P^{4}Bench: Contamination-Proof, Publicly Verifiable, and Privacy-Preserving LLM Evaluation via Zero-Knowledge Proofs

Enhan Zhao [1 2] **Yuanrui Zhang** [1] **Zhang Zhang** [1] **Wei Wu** [2] **Di He** [1]

Abstract

Mathematical benchmarks are central to evaluating the reasoning abilities of large language models (LLMs), but data contamination undermines their reliability when benchmark questions and answers appear in training data. The most common way to handle contamination is to keep benchmarks private, hiding both test problems and answers from model providers and the public. Although this reduces direct test-set leakage, it sacrifices transparency: the community must trust a third-party benchmark organizer to validate the problems, run the evaluation, verify the answers, and report scores, rather than being able to verify the results independently. To address this trade-off, we propose **P^{4}Bench**, a contamination-**p**roof, **p**ublicly verifiable, and **p**rivacy-**p**reserving evaluation framework based on zero-knowledge proofs (ZKPs). The key idea is to publish benchmark problems while keeping solutions private: model providers submit cryptographic proofs that their answers are correct, and anyone can verify these proofs without learning the answers. This simultaneously enables public benchmark release, answer privacy, and independent verification. We instantiate P^{4}Bench on Boolean satisfiability (SAT) and discrete logarithm problem (DLP) tasks, encoding answer verification as zk-SNARK circuits. Experiments with frontier LLMs show that the framework can effectively distinguish model capabilities while keeping proof generation and verification practical.

1. Introduction

Mathematical reasoning benchmarks have become an important driving force behind the development of large language models (LLMs). Widely used benchmarks such as GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), Omni-MATH (Gao et al., 2025), OlympiadBench (He et al., 2024), HARP (Yue et al., 2024), and OlymMATH (Sun et al., 2025) release test queries together with ground-truth answers, making evaluation open, accessible, and enabling broad community analysis. However, this openness also introduces a serious risk of data leakage. Test queries and answers may be accidentally crawled into the pretraining corpus or deliberately incorporated into supervised fine-tuning data, thereby compromising the validity of benchmark evaluation(Wang et al., 2025; Wu et al., 2026). This problem is known as data contamination (Li et al., 2024; Golchin & Surdeanu, 2024; Liang et al., 2026).

The most common way to prevent direct test-set leakage is to keep the test data private. Benchmarks such as FrontierMath (Glazer et al., 2024), TRUCE (Rajore et al., 2024), and FormalProofBench (Ravi et al., 2026) require model providers to expose their models through APIs, while benchmark organizers run evaluations on hidden test instances that are never released to the public. However, private benchmarks come at the cost of transparency. Users must trust that the benchmark organizers carefully verify each problem and its solution. In mathematical reasoning tasks, such verification is particularly demanding, since checking the correctness and rigor of proofs often requires substantial human expertise. As benchmark scales continue to grow, this manual verification process becomes increasingly costly and, in practice, difficult to sustain.

These limitations motivate a benchmark design that prevents test data from leaking into future training data while keeping the evaluation process transparent–publicly verifiable by the community. To this end, we propose **P^{4}Bench**, a benchmark framework based on zero-knowledge proofs (ZKPs). A ZKP is a cryptographic protocol in which a prover convinces a verifier that it knows hidden information satisfying a publicly specified relation, without revealing the hidden information itself.

In our scenario, the benchmark provider publishes problem instances but not answers. After solving a problem locally, the model provider does not need to send the answer to a verifier or make it public. Instead, each problem is

[1]Peking University [2]Ant International. Correspondence to: Wei Wu <wuwei19850318@gmail.com>, Di He <di_he@pku.edu.cn>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

paired with a public checker that specifies when an answer should be accepted, and the provider submits only a proof that it has an answer satisfying that checker. This submitted proof can then be checked by anyone using the public answer checker, increasing transparency for the community without relying on a trusted benchmark organizer. For example, for the graph 3-coloring problem, the Goldreich–Micali–Wigderson (GMW) protocol (Goldreich et al., 1986) allows a prover to convince a verifier that it knows a valid 3-coloring of a public graph without revealing the coloring.

This design addresses the limitations above in three ways. First, it reduces the contamination channel that affects standard benchmarks. The benchmark provider releases only problem instances and the answer-checking relation, while model providers demonstrate performance with proofs rather than answers; therefore, neither side has an incentive to publish reusable answers. Since data contamination arises when question–answer pairs enter future training data, keeping answers unpublished directly targets this leakage mechanism without making the benchmark private. Second, it protects valuable answers. By zero knowledge, verification reveals no more information than the fact that the hidden answer is accepted. This protection can encourage model providers to report performance even during model development, accelerating community feedback. Third, it restores the transparency often lost in private evaluation. The community can inspect the released problems and relations, verify submitted proofs, and recompute scores using public data, reducing reliance on a single benchmark organizer to adjudicate correctness behind closed doors.

We instantiate P^4Bench with Boolean satisfiability (SAT) and discrete logarithm problems (DLP). For each family, we construct 500 controlled-difficulty instances and implement the full pipeline from problem generation to LLM evaluation, proof generation, and public verification. We evaluate seven frontier models on the benchmark, and the results show that P^4Bench effectively distinguishes models by their reasoning and search capabilities: performance varies substantially across models and difficulty tiers, with Qwen 3.5 Plus performing best overall. We also find that proof generation is practical at submission time and that verification remains efficient, supporting the feasibility of ZKP-based, publicly auditable mathematical reasoning evaluation.

2. Related Work

2.1. Mathematical Benchmarks for LLMs

Mathematical reasoning benchmarks have progressed from elementary arithmetic datasets such as GSM8K (Cobbe et al., 2021) to competition-, Olympiad-, and research-level evaluations including MATH (Hendrycks et al., 2021), Omni-MATH (Gao et al., 2025), OlympiadBench (He et al., 2024), HARP (Yue et al., 2024), OlymMATH (Sun et al., 2025), and FrontierMath (Glazer et al., 2024). This progression raises both difficulty and trust concerns: public benchmarks are reproducible but vulnerable to contamination, while private or newly sourced evaluations can reduce leakage at the cost of transparency or sustained curation. For example, MathArena (Balunović et al., 2025) reports that models perform substantially better on older competition years than on newly released problems, highlighting the practical impact of contamination.

2.2. Contamination Mitigation and Detection Strategies

Several strategies have been proposed to address benchmark contamination:

Private benchmarks. Organizations maintain secret test sets and evaluate submitted models centrally. TRUCE formalizes this setting as *private benchmarking* and proposes secure evaluation protocols that keep the benchmark hidden from model developers (Rajore et al., 2024). Similar motivations also appear in recent contamination-aware evaluation work (Liang et al., 2026). While effective against direct leakage, private benchmarks require trust in the evaluating authority or execution environment, reduce transparency and reproducibility.

Live benchmarks. MathArena (Balunović et al., 2025), LiveCodeBench (Jain et al., 2025), and LiveBench (White et al., 2025) reduce contamination by evaluating models on newly released problems from competitions, code platforms or papers. This mitigates contamination for problems released after a model's training cutoff, but it requires sustained curation and can make cross-temporal comparisons difficult.

Perturbed benchmarks. Rather than hiding benchmarks entirely, some approaches perturb problems or answers to limit contamination. CapBencher (Ishida et al., 2025) publishes benchmarks with intentionally randomized answers, while perturbation-based methods (Mirzadeh et al., 2025; Zou et al., 2025) generate new instances by varying numerical values or visual elements of existing problems. These strategies reduce direct overlap with training data but face a shared limitation: the underlying reasoning patterns remain tied to the original problems, leaving residual contamination risks and limiting the ability to track fine-grained capability differences.

Table 1 summarizes these benchmark-design trade-offs. Public benchmarks maximize transparency but do not prevent contamination once question–answer pairs enter training data. Private benchmarks reduce leakage by hiding the test set, but they sacrifice transparency, require trust in a third-party evaluator, and expose submitted answers or

model outputs to the benchmark provider. Live benchmarks improve freshness, but they require continual curation and make comparisons across time less fair. Perturbed benchmarks are easy to release, but they do not provide answer privacy and, because they rewrite existing problems rather than fully decoupling from them, can still inherit contamination from the original tasks.

Contamination detection. A complementary line of work diagnoses contamination after a benchmark has been released. Depending on access, these methods include corpus-overlap audits using exact, fuzzy, or near-duplicate matching (Li et al., 2024); black-box prompt-completion tests that look for exact or near-exact continuations (Golchin & Surdeanu, 2024); gray-box membership signals based on loss, perplexity, zlib-normalized likelihood, reference-model scores, or Min-K% and Min-K%++ statistics (Shi et al., 2024; Zhang et al., 2025); calibration methods for reducing false positives (Zhang et al., 2024); and white-box gradient detectors such as GDS (Zhang et al., 2026). These techniques are useful diagnostics, but they remain access- and assumption-dependent: survey evidence shows sharp failures under distribution shift (Fu et al., 2025), while reinforcement learning can mask Min-K-style signals despite inflated benchmark performance (Wang et al., 2025), and spurious rewards may improve results only on contaminated benchmarks (Wu et al., 2026). Thus, detection can flag suspected leakage, but it cannot prevent or rule out answer-key exposure.

2.3. Zero-Knowledge Proofs for Large Language Models

Zero-knowledge proofs have recently been applied to LLMs, primarily for verifying model inference. zkLLM (Sun et al., 2024) proposed using ZKPs to prove that a specific LLM produced a given output, addressing model integrity rather than benchmark integrity. NANOZK (Wang, 2026) introduced a layerwise proof framework that decomposes transformer inference into per-layer proofs with commitment chains, achieving constant proof size per layer and enabling parallel proving, making ZKP-based verification practical for billion-parameter models. Both works focus on *inference verification*, confirming which model was used and that it ran correctly. Our work takes a complementary direction: we use ZKPs to verify that an LLM's *answer is correct* without revealing the answer itself, directly addressing the benchmark contamination problem.

3. Method

We describe P^{4}Bench from the benchmark interface down to its proof-system implementation. §3.1 formalizes each benchmark instance as a public relation R_i, namely the accept/reject rule for a hidden answer. §3.2 gives the submission and scoring protocol, and §3.3 states the zero-knowledge guarantees that make the resulting scores publicly verifiable while preserving answer privacy. §3.4 and §3.5 then show how these relations are encoded as R1CS constraints and realized with Groth16 proofs.

3.1. Benchmarking as a Public Relation

Let $\mathcal{P}$ denote the prover (e.g., the model developer) and $\mathcal{V}$ denote the verifier (e.g., the benchmark maintainer or any third party). For each benchmark instance, P_i is the public problem statement and a_i is the answer produced by the model. The private zero-knowledge witness is denoted by w_i: it contains a_i and, when needed, auxiliary values required to evaluate correctness. The public problem P_i induces an instance-specific public relation R_i, equivalently $R_i(w_i) \equiv R(P_i, w_i)$. Acceptance is written as

$$R_i(w_i) = 1, \tag{1}$$

Here, R_i encodes the accept/reject decision for a candidate hidden witness. Thus, proving $R_i(w_i) = 1$ proves that the hidden answer satisfies the public relation, without requiring the prover to reveal that answer.

3.2. P^{4}Bench Protocol

The evaluation workflow is intentionally close to ordinary benchmark submission, except that answers are replaced by proofs. For each public instance P_i, setup compiles the corresponding relation R_i and produces two keys: a proving key pk_i used to generate proofs, and a verification key vk_i used to check them. The relation R_i fixes the statement to be proved and determines the circuit-specific key material, but it does not by itself determine a proof: proof generation also requires a satisfying private witness. The benchmark provider publishes the problem instances $\{P_i\}_{i=1}^n$, the relation descriptions $\{R_i\}_{i=1}^n$, and verification keys $\{\mathsf{vk}_i\}_{i=1}^n$; the proving keys $\{\mathsf{pk}_i\}_{i=1}^n$ are made available to model providers for proof generation.

A model provider runs its model M on a public problem P_i, obtains an answer a_i, and forms a private witness w_i. Instead of submitting a_i, the provider computes $\pi_i \leftarrow \mathsf{Prove}(\mathsf{pk}_i; w_i)$ and publishes only (i, π_i). Any verifier then checks $\mathsf{Verify}(\mathsf{vk}_i, \pi_i) = 1$ using only public data.

Figure 1 contrasts this data flow with public and private benchmarking. The substantive change is what leaves the model provider's environment: conventional public evaluation releases or eventually reveals reusable answer keys, and private evaluation hides the benchmark but still requires a central judge to observe submitted answers. In contrast, P^{4}Bench releases proof objects instead of answer keys or model reasoning traces.

Table 1. Comparison of benchmark approaches across key desiderata. △ denotes partial satisfaction: private benchmarks hide answers from the public and model provider, but submitted outputs are visible to the central evaluator; perturbed benchmarks reduce direct overlap but may preserve reasoning patterns from the original tasks. P^{4}Bench is the only approach that simultaneously achieves all five properties.

Approach	Publicly Available	Contamination Resistant	Publicly Verifiable	Low Maintenance	Answer Privacy
Standard benchmark	✓	×	✓	✓	×
Private benchmark	×	✓	×	×	△
Live benchmark	✓	✓	✓	×	×
Perturbed benchmark	✓	△	✓	✓	×
P^{4}Bench (ours)	✓	✓	✓	✓	✓

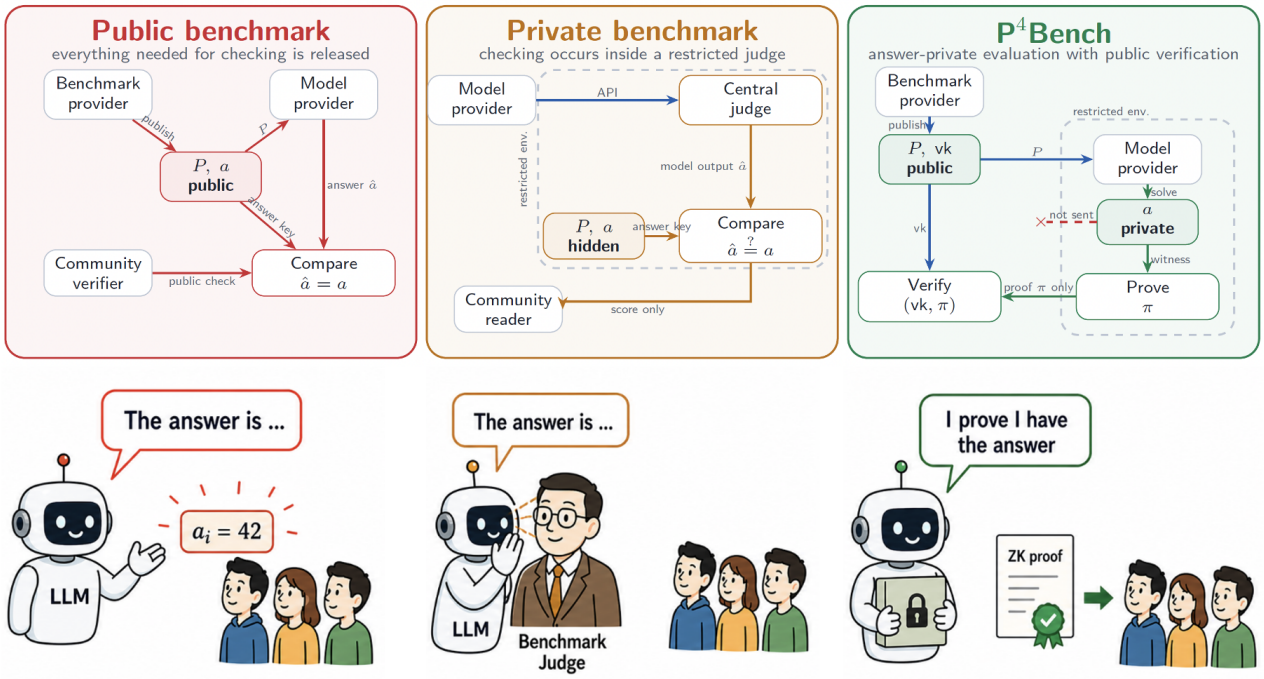

Figure 1. Comparison of benchmark data flows and their intuitive interpretation. Public benchmarks expose answers together with the problem instances. Private benchmarks hide the benchmark from the public, but the central judge still observes submitted answers. P^{4}Bench keeps each answer inside the prover's environment and releases only a zero-knowledge proof, which anyone can verify using the public verification key.

The benchmark score is the fraction of instances for which valid proofs are provided:

$$\mathrm{Score}(\mathcal{P}) = \frac{|\{i : \mathsf{Verify}(\mathsf{vk}_i, \pi_i) = 1\}|}{n}, \tag{2}$$

This data flow gives the contamination argument. The benchmark provider does not publish the answer key: only the problem statements and verification material are public. Model providers also do not need to reveal their answers to prove performance, because a zero-knowledge proof already certifies correctness. As a result, neither side has an incentive to release reusable answers; without exposed question–answer pairs, future training data should not contain the benchmark in a directly contaminating form.

Algorithm 1 summarizes the end-to-end workflow, from setup and local solving to proof publication and public score computation.

3.3. Protocol Guarantees

For each public relation R_i, the proof statement is that the prover possesses a private witness w_i such that $R_i(w_i) = 1$. In the benchmark setting, this means that the prover knows an answer accepted by the public relation, while the answer itself remains hidden. P^{4}Bench relies on three standard properties of zero-knowledge proofs, each of which corresponds to a benchmark requirement.

Completeness: fair scoring. If a model provider finds a valid answer and forms a witness w_i satisfying $R_i(w_i) = 1$, then the proof generated with pk_i is accepted under vk_i.

Algorithm 1 P^4Bench Submission and Verification Workflow

Require: Public instances $\{P_i\}_{i=1}^n$, relations $\{R_i\}_{i=1}^n$, model M, prover $\mathcal{P}$.
Ensure: Public score $\mathrm{Score}(\mathcal{P})$ and accepted proof set S.

$S \leftarrow \emptyset$.
for $i = 1, \ldots, n$ **do**
  Compile R_i into R1CS matrices $(\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i)$.
  Interpolate into $\mathsf{QAP}_i = (\{u_{i,j}\}, \{v_{i,j}\}, \{q_{i,j}\}, t_i)$.
  Sample hidden $\tau_i \xleftarrow{\$} \mathbb{F}$.
  $(\mathsf{pk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Setup}(\mathsf{QAP}_i; \tau_i)$; erase τ_i.
  Publish $(P_i, R_i, \mathsf{vk}_i)$ and provide pk_i to $\mathcal{P}$.
  $a_i \leftarrow M(P_i)$; $\mathcal{P}$ forms w_i and full assignment $\vec{z}_i$.
  if $R_i(w_i) = 1$ **then**
    Compute U_i, V_i, W_i and quotient h_i from $\vec{z}_i$.
    Ensure $U_i(x)V_i(x) - W_i(x) = h_i(x)t_i(x)$.
    $\pi_i = (A_i^\pi, B_i^\pi, C_i^\pi) \leftarrow \mathsf{Prove}(\mathsf{pk}_i; \vec{z}_i, h_i)$; see Eq. 8.
    $\mathcal{P}$ publishes (i, π_i) and keeps (a_i, w_i) private.
  end if
end for
for each submitted proof (i, π_i) **do**
  if $\mathsf{Verify}(\mathsf{vk}_i, \pi_i) = 1$ **then**
    $S \leftarrow S \cup \{(i, \pi_i)\}$.
  end if
end for
return $\mathrm{Score} = |S|/n$ and accepted set S.

Thus, hiding answers does not penalize correct submissions: a provider that solves an instance can still receive credit through a verifiable proof.

Knowledge soundness: public verifiability. If any efficient prover produces an accepted proof for instance i, then, except with negligible probability, the prover possesses some witness satisfying R_i. This property makes public verification meaningful. Anyone can recompute the score from the published verification keys and proof objects, without trusting a private judge or seeing the submitted answers.

Zero-knowledge: answer privacy. The proof reveals no information about the witness beyond the fact that it satisfies the public relation. In our setting, this means that verification confirms answer correctness without disclosing the answer, intermediate reasoning, or reusable witness values. As a result, the benchmark can publish problem statements and verification material while avoiding the release of answer keys; since model providers can prove performance without publishing answers, question–answer pairs are less likely to enter future training data.

What P^4Bench does not prove. A valid proof certifies only that the prover possesses a hidden witness satisfying the public relation. It does not, by itself, prove that the answer was independently generated by a particular LLM, that the model was run faithfully, or that the claimed model identity is correct.

3.4. Compiling the Relation into R1CS

The previous subsections assume public relations $\{R_i\}$. We now explain how such a relation is represented algebraically. The goal is to express the accept/reject predicate using only additions and multiplications over a finite field; once these primitive operations are available, the relation can be compiled into a Rank-1 Constraint System (R1CS). The following discussion gives the algebraic intuition behind the checks used by Groth16; the full protocol also includes additional randomization and setup terms.

Checking additions and multiplications. Groth16 works over a finite field $\mathbb{F}$ and represents hidden field elements using elliptic-curve group elements. If G is a generator of an elliptic-curve group, mapping a private scalar $a \in \mathbb{F}$ to the point $A = aG$ acts as a cryptographic commitment: recovering a from A is computationally infeasible under the discrete-logarithm assumption. Addition is compatible with this encoding. Given commitments $A = aG$ and $B = bG$, for any public coefficients $\alpha, \beta \in \mathbb{F}$ we have

$$\alpha A + \beta B = (\alpha a + \beta b)G. \tag{3}$$

Thus a claimed linear relation over hidden values can be checked by comparing the corresponding committed linear combination.

Multiplication is different: group operations alone do not let a verifier derive $(ab)G$ from aG and bG. Groth16 uses a bilinear pairing to check such products. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be additive elliptic-curve groups of order $|\mathbb{F}|$ with generators G_1 and G_2, and let $\mathbb{G}_T$ be a multiplicative target group. A pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfies

$$e(aG_1, bG_2) = e(G_1, G_2)^{ab}. \tag{4}$$

Consequently, if $A = aG_1$, $B = bG_2$, and $C = cG_1$, the verifier can check $e(A, B) = e(C, G_2)$, which holds exactly when $c = ab$ over $\mathbb{F}$. Linear combinations and product checks are therefore sufficient to represent arithmetic circuits over hidden values.

Having a way to check linear combinations and products is enough to check an R1CS. An R1CS is a list of constraints, where each row states that the product of two linear expressions equals a third linear expression:

$$L_A(\vec{z}) \cdot L_B(\vec{z}) = L_C(\vec{z}).$$

Equivalently, if all rows are collected into matrices A, B, C, the full assignment must satisfy

$$(A \cdot \vec{z}) \circ (B \cdot \vec{z}) = C \cdot \vec{z} \tag{5}$$

where $\circ$ denotes element-wise multiplication. Here $\vec{z}$ is the full assignment vector: it contains a constant 1, private witness values from w_i, auxiliary variables introduced by the circuit, and any public inputs required by the chosen relation representation. Appendix E gives a concrete SAT example showing what $\vec{z}$ looks like and how the matrices in Equation 5 are formed.

3.5. From R1CS to Groth16 Proofs

At this point, a verifier could check Equation 5 row by row, but that cost would scale with the number of constraints. Groth16 (Groth, 2016) avoids this row-by-row verification by interpolating the R1CS into a Quadratic Arithmetic Program (QAP): each constraint row becomes an evaluation point, and satisfying all rows becomes equivalent to satisfying one polynomial identity.

From R1CS to QAP. Suppose the R1CS has N constraint rows and assignment vector $\vec{z} = (z_0, \ldots, z_m)$. For each column j of matrix A, construct a polynomial $u_j(x)$ of degree less than N such that $u_j(k) = A_{k,j}$ for every row index $k \in \{1, \ldots, N\}$. Analogously, obtain polynomials $\{v_j(x)\}$ and $\{q_j(x)\}$ from B and C. Given $\vec{z}$, the prover constructs

$$\begin{aligned} U(x) &= \sum_{j=0}^{m} z_j u_j(x), \\ V(x) &= \sum_{j=0}^{m} z_j v_j(x), \\ W(x) &= \sum_{j=0}^{m} z_j q_j(x). \end{aligned} \tag{6}$$

By construction, $U(k)V(k) = W(k)$ at every row index k if and only if all R1CS constraints are satisfied. Let $t(x) = \prod_{k=1}^{N}(x - k)$ be the target polynomial. The R1CS is satisfied if and only if $t(x)$ divides $U(x)V(x) - W(x)$, i.e., there exists a quotient polynomial $h(x)$ such that

$$U(x)V(x) - W(x) = h(x)t(x). \tag{7}$$

Lemma 1 (Schwartz-Zippel)**.** *If $P(x)$ is a non-zero polynomial of degree d over $\mathbb{F}$, then for a uniformly random $\tau \in \mathbb{F}$, $\Pr[P(\tau) = 0] \leq d/|\mathbb{F}|$.*

Lemma 1 implies that evaluating the QAP identity at a hidden random point τ distinguishes a true polynomial identity from a false one with overwhelming probability over the 254-bit scalar field used in our implementation. The proof is given in Appendix A.

Groth16 verification. Groth16 uses a trusted setup phase for each compiled relation. The setup samples τ uniformly at random, publishes encrypted powers of τ, such as $\{\tau^k G_1\}_{k=0}^{d}$ and $\{\tau^k G_2\}_{k=0}^{d}$, and then destroys τ. Using the proving key, $\mathsf{Prove}(\mathsf{pk}_i; w_i)$ homomorphically evaluates the QAP polynomials at τ without learning τ. In a simplified presentation, the resulting proof consists of three elliptic-curve points:

$$\begin{aligned} A &= U(\tau)G_1 \in \mathbb{G}_1, \\ B &= V(\tau)G_2 \in \mathbb{G}_2, \\ C &= \big(W(\tau) + h(\tau)t(\tau)\big)G_1 \in \mathbb{G}_1. \end{aligned} \tag{8}$$

The proof $\pi = (A, B, C)$ contains a constant number of group elements regardless of the computation size. The verifier then checks a pairing equation of the form

$$e(A, B) \stackrel{?}{=} e(C, G_2). \tag{9}$$

By bilinearity (Equation 4), this holds if and only if $U(\tau)V(\tau) = W(\tau) + h(\tau)t(\tau)$, which by Lemma 1 holds with overwhelming probability only when the QAP identity (Equation 7) is satisfied. Thus, the interpolation step turns many local constraint checks into one succinct algebraic check. This is a simplified presentation of the Groth16 verifier: the full protocol includes additional setup constants and randomized proof terms for zero knowledge and soundness. We use this form only to highlight the key succinctness mechanism, namely that many R1CS constraints are reduced to a constant number of pairing checks.

The reason τ must remain hidden is that it acts as a setup trapdoor. If an adversary learns τ, it may be able to construct proofs that pass verification without a valid witness, breaking soundness. In deployed systems, this trust can be distributed through a multi-party setup ceremony: as long as at least one participant honestly samples and erases its secret randomness, the final trapdoor remains unknown. For simplicity, our prototype uses the single-setup abstraction above and assumes τ is destroyed after key generation.

4. Experiments

We instantiate P^{4}Bench with two representative problem families, Boolean Satisfiability (SAT) and the Discrete Logarithm Problem (DLP), as controlled testbeds. SAT tests symbolic search over discrete constraints, while DLP tests modular arithmetic and number-theoretic reasoning. We organize the experiments as follows: §4.1 introduces the two problem families, §4.2 describes instance generation and model evaluation, and §4.3 reports model performance together with the practical cost of proof generation and verification.

4.1. Benchmark Problem Families

Boolean Satisfiability (SAT). Given a Boolean formula in Conjunctive Normal Form (CNF), find a truth assignment that satisfies every clause. SAT is the canonical NP-complete problem (Cook, 2023): hard to solve, yet easy to verify. It tests an LLM's ability to perform systematic logical search and constraint reasoning. R1CS encoding details are in Appendix E.

Discrete Logarithm Problem (DLP). Given a prime p, a generator g, and $h = g^x \bmod p$, find the secret exponent x. DLP is a cornerstone of modern cryptography and tests an LLM's modular arithmetic and number-theoretic reasoning. Verification and proof details are in Appendix F.

4.2. Experimental Setup

We generate 500 random satisfiable SAT instances with a controlled difficulty distribution: 200 easy instances with 5 to 20 clauses and 3 to 8 variables, 200 medium instances with 20 to 50 clauses and 8 to 15 variables, and 100 hard instances with 50 to 100 clauses and 15 to 25 variables. Each instance is satisfiable by construction. We first sample a random assignment, generate clauses that are consistent with that assignment, and then discard the assignment before evaluation. This sampled assignment is only used to construct a valid instance; it is not treated as a unique answer key. During evaluation, any model-submitted assignment that satisfies the published CNF formula is accepted and can be used to form the private witness w_i. Full generation parameters are provided in Appendix C.

For DLP, we generate 500 instances using safe primes of varying bit sizes. The benchmark contains 200 easy instances with 5 to 8-bit primes, 200 medium instances with 8 to 14-bit primes, and 100 hard instances with 14 to 20-bit primes. Each instance publishes (p, g, h, q) with $h = g^x \bmod p$, while withholding the secret exponent x.

We evaluate seven frontier LLMs: Gemini 3 Flash, Claude Sonnet 4.5, GPT-4.1-mini (Singh et al., 2025), Kimi K2 (Team et al., 2026), Gemini 2.5 Pro (Comanici et al., 2025), Claude Opus 4.5, and Qwen 3.5 Plus (Qwen Team, 2026). Each model receives one problem at a time and is allowed to reason step by step before producing a final answer in a constrained format. We set the temperature to 0.2 for all models to reduce sampling variance while still allowing non-greedy reasoning. Prompt templates are listed in Appendix G.

4.3. Results

DLP solving performance. Table 2 reports results on 500 DLP instances. Performance varies substantially across models and drops sharply as the modulus size increases. Qwen 3.5 Plus performs best overall, reaching 77.2% accuracy, solving 89.5% of medium instances, and being the only model with nonzero accuracy on hard instances. The remaining models range from 19.2% to 53.0% overall and obtain 0.0% accuracy on the hard tier.

SAT solving performance. Table 3 reports results on 500 SAT instances. Accuracy is generally higher than on DLP, but the harder tiers still separate models. Qwen 3.5 Plus achieves the best overall performance at 99.6%, closely followed by Claude Opus 4.5 at 99.2%, while smaller models show larger drops on medium and hard instances.

ZKP proof generation. For each correctly solved instance, we generate a Groth16 zk-SNARK proof following Algorithm 1. This cost is incurred only at final submission time: during model development or prompt tuning, a provider can use the ordinary public checker locally and need not generate zero-knowledge proofs until it wants to publish a verifiable score. In our prototype, the median proof-generation time after setup is 0.14 ms for SAT and 0.11 ms for DLP. Verification uses a constant number of pairing checks and is independent of the number of constraints and witness size, typically taking about 5–10 ms per proof. The proving workload is also highly parallelizable because each benchmark instance can be proved independently. Appendix D reports additional timing details.

5. Discussion

In this section, we discuss two research questions regarding P^{4}Bench. First, what kinds of problems are within its natural scope (§5.1)? Second, how does this construction-oriented framework relate to formal proof systems such as Lean (§5.2)?

5.1. Scope: Verifiable Constructive Problems

The most suitable targets for P^{4}Bench are problems where finding a solution is difficult but checking a proposed solution is relatively easy. This includes constructive mathematical problems, combinatorial puzzles, constraint-satisfaction tasks, and search problems that may require broad exploration or strong insight to solve, but whose final answer can be verified by a deterministic checker. In this sense, the framework is not limited to SAT and DLP; these are controlled instantiations of a broader pattern in which a hidden object is certified against a public relation.

As a frontier-scale example, consider the Hadamard matrix problem: finding an $n \times n$ matrix $H \in \{+1, -1\}^{n \times n}$ such that $HH^\top = nI_n$. This is one of the open problems published by FrontierMath in March 2026[1]. The smallest open

[1] https://epoch.ai/frontiermath/open-problems/

Table 2. LLM performance on 500 DLP benchmark instances, broken down by difficulty tier. *Easy*: 5 to 8-bit primes (200 instances). *Medium*: 8 to 14-bit primes (200 instances). *Hard*: 14 to 20-bit primes (100 instances).

Model	Easy (%)	Medium (%)	Hard (%)	Overall (%)
Gemini 3 Flash	44.5	3.5	0.0	19.2
Claude Sonnet 4.5	95.0	34.0	0.0	51.6
GPT-4.1-mini	94.0	21.0	0.0	46.0
Kimi K2	**98.5**	34.0	0.0	53.0
Gemini 2.5 Pro	81.5	14.5	0.0	38.4
Claude Opus 4.5	93.5	32.0	0.0	50.2
Qwen 3.5 Plus	98.0	**89.5**	**11.0**	**77.2**

Table 3. LLM performance on 500 SAT benchmark instances, broken down by difficulty tier. *Easy*: 5 to 20 clauses, 3 to 8 variables (200 instances). *Medium*: 20 to 50 clauses, 8 to 15 variables (200 instances). *Hard*: 50 to 100 clauses, 15 to 25 variables (100 instances).

Model	Easy (%)	Medium (%)	Hard (%)	Overall (%)
Gemini 3 Flash	73.5	31.0	49.0	51.6
Claude Sonnet 4.5	97.5	49.5	25.0	63.8
GPT-4.1-mini	96.0	74.5	61.0	80.4
Kimi K2	98.0	87.5	33.0	80.8
Gemini 2.5 Pro	99.0	91.0	56.0	87.2
Claude Opus 4.5	**100.0**	99.5	97.0	99.2
Qwen 3.5 Plus	**100.0**	**100.0**	**98.0**	**99.6**

case is $n = 668$, and a solution would be a publishable mathematical result. Our framework lets a model provider prove possession of such a matrix via a constant-size Groth16 proof without revealing a single entry. We give the complete R1CS encoding in Appendix B.

5.2. Relationship to Formal Proof Systems

This construction-oriented scope also clarifies the relationship between P^{4}Bench and Lean (De Moura et al., 2015). Lean is designed for proof-oriented tasks, where the goal is to produce a fully formalized proof script that can be checked by a proof assistant. Extending our approach from constructive problems to theorem-proving benchmarks would therefore require zero-knowledge verification of formal proofs. Prior work on zkPi (Laufer et al., 2024) takes an important step in this direction, proving Lean theorems in zero knowledge: it successfully proves 57.9% of theorems in Lean's standard library and 14.1% of theorems in mathlib within 4.5 minutes per theorem. However, the large gap between standard-library and mathlib coverage indicates that many theorems used in Lean's broader mathematical ecosystem still require substantial additional work before this direction can support general-purpose proof benchmarks. In this sense, P^{4}Bench and Lean-based approaches are complementary: our current framework targets hidden-solution construction problems, while Lean-based ZKP systems point toward a possible but still technically demanding path for private theorem-proving evaluation.

6. Conclusion

We presented P^{4}Bench, a framework for LLM evaluation that uses zero-knowledge proofs to decouple public problem release from private answer disclosure. In P^{4}Bench, a benchmark provider can publish problem instances and public relations, while a model provider proves knowledge of a hidden answer satisfying the corresponding relation without revealing the answer itself. This turns benchmark scoring into a publicly verifiable computation problem: anyone can check submitted proofs and recompute scores, yet reusable question–answer pairs need not be exposed. Our SAT and DLP instantiations show that current LLMs can be efficiently evaluated within the P^{4}Bench framework, with proof generation and verification incurred only at final submission time. More broadly, ZKP-based evaluation offers a concrete path toward mathematical benchmarks that preserve answer privacy, reduce contamination channels, and remain auditable by the research community.

Impact Statement

This work aims to improve the reliability and transparency of LLM evaluation for mathematical reasoning. By enabling public verification without answer disclosure, it may reduce benchmark contamination and make evaluation more auditable; at the same time, deployments should ensure that cryptographic benchmark infrastructure remains accessible to the broader research community.

References

Balunović, M., Dekoninck, J., Petrov, I., Jovanović, N., and Vechev, M. Matharena: Evaluating llms on uncontaminated math competitions. *arXiv preprint arXiv:2505.23281*, 2025.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, Ł., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., Dhillon, I., Blistein, M., Ram, O., Zhang, D., Rosen, E., et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Cook, S. A. The complexity of theorem-proving procedures. In *Logic, automata, and computational complexity: The works of Stephen A. Cook*, pp. 143–152. 2023.

De Moura, L., Kong, S., Avigad, J., Van Doorn, F., and von Raumer, J. The lean theorem prover (system description). In *International Conference on Automated Deduction*, pp. 378–388. Springer, 2015.

Fiat, A. and Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pp. 186–194. Springer, 1986.

Fu, Y., Uzuner, O., Yetisgen-Yildiz, M., and Xia, F. Does data contamination detection work (well) for llms? a survey and evaluation on detection assumptions. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5235–5256, 2025.

Gao, B., Song, F., Yang, Z., Cai, Z., Miao, Y., Dong, Q., Li, L., Ma, C., Chen, L., Tang, Z., et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. In *International Conference on Learning Representations*, volume 2025, pp. 100540–100569, 2025.

Glazer, E., Erdil, E., Besiroglu, T., Chicharro, D., Chen, E., Gunning, A., Olsson, C. F., Denain, J.-S., Ho, A., Santos, E. d. O., et al. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *arXiv preprint arXiv:2411.04872*, 2024.

Golchin, S. and Surdeanu, M. Time travel in llms: Tracing data contamination in large language models. In *International Conference on Learning Representations*, volume 2024, pp. 43008–43029, 2024.

Goldreich, O., Micali, S., and Wigderson, A. How to prove all np statements in zero-knowledge and a methodology of cryptographic protocol design. In *Conference on the Theory and Application of Cryptographic Techniques*, pp. 171–185. Springer, 1986.

Groth, J. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pp. 305–326. Springer, 2016.

He, C., Luo, R., Bai, Y., Hu, S., Thai, Z., Shen, J., Hu, J., Han, X., Huang, Y., Zhang, Y., et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Ishida, T., Lodkaew, T., and Yamane, I. How can i publish my llm benchmark without giving the true answers away? *arXiv preprint arXiv:2505.18102*, 2025.

Jain, N., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

JJ, S. C. Q. and Vandewalle, J. Efficient identification and signatures for smart cards. In *Advances in Cryptology—EUROCRYPT '89*, pp. 688–689, 1990.

Laufer, E., Ozdemir, A., and Boneh, D. zkPi: Proving lean theorems in zero-knowledge. Cryptology ePrint Archive, Paper 2024/267, 2024. URL https://eprint.iacr.org/2024/267.

Li, Y., Guo, Y., Guerin, F., and Lin, C. An open-source data contamination report for large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 528–541, 2024.

Liang, Z., Yu, L., Shiyu, Z., Ye, Q., and Hu, H. How much do large language model cheat on evaluation? benchmarking overestimation under the one-time-pad-based framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 37636–37644, 2026.

Mirzadeh, I., Alizadeh-Vahid, K., Shahrokhi, H., Tuzel, O., Bengio, S., and Farajtabar, M. Gsm-symbolic: Understanding the limitations of mathematical reasoning

in large language models. In *International Conference on Learning Representations*, volume 2025, pp. 94743–94765, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, 2026.

Rajore, T., Chandran, N., Sitaram, S., Gupta, D., Sharma, R., Mittal, K., and Swaminathan, M. Truce: Private benchmarking to prevent contamination and improve comparative evaluation of llms, 2024. URL https://arxiv.org/abs/2403.00393.

Ravi, N., Ying, K., Nesterov, V., Krishnan, R., Uskuplu, E., Xia, B., Aswedige, J., and Nashold, L. Formalproofbench: Can models write graduate level math proofs that are formally verified? *arXiv preprint arXiv:2603.26996*, 2026.

Shi, W., Ajith, A., Xia, M., Huang, Y., Liu, D., Blevins, T., Chen, D., and Zettlemoyer, L. Detecting pretraining data from large language models. In *International Conference on Learning Representations*, volume 2024, pp. 51826–51843, 2024.

Singh, A. et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Sun, H., Li, J., and Zhang, H. zkllm: Zero knowledge proofs for large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pp. 4405–4419, 2024.

Sun, H., Min, Y., Chen, Z., Zhao, W. X., Fang, L., Liu, Z., Wang, Z., and Wen, J.-R. Challenging the boundaries of reasoning: An olympiad-level math benchmark for large language models. *arXiv preprint arXiv:2503.21380*, 2025.

Team, K., Bai, T., Bai, Y., Bao, Y., Cai, S., Cao, Y., Charles, Y., Che, H., Chen, C., Chen, G., et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Wang, H., Li, H., Ko, B., and Zhang, H. On the fragility of benchmark contamination detection in reasoning models. *arXiv preprint arXiv:2510.02386*, 2025.

Wang, Z. G. Nanozk: Layerwise zero-knowledge proofs for verifiable large language model inference. *arXiv preprint arXiv:2603.18046*, 2026.

White, C., Dooley, S., Roberts, M., Pal, A., Feuer, B., Jain, S., Shwartz-Ziv, R., Jain, N., Saifullah, K., Dey, S., et al. Livebench: A challenging, contamination-limited llm benchmark. In *International Conference on Learning Representations*, volume 2025, pp. 91595–91631, 2025.

Wu, M., Zhang, Z., Dong, Q., Xi, Z., Zhao, J., Jin, S., Fan, X., Zhou, Y., Lv, H., Zhang, M., et al. Reasoning or memorization? unreliable results of reinforcement learning due to data contamination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33944–33952, 2026.

Yue, A. S., Madaan, L., Moskovitz, T., Strouse, D., and Singh, A. K. Harp: A challenging human-annotated math reasoning benchmark. *arXiv preprint arXiv:2412.08819*, 2024.

Zhang, J., Sun, J., Yeats, E., Ouyang, Y., Kuo, M., Zhang, J., Yang, H., and Li, H. Min-k%++: Improved baseline for pre-training data detection from large language models. In *International Conference on Learning Representations*, volume 2025, pp. 64845–64862, 2025.

Zhang, R., Wang, L., Zhang, H., Zheng, Z., and Lan, Y. From unfamiliar to familiar: Detecting pre-training data via gradient deviations in large language models. *arXiv preprint arXiv:2603.04828*, 2026.

Zhang, W., Zhang, R., Guo, J., de Rijke, M., Fan, Y., and Cheng, X. Pretraining data detection for large language models: A divergence-based calibration method. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 5263–5274, 2024.

Zou, C., Guo, X., Yang, R., Zhang, J., Hu, B., and Zhang, H. Dynamath: A dynamic visual benchmark for evaluating mathematical reasoning robustness of vision language models. In *International Conference on Learning Representations*, volume 2025, pp. 48337–48383, 2025.

A. Proof of Lemma 1

Proof of Lemma 1. Let $P(x)$ be a non-zero polynomial of degree d over a finite field $\mathbb{F}$. A non-zero degree-d polynomial has at most d roots in $\mathbb{F}$. Therefore, for a uniformly random $\tau \in \mathbb{F}$,

$$\Pr[P(\tau) = 0] = \frac{|\{\tau \in \mathbb{F} : P(\tau) = 0\}|}{|\mathbb{F}|} \leq \frac{d}{|\mathbb{F}|}.$$

In our setting, $\mathbb{F}$ is the scalar field of the BN254 curve with $|\mathbb{F}| \approx 2^{254}$. Even for circuits with millions of constraints, the resulting soundness error is negligible. Thus, if a prover does not know a valid witness, the probability that a fraudulent proof passes the pairing check (Equation 9) is cryptographically negligible. □

B. ZKP Encoding for Hadamard Matrix Verification

We describe how to encode the verification of a Hadamard matrix of order n as an R1CS, enabling zero-knowledge proof of its existence.

Problem statement. A Hadamard matrix of order n is an $n \times n$ matrix H satisfying:

1. Every entry is ± 1: $h_{ij} \in \{-1, +1\}$ for all $i, j \in \{1, \ldots, n\}$.
2. All rows are mutually orthogonal: $HH^\top = nI_n$.

The Hadamard conjecture states that such matrices exist for every n divisible by 4. The smallest open case is $n = 668$.

Witness. The private witness consists of all n^2 matrix entries $\{h_{ij}\}_{i,j=1}^n$.

R1CS encoding. We encode the verification as follows:

(1) Entry constraints. Each entry must satisfy $h_{ij} \in \{-1, +1\}$, which is equivalent to $h_{ij}^2 = 1$. In R1CS form:

$$h_{ij} \cdot h_{ij} = 1 \qquad \text{for all } i, j \in \{1, \ldots, n\} \tag{10}$$

This yields n^2 constraints.

(2) Orthogonality constraints. The condition $HH^\top = nI_n$ requires that for all row pairs (i, j):

$$\sum_{k=1}^{n} h_{ik} \cdot h_{jk} = \begin{cases} n & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{11}$$

Each inner product $\sum_{k=1}^n h_{ik} h_{jk}$ involves n multiplications. Since R1CS allows only one multiplication per constraint, we compute the sum incrementally using auxiliary variables. Define $s_0 = 0$ and for $k = 1, \ldots, n$:

$$h_{ik} \cdot h_{jk} = p_k, \qquad s_k = s_{k-1} + p_k \tag{12}$$

The first equation is an R1CS multiplication constraint; the second is a linear constraint (absorbed into the R1CS matrices at no extra cost). We then enforce $s_n = n$ (for $i = j$) or $s_n = 0$ (for $i \neq j$) as a linear check.

This yields n multiplication constraints per row pair, and there are $\binom{n}{2} + n = \frac{n(n+1)}{2}$ row pairs (including diagonal self-orthogonality checks). The total number of R1CS constraints is:

$$\underbrace{n^2}_{\text{entry}} + \underbrace{\frac{n(n+1)}{2} \cdot n}_{\text{orthogonality}} = n^2 + \frac{n^2(n+1)}{2} \tag{13}$$

Concrete complexity for $n = 668$. Using Equation (13), for $n = 668$ the total is approximately 149.6 million constraints, yet the proof size remains 128 bytes and verification uses a constant number of pairing checks. Modern zk-SNARK implementations (e.g., Gnark) routinely handle circuits of this scale.

Generality. This encoding pattern applies broadly: any mathematical claim that can be verified by checking a system of polynomial equations over the entries of a solution can be encoded as an R1CS circuit. This includes problems involving matrix properties (determinants, eigenvalues, decompositions), combinatorial structures (Latin squares, graph colorings), number-theoretic objects (factorizations, discrete logarithms), and algebraic constructions, covering a wide swath of FrontierMath problems.

C. Benchmark Generation Details

SAT instance parameters. We generate 500 random satisfiable SAT instances across three difficulty tiers:

- **Easy** (200 instances): 5 to 20 clauses, 3 to 8 variables.
- **Medium** (200 instances): 20 to 50 clauses, 8 to 15 variables.
- **Hard** (100 instances): 50 to 100 clauses, 15 to 25 variables.

Clause width is a mix of 3-SAT and 4-SAT (3-literal clauses sampled with probability 0.75, 4-literal with probability 0.25 when $n \geq 4$). Random seed: 42 (for reproducibility).

Satisfiability guarantee. To ensure every instance is satisfiable, we first sample a random assignment $\mathbf{a} \in \{0, 1\}^n$,

then generate each clause so that at least one literal evaluates to true under $\mathbf{a}$. Specifically, for each literal in a clause, we match the assignment with probability 0.5 and choose a random polarity otherwise; if no literal in the clause is satisfied, we force one literal to match $\mathbf{a}$. The assignment is then discarded and only the CNF formula is published. We do not require the generated instance to have a unique satisfying assignment: any assignment that satisfies the public formula is a valid answer under our benchmark.

DLP instance parameters. We generate 500 DLP instances using safe primes p (where $(p-1)/2$ is also prime) across three difficulty tiers:

- **Easy** (200 instances): 5 to 8-bit primes.
- **Medium** (200 instances): 8 to 14-bit primes.
- **Hard** (100 instances): 14 to 20-bit primes.

For each instance, we select a generator g of the order-q subgroup (where $q = (p-1)/2$), sample a secret exponent $x \xleftarrow{\$} \mathbb{Z}_q$, and compute $h = g^x \bmod p$. The published instance is (p, g, h, q); the secret x is withheld. Random seed: 42 (for reproducibility).

Cryptographic parameters. We use the BN254 elliptic curve (also known as alt_bn128) for the Groth16 proof system, with prime field order:

$$\begin{aligned} p = {} & 21888242871839275222246405745 \\ & 25257275088548364400416034343 \\ & 69820418657580849561 7 \end{aligned} \tag{14}$$

This is a 254-bit prime, providing approximately 128 bits of security. The trusted setup generates per-instance proving and verification keys from the R1CS structure.

D. Proof Generation Timing

We report wall-clock timings from our Python prototype to give an order-of-magnitude estimate of the proof-generation overhead; these numbers are not intended to be optimized SNARK implementation benchmarks. For SAT, over 498 correctly solved instances, proof generation after setup takes 0.14 ms at the median and 0.25 ms at the 95th percentile. Including per-instance setup, the median end-to-end time is 8.2 ms, with a 95th percentile of 43.2 ms. The median end-to-end times by difficulty tier are 3.0 ms for easy instances, 11.0 ms for medium instances, and 33.9 ms for hard instances.

For the DLP R1CS prototype, over 500 instances, proof generation after setup takes 0.11 ms at the median and 0.13 ms at the 95th percentile. Including per-instance setup, the median end-to-end time is 1.7 ms, with a 95th percentile of 4.0 ms. Groth16 verification uses a constant number of pairing checks and is independent of the number of constraints and witness size.

E. SAT Public Relation and R1CS Encoding

We encode the verification of a satisfying assignment for a CNF formula as a Rank-1 Constraint System over $\mathbb{F}_p$.

Boolean constraints. For each variable x_i, we add:

$$x_i \cdot (1 - x_i) = 0 \tag{15}$$

This ensures $x_i \in \{0, 1\}$ in the finite field.

Clause constraints. For a clause $C = (\ell_1 \vee \ell_2 \vee \ell_3)$, let v_j denote the truth value of literal ℓ_j (where $v_j = x_k$ for a positive literal and $v_j = 1 - x_k$ for a negative literal). The clause is satisfied if and only if not all literals are false:

$$(1 - v_1)(1 - v_2)(1 - v_3) = 0 \tag{16}$$

Since R1CS constraints allow only one multiplication per row, we decompose this using an auxiliary variable t:

$$(1 - v_1) \cdot (1 - v_2) = t \tag{17}$$

$$t \cdot (1 - v_3) = 0 \tag{18}$$

For 4-literal clauses $(\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4)$, we introduce two auxiliary variables:

$$(1 - v_1) \cdot (1 - v_2) = t_1 \tag{19}$$

$$(1 - v_3) \cdot (1 - v_4) = t_2 \tag{20}$$

$$t_1 \cdot t_2 = 0 \tag{21}$$

The complete R1CS for a SAT instance with n variables and m clauses of width k requires $n + (k-1) \cdot m$ constraints and $n + (k-2) \cdot m$ auxiliary variables.

Worked example. Consider the formula $\phi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_3)$ with $n = 3$ variables and $m = 4$ clauses. With assignment vector

$\vec{z} = [1, x_1, x_2, x_3]^T$, the R1CS matrices are:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix},$$
$$B = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{22}$$
$$C = \mathbf{0}_{7\times 4}$$

The satisfying assignment $x_1 = 1, x_2 = 0, x_3 = 1$ yields $\vec{z} = [1, 1, 0, 1]^T$, which satisfies $(A\vec{z}) \circ (B\vec{z}) = C\vec{z}$.

F. DLP Public Relation and R1CS Encoding

Given a safe prime p, subgroup order $q = (p-1)/2$, generator g, and public value h, a DLP instance publishes (p, q, g, h) where $h = g^x \bmod p$ for a secret exponent x. In our benchmark, the answer is $a_i = x$, and the zero-knowledge witness w_i contains x together with the auxiliary values needed to verify modular exponentiation inside an R1CS circuit.

The public relation for DLP is:

$$R_{\mathrm{DLP}}((p, q, g, h), w_i) = 1 \iff 0 \le x < q \ \wedge \ g^x \equiv h \pmod{p}. \tag{23}$$

We encode this relation as an arithmetic circuit and use the same Groth16 zk-SNARK pipeline as for SAT.

The circuit first bit-decomposes the private exponent

$$x = \sum_{j=0}^{\ell-1} 2^j b_j, \qquad b_j \in \{0, 1\}, \tag{24}$$

where $\ell = \lceil \log_2 q \rceil$, and range-checks that $x < q$. The public constants $G_j = g^{2^j} \bmod p$ are precomputed. Starting from $A_0 = 1$, the circuit conditionally multiplies by G_j whenever $b_j = 1$:

$$A_{j+1} = \begin{cases} A_j, & b_j = 0, \\ A_j G_j \bmod p, & b_j = 1. \end{cases} \tag{25}$$

This conditional update is encoded arithmetically by computing the modular product residue $M_j = A_j G_j \bmod p$ and enforcing

$$A_{j+1} = (1 - b_j) A_j + b_j M_j. \tag{26}$$

Because the Groth16 field is not the same as the integer modulus p, each modular multiplication is represented using an auxiliary quotient variable K_j:

$$A_j G_j = M_j + K_j p, \tag{27}$$

with range constraints ensuring $0 \le M_j < p$. Finally, the circuit enforces $A_\ell = h$. Thus, a valid Groth16 proof certifies knowledge of an exponent x satisfying the public DLP instance without revealing x.

DLP-specific alternative. DLP also admits a simpler proof of knowledge via the Schnorr protocol, which can be made non-interactive using the Fiat-Shamir heuristic (JJ & Vandewalle, 1990; Fiat & Shamir, 1986).

G. LLM Prompt Templates

Each model receives one problem at a time and is allowed to reason step by step before providing a final answer.

SAT prompt.

```
Solve the following SAT problem. Find a boolean
assignment for variables x1..x{n} that satisfies
ALL clauses simultaneously.

Variables: x1, x2, ..., x{n}
Number of clauses: {m}

Formula (CNF):
(x1 OR ~x2 OR x3) AND (~x1 OR x4 OR x5) AND ...

Think step by step. Try assigning values and
checking which clauses are satisfied.
Backtrack if needed.

After your reasoning, give your final answer on
the last line in this exact format:
ANSWER: [true/false, true/false, ...]

The array must have exactly {n} elements,
where the i-th element is the value for x_i.
```

DLP prompt.

```
Solve the discrete logarithm problem:

Given:
  p = {p} (prime modulus)
  g = {g} (generator)
  h = {h} (target)
  q = {q} (group order)

Find x such that g^x = h (mod p), where
0 <= x < q.

Think step by step. After your reasoning,
give your final answer on the last line in
this exact format:
ANSWER: x
```