Exploring Structures in Physics Problems: Can AI Agents Discover Statistical Mechanical Mappings?

Wanyu Zhao*
University of Illinois Urbana-Champaign
wanyu2@illinois.edu

Wanbing Zhao*
Rice University
wz56@rice.edu

Abstract

An important skill in theoretical physics is to recognize when a new problem can be transformed into a known model. We study this skill as an AI-agent task: can LLM-based agents discover statistical mechanical mappings from a raw partition function to a tractable representation? To probe this question, we introduce StatMechBench-v0, a benchmark of six Ising-type problems covering transfer-matrix methods, gauge-removable disorder, planar/Pfaffian structure.

We evaluate a simple propose–verify–revise agent across multiple LLMs and problem phrasings. The results show that numerical feedback often helps agents repair code and recover correct partition functions on canonical tasks. However, agents can also pass small-size numerical checks while misidentifying the underlying tractable class or overstating computational complexity. These failures suggest that reliable scientific agents need more than numerical agreement: they require symbolic checks, structural invariants, and explicit complexity auditing. Our study provides an early benchmark and design direction for AI agents aimed at structural discovery in theoretical physics.

1 Introduction

A common mindset in theoretical physics is to relate a new problem to existing theories, such as mapping newly formulated models to well-studied ones. Statistical mechanics, a key branch in physics, has developed a collection of paradigmatic models to understand how microscopic degrees of freedom give rise to macroscopic phases and critical phenomena (Baxter [1982]). Examples include the Ising model (Lenz [1920], Ising [1925]), the Potts model (Potts [1952]), and many other exactly solved lattice models (Baxter [1982]). Developed and solved by generations of physicists over the past century, these models encode invaluable expert knowledge, and physicists seek transformations to connect new statistical mechanical problems to existing models, allowing known results, intuitions, and tools to be reused.

While successfully relating one statistical mechanical model to another can itself be a profound physical result, as exemplified by the Kramers–Wannier duality (Kramers and Wannier [1941a]), it also serves as one step in a broader research workflow. For example, physicists identify a mapping from a new problem to an existing model and use it to compute quantities of interest, simplify the analysis, and gain qualitative understanding of the original problem. However, recognizing or constructing such mappings is nontrivial: it often requires familiarity with existing models, mathematical or physical intuition about the structure of the original problem, and the ability to apply suitable transformations. Historically, many important mappings have required substantial expert effort, and this difficulty can become a key bottleneck in research, especially for those outside the relevant subfield.

*Equal contribution.

1st ACM Conference on AI and Agentic Systems (ACM CAIS 2026). Workshop: AI Agents for Discovery in the Wild (AID-Wild 2026).

In this work, we ask: *Can we build an LLM-based AI agent to discover statistical mechanical mappings?* Formally, we view a mapping as a transformation from the original problem representation to a reference model whose solution structure is already known (§2). Our key insight is that LLMs could have learned about all the existing physics models as they are described both mathematically and in natural language across textbooks, papers, and lecture notes, and an LLM-based agent may be able to propose plausible mapping hypotheses by combining model descriptions, symbolic patterns, and high-level physical analogies, with its reasoning and self-reflection abilities.

We first build our insight and understanding of current LLMs' ability in this domain through a systematic evaluation (§3). Specifically, we collect a set of representative statistical mechanical mapping problems and group them by difficulty into three tiers (§3.1). Our preliminary evaluation on a subset of problems shows that LLMs can find the mappings and construct tractable partition-function representations correctly with feedback loops, under both canonical and paraphrased problem descriptions (§3.2, §3.3). We also identify several failure modes. These observations both reveal the ability of current AI and shed light on our next step of building a discovery agent.

Based on our problem categories and evaluation findings, we propose future statistical mechanical discovery agent designs and challenges (§4). Our ultimate goal is to support frontier research, e.g., error thresholds for quantum codes (Dennis et al. [2002], Chubb and Flammia [2021]) and phase-transition questions in monitored quantum dynamics (Skinner et al. [2019], Bao et al. [2020]). More broadly, as an early attempt to concretely apply AI agents to open-ended theoretical physics problems, we hope this work encourages physicists to further explore the role of LLMs in research. By highlighting both the strengths and limitations of current agentic AI systems in assisting scientific discovery, this work aims to contribute to accelerating progress in science.

2 Background and Problem Formulation

The aim of *statistical mechanics* is to explain and predict macroscopic behavior of a system from its microscopic degrees of freedom and interactions.

In this section, we provide necessary background on statistical mechanics, following Ref. Baxter [1982], and then formalize the problem of "discovering statistical mechanical mappings". We end the section with a review of recent progress in AI agents for scientific discovery that inspires this work.

2.1 Partition function and exactly solvable models

We begin with the partition function Z, which allows macroscopic thermodynamic observables to be obtained from microscopic configurations and will serve as our starting point for constructing a statistical-mechanical mapping.

Classical equilibrium statistical mechanics is formulated in terms of the *partition function*[2]:

$$Z = \sum_{s} e^{-\beta H(s)}, \tag{1}$$

where s denotes a microscopic state, or configuration, of the system, e.g., for a system with n binary degrees of freedom, $s = (s_1, \ldots, s_n)$ with $s_i \in \{-1, +1\}$ for $i \in \{1, \ldots, n\}$; $H(s)$ denotes the energy of the configuration s; and β is the inverse temperature. Z is understood to depend on the physical model parameters entering $H(s)$ and on temperature; this dependence is left implicit here. The sum—for discrete systems—becomes an integral and a trace for continuous and quantum systems respectively. Partition function is central to statistical mechanics because normalizing $e^{-\beta H(s)}$ by Z gives the probability distribution over microscopic configurations s (i.e., $p(s) = \frac{e^{-\beta H(s)}}{Z}$); then, macroscopic observables can be obtained by taking weighted averages over microscopic states.

Although Z links microscopic states to macroscopic observables, directly evaluating it as in Eq. 1 is intractable for realistic interacting[3] systems of macroscopic size, which contain an enormous number of degrees of freedom, typically on the order of Avogadro's number (10^{23}). For example,

[2]For concreteness, we describe the canonical ensemble at fixed temperature. Other equilibrium ensembles have analogous partition functions.

[3]Here, "interacting" means that the energy function $H(s)$ contains terms involving more than one degree of freedom, for example, pairwise interactions between the i-th and j-th components of $s = (s_1, \ldots, s_n)$.

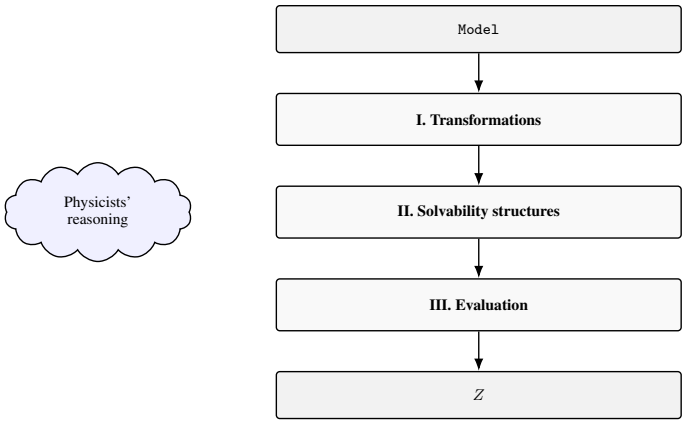

Figure 1: General logic underlying the exact evaluation of a partition function Z.

consider a system of n binary degrees of freedom, i.e., $s = (s_1, \ldots, s_n)$ with $s_i \in \{0, 1\}$, a direct computation of Eq. 1 involves a sum over 2^n ($n \sim 10^{23}$) configurations. Consequently, one or both of the following strategies is adopted: (i) replacing the real system by a simplified idealization (a *model*), specified by the states s and the energy function $H(s)$ chosen to capture the relevant physics while giving the partition function a structure that makes it tractable. (ii) introducing approximations to evaluate the sum.

In special cases, the structure of the model allows the partition function to be evaluated exactly; such models are usually called *exactly solvable models*. To date, the catalog of exactly solvable models remains limited, but it forms a valuable theoretical library for physicists to use, marking the current boundary of analytical control. Table 1 summarizes four groups of exactly solvable models.

2.2 Statistical mechanical mapping

We extract the general logic of exactly evaluating a partition function Z in Figure 1: one maps a model to an equivalent formulation where solvable structures become manifest, which then enables an explicit evaluation of the partition function. This logic also guides the study of new models: when a model is first introduced to explain a new phenomenon, one useful starting point is to seek transformations that reveal whether it has solvable structure or can be connected to a known model. To scale up the search for transformations that may open tractable solution paths, we ask:

Can an LLM agent uncover the solvability structure of a problem by mapping it onto a known statistical mechanical model?

We refer to such a task as a *statistical mechanical mapping*. As an initial study, this work focuses on cases where the quantity of interest is presented as a raw partition function, or a raw partition-function-type sum of the form in Eq. 1, i.e., a sum over exponentially many configurations arising from the problem; accordingly, "mapping" means the mathematical transformations bringing the raw partition function into the partition function of a known model, e.g., changes of variables, gauge transformations, and duality transformations. Such mapping plays a role analogous to Step I in Figure 1, and once the problem is mapped to an exactly solvable model, subsequent steps follow directly.

A successful partition-function mapping generally has both physical and methodological value. From a physical perspective, it can place a new problem within an existing universality[4] class. From a methodological perspective, analytical control over the mapped model can be transferred back to the original problem. Even when the mapped partition function is not exactly solvable, the mapping still provides physical insight and allows one to apply model-specific numerical methods, such as Monte Carlo or tensor network methods.

[4]In particular, if a model with the same dimensionality, symmetry, and interaction range as a given physical system can be solved, universality implies identical critical behavior, characterized by the same critical exponents near phase transitions.

Table 1: Representative classes of exactly solved models, adapted from Ref. Baxter [1982].

Class	Representative models and references
One-dimensional	Ising chain (Ising [1925]); one-dimensional Coulomb systems (Lenard [1961], Baxter [1963])
Infinite-dimensional	Mean-field and Bethe-lattice models; Kac-type long-range models (Kac et al. [1963], Uhlenbeck et al. [1963], Hemmer et al. [1964])
Spherical model	Berlin–Kac spherical model and its infinite-spin-dimensional interpretation (Montroll [1949], Berlin and Kac [1952], Stanley [1968])
Two-dimensional lattice	Ising model (Kramers and Wannier [1941b,c], Onsager [1944]); ferroelectric/six-vertex models Lieb [1967a,c,b]; eight-vertex model (Baxter [1971]); three-spin model (Baxter and Wu [1973])

To the best of our knowledge, no prior work have investigated using LLM agent to identify statistical mechanical mappings. The closest work Gupta et al. [2025] uses machine learning algorithms to learn the resulting function mapped by duality (a specific transformation), while this work asks if an LLM agent can discover the mapping itself.

2.3 AI agents for scientific discovery

Several LLM-based agents have been proposed for scientific and mathematical discovery through program search. FunSearch (Romera-Paredes et al. [2024]) introduced an LLM-guided evolutionary search framework that pairs a pretrained LLM with an automated evaluator to discover programs for mathematical and algorithmic problems. AlphaEvolve (Novikov et al. [2025]) extends this line of work to a more general evolutionary coding agent that can modify larger codebases and optimize algorithms across scientific, mathematical, and engineering tasks. In theoretical physics, Brenner and Colwell [2026] combine Gemini Deep Think, tree search, and automated numerical feedback to derive analytical solutions for a cosmic-string radiation problem by evaluating a core integral. It builds on a broader LLM–tree-search systems for scientific software generation called ERA (Aygün et al. [2025]), which formulate scientific software development as a scorable search problem and demonstrate results across multiple scientific domains. Song et al. [2025] evaluate LLMs on scenario-grounded scientific-discovery tasks across biology, chemistry, materials science, and physics (including solving an Ising model). This work studies a specific mechanism of theoretical discovery—statistical mechanical mapping, yet builds on many ideas developed in previous works.

3 Probing LLM Agents for Statistical Mechanical Mapping

As statistical mechanical mapping (§2.2) spans broad problem domains, touches diverse exactly solvable models and mathematical transformations, and can serve different purposes (e.g., computing quantity of interest vs. analyzing physics), and as today's LLM agent has a large design space (from prompt design to tool uses) with the LLM capability remaining opaque (e.g., memorization vs. reasoning), directly building a full end-to-end agent to achieve our task is unwieldy and challenging. Therefore, we conduct a probing evaluation to understand LLM capabilities and guide our agent design. We first categorize statistical mechanical problems into three tiers by difficulty and then evaluate a naive LLM agent on a small benchmark of easiest-tier problems. Our evaluation shows promising results of using feedback-guided LLM agents to identify the mappings. We also reveal several failure modes that inform the agent design directions in §4.

3.1 Three-tier problems

We classify the statistical mechanical model mapping problems into three tiers by difficulty. We instantiate a Tier 1 benchmark in §3.2 for evaluation, with Tier 2 and 3 problem study left for future work.

Tier 1: canonical models with known exact solution mechanisms. Tier 1 consists of textbook canonical examples of exactly solved statistical mechanical models as listed in Table 1. These models naturally compose a sanity-check benchmark because the mechanisms for evaluating their partition functions—e.g., transfer-matrix form, gauge-removable disorder, or planarity/Pfaffian structure—are

well understood and analytically controlled. In this tier, the mapping agent's task is to recover a known exact solution/reduction or exact tractable representation given the model's original formulation.

Tier 2: established mappings in quantum information literature. Tier 2 consists of known statistical mechanical mappings from the quantum information literature which are substantially more challenging to recover than those in Tier 1. Examples include toric-code threshold mappings with perfect or imperfect syndrome (Dennis et al. [2002]), correlated-noise or subsystem-code generalizations (Chubb and Flammia [2021]), coherent-noise extensions (Behrends and Béri [2025]), and replica mappings for monitored random circuits in the large local Hilbert-space dimension limit (Bao et al. [2020]). In this tier, the mapping agent's task, given a raw partition function, is to recover an effective energy function $H(s)$ and identify any special parameter limits that enable the mapping.

Tier 3: open but structured research problems. Tier 3 consists of open research problems for which statistical mechanical reformulations are expected to be useful, but for which no sufficiently-tractable mapping is known. One example is maximum-likelihood threshold problems for structured quantum code families under various noise models, for which useful reductions remain unclear after a raw partition-function representation is obtained; another example is analytical approaches to measurement-induced entanglement phase transitions at finite local Hilbert-space dimension. In this tier, a useful agent output may be a partial reduction, an effective Hamiltonian, a candidate universality class, a numerically testable observable, or evidence that no simple mapping is likely.

3.2 Experiment design

Table 2: StatMechBench-v0: six tasks, four tractable classes covered. n is the number of spins; L is the linear lattice size when applicable ($n = L^2$).

ID	Task	Probed n	Tractable class	Complex.
P01	1D Ising chain, OBC, uniform J	$\{6, 10, 14\}$	`transfer_matrix`	$O(n)$
P02	1D Ising chain w/ random site fields, OBC	$\{6, 10, 12\}$	`transfer_matrix`	$O(n)$
P03	2D ferromagnetic Ising on $L \times L$ torus	$\{9, 16\}$	`transfer_matrix`	$O(L\, 2^{2L})$
P04	2D Ising spin glass on planar grid	$\{9, 16\}$	`pfaffian`	$O(n^3)$
P05	Mattis spin glass, rank-1 disorder	$\{6, 10, 12\}$	`gauge`	$O(n)$
P06	Random 3-regular ± 1 Ising glass	$\{8, 10\}$	`no_efficient_known`	—

Task problems. We select six Tier 1 tasks, all classical Ising-type partition functions, listed in Table 2. We name this small benchmark StatMechBench-v0. The tasks are chosen to span diverse types: canonical transfer-matrix solvability (P01–P02), canonical gauge equivalence (P05), broadly-documented tractable reformulations via a two-dimensional transfer matrix (P03), a Pfaffian reduction for zero-field planar Ising instances (P04), and a documented negative case with no assumed efficient exact mechanism in our predefined library (P06). Each task is labeled with a *tractable class*[5] and the corresponding canonical complexity (as studied by earlier physics works).

A naive mapping agent.

- **Mapping proposer.** The proposer takes the natural-language task description and is asked to return a JSON object with fields `tractable_class`, `complexity_O`, `confidence`, `justification`, and `Z_efficient(params, n)` (Python function) [6]. Task description includes the statistical mechanical mapping problem formulation with a list of tractable class labels (system prompt), the specific problem description (initial user prompt), and failure feedback from earlier runs (feedback user prompt). We make sure no leakage of the tractable class label and corresponding complexity in the prompt. We provide nine tractable classes for each task to be classified into one. The full prompt template and tractable class list are provided in Appendix A.

[5] Tractable class is a unified name in operation level, it can be a target tractable model class, an algebraic transformation, an algorithm, or a graph structural property. Note that each task can be a multi-class problem because multiple tractability mechanisms could be applied. Here in StatMechBench-v0, exactly one label under our consideration applies.

[6] We also do a simple integrity check of code at the Abstract Syntax Tree (AST) stage (e.g., illegal library imports).

Table 3: LLM performance on StatMechBench-v0 across eight models with canonical and paraphrased problem descriptions. Highlighted entries are false positives.

Model	Variant	TC_0 (%) ↑	NM_0 (%) ↑	O (%) ↑	$k^\star$ ↓	Time ↓	Cost ↓
Haiku 4.5	canonical	100	50	100	4	1.6 m	\$0.08
	paraph.	67	50	100	4	1.7 m	\$0.09
Sonnet 4.6	canonical	100	83	83	1	1.7 m	\$0.13
	paraphrased	83	100	83	0	1.6 m	\$0.11
Opus 4.7	canonical	100	83	100	1	1.7 m	\$0.71
	paraphrased	**100**	100	**100**	0	1.2 m	\$0.51
DeepSeek V4-Flash Think	canonical	67	67	100	1	6.6 m	\$0.07
	paraphrased	67	100	83	0	4.5 m	\$0.05
DeepSeek V4-Pro	canonical	100	83	100	1	59.1 m†	\$0.26
	paraphrased	**100**	83	83	1	39.7 m†	\$0.19
Gemini 3.1 Pro	canonical	83	83	83	2	16.0 m	\$1.24
	paraphrased	**83**	83	83	2	18.8 m	\$1.44
GPT-5.4-mini minimal	canonical	100	100	83	0	0.5 m	\$0.01
	paraphrased	67	100	67	0	0.6 m	\$0.01
GPT-5.4-mini high	canonical	83	83	100	3	18.4 m	\$0.31
	paraphrased	**83**	83	100	2	12.6 m	\$0.21

- **Brute-force numerical verifier.** We build a simple verifier that checks numerical equality by brute-force: a `Z_brute(params, n)` function that returns the exact Z via summarization over all 2^n configurations. Given a proposed `Z_efficient(params, n)`, we launch an evaluation process with a timeout, write the candidate code into it, and pass a list of `(params, n)` probes. A trial passes only if all probe results are within a tolerance (relative error $\varepsilon_{\text{rel}} = |Z_{\text{eff}} - Z_{\text{brute}}|/|Z_{\text{brute}}| < 10^{-6}$). On failure a brief textual report listing whether the code ran, the number and rate of failed probes, and an order-of-magnitude relative error (but not Z_{brute} itself) is constructed as feedback of its round. We limit the number of rounds to k.

3.3 Experiments and results

Setup and evaluation metrics. We cover eight models across four vendors, with $k = 5$ verifier rounds, five parameter samples per probed n, and per-trial timeout 30 s.

How do different LLMs perform? Table 3 reports tractable class classification accuracy of the first round (TC_0), numerical evaluation pass rate of the first round (NM_0), post-hoc complexity honesty check score (O), the saturation round number $k^\star$ where the agent succeeds with 100% numerical evaluation pass and stops iteration, together with the wall-clock time and cost to complete the whole evaluation. We omit numerical pass rate with the total $k=5$ loops from the table as all reach 100%, so on canonical Tier 1 tasks the loop alone cannot discriminate frontier models from each other.

Overall, Opus 4.7 performs best, achieving the highest classification accuracy, numerical evaluation pass rate, and complexity honesty score with low k^*. The results also show a clear improvement across Claude models from earlier to later generations and from DeepSeek V4-Flash Thinking to stronger DeepSeek V4-Pro.

In several cases, models pass the probed numerical tests while submitting exponential row-transfer constructions under polynomial-time labels such as `pfaffian`/$O(n^3)$. The O audit detects these complexity mismatches and lowers the corresponding scores ($< 100\%$). Three models remain honest under O on both problem description variants: Haiku 4.5, Opus 4.7 and GPT-5.4-mini high. This also indicates that numerical correctness alone is insufficient to certify the claimed tractable class.

Does iterative refinement loop help? Figure 2 shows that numerical recovery usually occurs quickly. Across models and problem description variants, many saturate within one verifier round.

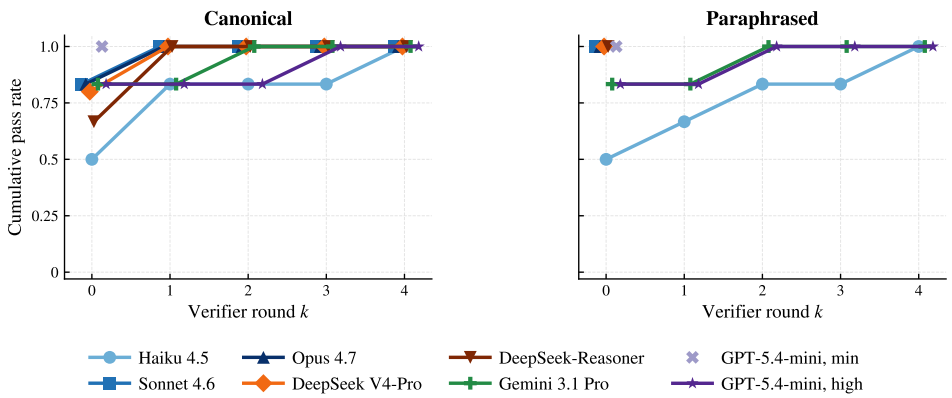

Figure 2: Cumulative numerical evaluation pass rates across self-refinement rounds, with eight LLMs on canonical (left) and paraphrased (right) problem descriptions. A trial first passing at round j is counted as passed for all $k \geq j$. Most reasoning models saturate by round 1, while only a few trajectories use much of the $k = 5$ budget.

Only a small number of trajectories use a substantial fraction of the $k = 5$ budget, typically when magnitude-only feedback cannot localize the structural source of the error.

The recovery traces also show the loop's limitation. We find that when the initial solution is numerically wrong, the verifier can often drive a successful fallback, such as replacing a flawed Pfaffian attempt with a row-transfer matrix. However, when the code is already numerically correct but carries the wrong tractable-class label, the loop has little signal to correct the label (e.g., DeepSeek V4-Flash Thinking and GPT-5.4-mini minimal on paraphrased description in Table 3). Thus the verifier is effective for numerical repair, but not for detecting class-only mismatches.

How does it perform when we paraphrase the problem description? By looking at $\Delta\mathrm{TC}_0 \equiv \mathrm{TC}_{0,\text{paraphrased}} - \mathrm{TC}_{0,\text{canonical}}$, a finding is that reasoning-enabled LLMs are largely paraphrase-invariant, whereas low-effort or non-reasoning ones lose TC_0 accuracy under paraphrase.

Another notable pattern is that paraphrasing can improve zero-shot numerical success (see NM_0 of Sonnet 4.6, Opus 4.7 and DeepSeek V4-Flash Thinking in Table 3). In P04, the canonical wording tends to trigger an ambitious Pfaffian construction, which some models fail to implement reliably. The paraphrased wording instead encourages a simpler row-transfer solution that passes at the probed sizes, but some still claim the `pfaffian` structure, making paraphrasing elicit lower complexity honesty scores. This reveals that LLMs may still rely on memorization rather than genuinely interpreting the mathematics and reasoning through the problem.

How did the agent go wrong? The evaluation reveals several failure modes. Sign errors produce large round-0 numerical mismatches but are usually repairable by magnitude-only feedback; class-only errors can pass numerically while retaining the wrong tractable-class label, leaving the refinement loop with no signal for correction and motivating richer verification checks such as symbolic checks. Some runs exhibit complexity false confidence: exponential implementations are paired with polynomial-time claims and high self-confidence. Canonical wording can induce memorized labels without reliable execution, which paraphrasing mitigates but falsely claims the complexity. The negative control behaves as intended: models consistently identify the no-known-efficient case and use brute force rather than forcing a spurious tractable label. Overall, these failures show that numerical verification alone is weak: it repairs many implementation bugs, but misses class-only and complexity-claim errors.

4 From v0 to the open problems

Although StatMechBench-v0 currently covers a limited set of statistical mechanical mapping problems, the probing evaluation already indicates that LLM-based agents can discover partition-

function representations in these settings. Below, we outline how StatMechBench-v0 can be systematically extended and discuss possible directions for improving the agent architecture.

Dataset

- **Task completion.** StatMechBench-v0 only covers a few tasks in Tier 1 and Tier 2. Moving forward, the benchmark should cover a larger set of both tiers' problems, which will expand the set of the structural classes as well, e.g., Tier 2 problems can require we do some algebraic transformation before having intuition about which tractable models they can potentially map to. Tier 3 open problems might not have mappings to existing physical models; thus, we pivot the agent output to giving a bound instead.
- **Task description.** We manually paraphrased the canonical problem description in StatMechBench-v0. In later version of the benchmark, we dynamically paraphrase with predefined templates like other benchmarks.
- **Task difficulty.** We can expand the benchmark with Tier 2 and 3 problems.
- **Task scale.** We only cover $n \leq 16$. Solving practical problems needs to scale n, which will disable our brute-force numerical verifier.

Agent.

- **Constrained action space.** We define a structured action library to restrict the agent's search space. Given a candidate partition function $Z(p)$, an LLM–CAS pipeline first analyzes its structure by extracting the interaction hypergraph G_f and computing invariants such as treewidth, planarity, symmetry group, and connectivity. These structural features determine a restricted set of admissible transformations, expressed in a domain-specific language (DSL). Within this constrained space, the agent performs symbolic manipulations—including gauge transformations, duality mappings, replica-limit evaluations, and Jordan–Wigner transformations—with each intermediate step checked for algebraic consistency. The overall design integrates multiple validation layers: numerical equivalence, static complexity checks, symbolic verification, and graph invariants.
- **Evaluation stack.** We employ a multi-layer evaluation stack combining static complexity analysis, symbolic verification via a CAS backend, and graph-theoretic invariants (e.g., treewidth, planarity, symmetry group, connectivity). For ground-truth validation, we extend brute-force enumeration with scalable numerical oracles, including MCMC sampling at the Nishimori temperature and tensor-network contraction on the Tanner graph, enabling reliable verification up to $n \sim 100$ for sparse codes. Additional diagnostic signals, such as traceback-style checks, are fed back into the agent context to improve robustness.
- **From tractability to bounds.** Beyond identifying tractable mappings, the framework can be extended to derive rigorous bounds when exact solvability is unavailable, by leveraging partial structure and approximate evaluation.
- **Search and evolution.** Following ERA(Aygün et al. [2025]), we could adopt a tree-search procedure. Each node corresponds to a candidate tractable model class. The LLM proposes symbolic derivations, the CAS executes algebraic steps, and numerical oracles provide verifiable feedback. If the error $\varepsilon < \delta_{\mathrm{tol}}$, the candidate is promoted to larger system sizes and tested for consistency (e.g., phase-transition behavior). Otherwise, failed derivations or $\varepsilon > \delta_{\mathrm{tol}}$ incur penalties, and the search explores alternative hypotheses.
- **Prompting.** Negative prompting (Brenner and Colwell [2026]) is used to enforce structural diversity across search branches and mitigate premature convergence.
- **Human oversight.** Human-in-the-loop supervision is incorporated to audit intermediate results, validate high-level reasoning, and guide exploration in ambiguous regimes.

5 Conclusion

This work takes a first step toward formulating statistical mechanical mapping as an agentic task for AI-assisted theoretical physics. We focus on a restricted setting: raw partition-function expressions whose tractable structure is already known, and ask whether LLM-based agents can recover useful representations from such inputs. To probe this question, we introduce a three-tier problem

taxonomy and instantiate its first tier in STATMECHBENCH-V0, a small benchmark of six Ising-type tasks covering transfer-matrix methods, gauge-removable disorder, planar/Pfaffian structure. Our experiments show that verifier feedback can help agents repair implementation errors and reproduce finite-size partition functions. At the same time, they reveal an important limitation: agreement with brute-force enumeration on small instances does not by itself certify that the agent has identified the correct tractable class, preserved the intended structure, or made an honest complexity claim. These results suggest that reliable agents for physics discovery should not rely on numerical checks alone, but should combine proposal generation with symbolic verification, graph-structural invariants, and explicit complexity auditing. In this sense, STATMECHBENCH-V0 is best viewed not as evidence that current agents can solve open mapping problems, but as a controlled testbed for studying the structural checks such agents will need.

References

Eser Aygün, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Renee Johnston Anton Kast, Cory Y McLean, Peter Norgaard, Zahra Shamsi, et al. An ai system to help scientists write expert-level empirical software. *arXiv preprint arXiv:2509.06503*, 2025.

Yimu Bao, Soonwon Choi, and Ehud Altman. Theory of the phase transition in random unitary circuits with measurements. *Physical Review B*, 101(10):104301, 2020.

Rodney J. Baxter. Statistical mechanics of a one-dimensional coulomb system with a uniform charge background. *Mathematical Proceedings of the Cambridge Philosophical Society*, 59(4):779–787, 1963. doi: 10.1017/S0305004100003790.

Rodney J. Baxter. Eight-vertex model in lattice statistics. *Physical Review Letters*, 26(14):832–833, 1971. doi: 10.1103/PhysRevLett.26.832.

Rodney J. Baxter. *Exactly Solved Models in Statistical Mechanics*. Academic Press, London, 1982. ISBN 978-0-12-083180-7.

Rodney J. Baxter and Fa-Yueh Wu. Exact solution of an ising model with three-spin interactions on a triangular lattice. *Physical Review Letters*, 31(21):1294–1297, 1973. doi: 10.1103/PhysRevLett.31.1294.

Jan Behrends and Benjamin Béri. Statistical mechanical mapping and maximum-likelihood thresholds for the surface code under generic single-qubit coherent errors. *PRX Quantum*, 6(4):040305, 2025.

Theodore H. Berlin and Mark Kac. The spherical model of a ferromagnet. *Physical Review*, 86(6):821–835, 1952. doi: 10.1103/PhysRev.86.821.

Michael P. Brenner and Lucy Colwell. Solving an open problem in theoretical physics using AI-assisted discovery, 2026.

Christopher T. Chubb and Steven T. Flammia. Statistical mechanical models for quantum codes with correlated noise. *Annales de l'Institut Henri Poincaré D*, 8(2):269–321, 2021. doi: 10.4171/AIHPD/105.

Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002. doi: 10.1063/1.1499754.

Prateek Gupta, Andrea E. V. Ferrari, and Nabil Iqbal. A machine learning approach to duality in statistical physics. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 21322–21341. PMLR, 13–19 Jul 2025. URL `https://proceedings.mlr.press/v267/gupta25b.html`.

Per C. Hemmer, Mark Kac, and George E. Uhlenbeck. On the van der waals theory of the vapor-liquid equilibrium. iii. discussion of the critical region. *Journal of Mathematical Physics*, 5(1):60–74, 1964. doi: 10.1063/1.1704065.

Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31:253–258, 1925. doi: 10.1007/BF02980577.

Mark Kac, George E. Uhlenbeck, and Per C. Hemmer. On the van der waals theory of the vapor-liquid equilibrium. i. discussion of a one-dimensional model. *Journal of Mathematical Physics*, 4(2): 216–228, 1963. doi: 10.1063/1.1703946.

Hendrik A. Kramers and Gregory H. Wannier. Statistics of the two-dimensional ferromagnet. part i. *Physical Review*, 60(3):252–262, 1941a. doi: 10.1103/PhysRev.60.252.

Hendrik A. Kramers and Gregory H. Wannier. Statistics of the two-dimensional ferromagnet. part i. *Physical Review*, 60(3):252–262, 1941b. doi: 10.1103/PhysRev.60.252.

Hendrik A. Kramers and Gregory H. Wannier. Statistics of the two-dimensional ferromagnet. part ii. *Physical Review*, 60(3):263–276, 1941c. doi: 10.1103/PhysRev.60.263.

Andrew Lenard. Exact statistical mechanics of a one-dimensional system with coulomb forces. *Journal of Mathematical Physics*, 2(5):682–693, 1961. doi: 10.1063/1.1703742.

W. Lenz. Beitrag zum verständnis der magnetischen erscheinungen in festen körpern. *Physikalische Zeitschrift*, 21:613–615, 1920.

Elliott H. Lieb. Exact solution of the f model of an antiferroelectric. *Physical Review Letters*, 18(24): 1046–1048, 1967a. doi: 10.1103/PhysRevLett.18.1046.

Elliott H. Lieb. Residual entropy of square ice. *Physical Review*, 162(1):162–172, 1967b. doi: 10.1103/PhysRev.162.162.

Elliott H. Lieb. Exact solution of the two-dimensional slater kdp model of a ferroelectric. *Physical Review Letters*, 19(3):108–110, 1967c. doi: 10.1103/PhysRevLett.19.108.

Elliott W. Montroll. Statistical mechanics of nearest neighbor systems. *Il Nuovo Cimento*, 6:264–281, 1949.

Alexander Novikov et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025.

Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physical Review*, 65(3–4):117–149, 1944. doi: 10.1103/PhysRev.65.117.

Renfrey B. Potts. Some generalized order-disorder transformations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48(1):106–109, 1952. doi: 10.1017/S0305004100027419.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, Jan 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06924-6. URL `https://doi.org/10.1038/s41586-023-06924-6`.

Brian Skinner, Jonathan Ruhman, and Adam Nahum. Measurement-induced phase transitions in the dynamics of entanglement. *Physical Review X*, 9(3):031009, 2019. doi: 10.1103/PhysRevX.9.031009.

Zhangde Song, Jieyu Lu, Yuanqi Du, Botao Yu, Thomas M. Pruyn, Yue Huang, Kehan Guo, Xiuzhe Luo, Yuanhao Qu, Yi Qu, Yinkai Wang, Haorui Wang, Jeff Guo, Jingru Gan, Parshin Shojaee, Di Luo, Andres M Bran, Gen Li, Qiyuan Zhao, Shao-Xiong Lennon Luo, Yuxuan Zhang, Xiang Zou, Wanru Zhao, Yifan F. Zhang, Wucheng Zhang, Shunan Zheng, Saiyang Zhang, Sartaaj Takrim Khan, Mahyar Rajabi-Kochi, Samantha Paradi-Maropakis, Tony Baltoiu, Fengyu Xie, Tianyang Chen, Kexin Huang, Weiliang Luo, Meijing Fang, Xin Yang, Lixue Cheng, Jiajun He, Soha Hassoun, Xiangliang Zhang, Wei Wang, Chandan K. Reddy, Chao Zhang, Zhiling Zheng, Mengdi Wang, Le Cong, Carla P. Gomes, Chang-Yu Hsieh, Aditya Nandy, Philippe Schwaller, Heather J. Kulik, Haojun Jia, Huan Sun, Seyed Mohamad Moosavi, and Chenru Duan. Evaluating large language models in scientific discovery, 2025. URL `https://arxiv.org/abs/2512.15567`.

H. Eugene Stanley. Spherical model as the limit of infinite spin dimensionality. *Physical Review*, 176 (2):718–722, 1968. doi: 10.1103/PhysRev.176.718.

George E. Uhlenbeck, Per C. Hemmer, and Mark Kac. On the van der waals theory of the vapor-liquid equilibrium. ii. discussion of the distribution functions. *Journal of Mathematical Physics*, 4(2): 229–247, 1963. doi: 10.1063/1.1703947.

Appendix

The appendices document everything needed to inspect the agent's behavior and replicate the probe evaluation. App. A reproduces the proposer prompts verbatim. App. B gives one card per task in StatMechBench-v0, with both the canonical and paraphrased descriptions. App. C documents the sandboxed verifier (AST allowlist, subprocess construction, feedback string template).

A Proposer prompts (verbatim)

The proposer is steered by a single system prompt plus one of two user templates (initial vs. feedback round). Templates and the closed tractable-class library live in `experiments/agent/prompts.py`. We reproduce them here unchanged.

System prompt. The string `SYSTEM_PROMPT`, concatenated with the library text below.

```
You are a research assistant participating in a benchmark of partition-function
mapping discovery.

Your job: given a description of a statistical-mechanics model H(s), produce
a JSON object that classifies it into a known tractable class and provides
an EXECUTABLE Python function `Z_efficient(params, n)` that computes the
partition function

    Z = sum_{s} exp(-beta * H(s))

ideally in time polynomial in n (or, where the problem is intractable in
the worst case, an honest brute-force fallback).

{LIBRARY}

Output FORMAT (a single JSON object; a Markdown ```json fence is fine but not required):

{
  "tractable_class": "<one of the library labels above>",
  "complexity_O": "<e.g. 'O(n)' or 'O(n^3)' or 'O(2^n)'>",
  "confidence": <a float in [0, 1]>,
  "justification": "<2-6 sentences: which transformation makes Z tractable and why>",
  "code": "<pure Python source defining Z_efficient(params, n) -> float; only standard libs +
        numpy>"
}

Rules:
  - The code MUST define a top-level function `Z_efficient(params, n)` returning a Python
      float (or any value losslessly convertible to one).
  - You may import `math`, `numpy`, `scipy`, `itertools`, `functools`. Do NOT import `os`, `
      sys`, `subprocess`, `socket`, or any networking/IO module, and do not read or write
      files.
  - If you genuinely believe no polynomial algorithm exists, set `tractable_class` to `"
      no_efficient_known"` and provide an honest brute-force implementation; do NOT pretend.
  - Your code will be executed in a subprocess with a 30 s timeout; budget your enumeration
      accordingly.
  - Your code will be checked numerically against brute-force enumeration with rtol=1e-6.
```

Tractable-class library (`LIBRARY`). Substituted into the system prompt. The labels here are the only legal values for `tractable_class`.

```
Library of tractable partition-function classes (you must classify into one):

  * "transfer_matrix" -- 1D / quasi-1D systems; row-by-row update; O(n) or O(n * 2^L).
  * "free_fermion" -- quadratic fermionic Hamiltonian after Jordan-Wigner; O(n^3) via
      diagonalization.
  * "pfaffian" -- planar Ising / dimer model; Kasteleyn orientation; O(n^3).
  * "gauge" -- disorder removable by a sign-flip gauge (e.g. Mattis); reduces to known model.
  * "percolation" -- bond/site percolation universality; analytic critical points.
```

```
 * "rbim_nishimori" -- random-bond Ising on the Nishimori line (e.g. toric code threshold).
 * "treewidth_bounded" -- junction-tree algorithm; O(2^k * n) for treewidth k.
 * "exactly_solvable_other" -- ANY OTHER known polynomial-time class; you must name it
     explicitly.
 * "no_efficient_known" -- no known polynomial-time algorithm; honest brute force is
     acceptable.
```

The library deliberately mixes categories; for the proposer, the unifying contract is operational ("emit code that returns Z in poly(n) and matches the oracle"). To show *why* the labels are not interchangeable, Table 4 further explains the underlying types.

Table 4: Details of tractable classes selected in this work.

Label	What kind of thing it actually is
`transfer_matrix`	Algorithm — works on any 1D / quasi-1D Hamiltonian
`pfaffian`	Algorithm (Kasteleyn) — applicable when the graph is planar
`treewidth_bounded`	Graph property — enables the junction-tree algorithm
`free_fermion`	Representation — quadratic-fermion form reachable by Jordan–Wigner; algorithm is diagonalisation
`gauge`	Algebraic transformation — sign-flip gauge that reduces to a known solvable model
`rbim_nishimori`	Model + special line — RBIM is generally hard; only on the Nishimori line is Z analytically pinned
`percolation`	Universality / mapping — Fortuin–Kasteleyn-style; gives analytic critical points but not a poly-time Z in general
`exactly_solvable_other`	Escape hatch for any other known poly-time class (Bethe ansatz, Lieb's ice, …)
`no_efficient_known`	Negative label — honest brute force only

Initial user prompt. Round 0 of every trial.

```
Here is the problem.

{task_description}

Please respond with ONLY the JSON object described in the system instructions.
```

Feedback user prompt. Rounds $k \geq 1$ of every trial. The `{prev_json}` slot is filled with the proposer's raw round-$(k-1)$ response; `{feedback}` is the verifier string from App. C.

```
You previously responded to this problem:

--- problem ---
{task_description}
--- end problem ---

Your previous response was:
{prev_json}

The verifier reports:
{feedback}

Please revise. Reply with a NEW JSON object only (same schema). If you can
identify the bug in your previous code, fix it. If your tractable class was
wrong, choose a different one.
```

B Task cards (StatMechBench-v0)

One card per task. Each card shows the gold class and complexity, the probed sizes n, the canonical description, the paraphrased description used in the contamination-control variant, and the optional hint (unused in the evaluation).

How the paraphrases were produced. All six paraphrases are *hand-authored* by the authors — there is no LLM rewriter, no automated transformation pass, and no offline generator. Each paraphrase is a Python string literal sitting next to the canonical description; the harness reads whichever string is selected by the `-paraphrased` flag and sends it as-is. Authors followed a fixed rewrite rule:

1. **Strip canonical model names.** "Ising", "spin glass", "Pfaffian", "Mattis", "Hamiltonian", "ferromagnet", "torus" are removed; geometric structure is described in plain words ("$L \times L$ grid of binary variables that wraps in both directions" in place of "$L \times L$ Ising torus").
2. **Rename the spin variable.** The canonical statement uses $s_i \in \{-1, +1\}$; the paraphrase uses x_i to remove the near-deterministic "spin" association.
3. **Preserve the Z expression literally.** The number of configurations (2^n), the index ranges, the coupling structure, and the exponent inside $\exp(\cdot)$ are reproduced unchanged. P05 is the strongest illustration: the canonical statement names the Mattis model and gives the rank-1 form $J_{ij} = \xi_i \xi_j$; the paraphrase substitutes the already-contracted scalar $M(x) = \sum_i \xi_i x_i$ and asks for $\sum_x \exp\big((\beta/2)(M(x)^2 - n)\big)$ without naming the model.
4. **Do not add or remove information.** The paraphrase contains no extra hint and no extra constraint relative to the canonical description; the optional `hint_md` field is the only place hints live.

The fixed-string approach is a deliberate choice: it makes the paraphrase set part of the benchmark artefact (re-runnable verbatim by any reader, against any future model) rather than a moving target generated on-the-fly. The cost is that paraphrase quality is bounded by author care; we view this as acceptable for v0 and discuss *programmatic* rewriting (templated variable renaming, structural isomorphism over a graph DSL) as a v1 extension in §4.

P01 — 1D Ising chain, OBC. L1. Gold class `transfer_matrix`, gold complexity $O(n)$. Probed sizes $n \in \{6, 10, 14\}$.

```
[BLIND]
You are given the **1D Ising chain with open boundary conditions**.
Hamiltonian: H(s) = -J * sum_{i=0}^{n-2} s_i * s_{i+1}, with s_i in {-1,+1}.
Goal: write a Python function
    def Z_efficient(params: dict, n: int) -> float:
        # params has keys "beta" (float) and "J" (float)
that computes the partition function
    Z(beta, J, n) = sum_{s in {-1,+1}^n} exp(-beta * H(s))
in time polynomial in n (NOT exponential).

[PARAPHRASED]
You are given a sequence of n binary variables x_1, ..., x_n with each
x_i in {-1, +1}, plus two real parameters beta and J. Compute
    Z(beta, J, n) = sum over all 2^n binary tuples (x_1,...,x_n) of
                      exp( beta * J * sum_{i=1}^{n-1} x_i * x_{i+1} ).
Naive enumeration of all 2^n tuples is too slow; your implementation should
run in time polynomial in n.
```

P02 — 1D Ising chain with random site fields, OBC. L1. Gold `transfer_matrix`, $O(n)$. Probed $n \in \{6, 10, 12\}$.

```
[BLIND]
You are given a **1D Ising chain with random site fields and open boundary conditions**.
Hamiltonian: H(s) = -J * sum_{i=0}^{n-2} s_i s_{i+1} - sum_{i=0}^{n-1} h_i s_i.
Implement
    def Z_efficient(params: dict, n: int) -> float:
        # params has "beta" (float), "J" (float), "h" (list of length n)

[PARAPHRASED]
You are given a sequence of n binary variables x_1, ..., x_n in {-1, +1},
a coupling constant J, and an array of n per-position weights h_1, ..., h_n.
Compute
    Z(beta, J, h, n) = sum over all 2^n tuples x of
        exp( beta * J * sum_{i=1}^{n-1} x_i * x_{i+1}
            + beta * sum_{i=1}^{n} h_i * x_i ).
```

P03 — 2D ferromagnetic Ising on $L \times L$ torus. L2. Gold `transfer_matrix`, gold complexity $\text{poly}(L) \cdot 2^L$. Probed $n \in \{9, 16\}$ (i.e. $L \in \{3, 4\}$).

```
[BLIND]
You are given the **2D ferromagnetic Ising model on an L x L torus**
(periodic boundary conditions in both directions).
H(s) = -J * sum_{<i,j>} s_i s_j over all nearest-neighbour pairs.
Z must be computed in time polynomial in L (so exponential in sqrt(n) is
acceptable, but exponential in n is not).

[PARAPHRASED]
L x L grid of binary variables x_{r,c} in {-1,+1}; the grid wraps in both
directions. E is the set of all unique neighbour pairs (2*L^2 entries).
Compute Z(beta, J, n) = sum over 2^n assignments of
      exp( beta * J * sum_{(i,j) in E} x_i * x_j ).
[HINT, unused] Consider a row-to-row transfer matrix of size 2^L by 2^L.
```

P04 — 2D Ising spin glass on planar grid (open). L2. Gold `pfaffian`, $O(n^3)$. Probed $n \in \{9, 16\}$.

```
[BLIND]
You are given a **2D Ising spin glass on an L x L OPEN-boundary planar
lattice** with random binary couplings on each edge.
H(s) = - sum_{horiz edges} Jh[r,c] * s[r,c] * s[r,c+1]
       - sum_{vert edges} Jv[r,c] * s[r,c] * s[r+1,c].
Z must be polynomial in n (NOT exponential).

[PARAPHRASED]
L x L grid of binary variables x_{r,c}; grid does NOT wrap. Each horizontal
edge carries a binary weight Jh[r][c] in {-1,+1}; each vertical edge
Jv[r][c] in {-1,+1}. Compute
   Z(beta, Jh, Jv, n) = sum over all 2^n assignments x of
      exp( beta * sum_{r,c: c<L-1} Jh[r][c] * x_{r,c} * x_{r,c+1}
         + beta * sum_{r,c: r<L-1} Jv[r][c] * x_{r,c} * x_{r+1,c} )
[HINT, unused] The interaction graph is planar.
```

P05 — Mattis spin glass (rank-1 disorder). L1. Gold `gauge`, $O(n)$. Probed $n \in \{6, 10, 12\}$.

```
[BLIND]
**Mattis-like spin glass on n spins** with all-to-all coupling.
J_{ij} = xi_i * xi_j for i != j, with xi a fixed +/- 1 pattern.
H(s) = -(1/2) * sum_{i!=j} J_{ij} s_i s_j = -(1/2) * [ (sum_i xi_i s_i)^2 - n ].
Z(beta, xi, n) in time polynomial in n.

[PARAPHRASED]
n binary variables x_1,...,x_n in {-1,+1} plus a fixed pattern vector xi.
For each x define M(x) = sum_i xi_i * x_i. Compute
   Z(beta, xi, n) = sum over 2^n assignments of exp( (beta/2) * (M(x)^2 - n) ).
[HINT, unused] The coupling matrix J_{ij} = xi_i * xi_j is rank-1.
```

P06 — Random 3-regular ± 1 Ising glass. L2 (negative control). Gold `no_efficient_known`, $O(2^n)$ (best known on general instances). Probed $n \in \{8, 10\}$.

```
[BLIND]
**Ising spin glass on a random 3-regular graph** with binary +/- 1 couplings.
Edges given as edge list; H(s) = -sum_k J[k] * s[edges[k][0]] * s[edges[k][1]].
"If you determine, after analysing the structure of the interaction graph,
that no polynomial-time algorithm in any class from the library applies,
state so explicitly in your justification and provide your best
implementation; the verifier will only call you at small n."

[PARAPHRASED]
Graph on n vertices specified by edge list, every vertex has degree 3.
Compute Z(beta, edges, J, n) = sum over 2^n assignments of
  exp( beta * sum_k J[k] * x[edges[k][0]] * x[edges[k][1]] ).
```

C Verifier internals

The verifier is an integrity check on a cooperative proposer, not an adversarial sandbox. Its job is to (i) reject obvious filesystem / network access at parse time, (ii) execute the candidate `Z_efficient` on a list of (n, params) probes in isolation, and (iii) compare each returned value against Z_{brute} at $\varepsilon_{\text{rel}} = 10^{-6}$.

AST import allowlist. `static_import_check` walks the candidate's AST and rejects any `import` or `from` statement whose top-level module is not in:

```
ALLOWED_IMPORTS = {
  "math", "numpy", "scipy", "itertools", "functools",
  "operator", "fractions", "collections", "cmath", "decimal",
  "dataclasses", "typing",
}
```

Sub-modules thereof (e.g. `scipy.linalg`) are allowed. `os`, `sys`, `subprocess`, `socket`, `pathlib`, `requests`, ...are all rejected before the subprocess starts. Adversarial bypasses (`__import__('os')`, monkey-patched `getattr` chains) are not blocked — the prompt asks the model to be honest, and the allowlist exists to surface accidental IO, not to defend against attacks.

Subprocess template. Every probe set is run in a freshly-launched Python subprocess via `subprocess.run` with a 30 s wall-clock timeout (`DEFAULT_TIMEOUT_SEC`), `stdin` carrying a JSON list of calls and `stdout` carrying a single sentinel-prefixed line:

```
import json, sys, math, itertools, functools
import numpy as np

def _to_float(x):
    try: return float(x)
    except (TypeError, ValueError):
        arr = np.asarray(x)
        if arr.size == 1: return float(arr.reshape(()).item())
        raise TypeError(f"Z_efficient returned non-scalar of shape {arr.shape}")

# ---------- candidate code starts ----------
__CANDIDATE_CODE__
# ---------- candidate code ends ----------

if 'Z_efficient' not in dir():
    print("__VERIFIER_RESULT__" + json.dumps({"error": "missing Z_efficient"}))
    sys.exit(0)

req = json.loads(sys.stdin.read())
out = []
for call in req["calls"]:
    try:
        z = Z_efficient(call["params"], call["n"])
        out.append({"ok": True, "z": _to_float(z)})
    except Exception as e:
        out.append({"ok": False, "error": f"{type(e).__name__}: {e}"})
print("__VERIFIER_RESULT__" + json.dumps({"results": out}))
```

The sentinel prefix `__VERIFIER_RESULT__` isolates the result line from any debug `print` the candidate may emit; the parser `_parse_subprocess_stdout` reads lines in reverse and returns the first match.

Environment-variable allowlist. The subprocess inherits only:

```
PATH, HOME, LANG, LC_ALL, LC_CTYPE,
TMPDIR, TEMP, TMP, PYTHONPATH,
LD_LIBRARY_PATH, DYLD_LIBRARY_PATH.
```

Every other environment variable is dropped, so API keys and other credentials in the parent shell are not exposed to candidate code.

Probe construction. For each n in the task's `n_range`, `_build_calls` draws `num_param_samples` independent parameter samples (default 5) using `np.random.default_rng(seed)` so the probe set is deterministic given `(seed, task)`. Z_{brute} is computed in the parent process; only the candidate's `Z_efficient` runs in the subprocess.

Pass criterion. A trial *passes* only if every probe returned a finite float and satisfied $\left|Z_{\text{eff}} - Z_{\text{brute}}\right| / \max\left(|Z_{\text{brute}}|, 10^{-300}\right) < 10^{-6}$ (`REL_ERR_TOL = 1e-6`, with `_REL_ERR_DENOM_FLOOR` guarding $Z = 0$).

Feedback string template. The string returned by `feedback_for_proposer` (with `leak_truth=False`, the pilot setting) is what gets pasted into the next round's user prompt. Three branches:

```
# (a) Code didn't run at all:
"Your code did not run. Error: {report.error}"

# (b) Code ran but every probe agreed:
"Verifier ran on {N} (n, params) samples.
All samples passed (max relative error {max_rel_err:.2e})."

# (c) Code ran and at least one probe disagreed:
"Verifier ran on {N} (n, params) samples.
{k}/{N} samples failed.
  - relative error {rel_err:.2e} >> 1e-06 -- your computed Z disagrees with brute force.
  - relative error {rel_err:.2e} >> 1e-06 -- your computed Z disagrees with brute force.
  - relative error {rel_err:.2e} >> 1e-06 -- your computed Z disagrees with brute force."
```