AutoBench: A Dynamic Benchmark for Auditing Scientific Reasoning in Large Language Models

Shima Imani[1]* Seungwhan Moon Adel Ahmadyan Lu Zhang
Kirmani Ahmed Babak Damavandi
Meta Reality Lab

Abstract

*Rigorous auditing of large language models (LLMs) demands dynamic, controllable evaluations, especially in scientific domains like physics where precise reasoning is essential. We introduce **AutoBench**, a novel synthetic benchmark for experimental auditing of LLM reasoning, robustness, and failure modes. AutoBench comprises 15K university-level physics problems generated through a rigourous process, each paired with structured step-by-step reasoning, symbolic Python code, and final computed answers. The dataset is fully dynamic: each problem is parameterized, allowing controlled variation of inputs with automatic solution updates via the associated code. Additionally, multiple paraphrased variants of each problem enable systematic perturbations of linguistic structure. These features support fine-grained experiments to test generalization, diagnose biases, and uncover brittle reasoning behaviors—advancing beyond traditional static benchmarks. In addition, since the dataset is fully dynamic, we can hide key components and measure hallucination, and how models behave in uncetainty. We further evaluate a range of state-of-the-art instruction-tuned LLMs on AutoBench, revealing new insights into their scientific reasoning capabilities and failure patterns. Our work highlights the importance of dynamic synthetic datasets for principled, experimental auditing of model behavior.*

1. Introduction

Effective reasoning, a cornerstone of human intelligence, enables systematic problem-solving, logical deduction, and structured decision-making. In science-driven domains such as physics, accurate reasoning demands the integration of theoretical principles, rigorous mathematical processes, and computational verification to ensure both dimensional and numerical correctness.

Recent advancements in large language models (LLMs) have markedly improved their reasoning capabilities. Innovations such as advanced prompting methods (e.g., chain-of-thought, tree-of-thought, and self-reflection), supervised fine-tuning on reasoning tasks, direct preference optimization (DPO), and reinforcement learning with human feedback (RLHF) have all contributed to these gains [9, 12–14, 18, 19]. Concurrently, a range of benchmarks has emerged to assess scientific reasoning in LLMs, particularly in STEM domains [8, 10, 15, 17].

However, existing benchmarks for evaluating scientific reasoning particularly in physics have several limitations. Many do not include step-by-step reasoning, do not reflect the complexity of university-level problems, lack systematic variation, and often do not provide executable Python code. For instance, an example from **ScienceQA** dataset [10]:

```
"question": "Select the solid."
"choices": ["rain", "water in a fishbowl", "
    hammer"]
"answer": 2
```

Similarly, in the **SciEval** [15] benchmark:

```
"question": "How can momentum be decreased?
A. Decrease mass or velocity, or transfer
    momentum through collision.
B. Keep mass and velocity constant, avoid
    collisions.
C. Increase mass and velocity, avoid collisions
    .
D. Increase mass, decrease velocity, and avoid
    collisions.
Answer: ["A"]
```

These benchmarks tend to be largely *static*, with fixed questions and limited support for controlled variation in input values, reasoning pathways, or linguistic formulations. This can make it challenging to conduct dynamic and fine-grained evaluations necessary for assessing model generalization, robustness, and reasoning under perturbation.

*Corresponding author: shimaimani@meta.com

Question

Determine the final velocities of two colliding objects after an elastic collision. Given that object A has a mass of $m_A = 0.500\,\text{kg}$ and an initial velocity of $u_A = 4.00\,\text{m/s}$, while object B has a mass of $m_B = 3.50\,\text{kg}$ and is initially at rest.

Solution

Key concepts & variables: An elastic collision conserves both momentum and kinetic energy. The target variables are the final velocities v'_A and v'_B, while the known quantities are m_A, m_B, u_A, u_B.
Set up: Using the conservation of momentum:

$$m_A u_A + m_B u_B = m_A v'_A + m_B v'_B$$

For kinetic energy:

$$\frac{1}{2}m_A u_A^2 + \frac{1}{2}m_B u_B^2 = \frac{1}{2}m_A v_A'^2 + \frac{1}{2}m_B v_B'^2$$

Execute: Solving the equations, we get:

$$v'_A = -3.00\,\text{m/s}, \quad v'_B = 1.00\,\text{m/s}$$

Evaluate: The negative v'_A means object A reverses direction, expected in an elastic collision. The results are consistent with physical intuition.

```python
import sympy as sp
from pint import UnitRegistry
ureg = UnitRegistry()
Q_ = ureg.Quantity

def elastic_collision(m_1, m_2, v_1, v_2):
    # Convert inputs to Pint quantities
    m_1 = Q_(m_1).to(ureg.kilogram) # Ensure kg
    m_2 = Q_(m_2).to(ureg.kilogram) # Ensure kg
    v_1 = Q_(v_1).to(ureg.meter / ureg.second) # Ensure m/s
    v_2 = Q_(v_2).to(ureg.meter / ureg.second) # Ensure m/s
    m_1 = m_1.magnitude
    m_2 = m_2.magnitude
    v_1 = v_1.magnitude
    v_2 = v_2.magnitude
    # Define symbolic variables
    v1_prime, v2_prime = sp.symbols('v1_prime␣v2_prime', real=
        True)
    # Conservation of momentum equation
    momentum_eq = sp.Eq(m_1 * v_1 + m_2 * v_2, m_1 * v1_prime +
        m_2 * v2_prime)

    # Conservation of kinetic energy equation
    kinetic_energy_eq = sp.Eq(0.5 * m_1 * v_1**2 + 0.5 * m_2 *
        v_2**2, 0.5 * m_1 * v1_prime**2 + 0.5 * m_2 *
        v2_prime**2)

    # Solve the system of equations
    solutions = sp.solve((momentum_eq, kinetic_energy_eq), (
        v1_prime, v2_prime))
    # Filter out the meaningful solution
    for sol in solutions:
        if sol[0] != v_1:
            v1_prime_val, v2_prime_val = sol
            break
    return {
        "v1'": v1_prime_val,
        "v2'": v2_prime_val
    }
```

Figure 1. An example from our **AutoBench** dataset includes a generalized question, a step-by-step solution, and the corresponding Python code. The dataset is dynamic, allowing any variable to be modified, providing flexibility for evaluating models across a range of input variations.

To address these limitations, we introduce **AutoBench**, a comprehensive benchmark for physics-based reasoning that includes 15,000 examples. Our benchmark contributes along three key dimensions:

- **Dynamic Generalization.** AutoBench offers systematically parameterized and extensively paraphrased physics problems, enabling tests of model robustness and generalization across variable inputs, conditions, and textual contexts.
- **Structured Step-by-Step Reasoning.** Every AutoBench instance includes carefully curated, detailed reasoning steps that explicitly illustrate the logical deduction process.
- **Integrated Computational Validation.** Each problem instance is accompanied by generalized, executable Python code that ensures computational correctness, numerical accuracy, and consistent dimensional analysis based on standard SI units. The code is designed to facilitate seamless generalization across different units of measurement while rigorously maintaining unit consistency and dimensional integrity.

Figure 1 illustrates an example demonstrating how AutoBench incorporates dynamically parameterized statements, structured reasoning steps, and corresponding computational validation through Python code.

By enabling controlled perturbations of both inputs and linguistic formulations, AutoBench supports experimental auditing of LLMs, allowing researchers to probe model behavior, diagnose biases, and uncover brittle reasoning patterns. We further benchmark several state-of-the-art instruction-tuned LLMs on AutoBench, revealing new insights into their scientific reasoning capabilities and exposing critical limitations that traditional static benchmarks fail to capture.

2. Related Work

Recent advancements in large language models (LLMs) have significantly expanded their capabilities in structured reasoning tasks. Pioneering work in methods such as Chain-of-Thought (CoT) prompting [18], instruction tuning [12], and reinforcement learning from human feedback (RLHF) [4, 12] has demonstrated strong performance across a wide range of tasks. However, despite these ad-

vances, challenges persist in complex, multi-step scientific reasoning, particularly in domains requiring deep domain knowledge, such as physics and engineering [8]. These challenges underscore the need for datasets and tools that better reflect the nuanced demands of STEM problem-solving.

In parallel, the field of auditing machine learning models has emerged as a critical area of research, focusing on understanding model behavior, performance, biases, and failure modes. One key area of interest is the use of synthetic data to provide reliable insights into model behavior. This approach helps evaluate how models generalize and behave under various conditions [5, 16]. Despite efforts to develop synthetic datasets for testing and evaluating machine learning models, many existing science benchmarks, particularly in physics, remain static. This static nature limits their ability to comprehensively probe model behavior and understand the underlying causes of failure modes.

The development of science benchmarks such as ScienceQA [10], SciBench [17], and physics-specific datasets like PhysBench from MMLU [8] has been instrumental in advancing the evaluation of LLMs on structured reasoning tasks. These benchmarks provide valuable resources for assessing baseline scientific knowledge and reasoning skills. However, their static nature fixed in scope and content—poses challenges for dynamically evaluating model behavior across varying conditions and diverse input scenarios. In addition, several benchmarks, particularly in physics such as PhysBench [8],SciEval [15], and JEEBench [2], predominantly utilize multiple-choice formats. While effective for standardized evaluation, this format can sometimes constrain a deeper assessment of model reasoning and generalization capabilities. Moreover, many existing resources lack detailed, programmatically verifiable step-by-step solutions and systematic support for techniques like dimensional analysis or symbolic computation. Expanding beyond static benchmarks could open new avenues for more robustly probing model behavior, diagnosing failure modes, and fostering a deeper understanding of scientific reasoning in dynamic, real-world contexts.

Our proposed **AutoBench** builds on these efforts by introducing a dynamic, parameterized benchmark designed to enable controlled perturbations of input variables. This flexibility allows for systematic exploration of model behavior under a wide range of conditions. Unlike traditional static benchmarks, AutoBench supports deeper investigation into model generalization, robustness, and failure modes by leveraging diverse problem variants coupled with programmatic validation. The ability to iteratively modify input scenarios and rigorously verify outputs is crucial for diagnosing weaknesses, understanding model limitations, and ultimately improving reliability in real-world scientific reasoning tasks.

3. AutoBench Dataset Overview

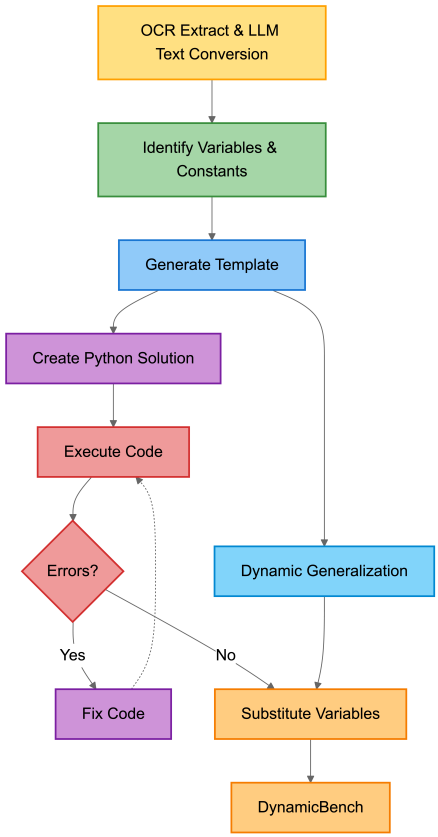

Figure 2. Step-by-step overview of the AutoBench dataset creation pipeline.

We constructed the AutoBench dataset through a rigorous, multi-stage pipeline designed to ensure scientific precision, generalization robustness, and computational verifiability. The process encompasses the extraction, generalization, paraphrasing, code generation, and validation of physics problems along with their reasoning steps and executable solutions. The following section outlines the methodology behind this pipeline, detailing each stage that contributes to the creation of the benchmark.

Methodology

Figure 2 provides an overview of the methodology used to create **AutoBench**, a comprehensive physics-based reasoning benchmark. This pipeline is designed to be scalable and adaptable to a variety of scientific domains, ensuring that the benchmark can evolve and be applied across different fields. Below, we detail each stage of the pipeline that contributes to the creation of the benchmark.

3.1. OCR and Problem Extraction

We begin by applying OCR techniques coupled with domain-specific heuristics to extract relevant physics problems from a variety of sources.

3.2. Image-to-Text Conversion

Next, we use the Llama-3.2-90B-Vision-Instruct model [7] selected for its proven reliability in prior benchmarks to convert the extracted problem into structured text. This output includes the problem statement and step-by-step reasoning steps in textual form.

3.3. Structured Information Extraction

Next, we employ the Llama-3.3-70B-Instruct [7] text-based language model to extract structured representations from the textual problem. The model is prompted to output information in a standardized format:

1. **Input variables** with numerical values and physical units (e.g., `"v1": [2, "m/s"]`).
2. **Output variables** with their expected final values and units (e.g., `"v_a_prime": [-3, "m/s"]`).
3. **Relevant constants**, such as gravitational acceleration (`"g": [9.8, "m/s$^2$"]`).

This structured format serves as a foundation for generating parameterized code and enables systematic variation across problem instances.

3.4. Dynamic Generalization

We then apply carefully designed templates to generalize each extracted problem. These templates emphasize:

1. The core physics concepts involved.
2. Known versus unknown quantities.
3. Selection and application of appropriate equations.
4. Logical step-by-step solution derivation.

Each generalized instance is used to generate multiple paraphrased variants, preserving physical integrity while enhancing dataset diversity and model robustness.

3.5. Python Code via Custom Scaffolding

Using the structured information and paraphrased problems, we prompt the Llama-3.3-70B-Instruct model to generate executable Python code. To ensure generalizability:

- Input variables and physical constants are passed as structured inputs.
- Output variables are initialized as placeholders and computed symbolically.
- Final answers are returned in a standardized dictionary format for consistent evaluation.

Use of Standard Libraries. To ensure correctness and dimensional consistency, all code utilizes:

- **Pint**: Enforces unit consistency across all numerical operations.
- **SymPy**: Enables symbolic algebra, equation solving, and analytical manipulation.

3.6. Computational Validation and Feedback

To ensure reliability, each generated Python script undergoes a two-stage validation process:

1. **Dimensional validation**: Verifies that all quantities conform to correct physical units using Pint.
2. **Numerical validation**: Confirms computed values match expected outputs using the original input values.

In case of discrepancies, the failure cases are automatically fed back into the model for iterative correction.

Final Dataset Construction

Once validated, each problem template is instantiated with randomized numerical values (within physically meaningful bounds) to generate diverse variants. For example, constraints are applied to prevent values that violate physical laws (e.g., speeds exceeding the speed of light). Each problem instance is accompanied by:

- A paraphrased textual problem.
- A structured reasoning explanation.
- Executable Python code with symbolic and numerical output.
- The generated input variables, constants, and the final answer computed through Python code.

AutoBench ultimately comprises over 15,000 rigorously validated physics problems, forming a dynamic and executable benchmark that tests model reasoning across both symbolic and computational dimensions.

Table 1 presents the distribution of our dataset across various physics topics.

Figure 1 provides a representative sample from the benchmark. Additional examples can be found in the Appendix (see Section 8).

Physics Topic	Percentage (%)	Number of Instances
Mechanics	33.56	5049
Electricity and Magnetism	21.35	3213
Modern Physics	11.53	1734
Thermodynamics	8.14	1224
Waves and Oscillations	10.16	1530
Fluid Mechanics	10.17	1530
Waves and Optics	5.09	765
Total	**100.00**	**15045**

Table 1. Distribution of dataset instances by physics topics in AutoBench. Percentages are rounded to two decimal places.

4. Experimental Results and Insights Beyond Accuracy

We evaluate a range of state-of-the-art instruction-tuned LLMs on **AutoBench** to assess their scientific reasoning capabilities under dynamic and perturbed conditions. To ensure an unbiased evaluation, we exclude LLaMA-based models from testing, as the dataset was partially generated using a LLaMA model. To this end, we measure several key metrics:

- **Exact Match Accuracy**: The proportion of problems for which the model produces a completely correct end-to-end solution.

$$\text{Exact Match Accuracy} = \frac{\text{\# fully correct solutions}}{\text{\# total problems}}$$

A substantial portion of the problems in our dataset consists of multiple subproblems (e.g., parts a, b, c, etc.). To compute exact match accuracy, we evaluate whether the model provides correct solutions for all parts of the problem. Given that many problems are composed of multiple sub-steps, we also introduce Partial Accuracy as a metric. This measures the fraction of sub-steps that the model answers correctly, providing a more granular assessment of performance. Partial accuracy captures cases where the model demonstrates progress by solving individual sub-steps correctly, even if it does not fully solve the entire problem. This metric is particularly useful for identifying areas where the model excels in certain aspects of a problem but requires further improvement in others.

- **Partial Accuracy**: The fraction of individual sub-steps within a structured solution that the model answers correctly.

$$\text{Partial Accuracy} = \frac{\text{\# correct sub-steps}}{\text{\# total sub-steps}}$$

Beyond accuracy, we also measure additional performance metrics:

- **Consistency Score**: The proportion of problem groups where the model consistently provides the correct answer across all perturbed variants, reflecting the stability and reliability of the model's performance. A high consistency score indicates that the model can reliably solve problems even with slight variations, showcasing its generalization ability.

$$\text{Consistency Score} = \frac{\text{\# groups with all correct variants}}{\text{\# total problem groups}}$$

To further assess the model's performance under uncertainty, we also compute the following metric:

- **Balance Score**: The Balance Score indicates the fraction of problem groups where the model's accuracy across variants is around 40%-60%, reflecting uncertainty in the model's reasoning. It provides insight into situations where the model may be guessing or uncertain about the correct approach.

$$\text{Balance Score} = \frac{\text{\# groups with } \sim 50\% \text{ accuracy}}{\text{\# total problem groups}}$$

- **Negative Consistency Score**: This metric tracks the proportion of problem groups where the model answers all variants incorrectly. A high Negative Consistency Score indicates areas of consistent failure, providing valuable diagnostic information for improving model performance.

$$\text{Negative Consistency Score} = \frac{\text{\# groups with all incorrect variants}}{\text{\# total problem groups}}$$

Table 2 provides a comprehensive evaluation of large language models on AutoBench, employing five crucial metrics: Partial Accuracy, Exact Match Accuracy, Consistency Score, Negative Consistency Score, and Balance Score. Notably, Anthropic Sonnet-3.7 and Gemini-2.0-Flash [1, 6] exhibit the strongest overall performance, achieving Exact Match Accuracy exceeding 64% alongside high Partial Accuracy, indicative of reliable step-by-step reasoning. Sonnet-3.7 stands out with the highest Consistency Score (42.42%), demonstrating exceptional robustness to paraphrased and perturbed variants of identical questions. The Balance Score, quantifying the proportion of problem groups where model accuracy across variants hovers around 50%, reflects the model's uncertainty in its reasoning; stronger models like Sonnet-3.7 display a lower Balance Score. While Qwen2.5-72B-Instruct [3] particularly in Partial Accuracy, they encounter challenges in maintaining consistency. In contrast, GPT-4-Turbo [11] presents solid overall metrics but underperforms in consistency compared to Sonnet-3.7 and Gemini. The significant disparity between the 7B and 72B versions of Qwen2.5 underscores the pivotal role of model scale in achieving robust physics reasoning. Ultimately, these results emphasize the critical necessity of evaluating not only accuracy but also consistency and robustness in reasoning tasks—qualities often neglected by traditional metrics yet paramount for real-world reliability in multimodal scientific problem-solving.

5. Measuring Hallucination

A key strength of our benchmark is its ability to systematically induce and detect hallucinations in LLM model responses by dynamically altering or concealing critical components of a physics problem, such as input variables or domain-specific constants. This functionality enables controlled testing of model behavior under uncertainty. When confronted with incomplete information: *Does the model seek clarification? Does it make reasonable assumptions and state them explicitly? Does it hallucinate values and proceed as if the input were fully specified?* Our benchmark

Table 2. Performance Metrics across LLMs on AutoBench. Includes Partial Accuracy, Exact Match Accuracy, Consistency Score, Negative Consistency Score, and Balance Score.

Model	Partial Accuracy	Exact Match Accuracy	Consistency Score	Negative Consistency Score	Balance Score
Qwen2.5-7B-Instruct	24.26%	16.44%	5.66%	41.51%	15.09%
Qwen2.5-72B-Instruct	66.57%	61.69%	37.74%	15.09%	11.32%
OpenAI GPT (gpt-4-turbo)	60.59%	53.73%	33.33%	18.18%	12.12%
Gemini-2.0-Flash	71.43%	64.49%	34.38%	9.38%	12.50%
Anthropic Sonnet-3.7	70.81%	65.48%	42.42%	18.18%	6.06%

is uniquely designed to probe these behaviors, allowing us to quantify reasoning integrity in under-specified scenarios and assess the robustness of models under challenging conditions.

We illustrate this with a representative example:

Example of Hallucination

Question: Determine the average time required for a glucose molecule to diffuse a distance of 0.00991 m in water.

This question omits the diffusion coefficient D, a necessary constant for computing the answer via the physical equation $t = \frac{x^2}{2D}$. However, when prompted with this version, Gemini does not request the missing value or flag the input as incomplete. Instead, the model fabricates a response by assuming an image is provided, stating: 'Based on the image,' and then extrapolates from a hallucinated example involving diffusion over 0.010 m in 7.5×10^4 seconds. It uses the proportionality $t \propto x^2$ to calculate:

$$t_2 = t_1 \cdot \left(\frac{x_2^2}{x_1^2} \right),$$

with $t_1 = 7.5 \times 10^4$ s, $x_1 = 0.010$ m, $x_2 = 0.00991$ m,

leading to an incorrect answer.

This response reflects a hallucinated reasoning chain. Instead of applying the correct physics or querying for D, the model infers a scenario that was never presented. Such behavior can be quantitatively evaluated in our benchmark by selectively omitting critical variables and analyzing how often models hallucinate versus recognize under-specified inputs.

In future work, we plan to formalize this capability and systematically benchmark hallucination rates across model families. This expands the scope of our dataset beyond correctness and robustness, making it a valuable tool for studying reasoning integrity under *partial* or *ambiguous* inputs.

6. Example-Level Insights

While aggregate metrics provide a high-level view of model performance, deeper insights emerge when examining specific examples and their variations. In this section, we analyze representative cases that highlight recurring success patterns, consistent failure modes, and surprising behaviors across paraphrased or perturbed inputs.

These quantitative and qualitative observations shed light on the limitations of current models in terms of robustness, generalization, and interpretability.

Case Study: Thermodynamics Question on Kinetic Energy and RMS Speed We analyze three semantically equivalent versions of a physics question requiring symbolic reasoning and precise numerical calculation. The first part of the task is to compute the average kinetic energy of a gas molecule nitrogen molecules at a given temperature. Constants are explicitly provided.

Version 1

Question: Determine the average kinetic energy of a gas molecule at a temperature of **182.0 K**. Additionally, calculate the root mean square (rms) speed of a nitrogen molecule (N_2).
Constants: Boltzmann constant = 1.38×10^{-23} J/K; Mass of N_2 molecule = 4.65×10^{-26} kg

LLM Response (Qwen2.5-7B):

Step 1: $KE = \frac{3}{2} k_B T = \frac{3}{2} \times 1.38 \times 10^{-23} \times 182.0 = \mathbf{8.06 \times 10^{-20}}$ J
Final Answer: **KE = 2.54 J** (incorrect by several orders of magnitude)

Ground Truth: 3.7674×10^{-21} J

Error: The model produces an answer that is several orders of magnitude incorrect, suggesting a fundamental error in the solution process.

Version 2

Question: a) Determine the average kinetic energy of a gas molecule at **202.0 K**. b) Calculate the RMS speed of N_2. Same constants provided.

LLM Response (Qwen2.5-7B):

Answer: KE: 4.786×10^{-21} J

Ground Truth: 4.1814×10^{-21} J
Observation: Despite minor numerical discrepancy, this response is approximately correct, indicating higher accuracy under this paraphrase.

Version 3

Question: Determine the average kinetic energy of a gas molecule at **270.0 K** and the RMS speed of N_2. Same constants provided.

LLM Response (Qwen2.5-7B):

Answer: KE: 2.127×10^{-20} J

Ground Truth: 5.589×10^{-21} J
Error: Kinetic energy is overestimated by more than **3**×, possibly due to arithmetic mistakes or symbolic misinterpretation.

Similar errors in Larger Models

Our observation is not limited to smaller models like Qwen2.5-7B. Even Qwen2.5-72B, despite using the correct physics formula and providing step-by-step reasoning, often produces numerically inconsistent results.

Consider the following example:

Qwen2.5-72B: Incorrect Electric Field Calculatio

Question: Determine the magnitude of the electric field E generated by a point charge of 2.09×10^{-9} C at a distance of 0.00567 m. Use Coulomb's constant $k = 8.99 \times 10^9\,\mathrm{N \cdot m^2/C^2}$.

Qwen2.5-72B Answer: Applies $E = \frac{kq}{r^2}$, computes $E \approx 587.5$ N/C.

Ground Truth: $E \approx 584{,}440$ N/C

The model uses the correct formula and walks through intermediate steps, but its final numeric output is off by nearly three orders of magnitude. This suggests that the issue is not conceptual misunderstanding but internal instability in arithmetic or symbolic execution. Similar inconsistencies were observed across other problems with slight input perturbations.

Implicit Simplification Bias: Another class of error arises in problems requiring more advanced topics. For example in question related to relativistic mechanics even when the scenario clearly demands relativistic treatment, the Qwen2.5-72B frequently defaults to oversimplified Newtonian expressions, for example using $a = \frac{F}{m}$ or $a = \frac{F}{\gamma m}$ without accounting for the orientation of the force relative to velocity. We call this behavior implicit simplification bias where the model superficially identifies relevant physical variables but fails to apply the correct governing equations when deeper conceptual distinctions are required. This suggests that such biases are not merely a consequence of model size but rather reflect fundamental gaps in their understanding of domain-specific complexities, highlighting the need for explicit training in these advanced areas.

7. Conclusion and Future Work

We introduced AutoBench, a benchmark designed to rigorously evaluate the scientific reasoning capabilities of large language models within the domain of physics. Each problem instance includes structured, step-by-step reasoning and is paired with executable Python code to ensure numerical accuracy and dimensional consistency, promoting scientific rigor and interpretability.

A key contribution of AutoBench is its emphasis on generalization. By incorporating diverse variations of each core problem, the benchmark assesses the robustness of model reasoning across different inputs and contexts. In addition to standard accuracy, we propose new evaluation metrics such as *consistency*, which captures output stability across paraphrased or perturbed inputs, and the *balance score*, which reflects uncertainty in model predictions to enable a more comprehensive analysis of model behavior.

In future work, we plan to expand AutoBench in several directions. Future iterations will incorporate multimodal reasoning tasks that require the integration of textual, visual, and symbolic information. We also aim to extend the benchmark to interdisciplinary STEM domains, supporting the evaluation of models capable of complex, cross-domain scientific reasoning. Through these efforts, we hope to advance the development of AI systems with robust, transparent, and reliable scientific reasoning capabilities.

References

[1] Anthropic. Claude (sonnet). https://www.anthropic.com/claude/sonnet. 5

[2] Daman Arora, Himanshu Gaurav Singh, et al. Have llms advanced enough? a challenging problem solving benchmark for large language models. *arXiv preprint arXiv:2305.15074*, 2023. 3

[3] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023. 5

[4] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017. 2

[5] Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. Retiring adult: New datasets for fair machine learning. *Advances in neural information processing systems*, 34: 6478–6490, 2021. 3

[6] Google DeepMind. Gemini. https://gemini.google.com/. 5

[7] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 4

[8] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021. 1, 3

[9] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022. 1

[10] Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521, 2022. 1, 3

[11] OpenAI. Chatgpt plugins, 2023. Accessed: 2025-03-18. 5

[12] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022. 1, 2

[13] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

[14] Matthew Renze and Erhan Guven. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024. 1

[15] Liangtai Sun, Yang Han, Zihan Zhao, Da Ma, Zhennan Shen, Baocai Chen, Lu Chen, and Kai Yu. Scieval: A multi-level large language model evaluation benchmark for scientific research. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 19053–19061, 2024. 1, 3

[16] Angelina Wang, Aaron Hertzmann, and Olga Russakovsky. Benchmark suites instead of leaderboards for evaluating ai fairness. *Patterns*, 5(11), 2024. 3

[17] 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. *arXiv preprint arXiv:2307.10635*, 2023. 1, 3

[18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022. 1, 2

[19] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. *URL https://arxiv. org/abs/2305.10601*, 3, 2023. 1

AutoBench: A Dynamic Benchmark for Auditing Scientific Reasoning in Large Language Models

Supplementary Material

8. Examples from AutoBench

Here are some examples from AutoBench, where each question is shown along with step-by-step reasoning and the corresponding Python code. We populate the problems with numerical values, and the relevant variables are generated as part of the pipeline.

8.1. Example A

Question

In a simplified atomic model, the most probable distance between the nucleus and an electron is $r = 3.33e - 11\,\text{m}$. The nucleus contains 1.3 protons. Determine the electric field due to the nucleus at the electron's position.
Here are constants:

$$\text{Permittivity of free space} = 8.85 \times 10^{-12}\,\frac{\text{C}^2}{\text{N}\cdot\text{m}^2}$$

$$\text{Elementary charge} = 1.6 \times 10^{-19}\,\text{C}$$

Solution

Identify Relevant Concepts

- The electric field due to a point charge is given by

$$\vec{E} = \frac{1}{4\pi\epsilon_0}\frac{q}{r^2}\hat{r}$$

  where ϵ_0 is the permittivity of free space, q is the charge, and r is the distance from the charge.
- The goal is to calculate the electric field at the electron's position.

Set Up the Problem

- The electric field at a distance r from a point charge is given by the formula above.
- The direction of the electric field is radially outward from the nucleus.

Execute the Solution

- Substituting the given values into the formula

Evaluate Your Answer

- The electric field is expected to be radially outward from the nucleus due to its positive charge.
- If r were very small, the electric field would be very large, and if r were large, the electric field would approach zero, which is physically reasonable.

Python code:

```
import sympy as sp
from pint import UnitRegistry
ureg = UnitRegistry()
Q_ = ureg.Quantity

def electric_field_at_electron(r, e, number_of_protons,
    epsilon_0):
   # Convert inputs to Pint quantities
   r = Q_(r).to(ureg.meter) # Ensure meters
   e = Q_(e).to(ureg.coulomb) # Ensure coulombs
   number_of_protons = Q_(number_of_protons).to(ureg.
       dimensionless)
   epsilon_0 = Q_(epsilon_0).to(ureg.farad / ureg.meter)
        # Ensure F/m

   r = r.magnitude
   e = e.magnitude
   number_of_protons = number_of_protons.magnitude
   epsilon_0 = epsilon_0.magnitude

   # Define symbolic variables
   q = e * number_of_protons
   E = sp.Symbol('E', real=True, positive=True)

   # Calculate the electric field
   E = (1 / (4 * sp.pi * epsilon_0)) * (q / r**2)

   return {
      'E': E.evalf()
   }
```

8.2. Example B

Question

Consider a solid metal cube with an edge length of $L = 0.0237\,\text{m}$.
(a) Determine the lowest energy level for an electron within this metal.
(b) Calculate the energy difference between this level and the next higher energy level.
Here are constants:

$$\text{Reduced Planck's constant } \hbar = 1.05 \times 10^{-34}\,\text{J}\cdot\text{s}$$

$$\text{Electron mass } m_e = 9.11 \times 10^{-31}\,\text{kg}$$

$$\text{Ground state quantum numbers: } n_x = n_y = n_z = 1$$

$$\text{Next state quantum numbers: } n_x = 2,\ n_y = 1,\ n_z = 1$$

Solution

Identify Relevant Concepts
- Model the electron as a particle in a 3D box.
- Energy levels are given by:

$$E(n_x, n_y, n_z) = \frac{\pi^2\hbar^2}{2m_eL^2}(n_x^2 + n_y^2 + n_z^2)$$

Set Up the Problem
- Ground state: $n_x = n_y = n_z = 1$
- Next higher level: $n_x = 2, n_y = 1, n_z = 1$

Execute the Solution
- Compute:

$$E_1 = \frac{\pi^2\hbar^2}{2m_eL^2}(1^2 + 1^2 + 1^2)$$

$$E_2 = \frac{\pi^2\hbar^2}{2m_eL^2}(2^2 + 1^2 + 1^2)$$

- Energy difference:

$$\Delta E = E_2 - E_1$$

Evaluate Your Answer
- Positive energy difference is expected since next level is higher.
- Larger cube size would reduce energy spacing, consistent with quantum model.

Python code:

```
import sympy as sp
from pint import UnitRegistry
ureg = UnitRegistry()
Q_ = ureg.Quantity

def electron_energy_levels(L, h_bar, m_e, n_x, n_y, n_z,
      n_x_next, n_y_next, n_z_next):
   L = Q_(L).to(ureg.meter)
   h_bar = Q_(h_bar).to(ureg.joule * ureg.second)
   m_e = Q_(m_e).to(ureg.kilogram)

   L = L.magnitude
   h_bar = h_bar.magnitude
   m_e = m_e.magnitude

   pi = sp.pi

   def energy(n_x, n_y, n_z, L, h_bar, m_e):
      return (pi**2 * h_bar**2 / (2 * m_e * L**2)) * (
           n_x**2 + n_y**2 + n_z**2)

   E1 = energy(n_x, n_y, n_z, L, h_bar, m_e)
   E2 = energy(n_x_next, n_y_next, n_z_next, L, h_bar,
        m_e)
   DeltaE = E2 - E1

   return {
      'E1': E1.evalf(),
      'E2': E2.evalf(),
      'DeltaE': DeltaE.evalf()
   }
```