TIRE: TOP-DOWN FORMULA INSTANTIATION WITH RELIABLE EXECUTION FOR PHYSICS PROBLEM SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Advancing physics reasoning in large language models requires scalable training data that connect physical laws, concrete problem scenarios, and verifiable numerical answers. Expert-designed problems offer top-down control over physical laws, conditions, and scenarios, but require costly human effort. LLM-based synthesis from texts scales efficiently, but model-generated answers and assessments leave supervision vulnerable to hallucinations. We introduce **TIRE**, a framework for Top-Down Formula Instantiation with Reliable Execution that brings knowledge-driven problem construction to large-scale data generation. Top-down instantiation uses reusable physical formulas and their applicability conditions to control the physical relations and numerical configurations of generated problems. Reliable execution establishes reference answers before problem generation, providing a common numerical foundation for problem construction and teacher distillation. Together, these mechanisms combine control over problem content with execution-grounded supervision for both supervised fine-tuning and reinforcement learning. Using TIRE, we construct **TIRE-19k**, spanning nine subfields of physics. Across **3** backbone LLMs and **4** benchmarks, TIRE-19k outperforms **5** representative datasets with equal amounts of SFT data, exceeding the strongest baseline by **2.86 percentage points** on average. Further GRPO raises Qwen3-8B's average accuracy by **1.52 percentage points** over its TIRE-19k SFT checkpoint. These results demonstrate the value of generating physics training data from explicit physical knowledge with reliable execution.

1 INTRODUCTION

Physics problem solving requires models to translate physical scenarios described in natural language into appropriate physical models and derive verifiable answers through symbolic derivation and numerical computation. It is therefore both an important task for evaluating the scientific reasoning capabilities of large language models (LLMs) and relevant to practical applications such as intelligent science education, engineering analysis, and research assistance (Wang et al., 2024; Xu et al., 2025). The current limitations of models in physics reasoning are largely due to a shortage of high-quality, verifiable training data. In mathematics, researchers have gradually established a systematic pipeline for generating verifiable data: they first construct executable programs, computational graphs, or formal proofs that encode valid solution processes, then convert them into natural-language problems and solutions, and finally use deterministic verifiers to independently and objectively check the correctness of the generated data (Huang et al., 2024; Ying et al., 2024; Ren et al., 2025; Wang et al., 2026). Compared with mathematical problem solving, in which derivations are typically carried out within a given formal structure, physics problem solving is more challenging. It requires not only formal mathematical derivation and computation, but also consistency among real-world physical scenarios, applicable laws, their conditions of validity, and formal models (Wang et al., 2024; Pang et al., 2025; Li et al., 2026a).

Existing efforts to build physics problem-solving data broadly follow two routes. One route relies on expert-designed problems from textbooks, examinations, and question banks (Chen et al., 2023; Xu et al., 2025). Experts construct problems top-down, starting from physical laws and specifying conditions, parameters, and scenarios. This gives direct control over problem content and quality, but the required human effort limits dataset scale. The other route uses LLMs to directly generate or complete problems, reasoning processes, and answers from existing texts (Zhang et al., 2024;

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

Fan et al., 2025; Yu et al., 2026). Although this approach scales efficiently, model hallucinations can introduce errors into problems and solutions, and quality assessment often relies on language models without an independent computational reference grounded in physics. The physical laws and parameter values represented in the data also depend on source texts and model preferences, limiting control over problem construction. The challenge is to combine top-down construction from physical knowledge with scalable generation and reliable supervision grounded in program execution.

Therefore, we propose TIRE, a framework for Top-Down Formula Instantiation with Reliable Execution that transforms structured physical knowledge into scalable training data for physics reasoning. Top-down instantiation starts from reusable physics formulas and their applicability conditions, giving explicit control over the physical relations and numerical configurations underlying each problem. TIRE translates these relations into validated executable programs, instantiates concrete parameter values, and constructs natural-language problems grounded in the resulting instances. Reliable execution establishes reference answers before problem generation, providing a common numerical foundation for both problem construction and teacher distillation. A teacher independently solves the generated problems, and its reasoning traces are selected by matching their final answers to these execution-derived references. This integration of physical knowledge, controlled instantiation, and teacher distillation supports systematic data construction with reliable, execution-grounded numerical supervision for both supervised fine-tuning and reinforcement learning. Using TIRE, we construct TIRE-19k, comprising 19,144 examples across nine subject areas and grounded in 3,051 formula records extracted from PHYSICS and SciBench.

Our main contributions are as follows:

1. We propose TIRE, a framework that combines *top-down formula instantiation* with *reliable execution* to generate physics training data at scale. Physical knowledge guides problem construction, and deterministic program outputs provide reference answers for supervision.
2. Based on TIRE, we construct the TIRE-19k dataset, covering nine subfields of physics with 19,144 examples grounded in 3,051 formula records. Each example includes an execution-derived reference answer and an answer-verified teacher reasoning trace.
3. We conduct extensive experiments. Across three backbones and four benchmarks, TIRE-19k outperforms existing datasets in both in-domain performance and out-of-domain generalization with equal amounts of training data. Ablations support the framework's components, teacher comparisons demonstrate its effectiveness with multiple teachers, and GRPO yields further gains after SFT.

2 Related Work

Physics Problem-Solving Datasets. Physics reasoning resources broadly follow two routes. Expert curation supports carefully specified problems, but collecting and validating instances is costly to scale (Chen et al., 2023; Wang et al., 2024; Rein et al., 2024). LLM-based approaches expand data through question annotation, document-grounded synthesis, and reasoning distillation (Zhang et al., 2024; Feng et al., 2025; Yuan et al., 2025; Fan et al., 2025; Yu et al., 2026). Templates and topic hierarchies provide structure for generation (Kashani, 2024; Wu et al., 2025), while FormulaReasoning supplies formula annotations and a formula database, and filters calculation errors in problems augmented from question–explanation exemplars (Li et al., 2026a). The challenge is to combine scalable construction with explicit control over physical instances and reliable numerical supervision. TIRE addresses this challenge through *top-down* synthesis: reusable physical formulas and their applicability conditions guide numerical instantiation and subsequent natural-language problem construction. Validated programs establish each instance's reference answer before question generation. The same references then select independently generated teacher traces by final-answer agreement, linking controlled problem construction with execution-grounded supervision.

Program-Guided Verifiable Data Synthesis. RV-Syn composes reusable mathematical functions into executable graphs and back-translates them into problems, while AMD synthesizes programs from mathematical knowledge and tools; both use execution outputs to check generated answers (Wang et al., 2026; Chen et al., 2025). Formal methods use proof assistants to check statements

and proofs when generating theorem-proving data (Huang et al., 2024; Ying et al., 2024; Ren et al., 2025); TP-as-a-Judge formalizes generated questions and answers and uses a prover to assess their reasoning steps (Leang et al., 2025). Physics additionally requires each numerical instance to correspond to a scenario in which the selected physical relation is applicable. TIRE builds on the execution-guided paradigm by using formula records to guide parameter assignments and scenario construction. Program interfaces specify physical meanings and units, and formula-derived tests check implementations before numerical instantiation. Program outputs provide a shared numerical reference for problem generation and teacher-answer selection, connecting controlled physics instance construction with answer-verified supervision for SFT and reward references for GRPO.

3 METHOD

3.1 OVERVIEW

TIRE generates physics problem-solving training data in a top-down manner from the domain knowledge encoded in physics formulas. Given a formula collection $\mathcal{F}$, it constructs a dataset $\mathcal{D} = \{(q_i, \tau_i, y_i^*)\}_{i=1}^N$, where q_i is a natural-language problem, y_i^* is an execution-derived reference answer, and τ_i is a teacher reasoning trace whose final answer agrees with y_i^*. The framework comprises three modules: Physical Knowledge, Top-Down Formula Instantiation, and Teacher Distillation. Physical Knowledge supplies reusable relations and their applicability conditions; formula instantiation turns these relations into numerical problems; teacher distillation supplies reasoning traces selected by numerical answer verification. Program-based operations support reliable numerical supervision across these modules: Program Validation checks formula implementations, Instance Execution computes reference answers, and Answer Verification selects teacher traces. LLMs make the semantic choices needed to instantiate physical relations and express problems, while program outputs provide a shared numerical reference across generation and selection. The resulting data connect each natural-language problem to the formula-based numerical instance from which it was constructed. We apply this framework to construct TIRE-19k, following the architecture in Figure 1.

3.2 PHYSICAL KNOWLEDGE

Formula Acquisition extracts general physical relations from source problems and their official solutions when available.[1] Each formula is represented as

$$\phi = (n, e, d, r), \qquad e : y = g(\mathbf{x}), \tag{1}$$

where n is its name, e its equation, d its physical description, and r its applicability conditions. The inputs $\mathbf{x}$ determine the target quantity y. Extraction favors general relations that support multiple numerical instances. An LLM plausibility filter checks equation well-formedness and consistency among the fields; deduplication then produces the formula library $\mathcal{F}$. The library makes each relation and its physical scope explicit, providing conditions for downstream generation. Source questions and answers serve only as extraction material and are not included as training examples; the training problems are generated through formula instantiation.

3.3 TOP-DOWN FORMULA INSTANTIATION

Program synthesis and validation. For each ϕ, the *Program Synthesiser* constructs an executable program P_ϕ mapping the independent inputs to y, together with a test set T_ϕ. Physical constants are incorporated into the program, and its input/output meanings and units are documented. Test inputs lie within the formula's applicable domain. Expected outputs are calculated from explicit formula-derived expressions without calling the program under test. *Program Validation* retains a program only if all tests pass:

$$\mathrm{Keep}(P_\phi) = \mathbb{1}[\text{all tests in } T_\phi \text{ pass}]\,. \tag{2}$$

This checks computational consistency on the tested inputs. The tests assess the implementation on designated formula-derived cases; they precede the execution of the parameter assignments used

[1] Sources are PHYSICS (Feng et al., 2025) and the physics subset of SciBench (Wang et al., 2024). Extraction uses DeepSeek-R1-250528 (Guo et al., 2025) (R1-0528 model card).

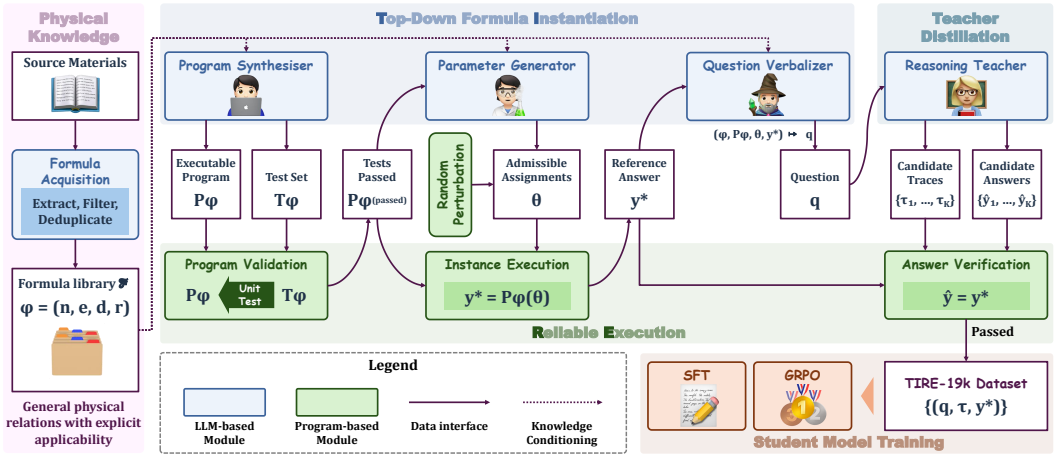

Figure 1: Overview of TIRE. Physical knowledge conditions formula instantiation, which produces questions for teacher distillation. Blue blocks denote LLM-based modules; green blocks denote program-based operations. Solid arrows indicate data transfer, and dotted arrows indicate knowledge conditioning. Execution-derived answers support both question construction and teacher-answer verification; retained examples support student training.

to construct training problems. The retained program supplies the interface for parameter generation and the implementation for numerical execution; subsequent uses of P_ϕ refer to this retained program.

Parameter generation and instance execution. Conditioned on the formula's name, equation, applicability conditions, and program interface, the *Parameter Generator* proposes input assignments. Instructions require physically admissible values, relevant variable constraints, and valid numerical operations. PHYSICS-derived assignments are used directly, while SciBench-derived assignments undergo rule-based perturbation to increase numerical diversity. We denote the final assignment by θ and use it consistently for execution and question construction. *Instance Execution* computes

$$y^* = P_\phi(\theta). \tag{3}$$

Instances that raise execution errors are discarded. Both source-specific instantiation routes therefore produce the same interface for downstream modules: an assignment paired with a successfully executed program and its numerical output. The reference y^* is fixed before question generation and teacher inference. Repeated instantiation yields different configurations under the same physical relation, with generation controlled through formula selection and parameter constraints. The full perturbation rules are given in Appendix A.

Question verbalization. The *Question Verbalizer* uses the formula information, retained program, θ, and y^* to generate an English problem q. Both parameter generation and verbalization withhold the extracted description d to limit source-specific context. The problem must preserve the supplied values, describe variables through their physical meanings, and specify a complete scenario consistent with the applicability conditions. It must provide enough information to recover the inputs and solve for the target while omitting the target equation. The reference answer constrains problem construction, while the final statement presents the physical scenario and the quantities needed for solving it. The verbalizer also supplies an auxiliary solution constrained to yield y^* and identifies the answer unit. This solution supports problem construction; training traces are obtained separately from the teacher. SciBench-derived problems undergo additional plausibility filtering, while PHYSICS-derived problems undergo LaTeX normalization before teacher inference, as detailed in Appendix A.

Table 1: Composition of the TIRE-19k training set. Formula counts are deduplicated within each source. Two records occur in both sources, so the total is their globally deduplicated union.

Formula source	Instantiation	Formulas	Examples
PHYSICS	Direct	2,561	15,133
SciBench	Perturbed	492	4,011
Total		3,051	19,144

3.4 Teacher Distillation

Candidate generation. A reasoning teacher independently solves each generated problem, receiving the problem and answer-format instructions but not y^*. We use Qwen3-235B-A22B-Thinking-2507 (Yang et al., 2025) as the default distillation teacher for both PHYSICS- and SciBench-derived problems. This teacher is used for TIRE-19k and all TIRE experiments unless an alternative teacher is explicitly specified in the teacher-sensitivity study (Section 5.4). The teacher is queried once per problem ($K = 1$), producing a candidate trace τ from which a final answer $\hat{y}$ is extracted. A request is re-issued, up to three attempts, only when the call fails or returns no complete final answer; incorrectly answered problems are not re-sampled. Source-specific prompts and sampling settings are provided in Appendices C.1 and A.3, respectively.

Answer verification and data selection. *Answer Verification* compares each candidate answer with the execution-derived reference. For scalar answers, acceptance is determined by

$$A(\hat{y}, y^*) = \mathbb{1}[|\hat{y} - y^*| \leq \epsilon \max(|\hat{y}|, |y^*|)], \tag{4}$$

with $\epsilon = 0.05$ for both sources. For vector-valued answers, the criterion is applied elementwise. Verification checks final numerical agreement; individual reasoning steps are not separately verified. Problems without an accepted trace are excluded, so answer verification filters problems rather than selecting among traces. After deduplicating by problem statement, we randomly select one accepted trace per retained problem. Answer extraction and final cleanup details appear in Appendix A.

Dataset and training use. After the training split and final formatting cleanup, TIRE-19k contains 19,144 training examples associated with 3,051 formula entries. Each example pairs a generated problem with an execution-derived reference and a selected teacher trace; Table 1 reports the source composition. The selected trace supplies a worked response for the generated question, and the reference remains the numerical target established before teacher sampling. For SFT, the target response contains the teacher's reasoning followed by its answer. For GRPO (Shao et al., 2024), y^* provides the reward reference, and agreement under the same tolerance gives a binary correctness reward. Student training settings are described in Appendix B.1.

4 Dataset Analysis

Scale and source composition. TIRE-19k comprises 19,144 generated problems grounded in 3,051 formula records. The formulas are drawn from PHYSICS and SciBench, with source composition summarized in Table 1. The dataset averages 6.27 retained examples per formula, expanding reusable physical relations into multiple concrete problem instances.

Broad subject coverage. Figure 2 shows coverage of nine subjects spanning classical and modern physics, physical chemistry, and quantum chemistry. Seven subjects each contribute more than 1,400 examples, and the largest accounts for 22.62% of the dataset. Four subjects, including classical and quantum mechanics, each contain more than 450 formula records, providing a substantial knowledge base across domains.

Question and answer lengths. Figure 3 shows length distributions measured with the Qwen3-8B tokenizer. Questions are compact, with a mean of 140 tokens and a median of 130; 99% contain at most 309 tokens. Teacher responses, including reasoning traces and final answers, average 5,136

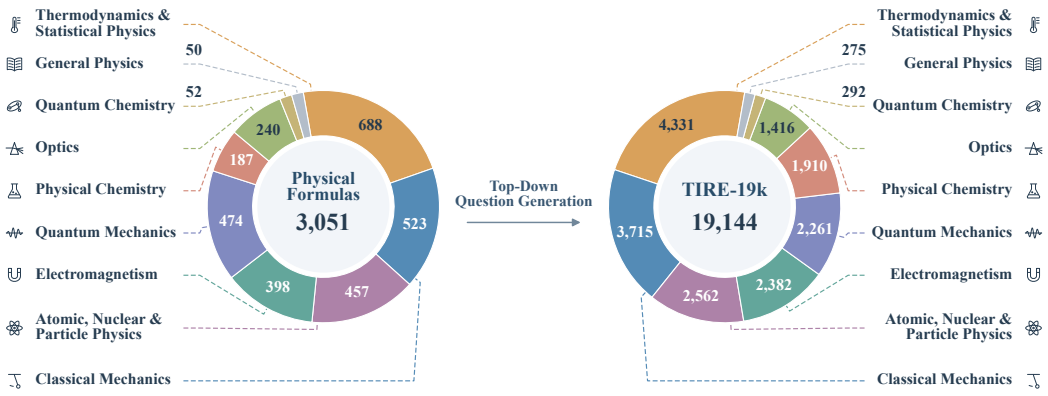

Figure 2: Subject distributions of formula records associated with TIRE-19k and training examples. Formula records can belong to multiple subjects, giving 3,069 subject memberships for 3,051 globally unique records; formula percentages use subject memberships as the denominator.

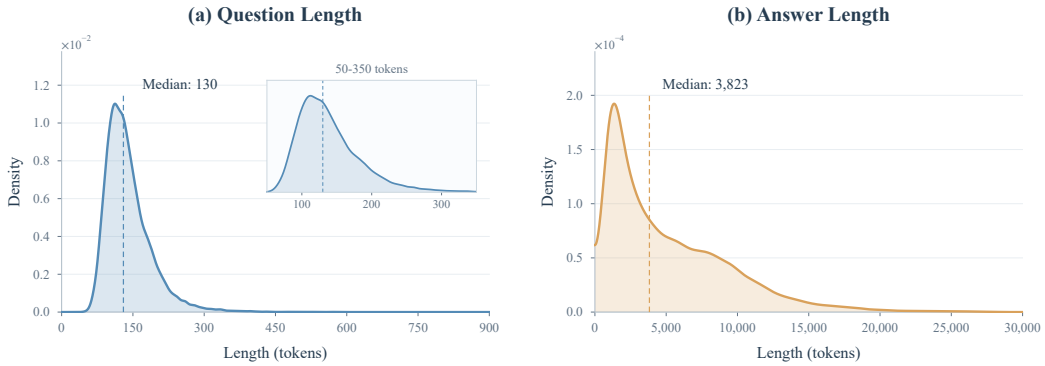

Figure 3: Question and answer length distributions in TIRE-19k. Dashed lines mark medians.

tokens with a median of 3,823, and 21.94% exceed 8,192 tokens. This pairing of concise questions with extended worked responses provides substantial reasoning supervision across a wide range of response lengths.

Illustrative example. Figure 4 illustrates the instantiation of the ideal-gas entropy-change formula as a piston-cylinder problem. The generated question specifies the gas amount, initial and final states, heat capacity, and gas constant, expressing an abstract relation as a concrete physical scenario with a clear target quantity and answer unit.

PHYSICAL FORMULA

Ideal-gas entropy change

$$\Delta S = nC_v \ln\frac{T_f}{T_i} + nR\ln\frac{V_f}{V_i}$$

GENERATED QUESTION

A piston-cylinder device contains $1.0\,\text{mol}$ of a diatomic ideal gas. Initially, the gas is at $300\,\text{K}$ and occupies a volume of $0.01\,\text{m}^3$. After a process, the gas is heated to $400\,\text{K}$ and expanded to a volume of $0.02\,\text{m}^3$. Calculate the change in entropy for the gas, given that the molar heat capacity at constant volume is $20.785\,\text{J mol}^{-1}\,\text{K}^{-1}$ and the ideal gas constant is $8.314\,\text{J mol}^{-1}\,\text{K}^{-1}$. The answer is expressed in the unit J K^{-1}.

Figure 4: An example of formula-to-question generation in TIRE.

5 EXPERIMENTS

We train 3 backbone LLMs on TIRE-19k and established physics datasets using the same number of training examples. We evaluate the resulting models on 4 representative physics benchmarks.

5.1 EXPERIMENTAL SETUP

The comparison spans three backbones: Qwen3-8B (Yang et al., 2025), DeepSeek-R1-Distill-Qwen-7B (Guo et al., 2025), and Qwen3.5-9B.[2] For each backbone, we compare TIRE-19k with five baseline datasets: MegaScience (Fan et al., 2025), PhysLogic (Yu et al., 2026), Spark-234K (Spark) (Li et al., 2026b), OpenSciReasoning-Physics-20K (OpenSci),[3] and SCP-116k (Lu et al., 2025). Each comparison uses 19,144 training examples per dataset. The main SFT settings are detailed in Appendix B.1.

Evaluation covers in-domain and out-of-domain performance. SciBench Physics (Wang et al., 2024) is in-domain because it contributes to TIRE's formula library; PhysReason (Zhang et al., 2025), OlympiadBench Physics (He et al., 2024), and TheoremQA-Physics (Chen et al., 2023) assess generalization beyond TIRE's sources. We report benchmark accuracies and their unweighted average as percentages; differences are computed from the displayed scores.

Component ablations train Qwen3.5-9B on four dataset configurations: full TIRE, TIRE without unit tests, TIRE without parameter perturbation, and Self-QA (*w/o all*), which generates question–answer pairs directly from the same formula records. Each ablation dataset contains 19,144 examples, and all configurations use the default teacher, Qwen3-235B-A22B-Thinking-2507; detailed ablation settings are given in Appendix B.2.

We assess teacher sensitivity with Qwen3.5-9B as the student, using the default Qwen3-235B-A22B-Thinking-2507 and three alternative teachers: Qwen3.7-plus, Qwen3.8-flash, and DeepSeek-v4.1-flash.

To assess further improvement after SFT, we continue training Qwen3-8B with GRPO on MegaScience, PhysLogic, and TIRE-19k. Each run starts from the corresponding SFT checkpoint and uses the same dataset, with numerical correctness as the reward.

Evaluation details appear in Appendix B.3.

5.2 SFT EXPERIMENTS

Tables 2, 3, and 4 show that TIRE-19k provides advantages over existing datasets in both in-domain performance and out-of-domain generalization across all three backbones. On in-domain SciBench Physics, it exceeds the best baseline by 5.59, 2.42, and 4.54 percentage points on Qwen3-8B, DeepSeek-R1-Distill-Qwen-7B, and Qwen3.5-9B, respectively.

The gains also extend to the three out-of-domain benchmarks: PhysReason, OlympiadBench Physics, and TheoremQA-Physics. Across these benchmarks, TIRE-19k's unweighted average accuracy exceeds that of the strongest baseline, SCP-116k, by 1.67, 2.06, and 3.25 percentage points on Qwen3-8B, DeepSeek-R1-Distill-Qwen-7B, and Qwen3.5-9B, respectively.

TIRE-19k has the highest four-benchmark average on every backbone. Its average accuracy exceeds that of the strongest baseline by 2.86, 2.15, and 3.57 percentage points on Qwen3-8B, DeepSeek-R1-Distill-Qwen-7B, and Qwen3.5-9B, respectively. **At the same training data scale, TIRE's formula-based, execution-verified data support stronger in-domain performance and out-of-domain generalization across backbones.**

A PHYSICS-only variant, trained under the same settings after removing all SciBench-derived examples and resampling the remaining data to the same size, still outperforms all five baselines on every benchmark for Qwen3.5-9B and Qwen3-8B (Appendix A.1, Table 8). **TIRE's advantage therefore persists when all four benchmarks are out-of-domain with respect to the training-data construction source.**

[2] https://huggingface.co/Qwen/Qwen3.5-9B

[3] https://huggingface.co/datasets/TerryJCZhang/OpenSciReasoning-Physics-20K

Table 2: Comparison of training datasets on Qwen3.5-9B. All scores are percentages.

Training data	SciBench	PhysReason	OlympiadBench	TheoremQA	Average
MegaScience	56.71	66.17	26.69	32.46	45.51
PhysLogic	66.97	68.58	26.69	64.04	56.57
Spark	74.56	74.31	43.64	65.79	64.58
OpenSci	78.84	82.26	47.88	64.04	68.26
SCP-116k	79.22	82.62	43.64	69.30	68.70
TIRE-19k	**83.76**	**82.99**	**51.27**	**71.05**	**72.27**

Table 3: Comparison of training datasets on Qwen3-8B. All scores are percentages.

Training data	SciBench	PhysReason	OlympiadBench	TheoremQA	Average
MegaScience	44.41	62.29	19.49	44.74	42.73
PhysLogic	61.01	62.85	23.31	52.63	49.95
Spark	56.03	68.02	22.88	50.88	49.45
OpenSci	72.49	76.71	37.71	61.40	62.08
SCP-116k	71.64	75.79	39.83	67.54	63.70
TIRE-19k	**78.08**	**78.19**	**40.68**	**69.30**	**66.56**

Table 4: Comparison of training datasets on DeepSeek-R1-7B. All scores are percentages.

Training data	SciBench	PhysReason	OlympiadBench	TheoremQA	Average
MegaScience	61.05	57.49	15.68	48.25	45.62
PhysLogic	60.18	52.50	15.25	39.47	41.85
Spark	57.06	63.40	15.25	35.09	42.70
OpenSci	54.07	68.21	18.64	57.02	49.49
SCP-116k	65.67	**71.72**	**27.54**	52.63	54.39
TIRE-19k	**68.09**	**71.72**	26.69	**59.65**	**56.54**

Table 5: Ablation study of TIRE on Qwen3.5-9B, all scores are percentages.

Configuration	SciB.	PhysR.	Olympiad.	TheoremQA.	Average
TIRE	**81.29**	**84.10**	**49.58**	**71.05**	**71.51**
w/o unit tests	79.69 (-1.60)	82.99 (-1.11)	47.46 (-2.12)	70.18 (-0.87)	70.08 (-1.43)
w/o perturbation	79.72 (-1.57)	83.18 (-0.92)	48.31 (-1.27)	69.30 (-1.75)	70.13 (-1.38)
w/o all	79.25 (-2.04)	81.33 (-2.77)	40.25 (-9.33)	70.18 (-0.87)	67.75 (-3.76)

5.3 Ablation Study

Table 5 shows that full TIRE achieves the highest accuracy on all four benchmarks, reaching an average of **71.51%** with equal amounts of training data across configurations. Removing unit tests lowers average accuracy by **1.43** percentage points, with the largest decrease on OlympiadBench (**2.12** points). Removing parameter perturbation lowers average accuracy by **1.38** percentage points, with the largest decrease on TheoremQA-Physics (**1.75** points). Both component ablations reduce accuracy on every benchmark, showing that their contributions extend to both in-domain performance and out-of-domain generalization.

Self-QA (*w/o all*) uses the same formula records to generate problems and reference answers directly with a teacher, without program synthesis or execution. TIRE instantiates formula-based programs and uses their outputs as independent reference answers. Its gains of **3.76** percentage points in average accuracy and **9.33** points on OlympiadBench highlight the benefit of program-based instan-

Table 6: Sensitivity to the distillation teacher with Qwen3.5-9B as the student.

Teacher	SciBench	PhysReason	OlympiadBench	TheoremQA	Average
Qwen3-235B-A22B	83.76	82.99	**51.27**	**71.05**	**72.27**
Qwen3.7-plus	81.76	84.66	47.46	**71.05**	71.23
Qwen3.8-flash	**84.52**	85.58	48.31	68.42	71.71
DeepSeek-v4.1-flash	82.74	**85.95**	46.61	64.04	69.84

Table 7: Qwen3-8B accuracy (%) after SFT and after further GRPO, with Average subscripts indicating absolute accuracy changes in percentage points from the corresponding SFT checkpoint.

Training data	Setting	SciBench	PhysReason	Olympiad.	TheoremQA	Average
MegaScience	SFT	44.41	62.29	19.49	44.74	42.73
	SFT→GRPO	46.09	65.25	16.95	42.11	42.60$_{(-0.13)}$
PhysLogic	SFT	61.01	62.85	23.31	52.63	49.95
	SFT→GRPO	62.38	62.85	19.92	50.88	49.01$_{(-0.94)}$
TIRE-19k	SFT	78.08	**78.19**	40.68	69.30	66.56
	SFT→GRPO	**78.64**	77.63	**43.22**	**72.81**	**68.08**$_{(+1.52)}$

tiation and execution-derived reference answers over direct teacher-generated QA. **These ablations support TIRE's instantiation and verification components.**

5.4 Sensitivity to the Teacher LLM

Table 6 shows that TIRE maintains strong performance across all four distillation teachers. Average accuracy ranges from 69.84% to 72.27%, with all four teachers exceeding the strongest baseline average of 68.70% in Table 2. The default teacher achieves the highest average accuracy, exceeding the strongest alternative by 0.56 percentage points. **TIRE remains effective with multiple distillation teachers, with the default teacher yielding the strongest overall results.**

5.5 Further Training with GRPO

On Qwen3-8B, further GRPO raises TIRE-19k's average accuracy from 66.56% after SFT to 68.08%, yielding the largest post-SFT average gain of **1.52** percentage points, versus -0.13 for MegaScience and -0.94 for PhysLogic (Table 7). The improvement spans in-domain and out-of-domain evaluation: TIRE-19k gains **0.56** percentage points on SciBench Physics, as well as **2.54** on OlympiadBench Physics and **3.51** on TheoremQA-Physics, where both baselines decline. **TIRE-19k is the most effective of the compared datasets for further GRPO training, supporting additional gains after SFT on both in-domain and out-of-domain benchmarks.**

6 Conclusion

TIRE constructs physics reasoning data through formula-guided problem instantiation and execution-based verification of teacher answers. With equal amounts of SFT data, TIRE-19k outperforms existing datasets in both in-domain performance and out-of-domain generalization, achieving the highest four-benchmark average accuracy across all three backbones. Further gains after SFT through GRPO and effectiveness across teachers support its use for scalable physics training-data construction.

AI USE STATEMENT

LLMs assisted with manuscript preparation and revision. Within TIRE's data-construction pipeline, LLMs serve solely as tools for training-data synthesis, consistent with a commonly used paradigm for constructing scientific question–answering datasets (Zhang et al., 2024; Fan et al., 2025). Their data-construction roles, including formula extraction and filtering, program and unit-test generation, parameter and problem generation, and teacher distillation, are documented in Section 3 and Appendices A and C.1. Separately, an LLM judge is used for PhysReason evaluation, as specified in Appendix B.3. **The research ideas, framework design, and code implementing the core pipeline and experimental logic were developed entirely by the human authors, without AI assistance.** The authors take full responsibility for the manuscript, code, data, and reported findings.

ETHICS STATEMENT

This work develops synthetic physics reasoning data for scientific research using established educational benchmarks as sources of physical knowledge. It does not involve human-subject experiments or the collection of personal information. The source datasets and models are identified in the paper, and the construction pipeline incorporates program validation, plausibility filtering, and numerical answer verification to support data quality. TIRE-19k and the resulting models are intended for research on physics reasoning; their use in consequential educational or engineering decisions requires independent validation by domain experts.

REPRODUCIBILITY STATEMENT

We document the methodology and experimental procedures in the main text and appendices to support reproducibility. Appendix A describes the source data, formula processing, construction configurations, teacher sampling, and data-selection rules. Appendix B details SFT and GRPO configurations, training hardware, ablation datasets, benchmark subsets, inference settings, answer extraction, and scoring procedures. Appendix C provides the data-generation and benchmark-evaluation prompt templates.

REFERENCES

Sirui Chen, Changxin Tian, Binbin Hu, Kunlong Chen, Ziqi Liu, Zhiqiang Zhang, and Jun Zhou. Arrows of math reasoning data synthesis for large language models: Diversity, complexity and correctness. In *Proceedings of CIKM*, CIKM '25, pp. 4665–4669, 2025.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. TheoremQA: A theorem-driven question answering dataset. In *Proceedings of EMNLP*, pp. 7889–7901, 2023.

Run-Ze Fan, Zengzhi Wang, and Pengfei Liu. MegaScience: Pushing the frontiers of post-training datasets for science reasoning. *arXiv preprint arXiv:2507.16812*, 2025.

Kaiyue Feng, Yilun Zhao, Yixin Liu, Tianyu Yang, Chen Zhao, John Sous, and Arman Cohan. PHYSICS: Benchmarking foundation models on university-level physics problem solving. In *Findings of ACL*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://doi.org/10.1038/s41586-025-09422-z.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of ACL*, pp. 3828–3850, 2024. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Yinya Huang, Xiaohan Lin, Zhengying Liu, Qingxing Cao, Huajian Xin, Haiming Wang, Zhenguo Li, Linqi Song, and Xiaodan Liang. MUSTARD: Mastering uniform synthesis of theorem and proof data. In *Proceedings of ICLR*, volume 2024, pp. 7358–7396, 2024.

Shlomo Kashani. QuantumLLMInstruct: A 500k LLM instruction-tuning dataset with problem-solution pairs for quantum computing. *arXiv preprint arXiv:2412.20956*, 2024.

Joshua Ong Jun Leang, Giwon Hong, Wenda Li, and Shay B. Cohen. Theorem prover as a judge for synthetic data generation. In *Proceedings of ACL*, pp. 29941–29977, 2025.

Xiao Li, Bolin Zhu, Kaiwen Shi, Sichen Liu, Yin Zhu, Yiwei Liu, and Gong Cheng. FormulaReasoning: A dataset for formula-based numerical reasoning. In *Findings of ACL*, pp. 31579–31601, 2026a.

Yu Li, Wei Li, Xin Gao, Mengyuan Sun, Xiaoyang Wang, Qizhi Pei, and Lijun Wu. SPARK: Skeleton-guided reasoning synthesis from large-scale scientific literature. *arXiv preprint arXiv:2608.30214*, 2026b. URL https://arxiv.org/abs/2608.30214.

Dakuan Lu, Xiaoyu Tan, Rui Xu, Tianchu Yao, Chao Qu, Wei Chu, Yinghui Xu, and Yuan Qi. SCP-116K: A high-quality problem-solution dataset and a generalized pipeline for automated extraction in the higher education science domain. *arXiv preprint arXiv:2501.15587*, 2025. URL https://arxiv.org/abs/2501.15587.

Xinyu Pang, Ruixin Hong, Zhanke Zhou, Fangrui Lv, Xinwei Yang, Zhilong Liang, Bo Han, and Changshui Zhang. Physics reasoner: Knowledge-augmented reasoning for solving physics problems with large language models. In *Proceedings of COLING*, pp. 11274–11289, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *Proceedings of COLM*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Jiapeng Wang, Jinhao Jiang, Zhiqiang Zhang, Jun Zhou, and Xin Zhao. RV-syn: Rational and verifiable mathematical reasoning data synthesis based on structured function library. In *Findings of the EACL*, pp. 1812–1827, 2026.

Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. SciBench: Evaluating college-level scientific problem-solving abilities of large language models. In *Proceedings of ICML*, pp. 50622–50649, 2024.

Xiong Jun Wu, Zhenduo Zhang, ZuJie Wen, Zhiqiang Zhang, Wang Ren, Lei Shi, Cai Chen, Deng Zhao, Qing Wang, Xudong Han, et al. SHARP: Synthesizing high-quality aligned reasoning problems for large reasoning models reinforcement learning. *arXiv preprint arXiv:2505.14147*, 2025.

Xin Xu, Qiyun Xu, Tong Xiao, Tianhao Chen, Yuchen Yan, Jiaxin Zhang, Shizhe Diao, Can Yang, and Yang Wang. UGPhysics: A comprehensive benchmark for undergraduate physics reasoning with large language models. In *Proceedings of ICML*, pp. 69849–69877, 2025.

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

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean Workbook: A large-scale lean problem set formalized from natural language math problems. In *Proceedings of NIPS*, pp. 105848–105863, 2024.

Zhaoxin Yu, Nan Xu, Kun Chen, Jiahao Zhao, Lei Wang, and Wenji Mao. Scientific logicality enriched methodology for LLM reasoning: A practice in physics. In *Proceedings of ICML*, 2026.

Weizhe Yuan, Jane Yu, Song Jiang, Karthik Padthe, Yang Li, Dong Wang, Ilia Kulikov, Kyunghyun Cho, Yuandong Tian, Jason Weston, and Xian Li. NaturalReasoning: Reasoning in the wild with 2.8M challenging questions. In *Proceedings of NIPS*, 2025.

Dan Zhang, Ziniu Hu, Sining Zhoubian, Zhengxiao Du, Kaiyu Yang, Zihan Wang, Yisong Yue, Yuxiao Dong, and Jie Tang. SciInstruct: a self-reflective instruction annotated dataset for training scientific language models. In *Proceedings of NIPS*, pp. 1443–1473, 2024.

Xinyu Zhang, Yuxuan Dong, Yanrui Wu, Jiaxing Huang, Chengyou Jia, Basura Fernando, Mike Zheng Shou, Lingling Zhang, and Jun Liu. PhysReason: A comprehensive benchmark towards physics-based reasoning. In *Proceedings of ACL*, pp. 16593–16615, 2025. doi: 10.18653/v1/2025.acl-long.811. URL https://aclanthology.org/2025.acl-long.811/.

A Data Construction Details

A.1 Source Data and Formula Library

Formulas are extracted from PHYSICS and SciBench; each extraction request contains a source problem together with its official solution.

PHYSICS. We use the official release of PHYSICS (Feng et al., 2025),[4] which contains 1,297 problems drawn from publicly available physics PhD qualifying examinations, organized in six subject files (`atomic`, `electro`, `mechanics`, `optics`, `quantum`, and `statistics`). Formula extraction covers the 999 text-only problems, and PHYSICS is not used for evaluation.

SciBench. SciBench (Wang et al., 2024)[5] collects college-level problems from ten textbooks in physics, chemistry, and mathematics. Its three physics textbooks, `fund`, `class`, and `thermo` (193 problems), form the SciBench Physics evaluation set and are also used for formula extraction, together with the textbooks of other disciplines whose problems apply physical laws; the mathematics textbooks are excluded. Both benchmarks serve here as sources of physical knowledge rather than of training problems: following Physics Reasoner (Pang et al., 2025), which collects formulas from the same three physics textbooks, we use SciBench for the laws of college physics curricula, and PHYSICS for those required at the level of PhD qualifying examinations. Because the evaluation textbooks contribute formulas, SciBench Physics is treated as in-domain for the full TIRE-19k dataset; the PHYSICS-only comparison below evaluates performance when none of the evaluation benchmarks is used as a construction source.

Formula records. Records are identified by their name and formula string, so algebraic variants of the same law, such as $F = ma$ and $a = F/m$, are distinct records: they have different target variables and therefore yield different programs and problems. After the plausibility filter, near-duplicate records within each source are removed: candidates are retrieved with MinHash locality-sensitive hashing over names and normalized formula strings, and a record is dropped when its name or formula nearly matches a retained record (Levenshtein ratio at least 0.82 or edit distance at most 5). Each TIRE-19k example is anchored to the formula record whose program computes its reference answer; solving the problem can still require further relations, such as unit conversions, physical constants, or intermediate quantities. Over the final data, PHYSICS-derived examples are linked to 2,561 records and SciBench-derived examples to 492, for 3,051 in total since two records occur in both sources (Table 1).

[4] https://github.com/yale-nlp/Physics/tree/main/PHYSICS

[5] https://github.com/mandyyyyii/scibench/tree/main/dataset/original

Evaluation with PHYSICS-only training data. To test whether TIRE-19k's advantages persist without using evaluation benchmarks as construction sources, we remove all 4,011 SciBench-derived examples and supplement the remaining 15,133 PHYSICS-derived examples by resampling them to restore the original size of 19,144. We train Qwen3.5-9B and Qwen3-8B on this PHYSICS-only variant with the same training and evaluation settings as for the full dataset. None of the four evaluation benchmarks supplies problems or solutions to the construction of the retained PHYSICS-derived examples; all four benchmarks are therefore out-of-domain with respect to this variant's construction source.

Table 8: Evaluation after excluding SciBench-derived training data. Both variants contain 19,144 examples before the validation split. All scores are percentages; Average is the unweighted mean of the four benchmark accuracies.

Construction source	SciBench	PhysReason	Olympiad.	TheoremQA	Average
Qwen3.5-9B					
PHYSICS + SciBench	83.76	82.99	51.27	71.05	72.27
PHYSICS only	82.01	83.73	48.73	72.81	71.82
Qwen3-8B					
PHYSICS + SciBench	78.08	78.19	40.68	69.30	66.56
PHYSICS only	74.98	80.59	40.68	71.05	66.82

Table 8 shows that PHYSICS-only training achieves average accuracies of 71.82% and 66.82% on Qwen3.5-9B and Qwen3-8B, respectively. The PHYSICS-only variant outperforms all five baseline datasets on every benchmark for both backbones (Tables 2 and 3), with its four-benchmark average exceeding the strongest baseline by 3.12 points in each case. Its average accuracy remains within 0.45 percentage points of the full configuration on both backbones. These results show that TIRE's performance advantage over the compared datasets persists when none of the evaluation benchmarks is used as a construction source. We include SciBench to broaden TIRE-19k's curricular coverage and data diversity. Its undergraduate-level physical knowledge complements the graduate-level knowledge drawn from PHYSICS qualifying examinations.

A.2 Construction Configuration

Table 9 lists the construction stages and settings for DeepSeek-R1-250528,[6] DeepSeek-V3.1,[7] Doubao-1.6-thinking,[8] and the distillation teacher described below. Unspecified sampling parameters follow the defaults of the serving endpoint.

Parameter perturbation. For SciBench-derived instances, every nonzero scalar parameter v is replaced by $v(1+u)$ with $u \sim \mathcal{U}(-0.05, 0.05)$ drawn independently, zero-valued scalars by a draw from $\mathcal{U}(0, 0.0005)$, and list elements are perturbed in the same way except that zeros are kept; values are rounded to three decimal places unless rounding would make them zero. Execution, verbalization, and verification use the perturbed assignment.

Problem filtering and LaTeX normalization. The problem filter reads only the generated problem statement and removes problems whose values are not physically meaningful (e.g., non-integer counts), whose units are inconsistent, or whose scenario violates basic scientific principles; it retains 4,553 of 5,911 SciBench-derived problems. LaTeX normalization rewrites units, constants, and symbols in LaTeX without changing the problem, and the rewrite is kept only if its ROUGE-L F1[9] with the original exceeds 0.5.

[6] https://huggingface.co/deepseek-ai/DeepSeek-R1-0528

[7] https://huggingface.co/deepseek-ai/DeepSeek-V3.1

[8] https://seed.bytedance.com/en/blog/introduction-to-techniques-used-in-seed1-6

[9] https://aclanthology.org/W04-1013/

Table 9: Construction configuration of TIRE-19k.

Stage	Model and main settings
Formula extraction	DeepSeek-R1-250528; at most 16,000 output tokens
Formula filtering	DeepSeek-R1-250528; True/False judgment; temperature 0.6
Program and unit-test generation	DeepSeek-V3.1, thinking mode; temperature 0.6
Parameter generation	DeepSeek-V3.1, thinking mode; temperature 0.6
Parameter perturbation	Rule-based, $\pm 5\%$
Question verbalization	DeepSeek-V3.1, thinking mode; temperature 0.6; at most 20,000 output tokens
Problem filtering	Doubao-1.6-thinking; YES/NO judgment; temperature 0.1; SciBench-derived problems only
LaTeX normalization	Doubao-1.6-thinking; temperature 0.1; PHYSICS-derived problems only
Teacher inference	Qwen3-235B-A22B, thinking mode; temperature 0.6; one trace per problem

A.3 Teacher Sampling and Data Selection

Sampling. Each problem is sent once to Qwen3-235B-A22B-Thinking-2507[10] ($K = 1$), with temperature 0.6 and at most 28,000 output tokens. A request is re-issued only when the call fails or the response contains no complete final answer; problems answered incorrectly are not re-sampled, so answer verification acts as a filter on problems.

Answer verification. The answer is the content of the last `\boxed{}` in the final response after the reasoning block, parsed as a number (with Math-Verify[11] as a fallback). It is accepted if it satisfies Equation 4 with $\epsilon = 0.05$; list-valued answers are compared element by element. The teacher prompt states the unit of the answer and asks for the boxed value without units, and no unit conversion is performed, so a value reported in another unit or scale is rejected. Responses without a parseable boxed answer are rejected.

Final format. Each example is a chat conversation whose assistant turn contains the teacher's reasoning in `<think>` tags followed by its final response; only untruncated traces that pass answer verification are kept.

B Experimental Protocols

B.1 Training Configurations

Table 10 summarizes the default training settings, with ablation-specific settings in Appendix B.2; benchmark and decoding settings are given in Appendix B.3. We use ms-swift[12] for SFT and GRPO,

[10] https://huggingface.co/Qwen/Qwen3-235B-A22B-Thinking-2507
[11] https://github.com/huggingface/Math-Verify
[12] https://github.com/modelscope/ms-swift

DeepSpeed ZeRO-2[13] for SFT, and vLLM[14] for GRPO rollouts. The alternative teachers in Table 6 are Qwen3.7-plus and Qwen3.8-flash,[15] and DeepSeek-v4.1-flash.[16]

Table 10: Training configurations.

Configuration	Setting
SFT optimization	Full-parameter ms-swift, 5 epochs; AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$); learning rate 10^{-6}, cosine decay, 10% warmup; weight decay 0.1; gradient clipping 1.0; global batch size 2; bf16, ZeRO-2, no packing; the epoch-5 checkpoint is evaluated
SFT data use	Main dataset comparisons: 2% of each 19,144-example set is held out for validation (seed 42), leaving 18,762 training examples; the ablation datasets also contain 19,144 examples and use the shorter configuration in Appendix B.2
Maximum length and thinking mode	40,960 with thinking (TIRE-19k, SCP-116k, and the Qwen3.7-plus teacher in Table 6); 20,480 with thinking (OpenSci; Qwen3.8-flash and DeepSeek-v4.1-flash teachers in Table 6); 16,384 without thinking (Spark); 8,192 without thinking (MegaScience) or with thinking (PhysLogic); rationale given below the table
GRPO (Table 7)	Qwen3-8B; each run starts from the epoch-5 SFT checkpoint of its dataset and trains for 3 epochs on the prompts whose four-sample solve rate under that checkpoint lies in $(0, 1)$; ms-swift GRPO with vLLM rollouts; binary reward (Equation 4); 8 rollouts per prompt, 128 completions per update; learning rate 10^{-6} (cosine, 10% warmup); 16,384 tokens, at most 14,336 generated; temperature 0.6, top-p 0.95, top-k 80; KL $\beta = 0.04$, clipping $\epsilon = 0.2$
Training hardware	SFT: 4 NVIDIA H200 GPUs (141 GB), some baselines on 2 GPUs at the same global batch size; ablations (Table 5): 8 H200 GPUs; GRPO: 8 H200 GPUs

Dataset-dependent settings. For the main dataset comparisons, SFT runs within each backbone share the optimization settings and the number of examples; only the maximum length and the thinking mode depend on the dataset. The maximum length is set from the longest example in the sampled data of each dataset. When only a few examples form a long tail, they are replaced by shorter examples from the same dataset so that a shorter limit suffices (e.g., PhysLogic at 8,192); when long examples are common, as in TIRE-19k and SCP-116k, they are kept under a larger limit (40,960). Datasets without reasoning traces (MegaScience and Spark) are fine-tuned with thinking disabled and evaluated in the same mode; all other datasets, as well as the backbones, use thinking mode.

B.2 Ablation Datasets

All four configurations in Table 5 use datasets of 19,144 examples built with the construction configuration in Table 9 and the Qwen3.5-9B backbone, with Qwen3-235B-A22B-Thinking-2507 as the teacher. Full TIRE serves as the common control for the two component ablations, each of which

[13] https://www.deepspeed.ai/tutorials/zero/
[14] https://github.com/vllm-project/vllm
[15] https://www.alibabacloud.com/help/en/model-studio/models
[16] https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash

disables the corresponding step in the main construction pipeline while retaining the other construction stages. To accommodate the shorter examples in the ablation datasets, all four configurations are trained with a maximum length of 24,576 tokens and a global batch size of 4; the other settings follow Table 10.

Unit tests. The *w/o unit tests* variant omits unit-test-based program validation: generated programs proceed to parameter generation and execution without requiring their unit tests to pass. Program execution still supplies the reference answers, and instances that raise execution errors or yield no usable reference answer are discarded. Question verbalization, source-specific processing, teacher distillation, and final-answer verification otherwise follow the full pipeline. The variant includes 2,256 verified examples from programs that fail their unit tests and is supplemented with examples sampled from TIRE-19k to reach 19,144 examples while preserving the source composition.

Parameter perturbation. The *w/o perturbation* variant replaces the perturbed SciBench-derived examples with unperturbed ones: the generated assignments are used directly for program execution and question verbalization, without the rule-based perturbation of Appendix A.2. PHYSICS-derived assignments remain unperturbed in both full TIRE and this variant.

Self-QA. The *w/o all* variant skips program synthesis, parameter generation, and execution: the teacher writes numerical problems directly from the same formula records and solves them with the formula in view to obtain the reference answers. The teacher then solves each problem again from its statement alone, and a trace is retained only if its final answer agrees with the reference.

B.3 Evaluation Protocols

Tables 11 and 12 specify the benchmark subsets and evaluation configurations. We use the official PhysReason data release,[17] OlympiadBench MathJudger,[18] and TheoremQA numerical scoring routines.[19] PhysReason responses are graded by DeepSeek-V4-Flash (0731 snapshot).[20] For Qwen3-8B, context extension uses YaRN[21] with the scaling factor specified in Table 12.

C Prompt Templates

C.1 Data Generation Prompts

This section gives one prompt for each construction stage; placeholders such as {`formula`} are filled at run time. Prompts marked as translated were written in Chinese and are rendered in English with their structure, constraints, and output formats unchanged; long in-prompt examples are abbreviated where noted.

Formula Extraction Prompt (translated)

You are a professional physicist. Based on the problem and its solution, list all the general physics formulas involved. Output each formula in the following format:
`{"name": "theorem name", "formula": "LaTeX formula", "description": "theorem description", "applicable_range": "applicable range"}`
Wrap the complete JSON array in a `json` code block, and write all fields in Chinese.
If you cannot solve the problem, still return at least one relevant formula.

Problem: {`problem_text`}

Solution: {`problem_solutions`}

[17] https://huggingface.co/datasets/zhibei1204/PhysReason

[18] https://github.com/OpenBMB/OlympiadBench/blob/main/inference/code/math_judger.py

[19] https://github.com/TIGER-AI-Lab/TheoremQA/blob/main/number_utils.py

[20] https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash-0731

[21] https://qwen.readthedocs.io/en/latest/deployment/vllm.html#context-length

Table 11: Benchmark subsets, question counts, and scoring.

Benchmark	Subset	Questions	Scoring
SciBench	Physics: textbooks `fund`, `class`, and `thermo` of the official release	193	Numeric match with 5% relative tolerance; macro-average over the three textbooks
PhysReason	Text-only questions with a single boxed reference answer	541	LLM judge (DeepSeek-V4-Flash, 0731 snapshot, thinking disabled)
OlympiadBench	Physics: `OE_TO_physics_en_COMP` (open-ended, text-only, English)	236	Official MathJudger with the per-question tolerance of the dataset
TheoremQA	Physics questions without images	114	Numeric match: 4% relative tolerance for reals, exact for integers, lists compared after sorting

Formula Plausibility Filter Prompt (translated)

{`formula_record`} *(the formula record as JSON inside a code block)*

Based on the content of the JSON fields above, strictly judge their plausibility:
- name: the formula name; it must be general and apt.
- formula: must be an equation in LaTeX format, scientifically sound and free of syntax errors.
- description: roughly describes the meaning of the formula.
- applicable_range: roughly specifies the scope in which the formula applies.

Judge comprehensively whether all four items are plausible and correspond to one another.
If all of them are plausible and consistent, return only True; if any item is implausible or inconsistent, return only False.
Do not explain the reason; return only True or False.

Table 12: Prompts, decoding, and answer processing.

Setting	SciBench	PhysReason	OlympiadBench	TheoremQA
Prompt	User prompt with unit sentence; zero-shot	System and user prompt; zero-shot	Official English open-ended prompt; zero-shot	System and user prompt; zero-shot
Reasoning mode	Thinking enabled for the backbones and for models fine-tuned on reasoning traces, including all TIRE models; disabled for models fine-tuned on MegaScience or Spark.			
Temperature / top-p	Model defaults: 0.6 / 0.95 (Qwen3-8B, R1-7B), 1.0 / 1.0 (Qwen3.5-9B); presence penalty 0	0.6 / 0.95; presence penalty 1.0; judge temperature 0	0.6 / 0.95; presence penalty 1.0	0.6 / 0.95; presence penalty 1.0
Output-token limit	Raised stepwise up to 60,000 tokens (server context 65,536; YaRN factor 2 for Qwen3-8B), regenerating only unfinished responses; responses still truncated are regenerated up to twice at 60,000 tokens. The judge uses at most 32,768 tokens.			
Responses per question	One response per question; accuracy is the percentage of correct responses.			
Answer extraction	Last `\boxed{}` after the reasoning block; scientific-notation normalization	Final response after the reasoning block, passed to the judge	Last `\boxed{}` after the reasoning block	Last `\boxed{}` after the reasoning block; numbers and lists parsed
Tolerance	5% relative	Decided by the judge against the reference	Per-question tolerance from the dataset; 10^{-8} otherwise	4% relative for reals; exact after rounding for integers
Failure handling	Empty, unparseable, truncated, or failed responses count as incorrect and remain in the denominator. A score is final only if the run is complete and both the truncation rate and the extraction-failure rate are at most 3%.			

Program Generation Prompt (translated)

Identity
You are an expert in physics and physical chemistry and are proficient in Python.
Task
- Given the information of a physics or physical chemistry formula, generate a Python function that represents the formula. The inputs of the function are all the variables on the right-hand side of the formula, and the output is the variable on the left-hand side. Constants in the formula must not be inputs; keep them implicit inside the function.
Formula information
`{"name": "{name}", "formula": "{formula}", "description": "{description}", "applicable_range": "{applicable_range}"}`
Requirements
- Output Python code, and do not output any explanation outside the code block.
- The code must include type annotations for the input parameters and the return value.
- The code must include a docstring describing the function and its parameters.
- Put the generated code in a JSON object whose key is python_code and whose value is the generated code.
Code requirements
- If a Python package is needed, import it inside the function, e.g., `def gravitational_acceleration(a: float, G: float, M: float, r: float) -> float: import numpy as np`
Return format
- Return a standard JSON object containing the following field: `{"python_code": "" // the corresponding Python code}`

Unit-Test Generation Prompt (translated; example abbreviated)

Identity
You are a rigorous physics expert, proficient in Python unit testing (unittest).
Goal
Based on the given "formula information" and the source code of the corresponding Python function, write high-quality unit-test code that verifies whether the function correctly implements the formula: the return value of the function should be equivalent to the physical quantity on the left-hand side of the formula, and the parameter list of the function should cover all independent variables on the right-hand side (physical constants in the formula are treated as internal constants and are not passed as parameters).
Formula information
`{"name": {name}, "formula": {formula}, "description": {description}, "applicable_range": {applicable_range}}`
Source code of the Python function under test
`{python_code}`
Generation requirements
1. Output format: return only one valid JSON object whose only key is `"test_cases"` and whose value is a string of pure Python test code. No other text, explanation, tags, backticks, or extra keys may appear.
2. Do not import or reference the module containing the function under test (no `import xxx` or `from xxx import` of the function itself); import only necessary standard libraries (usually unittest and math; do not import them if unused).
3. Call the function by name directly, as if it were defined above in the same file.
4. Use physically reasonable values: correct orders of magnitude, and no non-physical values such as negative masses or negative absolute temperatures.
5. Principles for computing expected values (key point; hard-coding values by intuition is strictly forbidden):
   - Every expected value must be computed on the fly from the structure of the formula; never write guessed approximate numbers (no magic numbers).
   - If the formula contains exponentials, logarithms, powers, or roots, rebuild the theoretical value with expressions such as `math.exp`, `math.log`, or `**`; do not write truncated decimal approximations.

- For half-life scenarios, write `expected = N0 * 0.5**n` or `N0 * math.exp(-lambda_ * t)`; do not hard-code division constants such as `N0/1024` unless the expression itself naturally produces them (e.g., two half-lives may be written as `N0 * 0.5**2`; prefer this form).
- "Approximately unchanged" tests (e.g., with a very small decay constant) must first compute expected from the formula and then assert actual against expected; never add a hand-estimated number from a source other than expected (e.g., 19999.999).
- Do not add a second assertion that artificially fits the initial value merely to be "close"; to show that a change is tiny, `self.assertLess(abs(expected - N0)/N0, 1e-5)` may be added.

6. Within one test, after asserting a return value against its formula-based expected value, do not assert the same return value again against a hand-written approximate constant.
7. Floating-point assertions: prefer `self.assertAlmostEqual(expected, actual, places=...)`, with the number of places chosen by magnitude (usually 6–10); for values spanning several orders of magnitude, use `rel = abs(expected - actual)/max(1.0, abs(expected))` and assert that rel is below a reasonable threshold, with justification.
8. If constants (such as G, c, or k_B) are not built into the function but are needed for the expected value, define them in the test with the same values and compute the expected value consistently with the structure of the formula.
9. Do not use randomness, I/O, networking, or third-party libraries.
10. If the formula contains powers, roots, exponentials, or logarithms, give an explicit physically derived expression for the expected value in the test instead of a hard-coded magic number.
11. Class name: Test + function name in upper camel case. Test method names: test_description.

Quality and self-check (performed internally, not written to the output)
- Check for hard-coded constants that do not correspond directly to the structure of the formula; replace any with symbolic expressions.
- Check for the pattern "repeated assertion + manual estimate"; if present, remove the manual estimate.
- Check that every expected value is defined independently of the function under test (using the function under test to derive its own expected value is forbidden).

Strict output specification
- Output no extra text, such as "Here is", "Below are", "Explanation", `<artifact>`, code fences, or "I'll create a set of test cases for the Hubble's Law function. Here's a JSON object containing the test cases".
- Output nothing outside the JSON.
- The test_cases string in the JSON may contain line breaks and indentation.

Example unit test (do not copy the example; rewrite it for the current function)
[One JSON example whose test_cases value is a unittest class for a photon-sphere-radius function; omitted here for space.]

Failure handling
If information is missing and tests cannot be constructed, still return valid JSON: put a minimal unittest structure in test_cases and mark it with `self.fail('insufficient data')` in setUp, but first try to generate tests normally.

Parameter Generation Prompt (translated)

Role
You are an expert in physical chemistry and scientific computing with Python.

Task
We provide the structured information of a physics or physical chemistry formula and its Python implementation. Based on the function signature and the physical meaning of the formula, construct one set of test input parameters that "can pass and are physically reasonable", and return only JSON. The return value of the function corresponds to the physical quantity on the left-hand side of the formula; all arguments of the function correspond to the variables on the right-hand side (and variables implicitly required by the implementation). Constants used in the code need not appear in the test input.

Formula Information

{"name": "{name}", "formula": "{formula}", "applicable_range": "{applicable_range}"}
Python Function
{function_code}
Generation requirements
1. Read only the arguments in the function signature and generate a reasonable physical value for each of them; do not omit any argument; avoid 0 or unreasonable extreme values unless the physics or the function logic explicitly allows them.
2. If an argument should be an array, vector, or matrix (evidence: the function body contains shape operations, slicing, broadcasting, or elementwise operations, or the variable is passed to np.array, np.asarray, np.stack, np.concatenate, or array arithmetic), represent it as a list, since it cannot otherwise be stored as JSON; element values must be physically reasonable, and the list should be short (2–4 elements).
3. If an argument denotes:
   - temperature: use a reasonable range of 250–1500 K (adjustable to the applicable domain of the formula);
   - pressure: use 10^3–10^7 Pa (or follow the formula description);
   - length (m): avoid 0 and negative values;
   - time (s): use positive values;
   - concentration or fraction: use the interval (0, 1];
   - energy, enthalpy, entropy, and similar quantities: use signs and magnitudes consistent with common physical situations;
   - frequency or rate constant: use positive values;
   - names containing ref, std, or baseline: common standard-condition values may be used;
   - names containing n, count, or steps: use a positive integer or a short array.
4. Ensure that the arguments do not create obvious physical contradictions (e.g., the sum of partial pressures must not exceed the total pressure, and mole fractions must not sum to clearly more than 1).
5. Do not invent arguments that the function does not declare, and do not rename arguments.
6. Do not output explanatory text, comments, or extra keys; output JSON only.
7. JSON structure: {"input_para": {"<param1>": <value>, "<param2>": <value>, ...}}, where array values are written directly as list literals (without quotes), string arguments keep their quotes, and Boolean arguments use True/False.
8. Avoid random numbers (do not use np.random).
9. If the function has *args or **kwargs, ignore them and handle only explicitly named arguments.
10. If there are dependencies between arguments (e.g., the length of one array must match another argument), satisfy the code logic.
11. Choose all values so that the computation does not produce numerical errors such as division by zero, the logarithm or square root of a negative number, or overflow from an overly large exponent.
12. If the applicable range of the formula imposes limits, choose values within that range; if it cannot be parsed, use conservative, conventional physical ranges.
Output
Output only the final JSON, with no prefix or suffix, no Markdown code block, and no additional explanation.
Example (illustrative, not a real function)
{"input_para": {"T": 298.15, "P": 101325.0, "y": [0.21, 0.79], "k0": 1.2e5}}
(Do not include this example in the formal output.)

Question Verbalization Prompt

Identity
You are an expert in both Physics and Physical Chemistry, with strong proficiency in quantitative problem design and solution explanation. You rigorously apply dimensional analysis, significant figures, and domain-appropriate terminology.
Tasks

Task 1: Create a calculation problem based on the provided formula metadata and input parameters (the parameters will be injected separately as structured data; do NOT list or restate them explicitly).
Task 2: Provide a detailed, logically ordered solution that applies the given formula correctly, with proper unit handling and clear justification of each step.
Task 3: Ensure the final numerical result exactly matches the provided answer parameter.
Scope
You may generate a problem in any of the following subjects:
- Classical mechanics
- Electromagnetism
- Optics
- Thermodynamics and statistical physics
- Quantum mechanics
- Atomic, nuclear, and particle physics
- General physics
- Physical chemistry
- Quantum chemistry
provided the selected scenario is realistic and consistent with the formula's applicable range.
Formula Information
The following structured block defines the formula context:
`{"name": "{name}", "formula": "{formula}", "applicable_range": "{applicable_range}"}`
Reference Code (for reference)
`{function_code}`
Parameter Context
The problem should be based on the following parameters:
`{param_list}`
Expected Answer Constraint
The computed final result MUST numerically match:
`{answer_para}`
Problem Construction Requirements
1. Scientific Accuracy:
   - Correctly apply the formula: `{formula}`
   - Maintain unit consistency; show conversions where non-trivial.
   - Select or invent a plausible scenario (experiment, natural process, or theoretical model) that justifies the formula's use.
2. Logical Clarity:
   - Present a coherent narrative (e.g., reaction mechanism, energy balance, motion description).
   - If a chemical reaction is used, ensure it is balanced and physically consistent.
   - If a physics scenario is used, ensure forces, fields, or energies are plausible.
3. Presentation Quality:
   - Use precise technical language.
   - Embed equations in LaTeX: e.g., `$F=ma$`, `$q=mc\Delta T$`, `$PV=nRT$`.
   - Use appropriate significant figures (justify if rounding).
4. Do NOT:
   - Expose raw input parameter names.
   - Fully provide the formula itself that needs to be asked about.
   - Ask the user for additional data.
   - Include meta-instructions or placeholders in the final output.
   - Deviate from the provided answer value.
Solution Requirements
- Show step-by-step derivation.
- Explicitly write the main governing formula before substitution.
- Show unit cancellation or transformation.
- Justify any approximations.
- Conclude by enclosing the final answer (that matches the target value) in `\boxed{}`.
- Both the generated questions and answers should be in English.
Output JSON Format

Return ONLY a JSON object with the following keys:
{"question": "A fully self-contained, well-structured Physics or Physical Chemistry problem statement (with LaTeX math expressions).", "answer": "Detailed solution showing reasoning, substitutions, unit analysis, and final matching result.", "answer_units": "LaTeX-formatted unit expression (e.g., \text{mol·L}^{-1}, \text{kJ·mol}^{-1}, \text{Pa·s}); use 'dimensionless' if no unit."}
Additional Notes
- Prefer realistic constants (e.g., $R=8.314\ \text{J·mol}^{-1}\text{K}^{-1}$) unless the scenario mandates a specific variant.
- If assumptions (e.g., ideal gas, negligible heat loss, non-relativistic motion) are required, state them explicitly.
- The style should be professional and suitable for advanced undergraduate or early graduate level.
Escaping requirement
- When showing LaTeX, JSON, file paths, or any string that normally contains backslashes, ensure each backslash is doubled (for example, write "\\Delta" for "\Delta", write "C:\\path\\my_file" for "C:\path\my_file").
- Validate final output to ensure no lone '\' characters remain; prefer double-escaping where ambiguity may occur.

Problem Plausibility Filter Prompt (translated)

[Task] Judge whether the following scientific problem is reasonable. Reasonableness is judged from the following aspects:
1. Whether the numerical values have practical meaning (e.g., a count cannot be a fraction, a weight cannot be negative, and the temperature range should be reasonable).
2. Whether units and dimensions match (e.g., a speed cannot be expressed in kilograms, and a number of molecules should not be negative).
3. Whether the problem conforms to basic scientific principles or common sense (e.g., an object cannot move faster than light, mass is conserved in chemical reactions, and the number of biological individuals cannot be negative).
[Problem]
{question}
Return the following format:
Conclusion: YES or NO
Reason: the specific reason (briefly explain why the problem is or is not reasonable).

LaTeX Normalization Prompt

Convert all units in the following text to proper LaTeX format with these requirements:
- Keep the original question structure and content unchanged.
- Express all units, constants, chemical formulas, and symbols with superscripts using LaTeX syntax (e.g., g/cm^3, mol/L, \(\Delta k\), H_2O).
- Ensure the text remains a natural and coherent question.
- Return only the modified question, without any explanation or extra content.

Original text:
{text}

Teacher Prompt (user message)

{problem} {unit_sentence}
Please reason step by step, and put your final answer within $\boxed{}$ without including units.

{unit_sentence} is drawn uniformly from ten paraphrases of "The unit of the answer is {u}.", where {u} is the answer unit in inline math, and is omitted when the answer is dimensionless.

C.2 BENCHMARK EVALUATION PROMPTS

All benchmarks are evaluated zero-shot, and no in-context examples are used. The SciBench unit-scaling convention follows its official evaluation code,[22] and the OlympiadBench prompt follows its official English open-ended template.[23]

SciBench Evaluation Prompt (user message only)

{problem_text} The unit of the answer is {unit}.
Please reason step by step, and put your final answer within $\boxed{}$ without including units.

Following the official SciBench code, the power-of-ten factor is removed from {unit} and handled by the scorer.

PhysReason Evaluation Prompt

System: You are a helpful assistant.

User:
Please answer this question: {query}.
Please wrap the final result at the end of the derivation process using `\boxed{}` to highlight it.

PhysReason Judge Prompt

Your job is to look at a question, a gold target, and a predicted answer, and return a letter "A" or "B" to indicate whether the predicted answer is correct or incorrect.

Question: {question}

Reference Answer: {gold}

Model Answer: {pred}

Evaluate the model's answer based on correctness compared to the reference answer.
Grade the predicted answer of this new question as one of:
A: CORRECT
B: INCORRECT

Just return the letters "A" or "B", with no text around it.

{pred} is the final response after the reasoning block. A response is counted as correct if the judge returns A.

OlympiadBench Evaluation Prompt (official English open-ended template)

System: You are an AI assistant. Please answer the following Physics competition questions as required.

User:
The following is an open-ended problem from an International Physics competition.
{answer_type_text}Please calculate the answer according to the given requirements and the information provided. Please use LaTeX format to represent the variables and formulas used in the solution process and results. Please end your solution with "So the final answer is {final_answer_text}." and give the result explicitly{unit_text}.
{context}
{question}

Slots follow the official evaluation code: they state the answer type (numerical value, expression, equation, or interval), the boxed final-answer format, and, when the answer has a unit, that the unit should not be included in \boxed{}. The context line is omitted when a problem has no context.

[22] https://github.com/mandyyyyii/scibench/blob/main/eval/eval_zeroCot.py

[23] https://github.com/OpenBMB/OlympiadBench/blob/main/inference/code/evaluators/evaluator.py

TheoremQA-Physics Evaluation Prompt

System: You are a science teacher. Provide a solution to the given problem. End with the final answer inside a single LaTeX `\boxed{...}`, without units or explanation inside the box. For multiple numerical answers, put the list of numbers inside that box.

User:
Problem:{`Question`}
Solution: