How Far Are We? Systematic Evaluation of LLMs vs. Human Experts in Mathematical Contest in Modeling

Anonymous ACL submission

Abstract

Real-world problem solving often requires end-to-end workflows that go beyond isolated reasoning benchmarks, making it unclear how well large language models (LLMs) can handle such tasks. Mathematical modeling competitions provide a stringent testbed for this capability. We propose a problem-oriented, stage-wise evaluation framework that assesses LLMs across modeling stages using expert-verified criteria. On problems from the China Postgraduate Mathematical Contest in Modeling, our automatic scores align substantially better with independent human expert judgments than existing evaluation schemes. Applying this framework, we reveal a comprehension–execution gap in state-of-the-art LLMs: they perform well in early stages such as problem identification and formulation, but remain weak in execution-oriented stages including model solving, code implementation, and result analysis. These gaps persist with increased model scale and stem from insufficient specification, missing verification, and weak validation, allowing errors to propagate across stages. Our findings suggest that bridging this gap requires approaches beyond scaling alone.

1 Introduction

Real-world problem solving requires more than isolated reasoning: solutions must be aligned with task-specific objectives, constraints, assumptions, and validation criteria. Mathematical modeling competitions (Banerjee, 2021; Bender, 2000) make this challenge explicit. They present open-ended problems with task-specific objectives, constraints, and success criteria, and require participants to frame the problem, construct a model, implement a solution, and validate the results. Since such problems often admit multiple valid approaches rather than predefined answers, high-quality solutions are not judged merely by plausible reasoning or well-written reports, but by whether the chosen assumptions, model, implementation, and validation are jointly appropriate for the specific problem. This makes mathematical modeling competitions a natural testbed for evaluating expert-level modeling competence. Although LLMs perform strongly on reasoning benchmarks (DeepSeek-AI et al., 2025; Team, 2025; OpenAI, 2023; xAI, 2025; OpenAI, 2025), existing evaluations largely isolate reasoning, coding, or writing skills (Hendrycks et al., 2021; Wang et al., 2025a; Wu et al., 2025). As a result, it remains unclear whether LLMs can solve open-ended problems through a coherent process from problem framing and modeling decisions to implementation and validation.

Although recent studies have shown that LLMs can generate end-to-end solution pipelines and complete mathematical modeling reports (Liu et al., 2025; Chou et al., 2025), most prior evaluations rely on a fixed set of generic criteria applied uniformly across different modeling problems (Chou et al., 2025). This *problem-agnostic* evaluation paradigm is largely insensitive to the global consistency among modeling objectives, assumptions, decisions, and validation (see Appendix C.2 for concrete examples), and therefore primarily measures surface-level report presentation quality rather than successful problem solving. As a result, locally plausible yet fundamentally misaligned solutions can still receive high scores. Thus, existing evaluations are incapable of accurately establishing how close such performance is to human expert modeling.

To address this challenge, we propose an evaluation framework that accounts for both problem specificity and the fact that mathematical modeling unfolds through multiple interdependent stages. Specifically, we introduce a *problem-oriented, stage-wise* evaluation framework for full-process mathematical modeling. Our approach replaces static rubrics with dynamically instantiated eval-

uation schemas that explicitly encode each problem' s objectives, constraints, and success criteria. This enables faithful measurement of modeling competence on real-world tasks and supports interpretable comparisons between LLMs and human expert performance. To ground our evaluation in an expert-level and realistic setting, we use Postgraduate Mathematical Contest in Modeling (PMCM) as our primary evaluation benchmark because it represents genuine expert-level difficulty, with only about 1.0 to 1.5% of participants achieving gold medals (China Graduate Mathematical Contest in Modeling Organizing Committee). We investigate the following research questions:

RQ1: Does the framework align with expert judgment? By implementing expert-verified, subtask- and stage-specific evaluation criteria, our proposed framework generates substantially more reliable and discriminative scores compared to problem-agnostic rubrics. Correlation analysis against expert judgments reveals consistently stronger alignment across all evaluated modeling stages, thereby confirming the enhanced evaluation validity of our approach.

RQ2: Where do LLMs fall short across modeling stages? Building upon the proposed evaluation framework, we conduct a stage-wise analysis of LLM-generated mathematical modeling reports under a unified 10-point scoring scheme, encompassing the modeling workflow from problem framing to result validation. Our findings reveal stage-dependent performance disparities. LLMs demonstrate near-expert performance in problem understanding and initial framing. Conversely, performance in the core modeling stage, encompassing assumption formulation, model construction, and model solving, remains at an intermediate level of approximately 4–6 out of 10. More critically, LLMs exhibit poor performance in code implementation and result validation, with scores typically falling within the 2–3 range. These performance patterns persist even for state-of-the-art large-scale models, suggesting that increased model scale alone may not resolve the challenges associated with full-process mathematical modeling.

RQ3: What causes LLM failures in modeling? Our stage-wise failure analysis reveals that performance degradation is primarily attributed to execution-level deficiencies rather than erroneous problem understanding or modeling intent. In the development of assumptions and model construction, failures predominantly arise from inadequate specification, justification, and verification, even when high-level modeling choices appear reasonable. These early-stage shortcomings are rarely revisited in subsequent stages, resulting in downstream failures characterized by missing verification, incomplete procedures, and a lack of validation. Concurrently, these findings suggest that LLMs encounter less difficulty in generating plausible modeling concepts than in executing, validating, and refining them across stages.

In summary, we propose a reliable, problem-oriented evaluation framework that grounds assessment criteria in problem semantics and modeling stages, enabling faithful measurement of mathematical modeling competence. Using this framework, we conduct a systematic comparison between LLMs and human experts, revealing substantial performance gaps particularly in execution-oriented stages such as model solving, code implementation, and result analysis. We further trace these gaps to specific failure patterns including insufficient specification, missing verification, and lack of validation, which persist across stages without corrective feedback. These findings suggest that future improvements should focus on enhancing procedural rigor and stage-wise consistency rather than solely scaling model size, pointing toward process-aware approaches that prioritize executability and validation throughout the modeling pipeline.

2 Problem-Oriented Stage-Wise Evaluation Framework

To accurately assess the performance of mathematical modeling, we introduce a *problem-oriented, stage-wise* evaluation framework. The framework operates in three steps. First, we decompose each mathematical modeling problem into a set of subtasks, where each subtask corresponds to a self-contained modeling requirement. Second, we derive a problem-oriented semantic abstraction for each subtask, capturing its core modeling goal, expected outputs, key inputs and constraints, modeling type, and role in the overall pipeline. Third, we construct evaluation criteria aligned with canonical modeling stages that span the full modeling workflow from problem understanding to result validation. These steps incorporate expert-in-the-

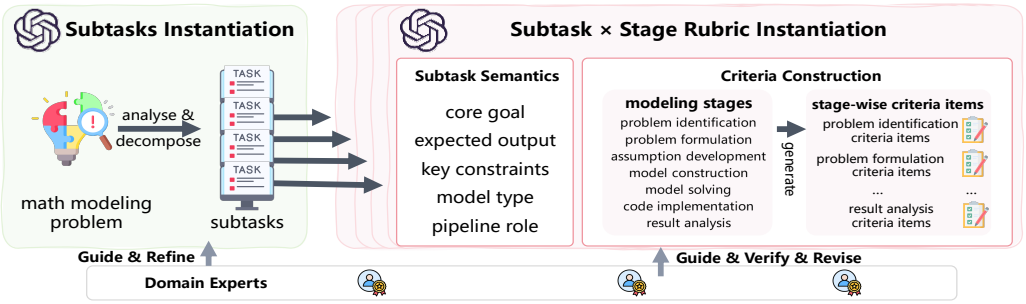

Figure 1: Overview of the problem-oriented, stage-wise evaluation framework. The LLM decomposes a mathematical modeling problem into subtasks, derives a problem-oriented semantic abstraction for each subtask, and instantiates stage-wise criteria. Domain experts provide lightweight guidance and verification, yielding a fixed, problem-specific rubric applied uniformly across models.

loop verification to ensure reliability, where domain experts review and refine the outputs before they are applied uniformly across all evaluated models.

2.1 Subtask Decomposition

A mathematical modeling problem often contains multiple requirements, making it difficult to evaluate the solution as a single unit. Following the structured decomposition strategy of Chou et al. (2025), we decompose each problem into a set of subtasks, where each subtask corresponds to a relatively self-contained modeling requirement. To improve decomposition quality, domain experts help refine the prompting guidelines on a small development set, ensuring that the generated subtasks better preserve the key objectives, constraints, and requirements in the original problem statements. The finalized prompting pipeline is then applied to all problems to obtain a common problem structure for subsequent report generation and evaluation. Examples are provided in Appendix B.

2.2 Stage-wise Evaluation Criteria Construction

For each decomposed subtask, we instantiate a stage-wise rubric by grounding criteria construction in the subtask's problem-oriented semantics, as illustrated in Figure 1. This semantic abstraction captures the subtask's core goal, expected output format, key inputs and constraints, modeling type (e.g., prediction, optimization, simulation, fitting, classification, evaluation, or hybrid), and role in the overall modeling pipeline. It makes explicit what the subtask is intended to achieve and what conditions a valid solution must satisfy. Second, we instantiate evaluation criteria for each subtask–stage pair. The modeling process is organized into seven stages, including *problem identification, problem formulation, assumption development, model construction, model solving, code implementation*, and *result analysis*, following standard decompositions of mathematical and scientific modeling practice (Bender, 2000; Banerjee, 2021). Detailed definitions are provided in Appendix C.1. For each stage, we prompt an LLM to generate checkable criteria grounded in the subtask semantics, so that each criterion specifies a necessary condition for successfully completing that stage of the subtask. Domain experts guide, verify, and revise the prompting instructions on a small development set to improve faithfulness and stage appropriateness. The finalized prompting pipeline is then applied to all subtask–stage pairs to produce the stage-wise rubric. A concrete example is provided in Appendix B.

Each generated criterion serves as an atomic unit for scoring. Model-generated reports are evaluated separately against every criterion associated with a given subtask and stage, yielding a structured score profile across the full modeling pipeline. Because the same criteria are applied uniformly to all models, this approach ensures consistent and comparable evaluation.

3 Related Work

Evaluation of LLMs on Reasoning Tasks A large body of work evaluates LLMs on reasoning-intensive benchmarks, including mathematical problem solving, logical reasoning, scientific question answering, and code generation (AIME, 2025; Wei et al., 2025; Song et al., 2025; Jimenez et al., 2024). These evaluations typically focus on answer correctness or the coherence of intermediate reasoning steps under well-defined problem set-

tings. While recent models achieve strong performance on narrowly scoped benchmarks, such evaluations largely isolate individual reasoning components and do not assess a model's ability to manage the full problem-solving lifecycle.

LLMs for End-to-End and Open-Ended Problem Solving Recent work has applied LLMs to more open-ended, multi-stage tasks such as scientific analysis, software development, and full mathematical modeling workflows (Rawat et al., 2025; Li et al., 2025; Applis et al., 2025; Qian et al., 2025). In these settings, evaluation remains largely coarse-grained, relying on holistic rubrics or preference judgments (Mohammadi et al., 2025; Yan et al., 2025; Wang et al., 2025b). As a result, existing evaluations provide limited insight into stage-wise execution and coordination, motivating the need for more fine-grained, problem-conditioned assessment frameworks.

4 Experimental Setup

Benchmark Construction Our benchmark is built from 97 graduate-level mathematical modeling problems from the China PMCM. Each problem requires an end-to-end modeling process, from problem formulation and assumption design to implementation and validation, making the dataset a challenging testbed for expert-level modeling evaluation. We convert the original multi-page PDFs into manually verified, machine-readable LaTeX problem statements and extract basic metadata for downstream analysis. Additional preprocessing details are provided in Appendix D.

Report Generation Setting To ensure comparable generation across LLMs, we use a unified mathematical modeling agent adapted from Liu et al. (2025). Each China PMCM problem is first decomposed into an ordered set of subtasks, as described in Section 2. For each subtask, the agent prompts an LLM to produce a subtask-level solution, and then aggregates the subtask outputs into a complete mathematical modeling report. To ensure fairness, the same subtask decomposition and ordering are used for all models on the same problem; only the underlying LLM is varied.

Within this fixed agent protocol, we evaluate a diverse set of LLMs to examine performance across different parameter scales and reasoning capabilities. Our evaluation includes multiple variants from the `Qwen` family ranging from 7B to 235B parameters (Yang et al., 2024, 2025a), the `DeepSeek-V3.2-Instruct` and `DeepSeek-V3.2-Thinking` models from the DeepSeek-V3.2-Exp series (DeepSeek-AI, 2025), and `o4-mini` (OpenAI, 2025). For all LLMs, we use the default generation hyperparameters provided by the corresponding model interface, including temperature and related sampling settings.

	Baseline	Ours
ICC(2,1) w.r.t. Expert	0.012	**0.673**

Table 1: Report-wise agreement between independent expert judgments and automatic scores.

Evaluation Protocol All generated reports are evaluated using the problem-oriented, stage-wise criteria instantiated in Section 2. In our implementation, we use `gpt-5.1` as the automatic evaluator for both our framework and the baseline rubric; this evaluator is not among the report-generation models, which helps avoid potential self-preference bias. The evaluator scores each report at the level of criterion items for every subtask–stage pair, producing a structured score profile across the full modeling process. We aggregate criterion-level scores within each stage and across subtasks to obtain **stage-wise scores** and **report-wise scores**. This hierarchy allows us to compare overall performance while preserving stage-wise diagnostic information.

To validate the reliability of this evaluation protocol, we compare automatic scores with independent expert judgments collected on the same subtask–stage criterion items. Agreement with expert judgments is measured using ICC(2,1), with detailed definitions provided in Appendix E. For comparison, we also implement the coarse-grained rubric of Liu et al. (2025) as a *baseline*, which evaluates each report using problem-agnostic dimensions including problem analysis, modeling rigor, practicality and scientificity, and result and bias analysis; the prompts used for this baseline comparison are provided in Appendix L.

5 Evaluation Reliability: Alignment with Expert Judgment

Before using our framework to evaluate LLM-generated reports, and thereby assess LLM modeling ability, we first examine whether its scores align with human expert judgment. A reliable eval-

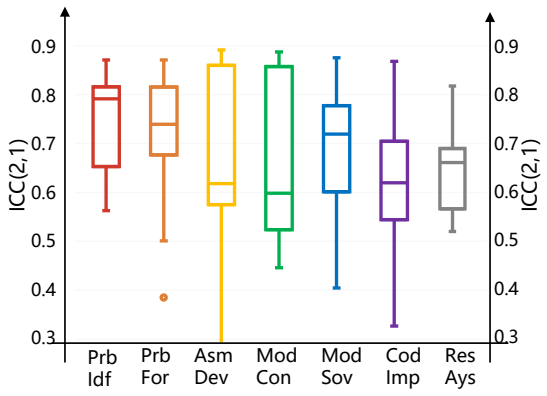

Figure 2: **Stage-wise agreement** between automatic and human expert scores measured by ICC(2,1).

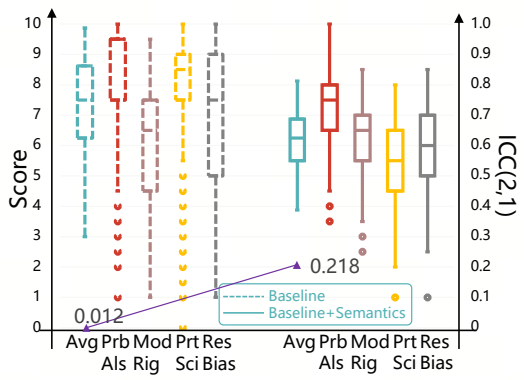

Figure 3: Ablation study on adding subtask semantics to the baseline rubric. BASELINE+SEMANTICS augments the original baseline rubric with subtask-level semantic information while preserving the baseline scoring dimensions.

uation should capture task-faithful modeling quality, rather than superficial report presentation. All results in this section are computed on the LLM-generated reports that received independent expert ratings.

To assess reliability, we recruit medal-winning mathematical modeling students as domain experts to independently score the same LLM-generated reports evaluated by our framework, using the subtask–stage criteria constructed in Section 2. These expert scores serve as the human reference signal. We compare expert and automatic scores in terms of both report-wise agreement, which reflects alignment on overall report quality, and stage-wise agreement, which reflects whether this alignment holds across modeling stages. Since the baseline rubric described in Section 4 produces only coarse-grained report-wise scores, we compare it with expert judgments only for report-wise agreement. We also evaluate an ablation, denoted as BASELINE+SEMANTICS, which augments the baseline rubric with subtask-level semantic information while preserving the baseline scoring dimensions.

Stage-wise framework matches expert judgment. Table 1 shows that the baseline rubric has almost no report-wise agreement with expert judgments, with an ICC of only 0.012. In contrast, our framework reaches an ICC of **0.673**, indicating substantially stronger alignment with expert assessment. This suggests that problem-oriented, fine-grained criteria better capture the modeling quality that experts consider important than coarse-grained holistic rubrics.

The same pattern holds beyond the overall report score. As shown in Figure 2, our framework maintains consistently strong stage-wise agreement with expert judgments across the modeling pipeline. The agreement is not limited to early, language-driven stages such as problem identification and formulation, but also extends to technically demanding stages including model construction, model solving, and code implementation. These results indicate that our framework provides reliable diagnostic scores at both report-wise and stage-wise granularities.

Subtask semantics are essential for problem-oriented evaluation. To isolate the contribution of problem-oriented conditioning, BASELINE+SEMANTICS provides the evaluator with each subtask's core goal, expected output, key constraints, modeling type, and pipeline role. As shown in Figure 3, it improves report-wise agreement with expert judgments from 0.012 to 0.218, indicating that subtask semantics help distinguish task-faithful modeling from superficially plausible exposition. The gain remains limited, however, because the stage-wise structure is weakened: the evaluator knows what a subtask should achieve, but does not separately verify formulation, assumptions, construction, solving, implementation, and analysis. This ablation suggests that problem-oriented semantics are necessary for evaluation validity, while explicit stage-wise criteria are needed for diagnostic reliability.

Stage-wise framework can distinguishes solution quality. We next examine whether our framework provides more discriminative assessment. As shown in Figure 4, baseline scores are

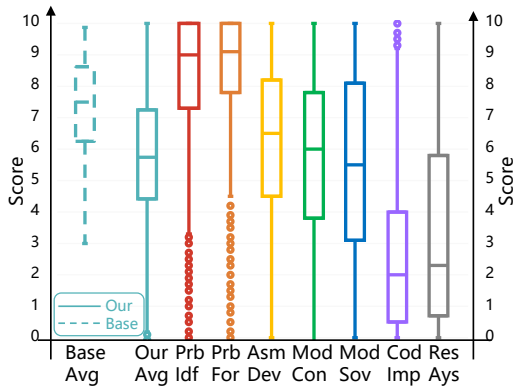

Figure 4: Distribution of report-wise and stage-wise evaluation scores under the baseline rubric and our framework.

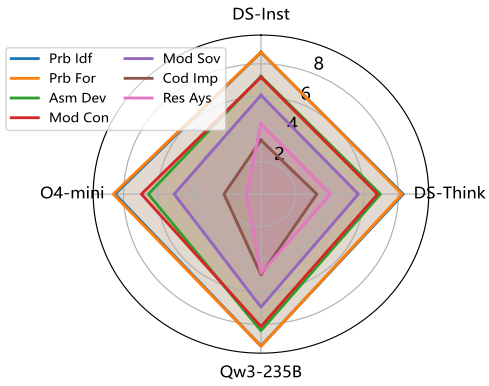

Figure 5: Stage-wise performance of four LLMs across the mathematical modeling pipeline. DS-Inst and DS-Think denote two DeepSeek-V3.2 variants, and Qw3-235B denotes a Qwen3-235B-Instruct model. Scores are averaged over reports and subtasks for each stage.

concentrated in a narrow mid-to-high range of approximately 6.5–8.5, even for reports with clear stage-specific deficiencies. Diagnostic cases in Appendix F further show that high baseline scores can coexist with violations of problem-grounded necessary conditions. In contrast, our subtask-stage evaluation produces more dispersed scores and prevents failures in model construction, solving, code implementation, or result analysis from being offset by fluent exposition elsewhere. This structure reduces score inflation and better distinguishes whether the modeling task has been substantively completed.

6 Stage-wise Performance: The Comprehension-Execution Divide

Having established the reliability of our evaluation framework, we now use it to assess how far current LLMs remain from expert-level mathematical modeling. Rather than relying on aggregate scores, we conduct a stage-wise analysis to identify where performance gaps emerge, revealing a clear comprehension–execution divide.

6.1 The Comprehension-Execution Gap

LLMs excel at comprehension but struggle with execution. We first examine how LLM performance varies across different modeling stages. For each modeling report, we compute a stage-wise score by averaging scores across all subtasks associated with that stage. We then aggregate these per-report stage-wise scores across all evaluated reports to obtain model-level stage-wise scores. Figure 5 reports mean stage-wise scores under our framework. Across all models, performance exhibits a clear monotonic decline as the modeling process progresses. Models achieve their highest scores in early, comprehension-oriented stages such as problem identification and problem formulation, but scores decrease steadily in subsequent stages. By the time models reach execution-heavy stages, most notably model solving, code implementation, and result analysis, average scores often fall to around 5.0 or below. This pattern indicates that, even when earlier reasoning steps are handled adequately, current LLMs struggle to sustain correctness and completeness through the later stages required for end-to-end mathematical modeling.

Understanding problems differs from modeling them. High performance in early stages primarily reflects the ability to interpret problem statements, restate objectives, and organize inputs into a coherent structure. However, these capabilities do not translate into reliable performance in stages that require producing checkable solutions, executing computational procedures, and validating results. The sharp score drop observed at the model-solving stage marks a critical transition point, beyond which purely language-driven reasoning becomes insufficient to support full-process modeling. As a result, current LLMs, when used in isolation, remain unable to consistently complete mathematical modeling tasks end to end.

6.2 Scaling Alone Is Not Enough

Given the substantial comprehension–execution gap identified above, a natural question arises: can these performance gaps be closed by scaling model size? To investigate this, we analyze models within

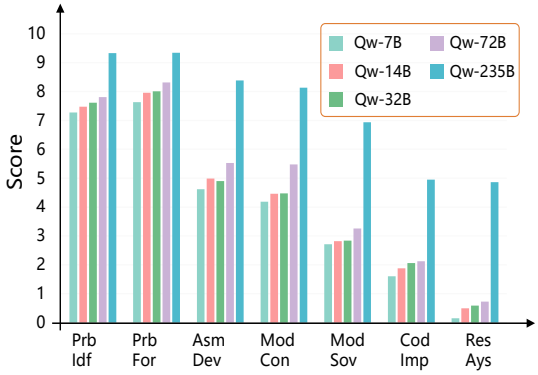

Figure 6: Stage-wise performance under within-family scaling for Qwen models. Bars correspond to `Qw-7B`–`Qw-72B` (Qwen2.5) and `Qw-235B` (Qwen3-235B).

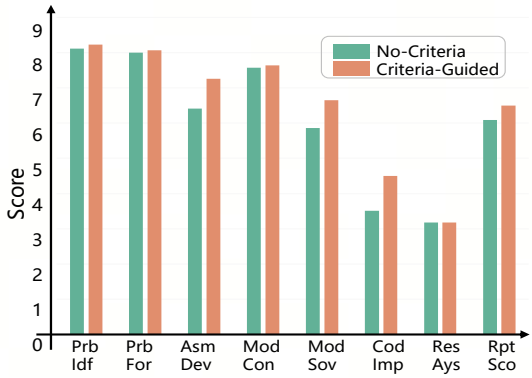

Figure 7: Effect of criterion-guided generation on 23 China PMCM problems using Qwen3-235B. **No-Criteria** denotes standard generation; **Criteria-Guided** provides problem-specific stage-wise criteria.

the Qwen family, ranging from 7B to 235B parameters. Figure 6 presents the stage-wise performance across different model scales.

Scaling improves comprehension but not execution. Figure 6 shows that increasing model scale mainly benefits early modeling stages, including problem identification and problem formulation. As model size grows from 7B to 72B, models become more reliable at understanding problem statements, extracting objectives, and organizing problem structures, although the gains gradually diminish at larger scales.

However, this improvement does not extend consistently to execution-oriented stages. Model solving, code implementation, and result analysis remain low and relatively flat across scales, suggesting that larger parameter counts alone do not yield reliable procedural execution, verification, or validation. These results indicate that the comprehension–execution divide persists under scaling, and that full mathematical modeling competence requires more than scaling language understanding.

6.3 Can Criteria Guide Generation?

Criteria substantially improve execution-oriented generation. Having shown that scaling alone does not close the execution gap, we further examine whether explicit stage-wise criteria can guide generation. We conduct a controlled subset experiment using Qwen3-235B on 23 randomly sampled China PMCM problems. The **No-Criteria** setting follows the standard report-generation pipeline, while the **Criteria-Guided** setting provides problem-specific stage-wise criteria before solving each subtask. Both settings are evaluated using the same fixed rubric, and the criteria contain no reference scores, gold solutions, or evaluator outputs. As shown in Figure 7, criterion-guided generation improves performance across most stages, with particularly clear gains in model solving and code implementation. This suggests that problem-specific stage-wise criteria can serve as effective process guidance, helping models translate task requirements into concrete modeling procedures and implementable solutions. Nevertheless, result analysis remains weak, indicating that criteria improve procedural execution but do not fully solve verification and validation.

7 Failure Analysis: Execution Deficiencies Dominate

Having identified substantial performance gaps in execution-oriented stages, we now investigate why these failures occur. We perform a stage-wise failure analysis by examining all subtasks whose stage-wise score falls below 8 points. For these low-scoring cases, we collect the associated evaluation comments and prompt an LLM to group recurring or semantically equivalent comments into coherent categories. The LLM then synthesizes each category into a representative failure cause for the corresponding stage. Table 2 summarizes the resulting failure causes, with detailed operational definitions provided in Appendix I. Within each stage, causes are ordered by their relative prevalence in reviewer comments, highlighting the dominant failure patterns at different points in the modeling pipeline.

Stage	Common Failure Causes
Assumption Development	Assumptions Not Checked; Missing Assumption Conditions; Assumption Impact Not Discussed; Hidden Idealized Assumptions; Unrealistic Assumptions
Model Construction	Incomplete Model Structure; Missing Model Derivation; Unclear Variables or Symbols; Solvability Not Justified; Model Deviates from Task Goal; Model-Assumption Conflict
Model Solving	No Checkable Solution; Key Solution Steps Missing; Solution Not Verified; Numerical Stability Not Analyzed; Computationally Infeasible; Inappropriate Solution Method
Code Implementation	Cannot be executed; Results Not Reproducible; No Code; Engineering or Numerical Risks; Code–Model Mismatch
Result Analysis	No Meaningful Results; No Validation or Comparison; Sensitivity Not Analyzed; Conclusions Not Supported; Results Miss the Goal; Limits Not Discussed

Table 2: Stage-specific failure causes across the mathematical modeling workflow. For each stage, failure causes are ordered from left to right by decreasing prevalence, based on their frequency of occurrence in reviewer comments associated with low-scoring cases. The detailed aggregation procedure of these frequencies is described in Appendix G.

Reasonable ideas, poor execution. Across all stages, the most prevalent failure causes are execution-level deficiencies rather than fundamentally flawed modeling ideas. LLMs generally propose reasonable high-level approaches, but fail to carry them out with sufficient rigor. In assumption development, models frequently state plausible assumptions without checking their validity, specifying necessary conditions, or analyzing their impact on downstream results. In model construction, models often produce structures that appear aligned with task objectives but lack complete derivations, clear variable definitions, or justification for solvability. These patterns reveal a consistent gap between modeling intent and modeling execution: LLMs can identify what to do, but struggle with how to do it properly.

Errors propagate without correction. Beyond individual stage failures, we observe that deficiencies introduced in earlier stages are rarely corrected in subsequent ones. Later stages such as model solving, code implementation, and result analysis are dominated by failures including solution not verified, no checkable solution, results not reproducible, and no validation or comparison. These downstream failures are consistent with earlier-stage issues such as underspecified assumptions and incomplete model formulations. This pattern suggests that LLMs treat intermediate artifacts as fixed inputs rather than revisiting or revising them based on later findings. As a result, early-stage weaknesses propagate and compound throughout the modeling pipeline.

Overall, these failure patterns reveal that current LLMs struggle to maintain procedural rigor across the full mathematical modeling workflow. While LLMs often exhibit reasonable modeling intent in early stages, they fail to sustain this quality through execution and validation. The core challenge is not generating plausible ideas, but carrying them out with sufficient specification, verification, and validation at each stage. These findings suggest that future improvements should target process-level consistency and self-correction mechanisms rather than focusing solely on enhancing individual stage capabilities.

8 Conclusion

This work investigates the extent to which LLMs can perform full-process mathematical modeling under expert-level competition settings. We propose a problem-oriented, stage-wise evaluation framework that grounds assessment criteria in problem semantics and modeling stages, enabling more reliable evaluation than problem-agnostic rubrics. Using this framework, we reveal a substantial comprehension–execution gap between LLMs and human experts, where LLMs perform well in understanding problems but struggle significantly in solving models, implementing code, and validating results. We further trace these gaps to specific failure patterns, showing that LLMs can generate reasonable modeling ideas but fail to carry them out with sufficient specification, verification, and validation, and that early-stage errors propagate throughout the pipeline without correction. These findings fundamentally reshape our understanding of how to advance LLMs toward expert-level problem solving. Rather than relying on model scaling alone, future progress requires new paradigms that bridge the comprehension-execution gap, such as process-aware reasoning and stage-wise self-correction.

9 Limitations

Our study is subject to the limitation that the proposed evaluation framework is instantiated and validated on mathematical modeling problems from the Postgraduate Mathematical Contest in Modeling (PMCM). While PMCM provides a realistic and expert-level setting that requires full-process reasoning, it represents a specific class of open-ended tasks with well-established modeling conventions and evaluation practices. As a result, the stage decomposition and evaluation criteria adopted in this work may not directly transfer to other end-to-end problem-solving domains with different structures or validation norms. In addition, the instantiation of problem-conditioned, stage-wise evaluation criteria relies on expert guidance to ensure faithful coverage of problem requirements and modeling semantics, which may limit scalability in settings where such expertise is unavailable. Extending the framework to a broader range of tasks while reducing reliance on expert intervention remains an important direction for future research.

References

AIME. 2025. Aime problems and solutions.

Leonhard Applis, Yuntong Zhang, Shanchao Liang, Nan Jiang, Lin Tan, and Abhik Roychoudhury. 2025. Unified software engineering agent as AI software engineer. *CoRR*, abs/2506.14683.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Ming-Hsuan Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. Qwen2.5-vl technical report. *CoRR*, abs/2502.13923.

Sandip Banerjee. 2021. *Mathematical modeling: models, analysis and applications*. Chapman and Hall/CRC.

Edward A Bender. 2000. *An introduction to mathematical modeling*. Courier Corporation.

China Graduate Mathematical Contest in Modeling Organizing Committee. **“华为杯”第二十二届中国研究生数学建模竞赛获奖名单公布** [award list of the 22nd china graduate mathematical contest in modeling]. Official announcement on the China Graduate Innovation Practice Competition website (in Chinese).

Jason Chou, Ao Liu, Yuchi Deng, Zhiying Zeng, Tao Zhang, Haotian Zhu, Jianwei Cai, Yue Mao, Chenchen Zhang, Lingyun Tan, Ziyan Xu, Bohui Zhai, Hengyi Liu, Speed Zhu, Wiggin Zhou, and Fengzong Lian. 2025. Autocodebench: Large language models are automatic code benchmark generators. *CoRR*, abs/2508.09101.

DeepSeek-AI. 2025. Deepseek-v3.2-exp: Boosting long-context efficiency with deepseek sparse attention.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Xuchen Li, Ruitao Wu, Xuanbo Liu, Xukai Wang, Jinbo Hu, Zhixin Bai, Bohan Zeng, Hao Liang, Leheng Chen, Mingrui Chen, Haitian Zhong, Xuanlin Yang, Xu-Yao Zhang, Liu Liu, Jia Li, Kaiqi Huang, Jiahao Xu, Haitao Mi, Wentao Zhang, and Bin Dong. 2025. Sciagent: A unified multi-agent system for generalistic scientific reasoning. *CoRR*, abs/2511.08151.

Fan Liu, Zherui Yang, Cancheng Liu, Tianrui Song, Xiaofeng Gao, and Hao Liu. 2025. Mm-agent: LLM as agents for real-world mathematical modeling problem. *CoRR*, abs/2505.14148.

Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. 2025. Evaluation and benchmarking of LLM agents: A survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, V.2, KDD 2025, Toronto ON, Canada, August 3-7, 2025*, pages 6129–6139. ACM.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

OpenAI. 2025. Gpt-5 system card. Technical report, OpenAI.

OpenAI. 2025. Introducing o3 and o4-mini. `https://openai.com/index/introducing-o3-and-o4-mini/`.

Cheng Qian, Hongyi Du, Hongru Wang, Xiusi Chen, Yuji Zhang, Avirup Sil, Chengxiang Zhai, Kathleen McKeown, and Heng Ji. 2025. Modelingagent: Bridging llms and mathematical modeling for real-world challenges. *Preprint*, arXiv:2505.15068.

Mrinal Rawat, Ambuje Gupta, Rushil Goomer, Alessandro Di Bari, Neha Gupta, and Roberto Pieraccini. 2025. Pre-act: Multi-step planning and reasoning improves acting in LLM agents. *CoRR*, abs/2505.09970.

Patrick E Shrout and Joseph L Fleiss. 1979. Intraclass correlations: uses in assessing rater reliability. *Psychological bulletin*, 86(2):420.

Zhangde Song, Jieyu Lu, Yuanqi Du, Botao Yu, Thomas M. Pruyn, Yue Huang, Kehan Guo, Xiuzhe Luo, Yuanhao Qu, Yi Qu, Yinkai Wang, Haorui Wang, Jeff Guo, Jingru Gan, Parshin Shojaee, Di Luo, Andres M Bran, Gen Li, Qiyuan Zhao, and 37 others. 2025. Evaluating large language models in scientific discovery. *Preprint*, arXiv:2512.15567.

Gemini Team. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *CoRR*, abs/2507.06261.

Zhexu Wang, Yiping Liu, Yejie Wang, Wenyang He, Bofei Gao, Muxi Diao, Yanxu Chen, Kelin Fu, Flood Sung, Zhilin Yang, Tianyu Liu, and Weiran Xu. 2025a. Ojbench: A competition level code benchmark for large language models. *CoRR*, abs/2506.16395.

Zi Wang, Xingqiao Wang, Sangah Lee, and Xiaowei Xu. 2025b. Arise: Agentic rubric-guided iterative survey engine for automated scholarly paper generation. *Preprint*, arXiv:2511.17689.

Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. 2025. Satbench: Benchmarking llms' logical reasoning via automated puzzle generation from SAT formulas. *CoRR*, abs/2505.14615.

Joseph P Weir. 2005. Quantifying test-retest reliability using the intraclass correlation coefficient and the sem. *The Journal of Strength & Conditioning Research*, 19(1):231–240.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, and Fei Huang. 2025. Writingbench: A comprehensive benchmark for generative writing. *CoRR*, abs/2503.05244.

xAI. 2025. Grok 4 model card. Technical report, xAI.

Xiangchao Yan, Shiyang Feng, Jiakang Yuan, Renqiu Xia, Bin Wang, Lei Bai, and Bo Zhang. 2025. SURVEYFORGE : On the outline heuristics, memory-driven generation, and multi-dimensional evaluation for automated survey writing. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 12444–12465. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 40 others. 2025a. Qwen3 technical report. *CoRR*, abs/2505.09388.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. 2025b. Optibench meets resocratic: Measure and improve llms for optimization modeling. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

A Ethical Considerations

We recruited three experts to perform the annotation tasks. All annotators are students majoring in mathematics and have received medals in mathematical modeling competitions, indicating recognized expertise in mathematical modeling. Among them, two are PhD candidates and one is a master' s student. All annotation procedures were conducted in accordance with institutional ethical guidelines and were approved by the Institutional Review Board (IRB).

B Examples of Subtask Instantiation from a Modeling Problem

To illustrate what constitutes a *subtask* in our evaluation framework, we present four representative subtasks instantiated from a real mathematical modeling problem on maneuvering target tracking. The purpose of these examples is not to reproduce a full solution, but to demonstrate how complex problem requirements are decomposed into self-contained modeling subtasks that admit stage-wise evaluation.

B.1 Original Problem Context (Excerpt).

The problem concerns tracking maneuvering targets using radar measurements. The provided datasets contain asynchronous observations from one or multiple radar sensors, with each observation specified in polar coordinates (range, azimuth, elevation, timestamp, and sensor identifier). The overall task involves coordinate transformation, uncertainty modeling, dynamic state estimation under maneuver, data association in multi-target scenarios, and long-horizon trajectory prediction.

B.2 Subtask 1: Measurement Preprocessing and Observation Uncertainty Modeling.

Subtask Definition. Transform raw radar observations from polar coordinates into a unified Cartesian coordinate system, quantify and propagate measurement uncertainty, handle asynchronous timestamps across sensors, detect and mark outliers, and output a time-ordered observation sequence suitable for downstream filtering and data association.

Subtask Scope and Inputs. The inputs to this subtask include raw radar measurements $(r, \mathrm{az}, \mathrm{el}, t, s)$, sensor metadata (sensor positions in geodetic or Earth-centered coordinates), and sensor-specific measurement error statistics (range and angular uncertainties). The subtask assumes known sensor locations and valid timestamps, while allowing observations to be asynchronous and heterogeneous across sensors.

Expected Outputs. The output is a structured, time-ordered observation sequence. Each record contains: (i) a Cartesian position estimate in a unified reference frame, (ii) an associated measurement covariance matrix representing propagated uncertainty, (iii) the observation timestamp and sensor identifier, and (iv) auxiliary metadata such as sensor geometry and quality or anomaly flags.

Role in the Modeling Pipeline. This subtask serves as the interface between raw sensor data and all subsequent modeling stages. Errors introduced here (e.g., incorrect coordinate conventions or missing uncertainty propagation) systematically affect downstream estimation and cannot be corrected later, making it a necessary and independently evaluable modeling requirement.

B.3 Subtask 2: Dynamic Model Design and State Estimation under Maneuver.

Subtask Definition. Design appropriate target motion models and estimate the time-varying target state (e.g., position and velocity) from noisy, nonlinear observations, while adapting to potential maneuvering behavior.

Subtask Scope and Inputs. The inputs include the preprocessed observation sequence from Subtask 1, together with modeling assumptions about target dynamics (e.g., constant velocity, constant acceleration, or stochastic acceleration models). Measurement functions and noise models are assumed known from preprocessing.

Expected Outputs. The output consists of a time-indexed sequence of state estimates and associated uncertainty measures (state covariance matrices). When multiple motion hypotheses are considered, the output may additionally include model probabilities or diagnostic indicators reflecting detected maneuvers.

Role in the Modeling Pipeline. This subtask constitutes the core estimation component of the tracking system. It translates observational data into latent target states and determines how quickly and reliably the tracker responds to maneuvers. Its performance directly affects the quality of subsequent data association and prediction tasks.

B.4 Subtask 3: Data Association and Track Management in Multi-Target Scenarios.

Subtask Definition. Associate incoming observations with existing target tracks, manage track initiation and termination, and maintain track continuity under ambiguous, missing, or overlapping measurements in multi-target environments.

Subtask Scope and Inputs. The inputs include predicted target states and uncertainties from Subtask 2, together with the current set of observations and their covariances from Subtask 1. The subtask may further rely on assumed detection probabilities and clutter characteristics.

Expected Outputs. The output includes updated target tracks with revised state estimates, measurement-to-track association decisions or probabilities, and track-level status indicators (e.g., confirmed, tentative, coasting, or terminated). In probabilistic formulations, association likelihoods or marginal probabilities are also produced.

Role in the Modeling Pipeline. This subtask ensures identity consistency over time and

prevents error accumulation caused by incorrect measurement-to-track assignments. Failures at this stage can lead to track swaps, fragmentation, or loss, which cannot be remedied by later prediction or smoothing.

B.5 Subtask 4: Trajectory Prediction and Outcome Estimation.

Subtask Definition. Predict future target motion beyond the observation window based on current state estimates and dynamic models, and estimate task-specific outcomes such as future trajectory envelopes or landing locations, together with associated uncertainty.

Subtask Scope and Inputs. The inputs consist of the latest state estimates and uncertainties from Subtask 2 (and, where relevant, track hypotheses from Subtask 3), along with assumed future dynamics and environmental constraints. This subtask may incorporate stochastic simulation or uncertainty propagation methods.

Expected Outputs. The output includes predicted trajectories or outcome distributions (e.g., expected landing points with confidence regions), temporal forecasts, and uncertainty summaries. In addition, computational cost or complexity estimates may be reported to assess feasibility under real-time constraints.

Role in the Modeling Pipeline. This subtask translates state estimation results into actionable predictions and decision-relevant quantities. It is the final consumer of all upstream modeling outputs and provides the basis for interpretation, planning, or strategic analysis.

Why These Constitute Subtasks. Each of the above subtasks addresses a distinct modeling requirement implied by the original problem, admits a complete modeling pipeline from formulation to output, produces well-defined artifacts consumed by downstream stages, and can be meaningfully evaluated using stage-aware criteria. Together, they illustrate how a complex modeling problem can be systematically decomposed into independently evaluable subtasks within our framework.

C Stage Definitions and Example

C.1 Stage Definitions

Problem Identification. Problem identification refers to the initial understanding of the real-world task, including recognizing the core objective, relevant entities, and success criteria implied by the problem description. At this stage, the modeler determines what is to be predicted, optimized, or explained, and distinguishes essential requirements from contextual background. Accurate problem identification is a prerequisite for all subsequent modeling decisions; misunderstandings at this stage fundamentally alter the nature of the task and cannot be compensated for by later technical sophistication.

Problem Formulation. Problem formulation translates the identified real-world task into a formal modeling problem. This includes defining inputs and outputs, specifying target variables, clarifying constraints, and determining the overall modeling scope. While problem identification focuses on understanding *what* is required, formulation focuses on deciding *how* the problem should be represented in analytical or mathematical terms. Errors at this stage often lead to internally consistent but misaligned solutions that fail to address the original task.

Assumption Development. Assumption development involves explicitly stating and justifying the simplifying assumptions necessary to make the problem tractable. These assumptions may concern data quality, noise characteristics, independence, stationarity, linearity, or domain-specific conditions. Assumptions play a central role in mathematical modeling by connecting real-world complexity to formal representations. Evaluating this stage is essential, as unstated, unjustified, or violated assumptions can invalidate an otherwise well-constructed model.

Model Construction. Model construction concerns the selection and specification of mathematical structures that operationalize the formulated problem under the stated assumptions. This includes defining equations, objective functions, probabilistic models, or algorithmic frameworks, as well as clarifying variables and parameters. This stage captures the core modeling choices and reflects whether the model structure is appropriate, coherent, and aligned with the problem formulation and assumptions.

Model Solving. Model solving addresses how the constructed model is analyzed or optimized to produce solutions. This includes deriving analytical solutions when possible, selecting numeri-

cal methods, specifying optimization procedures, and ensuring solvability. Even a well-designed model may fail at this stage if solution methods are ill-posed, unstable, or computationally infeasible. Separating model solving from model construction allows evaluation to distinguish conceptual modeling quality from solution execution quality.

Code Implementation. Code implementation refers to translating the proposed solution procedures into executable code. This stage includes algorithm realization, parameter initialization, data processing, and handling of practical issues such as numerical stability and runtime constraints. Implementation quality determines whether a model can be executed, reproduced, and applied in practice. Evaluating this stage is necessary because conceptual correctness does not guarantee executable or reliable implementation.

Result Analysis. Result analysis involves interpreting, validating, and critically examining the outputs produced by the implemented model. This includes checking correctness, assessing robustness, analyzing errors or uncertainty, comparing against baselines or expectations, and discussing limitations. Result analysis closes the modeling loop by determining whether the obtained results meaningfully address the original problem and whether conclusions are supported by evidence.

C.2 Example of Problem- and Stage-Aware Rubric Instantiation

Example Subtask. We present a concrete example of rubric instantiation for a representative modeling subtask involving *data preprocessing and observation uncertainty modeling*. The subtask requires transforming raw radar observations from multiple sensors into a unified local ENU coordinate system, propagating measurement uncertainty from polar to Cartesian space, handling asynchronous timestamps, and detecting abnormal observations. The following stage-wise criteria illustrate how evaluation is instantiated in a problem- and stage-conditioned manner.

In this example, the abstract dimensions discussed in the main text are instantiated concretely as follows. *Modeling objectives* correspond to the explicit preprocessing goals of coordinate unification, uncertainty propagation, and temporal alignment. *Modeling assumptions* include geometric conventions, noise distributions, independence hypotheses, and synchronization tolerances. *Modeling decisions* are reflected in concrete choices such as coordinate frames, uncertainty representations, linearization strategies, and handling of asynchronous observations. *Validation* refers to checks on numerical stability, uncertainty consistency, output completeness, and the impact of preprocessing on downstream tracking and filtering modules.

Stage: Problem Identification. Evaluation at this stage focuses on whether the model correctly recognizes the core objectives and requirements implied by the subtask.

- **Core processing objective recognition**: Whether the solution explicitly identifies the need to convert raw polar measurements into a local ENU coordinate system and to produce position estimates accompanied by quantified uncertainty.
- **Observation and uncertainty source identification**: Whether the solution correctly identifies the relevant observation components (range, azimuth, elevation, time, sensor identity) and acknowledges key sources of measurement uncertainty.
- **Downstream interface awareness**: Whether the solution recognizes that the processed outputs must be directly usable by subsequent filtering and data association modules, including requirements on data format, time consistency, and uncertainty representation.

Stage: Problem Formulation. This stage evaluates whether the identified task is reformulated into an explicit mathematical or algorithmic representation.

- **Measurement mapping specification**: Whether the solution clearly specifies the mathematical mapping from polar measurements to Cartesian coordinates under the given coordinate conventions.
- **Angle convention consistency**: Whether the solution explicitly addresses the transformation between problem-defined angle conventions and standard mathematical function inputs.
- **Uncertainty representation strategy**: Whether the solution articulates how measurement uncertainty is represented and

propagated, for example via linearization or by preserving uncertainty in the original measurement space.

Stage: Assumption Development. Evaluation at this stage focuses on whether simplifying assumptions are explicitly stated and justified.

- **Geometric and physical assumptions**: Whether assumptions regarding Earth model, sensor motion, and coordinate system stability are clearly stated.
- **Noise and independence assumptions**: Whether assumptions about measurement noise distributions and independence are made explicit and their limitations acknowledged.
- **Temporal synchronization assumptions**: Whether assumptions regarding acceptable time misalignment and synchronization strategy across sensors are clearly described.

Stage: Model Construction. This stage evaluates whether the formulated problem and assumptions are faithfully operationalized.

- **Coordinate transformation implementation**: Whether the solution specifies a complete and consistent transformation pipeline from geographic coordinates to the local ENU frame.
- **Uncertainty propagation realization**: Whether the solution defines how measurement uncertainty is operationalized in the constructed model, including Jacobian-based propagation or alternative representations.
- **Asynchronous observation handling**: Whether the model explicitly incorporates a strategy for handling observations arriving at different timestamps.

Stage: Model Solving. Evaluation at this stage focuses on whether the constructed model is solved or executed in a numerically and statistically sound manner.

- **Numerical stability considerations**: Whether the solution addresses potential numerical issues arising from extreme angles, near-singular configurations, or linearization errors.
- **Approximation validity checking**: Whether the solution considers the accuracy of linear approximations or validates them through alternative checks or simulations.
- **Time-consistency verification**: Whether the temporal handling of observations produces outputs consistent with the assumed time evolution.

Stage: Code Implementation. This stage evaluates whether the proposed procedures are realized in an executable and reproducible manner.

- **Executable transformation and propagation modules**: Whether the implementation correctly realizes coordinate transformations and uncertainty propagation as described.
- **Structured output generation**: Whether the implementation produces a structured observation sequence containing positions, uncertainties, timestamps, and sensor metadata.
- **Verification and testing practices**: Whether basic tests or checks are included to verify correctness of transformations, uncertainty propagation, and boundary cases.

Stage: Result Analysis. Evaluation at this stage focuses on whether the produced outputs are validated and interpreted.

- **Output completeness and consistency**: Whether the resulting observation sequence is complete, temporally ordered, and internally consistent.
- **Uncertainty consistency checking**: Whether the reported uncertainties are examined for statistical consistency or plausibility.
- **Impact on downstream modeling**: Whether the solution discusses how preprocessing choices and uncertainty handling affect subsequent filtering and data association performance.

D Dataset Preprocessing

Many original problem statements include complex mathematical notation and multi-page layouts. To produce a canonical, machine-processable representation while preserving mathematical fidelity, we adopt the following preprocessing pipeline:

1. **Page-level OCR-to-LaTeX conversion:** Each page of a raw problem PDF is converted into LaTeX markup using the multimodal model `Qwen2.5-VL-32B-Instruct`, instructed to output LaTeX-compatible representations for inline and display math, numbered equations, tables, and figures. Conversions are performed on a per-page basis to retain layout cues and to avoid cross-page concatenation errors.

2. **Stitching and structural normalization:** Page-level LaTeXfragments are programmatically concatenated into a single problem file. During stitching we normalize equation numbering, caption anchors, cross-references, and enumerations to produce a syntactically consistent LaTeXsource for each problem.

3. **Human verification and correction:** Each converted problem undergoes manual review by domain-expert annotators. Verification focuses on: (i) correct transcription of mathematical symbols and subscripts/superscripts, (ii) fidelity of displayed equations and matrices, (iii) correct mapping of figures/tables to captions, and (iv) semantic sanity checks (e.g., unit consistency where applicable). Annotators edit the LaTeXsource as needed to ensure near-perfect fidelity to the original problem statement.

4. **Canonicalization and metadata extraction:** We extract and store structured metadata for each problem (domain tags, problem type labels such as "prediction/optimization/simulation/evaluation", number of equations, presence of data tables, and estimated modeling complexity) to support downstream stratified analyses.

This pipeline yields a clean, human-verified LaTeXcorpus of graduate modeling problems that preserves symbolic content and is ready for model prompting, archival, and reproducible evaluation.

E Intraclass Correlation Coefficient: ICC(2,1)

E.1 Motivation and Use Case

To assess the reliability of human expert judgments in our evaluation framework, we employ the *intraclass correlation coefficient* (ICC) (Weir, 2005), which measures the degree of consistency or conformity among measurements made by multiple observers when assessing the same quantity. For continuous-valued ratings, ICC quantifies the proportion of total score variance that can be attributed to systematic differences among the evaluated items, as opposed to variability arising from individual raters or random measurement noise (Shrout and Fleiss, 1979).

In this work, ICC is used to evaluate the consistency among multiple expert judges when scoring the same set of mathematical modeling outputs. High ICC values indicate that the evaluation scores are stable and reproducible across judges, thereby supporting the validity of the proposed evaluation framework.

E.2 Model Selection: Why ICC(2,1)

We adopt ICC(2,1) following the classification of Shrout and Fleiss (1979), which corresponds to a two-way random-effects model with absolute agreement, based on single measurements. This choice is appropriate for our evaluation setting for the following reasons:

- **Two-way design.** Each modeling output is rated by the same group of judges, allowing score variance to be decomposed into item effects, rater effects, and residual error.

- **Random-effects assumption for raters.** Judges are treated as a random sample from a broader population of qualified experts. The goal is to assess whether evaluation scores would remain reliable if a different but comparable group of experts were used.

- **Absolute agreement.** ICC(2,1) measures absolute agreement rather than rank consistency. This is essential because the evaluation framework produces numerical scores whose absolute values are meaningful.

- **Single-measure reliability.** ICC(2,1) evaluates the reliability of a single judge' s score, providing a conservative estimate that aligns with realistic evaluation scenarios.

E.3 Statistical Formulation

Let n denote the number of items being rated (e.g., modeling reports), k the number of raters, and Y_{ij} the score assigned by rater j to item i. The two-way random-effects ANOVA model is defined as:

$$Y_{ij} = \mu + \alpha_i + \beta_j + \varepsilon_{ij},$$

where:

- μ is the grand mean,
- $\alpha_i \sim \mathcal{N}(0, \sigma_\alpha^2)$ represents the random effect of item i,
- $\beta_j \sim \mathcal{N}(0, \sigma_\beta^2)$ represents the random effect of rater j,
- $\varepsilon_{ij} \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ denotes residual error.

Based on the corresponding ANOVA mean squares:

- MS_R: mean square for items (rows),
- MS_C: mean square for raters (columns),
- MS_E: residual mean square,

the ICC(2,1) is computed as:

$$\text{ICC}(2,1) = \frac{MS_R - MS_E}{MS_R + (k-1)MS_E + \frac{k}{n}(MS_C - MS_E)}$$

F Diagnostic Case Analysis

This appendix presents three diagnostic case analysis to qualitatively examine the differences between the baseline rubric and our problem-oriented, stage-aware evaluation framework. The goal is to illustrate how high baseline scores may mask critical modeling deficiencies, and how such deficiencies are explicitly identified under our framework.

F.1 Case Overview

Table 3 summarizes the selected cases, reporting baseline evaluation scores along the dimensions of problem analysis, modeling rigor, practicality and scientificity, and result bias, together with the corresponding stage-wise scores under our framework.

F.2 Case 1: High Baseline Score Masking Fundamental Non-Executability

Baseline Evaluation Behavior. Under the baseline rubric, this report receives a high overall evaluation, with scores of 8.5 in problem analysis, modeling rigor, and result bias, and a score of 9.0 in practicality and scientificity (Table 3). The solution presents a comprehensive and technically sophisticated modeling pipeline for maneuvering target tracking, covering coordinate transformation, uncertainty propagation, state estimation, data association, and trajectory prediction. Advanced techniques such as IMM filtering, unscented Kalman filtering, probabilistic data association, and Monte Carlo-based prediction are correctly referenced and organized into a seemingly coherent workflow. The report is well structured, uses domain-appropriate terminology, and includes explicit assumptions and detailed mathematical formulations, which are strongly rewarded by the baseline evaluation scheme.

Problem-Grounded Modeling Deficiency. Despite its surface-level completeness, the solution fails to satisfy several necessary conditions for valid problem solving. Most critically, key components of the proposed pipeline are never operationalized or verified. Essential elements such as algorithmic parameter specification, statistical gating criteria, and consistency checks are either left unspecified or discussed only at a conceptual level. More importantly, the provided implementation code is not executable and produces no valid outputs. Consequently, none of the proposed modeling choices, including filtering, data association, or trajectory prediction, are empirically validated on the given datasets. Although the high-level modeling intent is reasonable, the solution does not constitute a functioning or verifiable problem-solving pipeline.

Stage-Aware Diagnosis. The proposed problem-oriented, stage-aware evaluation framework explicitly exposes these execution-level failures. While early stages such as problem identification and problem formulation consistently receive high scores (typically above 8), later stages exhibit a sharp performance drop. In particular, the code implementation stage receives scores of 0 across multiple subtasks, and result analysis scores fall as low as 0.4, indicating the absence of executable procedures, reproducible outputs, and quantitative validation (Table 3). Because the framework enforces stage-specific necessary conditions, these failures cannot be compensated by fluent exposition or sophisticated method descriptions. This case demonstrates how high baseline scores may mask fundamental non-executability, whereas the stage-aware framework reveals the gap between apparent modeling sophistication

F.3 Case 2: Methodological Sophistication Without Problem Compliance

Baseline Evaluation Behavior. Under the baseline rubric, this report receives consistently high

report-id	subtask-id	Prb Ana	Mod Rig	Pra Sci	Res Bias	Prb Idf	Prb Frm	Asm Dev	Mod Con	Mod Sol	Cod Imp	Res Ays
Case 1	1	8.5	8.5	9	8.5	10	9	8.5	8.5	4.3	0	1.5
	2					6.2	6.7	8.1	8.2	5.7	0	0.4
	3					8.5	8.6	8	8.9	4.8	2.5	5.6
	4					7.8	7.1	6.9	8.3	5.2	2.8	4.2
Case 2	1	9.5	8.5	9.5	9.5	4	5.5	1.5	3	2.5	0	1
	2					9	8.5	4.3	0.5	3.3	1.5	3.1
	3					8.4	9.6	6.1	8.7	5.3	5.9	2.8
	4					9.3	9.3	6.7	9	6	1.1	3
Case 3	1	9.5	8.5	9	10	9.5	10	9.5	6.7	3.5	1.5	1.5
	2					6.3	4.5	8.5	8	10	0.5	8
	3					10	10	9.3	9.8	9.1	2	8.9
	4					9.5	9.3	7.1	9.2	8.3	3.1	8.6

Table 3: Stage-wise evaluation scores for diagnostic cases under the baseline rubric and the proposed problem-conditioned framework. Each case consists of multiple subtasks, illustrating how high baseline scores may coexist with substantial deficiencies at specific modeling stages. Abbreviations: Prb Ana = Problem Analysis; Mod Rig = Modeling Rigor; Pra Sci = Practicality and Scientificity; Res Bias = Result Bias; Prb Idf = Problem Identification; Prb Frm = Problem Formulation; Asm Dev = Assumption Development; Mod Con = Model Construction; Mod Sol = Model Solving; Cod Imp = Code Implementation; Res Ays = Result Analysis.

scores across all baseline dimensions, including 9.5 in problem analysis, 8.5 in modeling rigor, 9.5 in practicality and scientificity, and 9.5 in result bias (Table 3). The solution presents a technically rich and well-articulated modeling pipeline for molecular property prediction, covering feature preprocessing, regression modeling, classification, uncertainty quantification, multi-objective screening, and interpretability analysis. Advanced techniques such as sparse group PLS, Gaussian process residual modeling, nested cross-validation, SMOTE-based class balancing, probabilistic calibration, Pareto optimization, and SHAP-based explanation are systematically introduced and mathematically formalized. The report is comprehensive, logically structured, and demonstrates strong familiarity with modern cheminformatics and machine learning practices, which aligns well with the surface-level criteria emphasized by the baseline evaluation.

Problem-Grounded Modeling Deficiency. Despite its methodological sophistication, the solution fails to satisfy key problem-specific modeling requirements. In multiple subtasks, the implemented procedures deviate substantially from the methods explicitly specified in the problem description. For example, fixed-threshold rules mandated by the task, such as missing-value and variance filtering criteria, correlation-based redundancy removal, and prescribed PLS–VIP–LASSO pipelines, are replaced by alternative strategies including adaptive quantile thresholds, hierarchical clustering, sparse group regularization, and hybrid models. While these alternatives are individually reasonable, they are not justified as equivalent to the required procedures, nor is their impact on task objectives analyzed. As a result, the solution effectively addresses a different modeling problem than the one posed, without verifying compliance with the original task constraints and acceptance criteria.

Stage-Aware Diagnosis. The stage-aware evaluation framework explicitly exposes this form of problem non-compliance. Although early stages such as problem identification and problem formulation often receive moderate to high scores (typically above 8 in several subtasks), sharp performance drops are observed in assumption development, model construction, and model solving. In particular, assumption development scores fall as low as 1.5, model construction scores drop to 0.5 in some subtasks, and model solving scores remain in the low range of approximately 2.5–3.3, reflecting unverified deviations from task-specified procedures (Table 3). Later stages further exhibit weak performance in code implementation and result analysis, with code implementation scores reaching 0 and result analysis scores as low as 1.0. By enforcing problem-conditioned, stage-specific criteria, the framework distinguishes methodological sophistication from task-faithful problem solving. This case demonstrates how baseline evaluation may reward technically advanced but misaligned solutions, whereas the proposed framework identifies failures arising from solving an unintended problem.

F.4 Case 3: Scientific Rigor Without Task-Level Executability

Baseline Evaluation Behavior. Under the baseline rubric, this report receives near-ceiling scores across all baseline dimensions, including 9.5 in problem analysis, 8.5 in modeling rigor, 9.0 in practicality and scientificity, and a perfect score of 10 in result bias (Table 3). The solution presents an exceptionally rigorous statistical modeling framework for longitudinal edema progression analysis, incorporating generalized additive mixed models, functional principal component analysis, clustering-based subgroup discovery, causal inference via propensity score weighting and marginal structural models, and joint modeling of hematoma–edema dynamics. Model formulations are mathematically precise, assumptions are explicitly discussed, and results are interpreted with strong domain knowledge and clinical insight. From a baseline perspective, the report closely resembles a polished scientific study and is therefore strongly rewarded.

Problem-Grounded Modeling Deficiency. Despite its high scientific quality, the solution fails to fully satisfy the task-specific execution requirements of the modeling problem. Several required deliverables are never concretely produced or verified. Although the report specifies detailed modeling pipelines and estimation procedures, it does not provide executable implementations, reproducible scripts, or concrete intermediate artifacts such as constructed long-format datasets, time-mapped measurement tables, or quality control logs. Moreover, key task-mandated outputs, including patient-level subgroup assignments, numerical result tables, and verifiable intermediate results, are either described only conceptually or reported in aggregated form. As a result, the solution remains at the level of a methodological proposal and empirical summary rather than a task-complete, checkable problem solution.

Stage-Aware Diagnosis. The stage-aware evaluation framework exposes this gap between analytical rigor and task-level execution. While early stages such as problem identification, problem formulation, and assumption development consistently receive very high scores (often above 9), later stages exhibit a pronounced performance drop. In particular, code implementation scores range from 1.5 to 3.1 across subtasks, and result analysis scores fall as low as 1.5, reflecting the absence of executable pipelines, reproducible artifacts, and task-specified outputs (Table 3). By enforcing problem-conditioned and stage-specific criteria, the framework prevents scientific sophistication from compensating for missing execution and verification. This case illustrates how baseline evaluation may reward analytically impressive but operationally incomplete solutions, whereas the proposed framework reliably identifies failures in task-level completion.

G Supplementary Analyses for RQ3

To better understand the reasons underlying low scores, we conduct a stage-wise analysis of failure causes across the full mathematical modeling workflow. For the *assumption development*, *model construction*, *model solving*, *code implementation*, and *result analysis* stages, we collect all reviewer comments associated with subtasks whose stage-wise scores fall below a fixed threshold (8 points). These comments are then summarized and grouped according to the recurring issues they explicitly identify, yielding a set of representative failure causes for each stage. Table 2 provides an overview of the resulting failure causes identified at each stage.

G.1 Assumption Development

Table 4 shows that low scores in the assumption development stage are primarily driven by execution-level shortcomings rather than fundamentally flawed assumptions. Across all models, *Not Checked* is by far the most prevalent failure cause, accounting for approximately 78–83% of low-scoring cases. This indicates that models frequently introduce assumptions that are, in principle, verifiable, yet fail to provide any accompanying tests, diagnostics, or concrete validation procedures.

The second most common issue is *Impact Ignored*, with frequencies ranging from roughly 42% to 66%. This suggests that even when assumptions are stated, models often do not analyze how these assumptions affect downstream modeling results or conclusions, nor do they discuss the conditions under which the assumptions may break down.

Missing Conditions also appears consistently across models (around 35–42%), revealing a systematic tendency to state assumptions without clearly specifying the prerequisites under which

Model	Not Checked	Miss Cond.	Impact Ignored	Hidden Ideal.	Unrealistic
Qwen3-235B	0.7000	0.5625	0.4250	0.4125	0.0000
DS-Inst	0.6743	0.5829	0.3771	0.4629	0.0057
DS-Think	0.6872	0.6089	0.4525	0.3799	0.0056
o4-mini	0.7412	0.6518	0.4441	0.4409	0.0128

Table 4: Distribution of primary failure causes in the assumption development stage. Model abbreviations: Qwen3-235B = Qwen3-235B-A22B-Instruct-2507; DS-Inst = DeepSeek-V3.2-Instruct; DS-Think = DeepSeek-V3.2-Thinking. Failure causes denote: *Not Checked* (Assumptions Not Checked), *Miss Cond.* (Missing Assumption Conditions), *Impact Ignored* (Assumption Impact Not Discussed), *Hidden Ideal.* (Hidden Idealized Assumptions), and *Unrealistic* (Unrealistic Assumptions). *Total* denotes the number of subtasks whose assumption-stage score is below 8.

they hold, such as required statistical properties, stability conditions, or parameter constraints. In contrast, *Idealized* assumptions occur at a moderate but non-negligible rate (approximately 11–17%), indicating that models sometimes rely on implicit idealized premises without explicitly acknowledging them.

Finally, *Unrealistic* assumptions are relatively rare, remaining below 6% for all models. This suggests that models generally avoid making assumptions that are obviously incompatible with the problem setting or underlying mechanisms.

Overall, failures in the assumption development stage are dominated by insufficient operationalization and reflection—namely, a lack of validation, condition specification, and impact analysis—rather than by the introduction of fundamentally unreasonable assumptions.

G.2 Model Construction

As shown in Table 5, low scores in the model construction stage are dominated by deficiencies in how models are formulated and justified, rather than by overtly incorrect modeling choices. The most prevalent failure cause is *Lack Derivation*, which accounts for more than half of the low-scoring cases across all models. This indicates that models often present formal equations or structural forms without clearly articulating the intermediate reasoning steps that connect the problem description to the proposed model.

Another major contributor is *Incomplete*, with proportions ranging from approximately 48% to 69%. This suggests that essential components of the model, such as key variables, constraints, or objective terms, are frequently missing or only partially specified, resulting in models that are structurally insufficient even if their overall direction is plausible.

By contrast, failures related to *Goal Mismatch*, *Variables*, *Solvability*, and *Assump. Conflict* occur at substantially lower rates. This implies that, once a model structure is proposed, it is usually aligned with the task objective and internally coherent, but often lacks completeness or adequate justification.

Overall, these findings indicate that low scores in the model construction stage are primarily driven by underspecified and weakly motivated formulations, rather than by fundamentally misguided modeling decisions. For completeness, Appendix I provides formal definitions of each failure cause used in this stage, together with representative reviewer comment excerpts that illustrate how these issues manifest in practice.

G.3 Model Solving

As shown in Table 6, low scores in the model solving stage are primarily driven by deficiencies in solution verification and procedural completeness, rather than by the absence of solution attempts. Across all models, *Unverified* is the most prevalent failure cause, reaching up to 72% for some models. This indicates that although solutions are often proposed, their feasibility, optimality, or convergence is rarely demonstrated or justified.

A second major contributor is *Missing Steps*, which appears in approximately 30–47% of low-scoring cases. This suggests that solution procedures are frequently described only at a high level, omitting critical intermediate steps needed to assess correctness or ensure reproducibility.

Failures related to *No Stability* are also non-negligible, reflecting a systematic lack of sensitivity analysis or numerical stability assessment, even when such analyses are necessary for validating the robustness of the proposed solution. By contrast, *Invalid Method*, *No Output*, and *Infeasible* occur at substantially lower rates. This implies that models generally avoid proposing solutions that are outright inappropriate, absent, or computationally im-

Model	Incomp. Struct.	Var. Unclear	Miss Deriv.	Solv. Unjust.	Goal Deviat.	Assump. Conflict
DS-Inst	0.6604	0.4717	0.2075	0.0377	0.0377	0.0566
DS-Think	0.8030	0.3182	0.0909	0.0303	0.1364	0.0303
Qwen3-235B	0.8125	0.3750	0.1250	0.0625	0.0625	0.0000
o4-mini	0.4560	0.6080	0.1840	0.1920	0.0320	0.0240

Table 5: Distribution of primary failure causes in the model construction stage. Model abbreviations: DS-Inst = DeepSeek-V3.2-Instruct; DS-Think = DeepSeek-V3.2-Thinking; Qwen3-235B = Qwen3-235B-A22B-Instruct-2507. Failure causes include: *Miss Deriv.* (Missing Model Derivation), *Incomp. Struct.* (Incomplete Model Structure), *Goal Deviat.* (Model Deviates from Task Goal), *Var. Unclear* (Unclear Variables or Symbols), *Solv. Unjust.* (Solvability Not Justified), and *Assump. Conflict* (Model–Assumption Conflict). *Total* denotes the number of subtasks whose model-construction score is below 8. f

Model	No Output	Steps Missing	Not Verified	No Stability	Infeasible	Wrong Method
DS-Inst	0.3592	0.4296	0.3099	0.3310	0.0704	0.0070
DS-Think	0.3896	0.4481	0.3506	0.2727	0.0844	0.0260
Qwen3-235B	0.5287	0.3333	0.2414	0.1954	0.0345	0.0115
o4-mini	0.5855	0.2566	0.5263	0.2730	0.0658	0.0197

Table 6: Distribution of primary failure causes in the model solving stage. Model abbreviations: DS-Inst = DeepSeek-V3.2-Instruct; DS-Think = DeepSeek-V3.2-Thinking; Qwen3-235B = Qwen3-235B-A22B-Instruct-2507. Failure causes include: *No Output* (No Checkable Solution), *Steps Missing* (Key Solution Steps Missing), *Not Verified* (Solution Not Verified), *No Stability* (Numerical Stability Not Analyzed), *Infeasible* (Computationally Infeasible), and *Wrong Method* (Inappropriate Solution Method). *Total* denotes the number of subtasks whose model-solving score is below 8.

practical, but often fail to rigorously justify the solutions they do present.

Overall, low scores in the model solving stage arise not from missing solutions, but from insufficient verification, incomplete procedural exposition, and limited robustness analysis. For clarity and reproducibility, Appendix I provides formal definitions of each failure cause used in this stage, together with representative reviewer comment excerpts that illustrate how these issues manifest in practice.

G.4 Code Implementation

Table 7 indicates that low scores in the code implementation stage are overwhelmingly driven by reproducibility and executability issues, rather than by discrepancies between the implementation and the underlying model. Across all models, the most dominant failure cause is *Results Not Reprod.*, accounting for approximately 74–79% of low-scoring cases. This suggests that, even when code is provided, the reported results cannot be reliably reproduced due to missing configuration details, unstable execution behavior, or insufficient documentation of experimental settings.

A second major contributor is *No Executable*, which appears at particularly high rates for some models. This reflects a frequent absence of inspectable or runnable implementations, indicating that solutions are often described at a conceptual or pseudo-code level without providing executable artifacts that allow verification. In addition, *Missing Modules* occurs consistently across models (around 29–38%), revealing that critical components of the implementation pipeline, such as key functions, data processing steps, or parameter initialization routines, are often omitted or left unspecified.

By contrast, failures related to *Engineering Risks* appear at moderate levels, suggesting occasional issues such as numerical instability or unsafe engineering practices. Meanwhile, *Model Mismatch* and *Code Not Reprod.* occur at very low rates, indicating that implementations are generally aligned with the intended model structure and that the code itself is usually provided in a consistent form. Taken together, these results imply that low scores in the implementation stage stem primarily from insufficient support for execution and reproducibility, rather than from fundamental errors in translating models into code.

For completeness and transparency, Appendix I provides formal definitions of each failure cause used in this stage, along with representative reviewer comment excerpts that illustrate how these implementation-level issues arise in practice.

Model	Results N/R	No Code	Miss Comp.	Eng./Num. Risk	Code–Model Mis.
DS-Inst	0.7564	0.5782	0.1564	0.0800	0.0036
DS-Think	0.7600	0.6133	0.2100	0.0933	0.0000
Qwen3-235B	0.7238	0.6000	0.2333	0.1143	0.0048
o4-mini	0.7646	0.8915	0.3175	0.0556	0.0026

Table 7: Distribution of primary failure causes in the code implementation stage. Model abbreviations: DS-Inst = DeepSeek-V3.2-Instruct; DS-Think = DeepSeek-V3.2-Thinking; Qwen3-235B = Qwen3-235B-A22B-Instruct-2507. Failure causes include: *Results N/R* (Results Not Reproducible), *No Code* (No Usable Code), *Miss Comp.* (Missing Key Code Components), *Eng./Num. Risk* (Engineering or Numerical Risks), and *Code–Model Mis.* (Code–Model Mismatch). *Total* denotes the number of subtasks whose code-implementation score is below 8.

G.5 Result Analysis

As shown in Table 8, low scores in the result analysis stage are predominantly driven by insufficient empirical validation rather than by isolated presentation errors. Across all models, *No Validation* is the most frequent failure cause, exceeding 65% in every case. This indicates that results are often reported without accompanying baselines, comparative evaluations, or validation experiments that would substantiate the claimed conclusions.

A second prominent issue is *Weak Evidence*, with substantial proportions across models. This suggests that conclusions are frequently overstated relative to the strength or completeness of the reported results. In addition, *No Results* remains non-negligible, particularly for some models, reflecting cases where outputs are either absent or insufficiently structured to support meaningful analysis.

Failures related to *No Sensitivity* and *Goal Misaligned* occur at moderate levels, indicating that models often do not analyze how results depend on assumptions or parameters, nor do they consistently align reported outcomes with the stated modeling objectives. By contrast, *No Limits* appears infrequently, implying that most submissions do not progress to a stage where applicability boundaries or failure conditions are explicitly discussed.

Overall, low scores in the result analysis stage primarily reflect a disconnect between reported outcomes and rigorous empirical grounding, rather than a lack of attempted analysis. For clarity and reproducibility, Appendix I provides formal definitions of each failure cause used in this stage, along with representative reviewer comment excerpts illustrating how these issues manifest in practice.

G.6 Summary of Stage-wise Failure Characteristics

Across stages, a consistent pattern emerges: low scores are rarely caused by fundamentally invalid ideas, but instead by insufficient specification, justification, validation, and reproducibility. Early-stage failures tend to involve underspecified assumptions and incomplete model derivations, while later-stage failures increasingly reflect missing verification, weak empirical support, and reproducibility issues. This progression suggests that errors propagate through the modeling pipeline, accumulating rather than being corrected in subsequent stages.

H Distribution of China Postgraduate Mathematical Contest in Modeling Problems

The dataset consists of 97 problems drawn from the China Postgraduate Mathematical Contest in Modeling (China PMCM), which cover a wide range of real-world modeling scenarios. To provide a descriptive overview of the modeling tasks used in our evaluation, we analyze their distribution across predefined mathematical method categories and application domains. The *mathematical method categories* describe the primary modeling techniques and mathematical structures required to formulate and solve a problem. Specifically, we consider differential equation models, optimization models, statistical models, probabilistic and stochastic models, discrete and graph-based models, simulation-based models, data-driven and machine learning models, as well as hybrid modeling approaches that combine multiple methods. The *application domains characterize* the real-world systems or contexts from which the modeling problems arise. We adopt a coarse-grained domain taxonomy that includes physical and natural systems, engineering and industrial systems, information and communication systems, biological and life systems, healthcare and public health systems, environmental and resource systems, economic and financial systems, social and public systems, and

Model	No Valid.	No Results	No Sensit.	Weak Concl.	Goal Miss.	No Limits
DS-Inst	0.5417	0.4722	0.3565	0.2639	0.0417	0.0463
DS-Think	0.6161	0.4420	0.3839	0.2991	0.0536	0.0313
Qwen3-235B	0.6182	0.5818	0.3333	0.2667	0.0061	0.0364
o4-mini	0.6183	0.7984	0.3495	0.2957	0.0323	0.0349

Table 8: Distribution of primary failure causes in the result analysis stage. Model abbreviations: DS-Inst = DeepSeek-V3.2-Instruct; DS-Think = DeepSeek-V3.2-Thinking; Qwen3-235B = Qwen3-235B-A22B-Instruct-2507. Failure causes include: *No Valid.* (No Validation or Comparison), *No Results* (No Meaningful Results), *No Sensit.* (Sensitivity Not Analyzed), *Weak Concl.* (Conclusions Not Supported), *Goal Miss.* (Results Miss the Goal), and *No Limits* (Limits Not Discussed). *Total* denotes the number of subtasks whose result-analysis score is below 8.

urban and transportation systems.

Each modeling problem is annotated from these two complementary perspectives. Because mathematical modeling problems are inherently multifaceted, a single problem may involve multiple mathematical methods and span multiple application domains. Figure 8 visualizes the resulting co-occurrence statistics using a heatmap representation. Rather than enforcing exclusive assignments, we adopt a multi-label annotation scheme to reflect the integrated nature of real-world modeling tasks. Accordingly, the reported counts indicate co-occurrence frequencies between method and domain categories and should be interpreted as descriptive statistics rather than mutually exclusive frequencies. This analysis is intended to contextualize the diversity of the evaluation tasks and does not imply any causal or performance-related relationships. The distribution further shows that problems associated with engineering and industrial systems account for a substantial proportion of the dataset. This reflects the practical orientation of the China PMCM, where many tasks are grounded in real-world engineering scenarios involving system design, resource allocation, process optimization, and operational decision-making.

Correspondingly, optimization models and statistical models appear with relatively high frequency among the mathematical method categories. Optimization models are commonly employed to formalize decision-making objectives under practical constraints, which are prevalent in engineering and industrial contexts. Statistical models, in turn, are frequently used to handle uncertainty, variability, and data-driven inference arising from real-world measurements and observations. These patterns are consistent with the applied nature of the competition problems and provide contextual insight into the methodological and domain diversity of the evaluation tasks, rather than indicating any methodological preference or performance implication.

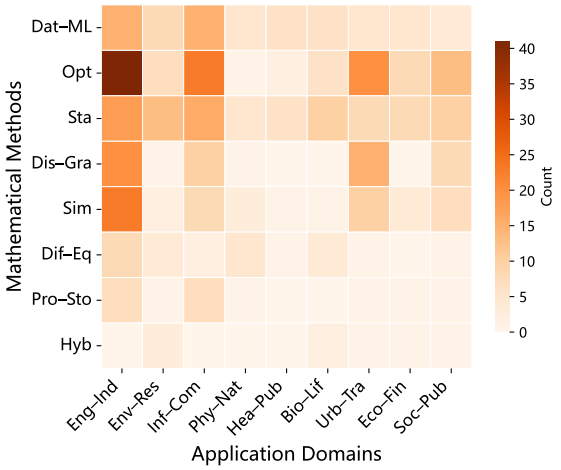

Figure 8: Heatmap of co-occurrence between mathematical method categories and application domains. Color intensity indicates the number of modeling problems associated with each method–domain pair. Each problem may belong to multiple mathematical methods and application domains; counts therefore reflect multi-label co-occurrences. Abbreviations are used for compact visualization: Dat–ML (Data-driven and Machine Learning), Opt (Optimization), Sta (Statistical), Dis–Gra (Discrete and Graph-based), Sim (Simulation-based), Dif–Eq (Differential Equation), Pro–Sto (Probabilistic and Stochastic), Hyb (Hybrid); Eng–Ind (Engineering and Industrial), Env–Res (Environmental and Resource), Inf–Com (Information and Communication), Phy–Nat (Physical and Natural), Hea–Pub (Healthcare and Public Health), Bio–Lif (Biological and Life), Urb–Tra (Urban and Transportation), Eco–Fin (Economic and Financial), and Soc–Pub (Social and Public).

I Operational Definitions of Failure Dimensions

Table 9: Operational Definitions of Stage-Specific Failure Causes Corresponding to Table 2

Stage	Failure Cause	Operational Definition
Assumption Development	Assumptions Not Checked	Assumptions that are in principle testable are stated but no validation, diagnostic procedure, or empirical check is provided.
Assumption Development	Missing Assumption Conditions	Necessary conditions under which an assumption holds (e.g., constraints, statistical tests, parameter bounds) are not explicitly specified.
Assumption Development	Assumption Impact Not Discussed	The influence of assumptions on model behavior, results, or conclusions is not analyzed or discussed.
Assumption Development	Hidden Idealized Assumptions	Idealized premises (e.g., noise-free data, full observability, stationarity) are implicitly adopted without being explicitly declared.
Assumption Development	Unrealistic Assumptions	Assumptions are incompatible with the problem context, data-generating process, or domain knowledge, even if explicitly stated.
Model Construction	Incomplete Model Structure	The model lacks essential components such as key variables, constraints, objective functions, or state relationships required to represent the task.
Model Construction	Missing Model Derivation	The transition from problem description to formal model lacks sufficient derivation steps or logical justification.
Model Construction	Unclear Variables or Symbols	Variables, symbols, or dimensional definitions are ambiguous, inconsistent, or insufficiently specified.
Model Construction	Solvability Not Justified	The mathematical feasibility, identifiability, or solvability of the model is not established or argued.
Model Construction	Model Deviates from Task Goal	The chosen model formulation or abstraction fails to align with the stated modeling objectives or task requirements.
Model Construction	Model–Assumption Conflict	The formal model contradicts, ignores, or fails to operationalize the stated assumptions.
Model Solving	No Checkable Solution	No numerical solution, concrete output, or verifiable result is provided for the constructed model.
Model Solving	Key Solution Steps Missing	Critical derivation steps, algorithmic procedures, or implementation details required to reproduce the solution are absent.
Model Solving	Solution Not Verified	The correctness, feasibility, convergence, or optimality of the solution is not validated or demonstrated.
Model Solving	Numerical Stability Not Analyzed	Sensitivity, robustness, or numerical stability with respect to parameters or numerical settings is not examined.
Model Solving	Computationally Infeasible	The proposed solution approach is computationally impractical at the intended scale, without mitigation or discussion.
Model Solving	Inappropriate Solution Method	The solution method or algorithm is theoretically unsuitable for the given model structure.

Stage	Failure Cause	Operational Definition
Code Implementation	Cannot Be Executed	The provided code or implementation cannot be executed due to errors, missing components, or invalid configurations.
Code Implementation	Results Not Reproducible	The implementation lacks sufficient information (e.g., parameters, seeds, environment details) to reproduce reported results.
Code Implementation	No Code	No executable code, pseudocode, or inspectable implementation is provided.
Code Implementation	Engineering or Numerical Risks	The implementation exhibits serious issues related to numerical stability, efficiency, resource usage, or error handling.
Code Implementation	Code–Model Mismatch	The implementation does not faithfully realize the mathematical model or solution logic.
Result Analysis	No Meaningful Results	No analyzable numerical results, experimental outputs, or substantive analysis targets are presented.
Result Analysis	No Validation or Comparison	No baseline comparison, validation experiment, or reference analysis is conducted to contextualize results.
Result Analysis	Sensitivity Not Analyzed	Dependence of results on key assumptions, parameters, or inputs is not examined.
Result Analysis	Conclusions Not Supported	The reported results are insufficient to justify the stated conclusions.
Result Analysis	Results Miss the Goal	The analysis fails to address the original modeling objectives or evaluation criteria.
Result Analysis	Limits Not Discussed	The scope of applicability, failure conditions, or limitations of the results are not discussed.

J Criteria Generation

Evaluation Criterias Prompt Design

背景信息:
{{background}}
问题描述:
{{question}}

你现在的任务是：针对一个已明确的数学建模子任务，生成 基于任务语义理解的细粒度评分细则。

{% if previous_subtasks %}
依赖子任务: {{previous_subtasks}}
{% endif %}
当前子任务:
{{subtask}}

——任务说明（请严格遵守）——
你的任务是：针对上面已明确的 单一数学建模子任务，生成 基于任务语义理解的细粒度评分细则（JSON），具体要求如下。

一、先进行 任务语义理解与抽象（输出项位于 `task_understanding`**）:**

1. `core_goal`: **一句话概括该子任务的 核心建模目标，必须包含子任务中的关键动作词**

（如“预测”“优化”“拟合”“评估”）和对象（如“AQI”“功率曲线”“成本”）。

2. expected_output：明确列出预期输出的形式（数值序列、最优解、图表、报表、模型文件、代码等）。

3. key_inputs_constraints：列出具体输入变量名 / 数据类型 / 必要前置条件 / 关键约束（至少 3 项；若不足请写“无”并说明原因）。

4. modeling_type：从 {预测/优化/仿真/拟合/分类/评价/混合} 中选择最贴切的一项（可复合，如“预测 + 优化”）。

5. role_in_pipeline：说明该子任务在整体建模流程中的定位，并明确其直接依赖的前序任务与后续任务（例如“作为特征工程后、模型训练前的子任务”）。

二、基于上述 task_understanding，为以下 7 个维度生成细化评分细则（输出项位于 evaluation_criteria）：

- 维度列表（顺序不可更改）：问题识别、问题复述、假设建立、模型构建、模型求解、代码实现、结果分析。
- 每个维度必须包含 3–5 个具体 sub_criteria，且每个 sub_criteria 必须包含以下字段：
  - sub_criteria：评分点主题（必须包含任务关键词）
  - description：与该子任务语义深度绑定的评分说明
  - score：该评分点在该维度中的整数分值（同一维度内总分必须等于 100）
  - evaluation_focus：考察能力标签（不超过 6 个词）
  - scoring_hint：区分高分 / 中分 / 低分的可判定标准
- 评分点撰写规则：
  1. 必须直接使用 task_understanding 中的任务关键词。
  2. 禁止使用泛化或空泛表达（如“目标是否明确”）。
  3. scoring_hint 应给出可操作、可检验的判定标准；优先使用量化指标。
  4. 若某维度不适用，该维度应返回空数组，并增加 not_applicable_reason 说明原因。

三、输出格式与其它要求：

- 只返回一个合法的 JSON 对象，不得输出任何多余文字
- JSON 编码必须为 UTF-8
- 字段顺序必须为（task_understanding 在前，evaluation_criteria 在后）

输出格式示例如下：

```json
{
  "task_understanding": {
    "core_goal": "...",
    "expected_output": "...",
    "key_inputs_constraints": "...",
    "modeling_type": "...",
    "role_in_pipeline": "..."
```

```
  },
  "evaluation_criteria": {
    "问题识别": [],
    "问题复述": [],
    "假设建立": [],
    "模型构建": [],
    "模型求解": [],
    "代码实现": [],
    "结果分析": []
  }
}
```

Evaluation Criteria Prompt Design

Background Information:
{{background}}
Problem Description:
{{question}}

Your task is to generate **fine-grained evaluation criteria grounded in task-level semantic understanding** for a clearly specified mathematical modeling subtask.

{% if previous_subtasks %}
Dependent Subtasks: {{previous_subtasks}}
{% endif %}
Current Subtask:
{{subtask}}

——Task Instructions (Must Be Strictly Followed) ——
Your task is to generate **fine-grained, task-aware evaluation criteria (in JSON format)** for the above specified **single mathematical modeling subtask**, according to the requirements below.

I. Task Semantic Understanding and Abstraction (Output under the field `task_understanding`):

1. `core_goal`: Summarize the **core modeling objective** of the subtask in one sentence. The description must include key action verbs (e.g., "predict", "optimize", "fit", "evaluate") and task-specific objects (e.g., "AQI", "power curve", "cost").

2. `expected_output`: Clearly specify the expected form of the output (e.g., numerical sequences, optimal solutions, plots, reports, model files, or code).

3. `key_inputs_constraints`: List the **concrete input variables / data types / required preconditions / key constraints** (at least three items; if fewer, explicitly state "none" and explain why).

4. `modeling_type`: Select the most appropriate type from `{prediction/optimization/simulation/fitting/classification/evaluation/hybrid}`. Composite types (e.g., "prediction+optimization") are allowed.

5. `role_in_pipeline`: Describe the role of this subtask within the overall modeling pipeline, and explicitly state its direct upstream and downstream dependencies (e.g., "performed after feature engineering and before model training").

II. Fine-Grained Evaluation Criteria Construction
Based on the above `task_understanding`, generate evaluation criteria for the following **seven dimensions** (output under `evaluation_criteria`):

- Dimension list (order must not be changed): Problem Identification, Problem Restatement, Assumption Formulation, Model Construction, Model Solving, Code Implementation, Result Analysis.
- Each dimension must contain **3–5 specific sub_criteria**. Each `sub_criteria` must include the following fields:
  - `sub_criteria`: the evaluation focus (must include task-specific keywords)
  - `description`: a description tightly grounded in the semantics of this subtask
  - `score`: the integer score assigned to this criterion (the total score within each dimension must equal 100)
  - `evaluation_focus`: a short capability label (no more than six words)
  - `scoring_hint`: concrete, checkable indicators distinguishing high / medium / low scores
- Rules for writing evaluation criteria:
  1. Task-specific keywords must be explicitly reused from `task_understanding`.
  2. Generic or vague criteria (e.g., "whether the goal is clear") are not allowed.
  3. `scoring_hint` should provide operational and verifiable standards; quantitative thresholds are preferred whenever possible.
  4. If a dimension is not applicable, return an empty list for that dimension and add a `not_applicable_reason` explaining why.

III. Output Format and Additional Requirements:

- Only return a single valid JSON object; no extra text is allowed.
- The JSON encoding must be UTF-8.
- Field order must be preserved (`task_understanding` first, followed by `evaluation_criteria`).

An example output format is shown below:

```json
{
  "task_understanding": {
    "core_goal": "...",
    "expected_output": "...",
    "key_inputs_constraints": "...",
    "modeling_type": "...",
    "role_in_pipeline": "..."
  },
  "evaluation_criteria": {
    "Problem Identification": [],
    "Problem Restatement": [],
    "Assumption Formulation": [],
    "Model Construction": [],
```

```
    "Model Solving": [],
    "Code Implementation": [],
    "Result Analysis": []
  }
}
```

K Evaluation Prompt

Stage-wise Evaluation

当前子问题:
{{subproblem}}

数学建模论文内容:
{{report_content}}

评分准则（JSON）:

```json
{{report_criteria}}
```

——评分任务说明（请严格遵守）——

你将作为一名 严格、审慎、完全基于证据的数学建模竞赛评审专家，依据给定的评分准则，对上述论文内容进行 逐维度、逐评分项的客观评分。

核心原则（必须遵守）:

- **只依据论文文本，绝不脑补。**
- **论文未明确写出的内容一律视为不存在。**
- **论文仅模糊提及、缺乏公式、推导、结构说明或方法细节的内容，视为“部分满足”或“不满足”。**
- **若模型结构、算法流程、关键假设或变量定义未明确给出，视为“不满足”。**

逐项评分规则:

1. **按照评分准则中给定的顺序，对每一个评分项分别给出:**
   - **评价等级（六级）**
   - **得分（必须落在该项规定的分值区间内）**
   - **基于论文文本的证据引用**
   - **明确判断理由（满足 / 基本满足 / 部分满足 / 基本不满足 / 不满足 / 完全不满足）**
2. **每一评分项必须明确回答:**
   - **论文中是否出现与该评分项直接相关的内容?**
   - **是否给出了必要的公式、模型结构、参数含义、推导、算法流程或变量定义?**
   - **内容是否完整、严谨、可复现?**
   - **若未出现，必须明确说明“论文未出现该内容”，并按规则扣分。**

六级评价体系（必须严格使用）：

1. **满足（Full）：内容完整规范，包含公式、模型、推导与解释，可完整复现。**
2. **基本满足（Almost）：核心内容齐全，但细节略有不足，仍可基本复现。**
3. **部分满足（Partial）：有相关描述但明显不完整，无法完整复现。**
4. **基本不满足（Barely Not Met）：仅有极为浅显的提及，缺乏关键结构与方法细节。**
5. **不满足（Not Met）：基本未描述相关内容或明显偏离评分要求。**
6. **完全不满足（Completely Not Met）：论文中完全未出现任何相关内容。**

判分强制要求：

- **得分必须严格落在对应评分项的分值区间内。**
- **评分等级必须与得分保持一致。**
- **不得使用“推测”“可能”“应当”等非证据性表述。**
- **所有评语必须引用论文中出现的真实描述（可节选关键词或短句）。**

输出格式（仅限 JSON）：

```
```json
{
  "问题识别": [
    {
      "dimension": "[评分维度名称]",
      "comment": "[基于论文文本的评语说明]",
      "score": [得分]
    }
  ],
  "问题复述": [],
  "假设建立": [],
  "模型构建": [],
  "模型求解": [],
  "代码实现": [],
  "结果分析": []
}
```
```

Stage-wise Evaluation (English)

Current Subtask:
{{subproblem}}

Mathematical Modeling Report Content:
{{report_content}}

Evaluation Criteria (JSON):

```json
{{report_criteria}}
```

——Evaluation Instructions (Must Be Strictly Followed) ——

You will act as a **strict, cautious, and evidence-based judge in a mathematical modeling competition**. Based solely on the provided evaluation criteria, you are required to conduct **stage-wise and criterion-wise objective scoring** of the report above.

Core Principles (Mandatory):

- **Rely exclusively on the report text; do not infer or hallucinate.**
- Any content not explicitly stated in the report must be treated as nonexistent.
- Vague mentions without formulas, derivations, structural descriptions, or methodological details must be considered *partially satisfied* or *not satisfied*.
- If model structures, algorithmic workflows, key assumptions, or variable definitions are not explicitly specified, the criterion must be marked as *not satisfied*.

Criterion-wise Scoring Rules:

1. Follow the order of evaluation criteria strictly. For each criterion, you must provide:
   - an evaluation level (six-level scale),
   - a numerical score (must fall within the specified score range),
   - explicit evidence cited from the report text,
   - a clear justification (Fully Met / Almost Met / Partially Met / Barely Not Met / Not Met / Completely Not Met).
2. For each criterion, you must explicitly address:
   - Whether the report contains content directly relevant to this criterion;
   - Whether necessary elements are provided, including formulas, model structures, parameter definitions, derivations, algorithmic procedures, or variable specifications;
   - Whether the description is sufficiently complete, rigorous, and reproducible;
   - If absent, explicitly state *"The report does not contain this content"* and deduct points accordingly.

Six-Level Evaluation Scale (Must Be Used Strictly):

1. **Fully Met:** The report provides a complete and 规范 description, including formulas, models, derivations, and explanations; the method is fully reproducible.
2. **Almost Met:** Core components are present, but some details are insufficient; the method is largely reproducible.
3. **Partially Met:** Relevant content is mentioned but clearly incomplete; the method cannot be fully reproduced.

4. **Barely Not Met:** Only superficial mentions are present, with major missing components such as model structure or algorithmic details.

5. **Not Met:** Relevant content is largely absent or deviates substantially from the criterion requirements.

6. **Completely Not Met:** The report contains no information related to this criterion.

Mandatory Scoring Constraints:

- Scores must strictly fall within the predefined range of each criterion.
- Evaluation levels must be consistent with the assigned scores.
- Do not use speculative language such as 'might", 'could", or "should".
- All comments must cite explicit descriptions from the report text (keywords or short excerpts are acceptable).

Output Format (JSON Only):

```json
{
  "Problem Identification": [
    {
      "dimension": "[Criterion name]",
      "comment": "[Evidence-based justification from the report]",
      "score": [Score]
    }
  ],
  "Problem Formulation": [],
  "Assumption Development": [],
  "Model Construction": [],
  "Model Solving": [],
  "Code Implementation": [],
  "Result Analysis": []
}
```

L Baseline Evaluation Prompts

Problem Analysis Evaluation Prompt

你的任务是：在数学建模背景下，基于问题背景与问题要求，评估建模者对问题进行子问题拆解时的合理性与整体一致性。

问题背景：
{{background}}

问题要求：
{{requirements}}

以下是建模者给出的任务分析:
任务分析:
{{task_analysis}}

评估维度:
1. 问题分析与理解

1.1 问题定义与目标
要求问题定义清晰、分析准确、目标明确。
- 问题的研究范围与目标是否清晰界定?
- 是否有效识别了问题的关键组成部分?
- 模型试图解决的实际目标是否被清楚陈述?

评分标准:
1 - 2 = 完全不清晰;
3 - 4 = 不够清晰;
5 - 6 = 基本清晰;
7 - 8 = 清晰;
9 - 10 = 非常清晰。

1.2 相关性与覆盖性
评估是否紧扣问题核心，以及子任务之间是否相互关联并完整覆盖总体目标。
- 各子任务之间是否存在合理的依赖关系?
- 各子任务是否直接服务于最终目标?
- 是否存在关键部分缺失或偏离真实目标的情况?

评分标准:
1 - 2 = 完全偏离目标;
3 - 4 = 部分偏离;
5 - 6 = 基本覆盖;
7 - 8 = 大部分覆盖;
9 - 10 = 完全覆盖。

输出格式要求（JSON）:
请严格按照以下 JSON 结构输出，不要包含任何额外文本说明，不要使用 Markdown，
不要添加注释。

```
{
  "problem_definition_and_goals": {
    "reason": "对 1.1 问题定义与目标 的评价理由",
    "score": 1-10 的整数
  },
  "scope_and_coverage": {
    "reason": "对 1.2 相关性与覆盖性 的评价理由",
    "score": 1-10 的整数
  }
}
```

请根据上述标准，对问题分析与理解进行客观且详细的评价，
并仅返回符合上述结构的 JSON 结果。

1833

Modeling Rigor Evaluation Prompt

你的任务是：在数学建模背景下，基于问题背景与问题要求，
评估建模过程在假设设定与模型构建方面的严谨性与合理性。

问题背景：
{{background}}

问题要求：
{{requirements}}

以下是建模者的建模分析内容：
建模分析：
{{modeling_analysis}}

评估维度：
2. 建模的严谨性与合理性

2.1 假设
模型假设应当清晰明确，并构成模型的理论基础。
- 模型假设是否清楚说明？
- 假设是否与实际问题背景相符且合理？
- 是否讨论了假设的合理性及其对结果的影响？

评分标准：
1 - 2 = 完全不合理；
3 - 4 = 部分合理；
5 - 6 = 一般；
7 - 8 = 合理；
9 - 10 = 非常合理。

2.2 合理性
关注模型选择与结构是否科学地反映了实际问题。
- 是否选择了合适的建模方法与指标？
- 模型结构是否能够真实刻画问题本质？

评分标准：
1 - 2 = 完全不合理；
3 - 4 = 部分合理；
5 - 6 = 一般；
7 - 8 = 合理；
9 - 10 = 非常合理。

输出格式要求（JSON）：
请严格按照以下 JSON 结构输出，不要包含任何额外文本说明，不要使用 Markdown，
不要添加注释。

```
{
  "assumptions": {
    "reason": "对 2.1 假设 的评价理由",
```

```
    "score": 1-10 的整数
  },
  "model_rationality": {
    "reason": "对 2.2 合理性 的评价理由",
    "score": 1-10 的整数
  }
}
```

请根据上述标准，对建模的严谨性与合理性进行客观且详细的评价，
并仅返回符合上述结构的 JSON 结果。

Practicality and Scientificity Evaluation Prompt

你的任务是：在数学建模背景下，基于问题背景与问题要求，
评估建模过程的实用性与科学性，重点关注模型是否具备实际问题求解能力，
以及是否遵循科学原则。

问题背景：
{{background}}

问题要求：
{{requirements}}

以下是建模者的建模过程描述：
建模过程：
{{modeling_process}}

评估维度：
3. 实用性与科学性

3.1 实用性
- 建模方法是否匹配问题特征与需求？
- 是否提供了超越简单拟合的数据洞见，能够支持决策、解释或预测？
- 是否超越常规机器学习或数据处理方法？
- 是否引入了新的框架、约束、目标或数据表示？
- 模型是否可实际运行，应用场景是否清晰？

评分标准：
1 - 2 = 完全不具备实用性；
3 - 4 = 部分实用；
5 - 6 = 一般；
7 - 8 = 具备实用性；
9 - 10 = 非常实用。

3.2 科学性
- 是否遵循科学原理并具有理论基础？
- 模型假设与方法是否科学合理？
- 是否体现对问题本质的深入理解，而非套用通用方法？

评分标准：

1 - 2 = 完全不科学；
3 - 4 = 部分科学；
5 - 6 = 一般；
7 - 8 = 科学；
9 - 10 = 非常科学。

输出格式要求（JSON）：
请严格按照以下 JSON 结构输出，不要包含任何额外文本说明，不要使用 Markdown，
不要添加注释。

```
{
  "practicality": {
    "reason": "对 3.1 实用性 的评价理由",
    "score": 1-10 的整数
  },
  "scientificity": {
    "reason": "对 3.2 科学性 的评价理由",
    "score": 1-10 的整数
  }
}
```

请根据上述标准，对建模过程的实用性与科学性进行客观且详细的评价，
并仅返回符合上述结构的 JSON 结果。

Result and Bias Analysis Evaluation Prompt

你的任务是：评估建模报告中的结果分析与偏差分析，重点关注模型输出的合理性、
可解释性，以及对潜在偏差的识别与修正。

问题背景：
{{background}}

问题要求：
{{requirements}}

以下是建模者的结果与分析内容：
建模报告：
{{modeling_report}}

评估维度：
4. 结果分析与偏差分析

4.1 结果分析
- 模型输出结果是否清晰、符合预期？
- 是否对模型推理过程进行了充分解释？
- 结果是否具有可解释性，并帮助理解问题本质？
- 是否明确总结了模型的优势与不足？

评分标准：
1 - 2 = 完全不清晰；

3 - 4 = 部分清晰；
5 - 6 = 一般；
7 - 8 = 清晰；
9 - 10 = 非常清晰。

4.2 偏差分析
- 是否识别并分析了潜在偏差？
- 是否考虑了数据偏差、模型偏差等因素？
- 是否采取了合理的偏差修正措施？

评分标准：
1 - 2 = 完全忽略偏差；
3 - 4 = 部分考虑；
5 - 6 = 一般；
7 - 8 = 考虑并修正；
9 - 10 = 非常全面，偏差控制良好。

输出格式要求（JSON）：
请严格按照以下 JSON 结构输出，不要包含任何额外文本说明，不要使用 Markdown，
不要添加注释。

```
{
  "result_analysis": {
    "reason": "对 4.1 结果分析 的评价理由",
    "score": 1-10 的整数
  },
  "bias_analysis": {
    "reason": "对 4.2 偏差分析 的评价理由",
    "score": 1-10 的整数
  }
}
```

请根据上述标准，对结果分析与偏差分析进行客观且详细的评价，
并仅返回符合上述结构的 JSON 结果。