# INTERNALIZING GEOMETRIC LAW: LEARNING FROM SOLVER RESIDUALS FOR PRECISION-CRITICAL GENERATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Large Language Models frequently hallucinate in precision-critical domains such as technical diagramming and mechanical design, where outputs must satisfy strict geometric constraints. We study this failure where it can be checked exactly: open-ended geometric synthesis from natural language, in which an LLM must turn a free-form description into a precise geometric construction whose objects simultaneously satisfy dozens of interacting constraints. To make it trainable we release *PyGeoX*, a programmable geometric DSL that compiles declarative constraints into a differentiable loss, an agent harness that exposes PyGeoX to any LLM, and a 900-problem evaluation suite with per-constraint verifiable rewards spanning procedurally generated, examination and engineering problems. Using PyGeoX as a verifier, we identify a failure mode we call *Outlier Gradient Masking*: when residuals are aggregated through a single norm, for example $\exp(-\text{MSE})$, one outlier constraint can nullify the learning signal from all the others. To address this we propose *Saturating Additive Rewards* (SAR), which decompose the reward into bounded per-constraint terms so that progress on satisfied constraints survives. Against MSE-based rewards, the natural baseline for geometry solvers, SAR improves the hard-tier solving rate by $2.4\times$, and the resulting 8B model is competitive with much larger frontier systems.

## 1 INTRODUCTION

Large Language Models have achieved remarkable proficiency in semantic tasks, from code synthesis to literature summarization. However, their capability to adhere to rigorous, precision-critical constraints remains brittle. This phenomenon, which we term "precision hallucination," diverges fundamentally from the "semantic hallucinations" of earlier models: outputs are syntactically coherent and semantically plausible but violate the exact laws of geometry, physics, or logical consistency (Trinh et al., 2024).

For researchers aiming to deploy generative models in engineering domains, specifically technical diagramming, computer-aided design (CAD) and kinematic mechanism design, this limitation is existential. We consider all such problems under the umbrella of *Geometric Constraint Solving* (GCS), as they involve finding a configuration of geometric entities that satisfies a set of relational constraints.

Current approaches to mitigate this limitation often rely on delegating reasoning to an external numerical or symbolic solver. While this ensures strict constraint satisfaction, it restricts the LLM to a translation-only role. The model merely converts natural language into solver-specific syntax, which hinders its ability to internalize spatial logic or perform creative, constrained synthesis. Existing symbolic solvers and geometry constraint engines, such as FormalGeo (Zhang et al., 2024c) and the static predicate systems used in AlphaGeometry (Trinh et al., 2024) or FGeo-DRL (Zou et al., 2024), impose inherent limitations on the expressivity of objects and relationships. For instance, some of these systems cannot deal with inequality relationships (e.g., "line 1 is larger than line 2"). Crucially, this creates a bottleneck across both pure geometric solving and engineering domains like kinematic synthesis. Current symbolic engines lack the vocabulary to define dynamic functional requirements, which limits their utility to textbook geometry problems rather than real-world engineering design.



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

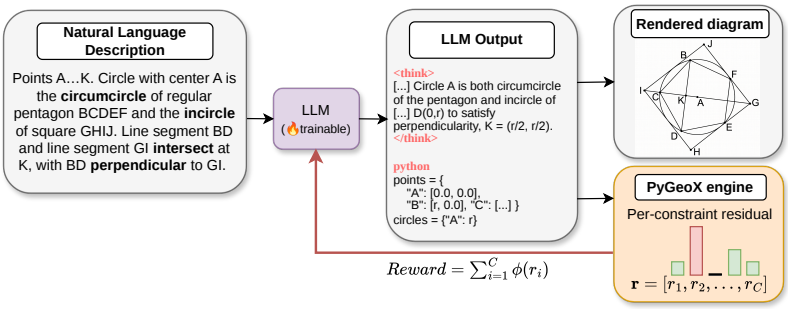



Figure 1: Task overview. The agent thinks and emits Python code containing exact coordinates and radii.

**Our approach.** We formulate end-to-end GCS as a new LLM alignment task and build the infrastructure to train on it: *PyGeoX*, a geometry library with auto-compiled differentiable rewards (Figure 3), a procedural data pipeline yielding $\sim 100k$ validated training problems, and the benchmarks needed to measure progress. With this infrastructure in place, the remaining question is reward design. Standard Reinforcement Learning with Verifiable Rewards (RLVR) aggregators, such as global norms (e.g., $\exp(-\|\mathbf{r}\|_2^2)$, where $\mathbf{r} = [r_1, \ldots, r_C]$ is the per-constraint residual vector returned by the solver) and binary success indicators, suffer a failure mode we call *Outlier Gradient Masking*: a single severely violated constraint drives the aggregated reward to zero, zeroing out the policy gradient and destroying the learning signal from every satisfied constraint. We propose *Saturating Additive Rewards* (SAR), which sum independent bounded kernels per constraint so that progress on satisfied constraints survives. Figure 1 summarizes the overall framework.

### 1.1 Preliminaries

**Task.** Given a natural-language description x specifying geometric objects (points, lines, circles, polygons) and relational constraints (incidence, perpendicularity, tangency, length, area, ...), the model must reason about the geometry and produce a Python program whose output is a structured dictionary containing exact metric values for every free parameter: point coordinates and circle radii, such that all constraints are simultaneously satisfied. The model operates as a single-turn code agent with access to standard scientific libraries (`numpy`, `scipy`, `sympy`) and may choose any solving strategy: constructive ruler-and-compass derivations, numerical optimization, or hybrids.

**Verification.** The PyGeoX symbolic-numeric solver (Section 4.1) compiles the constraints into a residual vector $\mathbf{r} = [r_1, \ldots, r_C]$, where each $r_i \geq 0$ measures the violation magnitude of one constraint (e.g., Euclidean distance for incidence, angular deviation for parallelism). A solution is judged correct when $\|\mathbf{r}\|_2^2 < 10^{-3}$ (allowing for floating-point error propagation through the constraint equations). Crucially, the LLM cannot use the PyGeoX library. PyGeoX is used only off-policy: for procedural training-data generation (Section 4.2) and for computing per-constraint residuals during RL reward evaluation.

**Why this is a new task.** No existing public work targets this exact input/output signature. Geometry problem-solving models and benchmarks (Chen et al., 2021; 2022; Zhang et al., 2023; 2024c;a; Xu et al., 2025) output scalar answers to textbook geometry problems or proofs; diagram-generation benchmarks (Wei et al., 2026) evaluate rendered images; theorem-proving systems (Trinh et al., 2024; Zou et al., 2024) output discrete proof steps. None require translating natural language geometric descriptions to numerical coordinates directly using LLMs.

**Contributions.** In this work, we introduce **PyGeoX-RL**, a neuro-symbolic framework that teaches LLMs to *internalize geometric law*. Our contributions are:

1. **End-to-end GCS as an LLM alignment task.** We formulate Geometric Constraint Solving as a new alignment problem: natural language to exact metric coordinates verified by per-constraint residuals, a task formulation that no existing benchmark targets and that even frontier LLMs cannot reliably solve out of the box.
2. **The PyGeoX engine.** We release a programmable geometric environment covering 35 object types and 38 relationships, with auto-compiled differentiable reward functions. PyGeoX

serves as both a scalable data generation engine and an RL Gym environment, and supports expressivity (inequality constraints, polygon relationships) absent from prior solvers.

3. **Data pipeline and benchmarks.** We provide a procedural pipeline yielding $\sim$100k validated training problems and a 900-problem evaluation suite. PyGeoX-Bench contributes 450 problems stratified into three difficulty levels, obtained from the same procedural pipeline. We also provide three out-of-distribution suites of 150 problems each: PyGeoX-Wild, adapted from public geometry benchmarks, PyGeoX-CAD, phrased as engineering drawing notes, and PyGeoX-Mech, which poses inverse mechanism synthesis problems.
4. **Reward design that makes RL on GCS viable.** We identify a failure mode, Outlier Gradient Masking, common to global-norm and sparse rewards, and introduce Saturating Additive Rewards (SAR). To our knowledge, this is the first application of per-constraint solver residuals as a dense reward for autoregressive LLMs. SAR substantially outperforms field-standard sparse and global-norm rewards on hard GCS problems.
5. **An agent harness.** We contribute an agent harness that lets a model formalize a problem in PyGeoX and solve it with the engine's numerical optimizer, escalating through a range of strategies. This is deliberately the setting our task definition excludes.

## 2 Related Work

**Neuro-Symbolic Geometric Reasoning.** Integrating geometric engines with LLMs mitigates the precision hallucinations of pure neural models. "Code-as-Reasoning" frameworks like *GeoCoder* (Sharma et al., 2025), *ToRA* (Gou et al., 2024), and *CAD-Llama* (Li et al., 2025b) effectively reduce the LLM to a translator from natural language into solver-specific syntax, delegating all geometric reasoning to the external engine. Recent work adds RL to this paradigm (Li et al., 2025a; Yin et al., 2025), but although these systems benefit from solver feedback during training, the LLM still remains a DSL translator. Discrete deductive systems such as *AlphaGeometry* (Trinh et al., 2024) and *FormalGeo* (Zhang et al., 2024c) offer logical rigor but are designed for theorem proving in discrete spaces rather than constructive synthesis. Lastly, traditional CAD tools like *SolveSpace* (Westhues & SolveSpace Contributors, 2022) and *Parasolid* (Sears & Allen, 1991) are engineered for final precision rather than intermediate learning, lacking granular feedback necessary for agent improvement.

**Physics-Informed and Constraint-Based Learning.** Our approach aligns theoretically with Physics-Informed Neural Networks (PINNs, Raissi et al., 2017), which embed physical laws into deep learning by treating PDE residuals as loss functions. While recent works like *PIRF* (Yuan et al., 2025) have adapted this paradigm to diffusion models, to the best of our knowledge, we are the first to translate this "residual-as-supervision" framework into the domain of autoregressive LLMs. Current state-of-the-art reasoning models, such as *DeepSeek-R1* (Guo et al., 2025) and other RLVR approaches (Wang et al., 2026), typically discard this fine-grained residual information, relying instead on sparse, binary outcome supervision ($r \in \{0, 1\}$).

**Generative Geometric Design.** Wang et al. (2025) target diagram generation from natural language but delegate coordinate computation to an algorithmic solver; the LLM autoformalizes constraints into a formal specification, and the system is training-free. Casey et al. (2025) fine-tune LLMs to predict *constraint labels* given primitives with known coordinates, a classification task on existing geometry rather than coordinate generation. In contrast, our LLM directly emits metric coordinates satisfying every constraint, and the solver is used only for verification and reward computation.

## 3 Reward Design for GCS

### 3.1 From Residuals to Rewards

A source of confusion in this domain arises from conflating the role of *Reward Functions* in Reinforcement Learning with *Loss Functions* in Supervised Learning. One might intuitively assume that because the sum of squared errors (SSE) works well for regression, a simple transformation (such as negation) will serve as an effective reward.

In supervised regression, minimizing a global norm loss like SSE ($\mathcal{L} = \|\mathbf{r}\|_2^2$) is effective because the gradient is directly proportional to the residual ($\nabla\mathcal{L} \propto \mathbf{r}$). In RL, the mechanics of the update

are fundamentally different. Let $\pi_\theta$ denote the language model policy with parameters $\theta$, mapping a prompt $\mathrm{x}$ to a distribution over generated sequences $\mathrm{y}$, and let $\mathcal{R}(\mathrm{x}, \mathrm{y})$ be a scalar reward that quantifies how well $\mathrm{y}$ satisfies the geometric constraints specified by $\mathrm{x}$. Given prompts $\mathrm{x} \sim \mathcal{D}$ and generated answers $\mathrm{y} \sim \pi_\theta(\cdot \mid \mathrm{x})$, the policy gradient (omitting KL regularization for clarity) is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathrm{x}\sim\mathcal{D},\, \mathrm{y}\sim\pi_\theta(\cdot|\mathrm{x})} \left[\mathcal{R}(\mathrm{x}, \mathrm{y}) \cdot \nabla_\theta \log \pi_\theta(\mathrm{y} \mid \mathrm{x})\right].$$

The update direction is determined solely by the log-probability gradient $\nabla_\theta \log \pi_\theta(\mathrm{y} \mid \mathrm{x})$, while the reward $\mathcal{R}(\mathrm{x}, \mathrm{y})$ acts only as a scalar multiplier. This creates a critical vulnerability when the reward depends on the residuals through a global aggregation, such as a sum of squared errors $\|\mathbf{r}\|_2^2$, any $\ell_p$ norm, or a transformation thereof (e.g., $\exp(-\|\mathbf{r}\|_2^2)$ or $\exp(-\mathrm{MSE})$). In all such cases, a single large outlier residual $r_i \gg 0$, common in complex GCS tasks, drives the aggregated reward to near-zero. This "all-or-nothing" scaling nullifies the entire policy gradient term, masking the agent's partial progress on the remaining constraints (Figure 2).

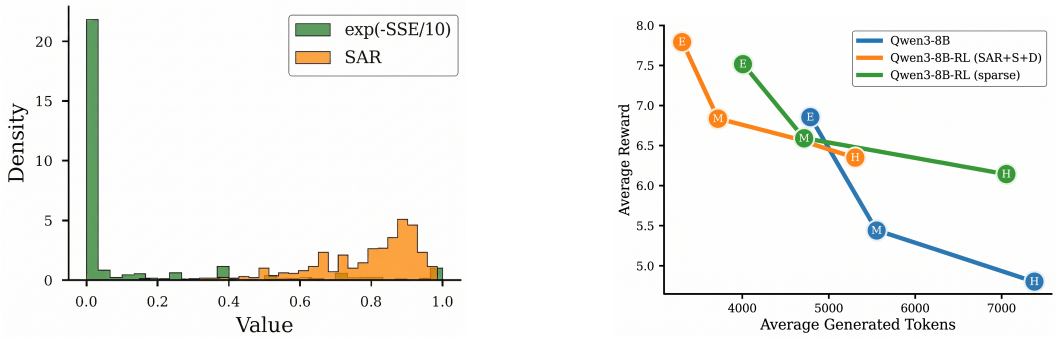



Figure 2: Reward signal analysis for Qwen3-8B on PyGeoX-Bench. (Left) The global-norm reward collapses to zero on partially correct base-model outputs, providing no learning signal and SAR preserves dense signal even when some constraints remain violated. (Right) SAR-trained models achieve higher reward with fewer tokens across all difficulty tiers (E-Easy, M-Medium, H-Hard) compared to the sparse baseline.

### 3.2 Saturating Additive Rewards

To resolve outlier gradient masking, we propose summing independent bounded kernels, one per constraint, so that no single residual can collapse the reward signal.

**Definition 1.** A reward function $\mathcal{R}(\mathbf{r})$ is a *Saturating Additive Reward* (SAR) if it decomposes as

$$\mathcal{R}_{\mathrm{SAR}}(\mathbf{r}) = \sum_{i=1}^{C} \phi(r_i), \tag{1}$$

for a monotonically decreasing kernel $\phi : \mathbb{R}_{\geq 0} \to [0, 1]$ with $\lim_{r\to\infty} \phi(r) = 0$.

We compare global-norm rewards $\mathcal{R}_G = \phi(\|\mathbf{r}\|_p)$ against SAR in two regimes. The first shows global-norm rewards are almost surely zero at random initialization in high dimensions, so RL cannot start. The second shows that even when the agent makes local progress on a single constraint, global-norm rewards mask it whenever the rest of the configuration is disordered, so RL cannot continue. SAR avoids both pathologies. Full statements and proofs are given in Theorems A.1 and A.3 of Appendix A.

**Signal availability in high dimensions.** For the policy gradient to be non-zero, the agent must encounter a non-negligible reward signal. Consider the residual space $\Omega = [0, M]^C$ (all possible per-constraint violation vectors), where $C$ is the constraint count. Also, consider the *effective reward volume* $V_\epsilon \subset \Omega$ where $\mathcal{R}(\mathbf{r}) > \epsilon$ (the region where reward exceeds threshold $\epsilon$). We prove (Theorem A.1) that as $C \to \infty$, $\mathrm{Vol}(V_\epsilon^G)/\mathrm{Vol}(\Omega) \to 0$ for global-norm rewards, while $\mathrm{Vol}(V_\epsilon^{\mathrm{SAR}})/\mathrm{Vol}(\Omega) \to 1$ for SAR. While pretrained LLMs are not truly random, base model performance on GCS remains weak (Table 10: Qwen3-8B achieves only 0.153 Hard SR), and early RL rollouts still sample broadly across $\Omega$. The probability of obtaining useful gradient signal in this regime decays exponentially for global norms and converges to one for SAR.

**Gradient sensitivity.** Learning efficiency also depends on whether a local improvement in constraint satisfaction produces a measurable change in the reward, captured by $\|\nabla_{r_i}\mathcal{R}\|$. We show (Theorem A.3) that the global-norm gradient $\|\nabla_{r_i}\mathcal{R}_G\| = |\phi'(L)| \cdot (r_i/L)^{p-1}$ is structurally coupled to the total error $L = \|\mathbf{r}\|_p$, whereas the SAR gradient $\|\nabla_{r_i}\mathcal{R}_{\text{SAR}}\| = |\phi'(r_i)|$ is strictly local. When the global configuration remains disordered ($L \gg r_i$), whether due to a single divergent outlier or aggregate error from a worsening subset, the global-norm signal for the successful constraint $i$ vanishes. On the other hand, the SAR signal remains non-zero and independent of $L$, allowing valid sub-solutions to be protected from global noise.

**Why compare SAR with global-norm rewards?** No prior method exists for this task, so we use *baseline* to mean a well-motivated alternative rather than a previous best method. Two considerations motivate using $\exp(-\text{SSE}/T)$ as a baseline. First, it is what the existing machinery provides, since SSE is the native objective of production geometry constraint solvers such as FreeCAD and SolveSpace. A practitioner who wires any of these into RLVR inherits $\|\mathbf{r}\|_2^2$ as the scalar feedback, and $\exp(-\|\mathbf{r}\|_2^2/T)$ is the standard way to map it into a bounded reward. Second, it is what the closest prior work uses: motion-imitation RL passes tracking errors through $\exp(-w\|\cdot\|^2)$ kernels (Peng et al., 2018), and physics-informed learning supervises on the mean squared error of physical residuals (Raissi et al., 2017). We tune it favourably, choosing $T_{\text{mse}} = 10$ to maximize reward spread on partially correct solutions, and it still collapses $60\%$ of them to near-zero (Appendix C.5 and Figure 2), so the comparison is conservative rather than a strawman.

From now on, we use the exponential kernel on both sides of the comparison, so what we contrast is $\exp(-\|\mathbf{r}\|_2^2/T)$ against $\frac{1}{C}\sum_i \exp(-r_i/T)$. In the limit, as $T \to 0$ the first collapses to the sparse indicator $\mathbb{I}[\|\mathbf{r}\|_2^2 < \epsilon]$, which discards everything except whether the whole diagram is correct, while the second tends to the fraction of constraints satisfied.

### 3.3 Reward function for GCS

A pure SAR reward is robust to outlier gradient masking, but two domain-specific failure modes prevent it from being sufficient on its own.

**Reward plateau.** The dense SAR component rewards partial constraint satisfaction monotonically: a configuration that satisfies 14 of 16 constraints receives high reward even though it is, by GCS standards, wrong. The agent has little incentive to push the final residuals through the strict $\|\mathbf{r}\|_2^2 < 10^{-3}$ threshold required for a valid solution. A sparse outcome bonus, activated only when all constraints are simultaneously satisfied, supplies that force.

**Geometric degeneracy.** Geometric constraints can be satisfied by trivial configurations, for instance, by collapsing all points to a single coordinate or placing every line on the same axis. These solutions can have no residual but are useless as geometric constructions. A degeneracy penalty, deducting reward proportional to the number of detected degenerate substructures, prunes them.

**Composite reward.** Combining the three components, we use:

$$\mathcal{R} = \underbrace{\frac{w}{C}\sum_{i=1}^{C} e^{-r_i/T}}_{\text{Dense Shaping (SAR)}} + \underbrace{\mathbb{I}_{\text{suc}} \cdot R_{\text{bonus}}}_{\text{Sparse Bonus}} - \underbrace{\min(4, C_{\text{deg}})}_{\text{Degeneracy Penalty}} \tag{2}$$

where $\mathbb{I}_{\text{suc}}$ activates when $\|\mathbf{r}\|_2^2 < 10^{-3}$, $C_{\text{deg}}$ counts detected degenerate substructures (capped at 4), and we set $w = 6.0$, $T = 0.1$, and $R_{\text{bonus}} = 4.0$ to bound the reward in $[-4, 10]$ and create a sharp landscape that demands high precision. Hybrid dense and sparse structures have improved performance in general mathematical reasoning (Tao et al., 2025). Our design adapts this pattern to the precision-critical GCS setting. Appendix B illustrates how the reward decreases as constructions deviate from valid geometric configurations (Figure 4).

## 4 The PyGeoX-RL Framework

### 4.1 PyGeoX: A Programmable Geometric Constraint Environment

GCS problems are typically under-constrained and possess an infinite solution space. As seen in Figure 3, the construction remains valid regardless of translation, rotation or scaling, meaning a

direct coordinate-wise comparison with a ground-truth diagram is impossible. To align an LLM with geometric laws, we require an environment capable of rigorously quantifying "correctness" while maintaining the flexibility to express complex design intent. We introduce PyGeoX, a lightweight, object-oriented Python framework for geometric constraint solving.

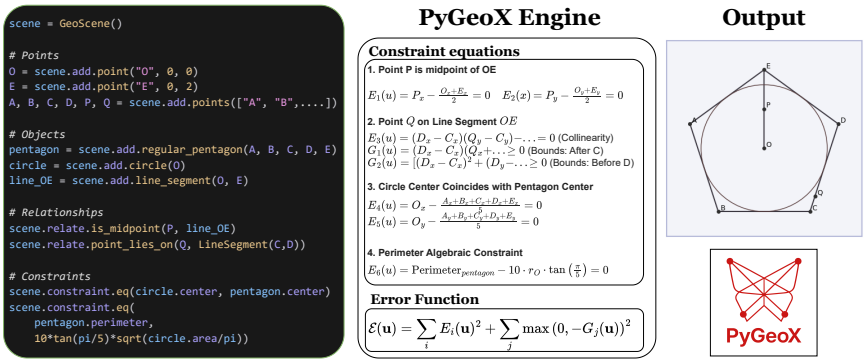



Figure 3: The PyGeoX Symbolic Representation of a geometric figure (a pentagon ABCDE with incircle O and related line segment OE with midpoint P). The PyGeoX engine translates each geometric relationship into a system of symbolic constraints, $E_i = 0$ (equality) or $G_i > 0$ (inequality). These constraints are aggregated into a single error function $\mathcal{E}(\mathbf{u})$, where the input vector $\mathbf{u}$ comprises all geometric variables.

To overcome the limitations of the discrete and opaque engines discussed in Section 2, PyGeoX unifies symbolic expression with numeric feedback by implementing a three-stage Declarative-to-Symbolic-to-Differentiable Pipeline. As illustrated in Figure 3, the engine first captures geometric intent through a high-level declarative DSL (left), which is lazily mapped into a stack of symbolic constraint equations via a SymPy backend (middle, top). These constraints are then aggregated into a single error function $\mathcal{E}(u)$ (middle, bottom), which serves two distinct roles: during data generation, PyGeoX actively optimizes $\mathcal{E}(u)$ to compute valid coordinates and during RL training, the engine evaluates the per-constraint residuals $\mathbf{r} = [r_1, \ldots, r_C]$ extracted from $\mathcal{E}(u)$ and applies the composite reward function (Eq. 2) to provide dense training signal. Crucially, PyGeoX handles both equalities and inequalities as residuals, enabling the reward to capture strict, non-strict, and not-equal constraints. The declarative interface supports 35 geometric objects and 38 relationships, with arbitrary algebraic constraints compiled via Numba's JIT for 10–50$\times$ speedups over Python. Further details on PyGeoX are given in Appendix B.

### 4.2 Data Generation for RL and PyGeoX-Bench

We utilized PyGeoX to construct a synthetic curriculum of 100k geometric problems. To ensure diversity and mitigate the repetitive outputs typical of unguided prompting, the pipeline seeds a Qwen3-30B-A3B model with $n$ objects, $m$ relationships and $k$ extra constraints randomly sampled from a weighted vocabulary of 35 object and 38 relationship types. The LLM expands these seeds into fully defined geometric specifications which are subsequently validated against the PyGeoX numerical solver and any configuration that fails to converge or exhibits degeneracy is automatically discarded. The pipeline, the sample format and the difficulty tiers are detailed in Appendix C.1.

### 4.3 Training Methodology

We evaluate SAR both as an RL reward and as an SFT sample-weighting scheme, using the same base model, Qwen3-8B. All agent outputs are executed within a sandboxed environment equipped with standard scientific libraries (`numpy`, `scipy`, `sympy`) under a 90-second timeout. Failure to produce a valid coordinate dictionary results in a reward of zero. The system prompt (Appendix C.3) provides successful examples illustrating both numerical optimization and constructive algebraic strategies.

**Supervised fine-tuning.** Standard SFT protocols filter data for strict correctness, discarding any solution that fails the final check. We hypothesized that this binary filtering throws away valuable logic contained in partially correct reasoning traces, where the model gets most constraints right

Table 1: Solving rate on the two merged sets, each pooling 450 problems, with 95% Wilson intervals. The $p$ column is an exact paired McNemar test against the SAR+S+D reward of the same training stage, with the base model tested against the RL variant.

| Model | Reward | PyGeoX-Bench 450 | $p$ | PyGeoX-OOD 450 | $p$ |
|---|---|---|---|---|---|
| Qwen3-8B (Base) | none | 0.362 [0.32, 0.41] | <0.0001 | 0.429 [0.38, 0.47] | <0.0001 |
| Qwen3-8B-SFT | SAR | 0.496 [0.45, 0.54] | 0.0140 | 0.429 [0.38, 0.47] | 0.20 |
| Qwen3-8B-SFT | MSE | 0.289 [0.25, 0.33] | <0.0001 | 0.377 [0.33, 0.42] | 0.0003 |
| Qwen3-8B-SFT | Sparse | 0.349 [0.31, 0.39] | <0.0001 | **0.467** [0.42, 0.51] | 0.64 |
| Qwen3-8B-SFT | SAR+S+D | **0.444** [0.40, 0.49] | — | 0.455 [0.41, 0.50] | — |
| Qwen3-8B-SFT | MSE+S+D | 0.169 [0.14, 0.21] | <0.0001 | 0.290 [0.25, 0.33] | <0.0001 |
| Qwen3-8B-RL | SAR | 0.440 [0.39, 0.49] | 0.0035 | 0.545 [0.50, 0.59] | 0.0207 |
| Qwen3-8B-RL | MSE | 0.331 [0.29, 0.38] | <0.0001 | 0.509 [0.46, 0.55] | 0.0004 |
| Qwen3-8B-RL | Sparse | 0.451 [0.41, 0.50] | 0.0088 | 0.516 [0.47, 0.56] | 0.0001 |
| Qwen3-8B-RL | SAR+S+D | **0.509** [0.46, 0.55] | — | **0.605** [0.56, 0.65] | — |
| Qwen3-8B-RL | MSE+S+D | 0.391 [0.35, 0.44] | <0.0001 | 0.540 [0.49, 0.59] | 0.0100 |

but fails on a few. We generated 15,666 reasoning traces on medium-difficulty problems with a Qwen3-32B teacher and trained a Qwen3-8B student using the weighted SFT objective

$$\mathcal{L} = -\sum_i \tilde{\mathcal{R}}_i \sum_j \log p(y_{i,j} \mid y_{i,<j}, \mathbf{x}_i),$$

where each sample is weighted by its normalized reward $\tilde{\mathcal{R}}_i \in [0, 1]$. We adjust the number of epochs across variants so each model sees the same total token volume, and select the best checkpoint by validation loss. We use LoRA ($r$=8, $\alpha$=32, dropout 0.05) with the `ms-swift` (Zhao et al., 2025) framework, learning rate $1 \times 10^{-5}$, cosine decay with 10% warmup, and maximum sequence length 18,196 tokens.

**Reinforcement learning.** We train each reward variant with Group Relative Policy Optimization (GRPO) via `OpenRLHF` (Hu et al., 2025) on 10k medium-difficulty problems drawn from the procedural corpus, generating $G = 8$ rollouts per problem with a maximum generation length of 8,192 tokens. Crucially, all RL runs are cold-started from the base Qwen3-8B model, not from any SFT checkpoint; this isolates the effect of reward design from any SFT initialization bias. To further isolate the impact of reward design, we keep optimization controlled across variants: learning rate $5 \times 10^{-6}$ with cosine decay, KL coefficient 0.01, asymmetric clipping $\epsilon \in [0.2, 0.3]$, and discount factor $\gamma = 1.0$. Batch composition and the remaining hyperparameters are in Appendix C.4.

## 5 Experimental results

### 5.1 Benchmarks and metrics

We evaluate on 900 problems split across two suites. **PyGeoX-Bench** comprises 450 problems, 150 each at Easy, Medium and Hard, drawn from the same procedural pipeline as the training data but disjoint from the problems seen during training. On the other hand, **PyGeoX-OOD** has 450 problems to test whether that progress survives on descriptions written by people for real tasks. It is split into three benchmarks holding 150 problems each with manually verified natural-language and PyGeoX code pairs. **PyGeoX-Wild** adapts items from the public ZhongKaoGeo (Dai et al., 2026), MathVerse (Zhang et al., 2024b) and MathVista (Lu et al., 2024) benchmarks, keeping only the diagram description and discarding the question that followed it. **PyGeoX-CAD** covers detail-drawing geometry such as bolt-hole pitch circles, fillets, and tangent belt runs, phrased the way a drawing is annotated. **PyGeoX-Mech** poses inverse mechanism synthesis, for instance locating a fixed pivot so that a linkage reaches a set of prescribed positions. Appendix C.1 provides more detail.

### 5.2 Comparative analysis

In Table 1 we compare five signals, all evaluated on the same residual vector. *SAR* is the per-constraint reward of Definition 1, $\frac{1}{C}\sum_i \exp(-r_i/0.1)$. *MSE* is its global-norm counterpart, $\exp(-\|\mathbf{r}\|_2^2/10)$ and *Sparse* is the field-standard RLVR signal $\mathbb{I}[\|\mathbf{r}\|_2^2 < \epsilon]$, which equals 1 only when all constraints are satisfied up to tolerance. *S+D* denotes adding the sparse success bonus and the degeneracy penalty

Table 2: Frontier systems evaluated one-shot and inside the agent harness. Confidence intervals in Appendix C.6.

| Setting | Model | Bench (E) | Bench (M) | Bench (H) | Wild | CAD | Mech |
|---|---|---|---|---|---|---|---|
| one-shot | DeepSeek-V3.2 | 0.52 | 0.39 | 0.25 | 0.64 | 0.64 | 0.69 |
| one-shot | Claude-Sonnet-4.5 | 0.53 | 0.57 | 0.37 | 0.65 | 0.77 | 0.90 |
| one-shot | Gemini-3-Flash | 0.55 | 0.54 | 0.38 | 0.77 | 0.73 | 0.98 |
| one-shot | Gemini-3-Pro | 0.61 | 0.49 | 0.51 | 0.81 | — | — |
| one-shot | Claude-Haiku-4.5 | 0.57 | 0.46 | 0.32 | 0.61 | 0.62 | 0.78 |
| +harness | DeepSeek-V3.2 | 0.80 | 0.84 | **0.86** | 0.79 | **0.80** | 0.92 |
| +harness | Gemini-3-Flash | 0.79 | 0.81 | 0.78 | **0.83** | 0.79 | **0.95** |
| +harness | Claude-Haiku-4.5 | 0.79 | 0.79 | 0.72 | 0.70 | 0.74 | 0.89 |

of Eq. 2 to a dense signal. For RL training, the composite SAR reward (SAR+S+D) is the best configuration both in distribution (0.509) and out of distribution (0.605), and is significantly ahead of the base model and every other reward on both sets ($p \leq 0.021$, paired McNemar).

**Headline finding: SAR+S+D beats the field-standard RLVR baselines.** Against *sparse* the margin appears on the hard tier (0.360 vs. 0.260 in RL, $p$=0.017, and 0.287 vs. 0.180 in SFT, $p$=0.009) and on PyGeoX-Wild (0.709 vs. 0.649 in RL, 0.689 vs. 0.628 in SFT). Against *MSE+S+D* the gap is far wider, 0.360 vs. 0.153 on the hard tier in RL and 0.287 vs. 0.047 in SFT. Also, the composite design is essential. Under RL, MSE alone reaches 0.080 on the hard tier and SAR alone 0.200, against 0.260 for sparse alone, but adding S+D lifts those two to 0.153 and 0.360.

**MSE results corroborate the gradient-masking analysis.** Strikingly, MSE+S+D underperforms sparse alone on the hard tier (0.153 vs. 0.260 in RL), exactly as the analysis of Section 3 predicts: replacing SAR's per-constraint kernel with the global-norm objective actively harms training. Two measurements confirm that the cause is signal loss rather than optimization noise. An empirical analysis of 3,893 partially correct solutions finds that 97% of SAR rewards fall in the informative range $[0.1, 0.9]$, whereas 60% of MSE rewards collapse to near-zero, leaving GRPO nothing to rank (Appendix C.5). The same effect appears during training on a second model family, Gemma-3-4B (Appendix C.8).

**Cross-distribution stability.** The ranking holds out of distribution. SAR+S+D leads on PyGeoX-Wild in both SFT (0.689) and RL (0.709), while MSE+S+D collapses there to 0.162 against 0.628 for sparse alone in SFT. On the two engineering suites the trained models still improve substantially over the base model (0.367 against 0.207 on PyGeoX-CAD, $p$=0.0002, and 0.740 against 0.473 on PyGeoX-Mech, $p < 0.0001$, both for RL SAR+S+D), but the reward designs stop separating from one another. These engineering benchmarks test two capabilities at once: translating an engineering requirement into a constraint system, and solving that system. Our training targeted only the second, and the failures concentrate in the first. A representative case is a slider mechanism shown at both ends of its travel, where the model builds the inner dead-centre position as though it were the outer one.

**Token efficiency.** As shown in Figure 2, the SAR+S+D-trained model averages ∼4,060 generated tokens per task, against ∼5,260 for the sparse baseline (a 22.8% reduction).

### 5.3 Frontier-model evaluation

Table 2 calibrates how hard these benchmarks are for systems far larger than 8B. We evaluate five frontier models in two settings. In the *one-shot* setting the model sees the problem once and returns coordinates, which is the protocol used for our own models throughout the paper. In the *harness* setting the model instead works through a staged ladder with access to the PyGeoX verifier: (1) translate the natural-language problem into PyGeoX code, (2) run a *cold solve*, meaning a numerical optimizer started from a random configuration that searches for a minimum of the objective $\mathcal{E}(\mathbf{u})$, (3) reseed that optimizer with a starting configuration the model derives itself after reading which constraints failed, (4) construct and refine coordinates directly over rounds of per-constraint residual feedback, and (5) decompose the constraint set and reintroduce it incrementally. This is the setting deliberately excluded from the task definition in Section 1.1, studied here to measure how much of the difficulty a solver can absorb on its own. Appendix C.9 specifies the ladder, the optimizer and the tools exposed at each stage.

On the hard tier our 8B model (0.360) exceeds DeepSeek-V3.2 (0.25) and sits inside the intervals of Claude-Sonnet-4.5 (0.37) and Gemini-3-Flash (0.38), with only Gemini-3-Pro (0.51) clearly ahead, although these systems differ from ours in scale and inference budget, so we read this as difficulty calibration rather than controlled comparison. The harness then lifts the hard tier from a one-shot best of 0.51 to 0.86, at roughly an order of magnitude more cost per problem. Most informative is which stage produced each answer (Appendix C.9): on PyGeoX-Bench a cold solve handles most solved problems once the formalization is right, but on PyGeoX-CAD it handles a minority, and between 57% and 75% of solutions required the LLM to derive the starting geometry itself, to unstick the optimizer. Geometric reasoning capability therefore pays off inside solver-in-the-loop pipelines and not only in end-to-end generation, which is the capability our reward design trains.

## 6 Discussion and Limitations

A natural question is whether the model actually internalized geometric law or merely memorized training examples. Three lines of evidence support internalization:

1. **Constructive reasoning traces.** About 90% of successful Hard-tier traces follow constructive ruler-and-compass-style derivations, reasoning step-by-step through geometric properties rather than emitting literal lookups or calling a black-box optimizer (Appendix C.2).
2. **OOD transfer.** SAR+S+D obtains the highest PyGeoX-Wild SR in both SFT (0.689) and RL (0.709), despite three-way distribution shift: human-authored prose, unseen constraint types (angle bisectors, equal arc lengths), and novel multi-step reasoning chains. The gain over the base model also holds on the two engineering suites (Section 5.2).
3. **Combinatorial infeasibility of memorization.** A Hard-tier problem draws 3 polygons from $\sim$15 subtypes, $\sim$23 objects, and 16 constraints from 38 relationship types. A conservative lower bound on distinct configurations exceeds $10^{17}$, dwarfing the $\sim$10k RL training problems by 13 orders of magnitude.

Our work has limitations that frame the scope of these claims. The main RL and SFT results use Qwen3-8B as the base model. We replicate the training-signal analysis on a second family, Gemma-3-4B, in Appendix C.8. We also attempted training on Qwen3-1.7B and Llama-3.1-8B-Instruct, but found both base models too weak at math and instruction following for this task, with near-zero base performance across every reward configuration (Appendix C.10). This suggests strong math and code base model capabilities are a prerequisite for GCS. The current engine also targets 2D static geometry only, while 3D GCS requires additional symbolic primitives.

## 7 Conclusion

We presented PyGeoX-RL, a framework that formulates end-to-end Geometric Constraint Solving as an LLM alignment task: natural language to exact metric coordinates verified by per-constraint solver residuals. Emitting coordinates directly, rather than delegating to a DSL or an external solver API, leaves the synthesis unconstrained by a predefined vocabulary. We release the PyGeoX engine and DSL, the four evaluation suites, the agent harness and the data-generation pipeline. Within this infrastructure we identified Outlier Gradient Masking and showed that SAR+S+D outperforms field-standard rewards on hard GCS problems. Beyond GCS, SAR applies to any RLVR setting where a solver returns a multi-dimensional residual over many constraints: physical simulation, scene layout or robotic manipulation.

### AI use statement

Large language models are a core component of the methodology rather than only a writing aid, and every such use is documented in the paper. In this work, we used generative AI tools to generate synthetic datasets: Qwen3-30B-A3B produces the 100k-problem training curriculum and the PyGeoX-Bench problems, and every generated problem is validated by the PyGeoX solver before it is admitted (Section 4.2); Qwen3-32B serves as the SFT teacher (Section 5.2); Qwen3-8B is the trained policy; and frontier LLMs (DeepSeek-V3.2, Claude-Sonnet-4.5, Claude-Haiku-4.5, Gemini-3-Flash, Gemini-3-Pro) are evaluated in Section 5.3 for difficulty calibration only. We have not used generative AI tools to develop the theoretical results, to formulate or prove the mathematical claims, to design the methodology or experiments, or to interpret the results; the remaining tasks with required disclosure

(translation, qualitative or thematic data analysis, survey or interview questions) are not applicable to this work. Additionally, we used generative AI tools to edit the paper for readability (rephrasing and grammar), for LaTeX formatting, to identify relevant literature, to format references, and to assist in writing and refactoring parts of the evaluation and analysis scripts. We have reviewed all AI-assisted work: AI-assisted code was verified and tested by the authors, and every cited work was checked by the authors against its publisher, arXiv or ACL Anthology record, from which the bibliography entries were taken. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### Ethics statement

This work uses only synthetic geometry problems and a publicly available middle-school benchmark adapted with attribution; it involves no human subjects, no crowdsourcing and no scraped personal data. The released artefacts (a synthetic geometry benchmark, an out-of-distribution diagnostic adapted from a published middle-school benchmark, the symbolic-numeric engine, the agent harness and an 8B model specialized to geometric coordinate generation) pose no elevated misuse risk beyond commodity LLMs, and we do not foresee direct societal impacts beyond those generally applicable to LLM fine-tuning research. Third-party assets (the Qwen3 model family under Apache-2.0; the SymPy, NumPy and SciPy stack under permissive BSD-style licenses; the published middle-school geometry benchmark from which PyGeoX-Wild derives) are cited and used in accordance with their licenses. All code and data introduced by this paper will be released under permissive open-source licenses.

### Reproducibility statement

Precise assumptions and complete proofs of Theorems A.1 and A.3 are given in Appendix A. The training and evaluation protocol (base model, optimizer, learning rate, batch composition, KL coefficient, generation budget, reward-function coefficients and compute) is reported in Section 5 and Appendix C.4; the construction of the training corpus and of the four evaluation suites, including the data splits that keep PyGeoX-Bench disjoint from training and PyGeoX-Wild out of distribution, is described in Sections 4.2 and 5.1 and in Appendix C. Every solving rate carries a 95% Wilson interval, and exact paired McNemar tests with Holm correction are reported in Appendix C.7.

## References

Evan Casey, Tianyu Zhang, Shu Ishida, John Roger Thompson, Amir Khasahmadi, Joseph George Lambourne, Pradeep Kumar Jayaraman, and Karl DD Willis. Aligning constraint generation with design intent in parametric cad. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8613–8622. IEEE, 2025.

Jiaqi Chen, Jianheng Tang, Jinghui Qin, Xiaodan Liang, Lingbo Liu, Eric Xing, and Liang Lin. Geoqa: A geometric question answering benchmark towards multimodal numerical reasoning. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 513–523, 2021.

Jiaqi Chen, Tong Li, Jinghui Qin, Pan Lu, Liang Lin, Chongyu Chen, and Xiaodan Liang. Unigeo: Unifying geometry logical reasoning via reformulating mathematical expression. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 3313–3323, 2022.

Shang-Ching Chou, Xiao-Shan Gao, and Jing-Zhong Zhang. An introduction to geometry expert. In *International Conference on Automated Deduction*, pp. 235–239. Springer, 1996.

Weichen Dai, Rafael Medeiros Cabral, Ziyi Shou, Yan Cao, Xin Shen, Dongcai Lu, and Yi Zhou. From symbolic perception to logical deduction: A framework for guiding language models in geometric reasoning, 2026. URL https://arxiv.org/abs/2609.10335.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340. Springer, 2008.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations*, volume 2024, pp. 48362–48395, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Jian Hu, Xibin Wu, Wei Shen, Jason Klein Liu, Zilin Zhu, Weixun Wang, Songlin Jiang, Haoran Wang, Hao Chen, Bin Chen, Weikai Fang, Xianyu, Yu Cao, Haotian Xu, and Yiming Liu. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework, 2025. URL https://arxiv.org/abs/2405.11143.

Jiahao Li, Yusheng Luo, Yunzhong Lou, and Xiangdong Zhou. Recad: Reinforcement learning enhanced parametric cad model generation with vision-language models, 2025a. URL https://arxiv.org/abs/2512.06328.

Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. Cad-llama: leveraging large language models for computer-aided design parametric 3d model generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18563–18573. IEEE, 2025b.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations*, volume 2024, pp. 23439–23554, 2024.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason Keith Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony M. Scopatz. Sympy: Symbolic computing in python. *PeerJ Prepr.*, 4:e2083, 2017. URL https://api.semanticscholar.org/CorpusID:2272000.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, 37(4):1–14, 2018.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017.

Ken Sears and George Allen. Curves and surfaces in unigraphics and parasolid. In Josef Hoschek (ed.), *Freeform Tools in CAD Systems: A Comparison*, pp. 129–145. Vieweg+Teubner Verlag, Wiesbaden, 1991. ISBN 978-3-322-86773-5. doi: 10.1007/978-3-322-86773-5_8. URL https://doi.org/10.1007/978-3-322-86773-5_8.

Aditya Sharma, Aman Dalmia, Mehran Kazemi, Amal Zouaq, and Christopher Pal. GeoCoder: Solving geometry problems by generating modular code through vision-language models. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7355–7371, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.410. URL https://aclanthology.org/2025.findings-naacl.410/.

Leitian Tao, Ilia Kulikov, Swarnadeep Saha, Tianlu Wang, Jing Xu, Sharon Li, Jason E Weston, and Ping Yu. Hybrid reinforcement: When reward is sparse, it's better to be dense, 2025. URL https://arxiv.org/abs/2510.07242.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Viet Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625:476 – 482, 2024. URL https://api.semanticscholar.org/CorpusID:267032902.

Junxiao Wang, Ting Zhang, Heng Yu, Jingdong Wang, and Hua Huang. Magicgeo: Training-free text-guided geometric diagram generation, 2025. URL https://arxiv.org/abs/2502.13855.

Yikun Wang, Yibin Wang, Dianyi Wang, Zimian Peng, Qipeng Guo, Dacheng Tao, and Jiaqi Wang. GeometryZero: Advancing geometry solving via group contrastive policy optimization. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 27948–27963, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1392. URL https://aclanthology.org/2026.findings-acl.1392/.

Jingxuan Wei, Caijun Jia, Xi Bai, Xinglong Xu, Siyuan Li, Linzhuang Sun, Bihui Yu, Conghui He, Lijun Wu, and Cheng Tan. Ggbench: A geometric generative reasoning benchmark for unified multimodal models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5199–5210, 2026.

Jonathan Westhues and SolveSpace Contributors. Solvespace: Parametric 2d/3d cad, 2022. URL https://solvespace.com. Accessed: 2026-01-23.

Liangyu Xu, Yingxiu Zhao, Jingyun Wang, Yingyao Wang, Bu Pi, Chen Wang, Mingliang Zhang, Jihao Gu, Xiang Li, Xiaoyong Zhu, et al. Geosense: Evaluating identification and application of geometric principles in multimodal reasoning. *arXiv preprint arXiv:2504.12597*, 2025.

Xiaolong Yin, Xingyu Lu, Jiahang Shen, Jingzhe Ni, Hailong Li, Ruofeng Tong, Min Tang, and Peng Du. Rlcad: Reinforcement learning training gym for revolution involved cad command sequence generation, 2025. URL https://arxiv.org/abs/2503.18549.

Mingze Yuan, Pengfei Jin, Na Li, and Quanzheng Li. Pirf: Physics-informed reward fine-tuning for diffusion models, 2025. URL https://arxiv.org/abs/2509.20570.

Jiaxin Zhang, Zhong-Zhi Li, Ming-Liang Zhang, Fei Yin, Cheng-Lin Liu, and Yashar Moshfeghi. Geoeval: benchmark for evaluating llms and multi-modal models on geometry problem-solving. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 1258–1276, 2024a.

Ming-Liang Zhang, Fei Yin, and Cheng-Lin Liu. A multi-modal neural geometric solver with textual clauses parsed from diagram. *arXiv preprint arXiv:2302.11097*, 2023.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, et al. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems? In *European Conference on Computer Vision*, pp. 169–186. Springer, 2024b.

Xiaokai Zhang, Na Zhu, Yiming He, Jia Zou, Qike Huang, Xiaoxiao Jin, Yanjun Guo, Chenyang Mao, Yang Li, Zhe Zhu, Dengfeng Yue, Fangzhen Zhu, Yifan Wang, Yiwen Huang, Runan Wang, Cheng Qin, Zhenbing Zeng, Shaorong Xie, Xiangfeng Luo, and Tuo Leng. Formalgeo: An extensible formalized framework for olympiad geometric problem solving, 2024c. URL https://arxiv.org/abs/2310.18021.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, et al. Swift: a scalable lightweight infrastructure for fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 29733–29735, 2025.

Rushan Ziatdinov and James R. Valles. Synthesis of modeling, visualization, and programming in geogebra as an effective approach for teaching and learning stem topics. *Mathematics*, 10(3):398, January 2022. ISSN 2227-7390. doi: 10.3390/math10030398. URL http://dx.doi.org/10.3390/math10030398.

Jia Zou, Xiaokai Zhang, Yiming He, Na Zhu, and Tuo Leng. Fgeo-drl: Deductive reasoning for geometric problems through deep reinforcement learning. *Symmetry*, 16(4):437, 2024.

## APPENDIX OVERVIEW

The appendix has three parts. Appendix A states and proves the two formal results on signal availability and gradient sensitivity that Section 3.2 summarizes informally. Appendix B documents the PyGeoX engine: its object model, the symbolic-to-differentiable compilation pipeline, the global optimizer, and a feature comparison against existing geometry and symbolic frameworks. Appendix C covers everything empirical: benchmark provenance, worked examples from all four suites, the system prompt, reproducibility details, and the additional results referenced from Section 5, including confidence intervals, paired significance tests, the cross-family replication and the agent harness.

## A THEOREM STATEMENTS AND PROOFS

We restate the two formal results summarized informally in Section 3.2 and provide full proofs.

**Theorem A.1** (Vanishing vs. Concentrating Signal Volumes)**.** *Consider a residual space bounded by a hypercube $\Omega = [0, M]^C$, the global-norm reward $\mathcal{R}_G = \phi(\|\mathbf{r}\|_p)$, and the SAR reward $\mathcal{R}_{SAR} = \sum_i \phi(r_i)$, where $\phi : [0, \infty) \to [0, 1]$ is continuous, monotonically decreasing to zero with $\lim_{x\to\infty} \phi'(x) = 0$. Define the effective reward volume $V_\epsilon = \{\mathbf{r} \in \Omega : \mathcal{R}(\mathbf{r}) > \epsilon\}$. Then as the constraint count $C \to \infty$,*

$$\lim_{C\to\infty} \frac{\text{Vol}(V_\epsilon^G)}{\text{Vol}(\Omega)} = 0 \quad \textit{and} \quad \lim_{C\to\infty} \frac{\text{Vol}(V_\epsilon^{SAR})}{\text{Vol}(\Omega)} = 1.$$

*Proof.* **Part 1: Global Norm (Vanishing Volume)** For a Global Norm reward $\mathcal{R}_G = \phi(\|\mathbf{r}\|_p)$ to exceed $\epsilon$, we require $\|\mathbf{r}\|_p < \phi^{-1}(\epsilon) := R_{\max}$. The effective reward region $V_\epsilon^G \subseteq \Omega = [0, M]^C$ is the set of all residual vectors satisfying this constraint. This region is bounded by a $C$-dimensional $\ell_p$-ball of radius $R_{\max}$. The volume ratio can be bounded as:

$$\frac{\text{Vol}(V_\epsilon^G)}{\text{Vol}(\Omega)} \leq \frac{\text{Vol}(C\text{-ball of radius } R_{\max})}{M^C} \propto \left(\frac{R_{\max}}{M}\right)^C$$

where the proportionality constant depends on the dimension $C$ and norm type $p$, but crucially, $R_{\max}$ is a constant determined solely by $\epsilon$ and the kernel function $\phi$. Since $0 < R_{\max} < M$ (assuming the effective reward region is strictly contained in the domain), we have:

$$\lim_{C\to\infty} \left(\frac{R_{\max}}{M}\right)^C = 0$$

**Part 2: SAR (Concentrating Volume)** For SAR, the condition is $\sum_{i=1}^C \phi(r_i) > \epsilon$. Let $X_i = \phi(r_i)$ be a random variable corresponding to the kernel value of a randomly sampled point in $\Omega$. Since $\phi$ is strictly decreasing and bounded $[0, 1]$, and the domain is $[0, M]$, the expected value $\mu = \mathbb{E}[X_i]$ is strictly positive. By the Law of Large Numbers, the sum $\sum_i X_i$ concentrates around $C\mu$. For any fixed threshold $\epsilon$, the condition $\epsilon < C\mu$ is eventually satisfied as $C \to \infty$ since $C\mu$ grows without bound while $\epsilon$ remains fixed. Formally, using Hoeffding's inequality for $C$ bounded i.i.d. variables in $[0, 1]$, the measure of the complement set (where reward $\leq \epsilon$) decays exponentially:

$$\frac{\text{Vol}(\Omega \setminus V_\epsilon^{SAR})}{\text{Vol}(\Omega)} \leq \exp\left(-\frac{2(C\mu - \epsilon)^2}{C}\right)$$

Thus, the relative volume of the reward region approaches 1. □

**Corollary A.2** (Initialization Success)**.** *Under random initialization on $\Omega$, the probability of obtaining a meaningful learning signal ($\mathcal{R} > \epsilon$) decays exponentially to $0$ for global-norm rewards and converges to $1$ for SAR.*

**Theorem A.3** (Robustness to Global Error)**.** *Let $\mathbf{r} \in \mathbb{R}^C$ be the residual vector and $L = \|\mathbf{r}\|_p$ the global $p$-norm ($p \geq 1$). The gradient magnitudes of the global-norm and SAR rewards with respect to a single residual $r_i$ are*

$$\|\nabla_{r_i} \mathcal{R}_G\| = |\phi'(L)| \cdot (r_i/L)^{p-1}, \qquad \|\nabla_{r_i} \mathcal{R}_{SAR}\| = |\phi'(r_i)|.$$

*The global-norm gradient is structurally coupled to the global error $L$, whereas the SAR gradient is strictly local.*

*Proof.* **Part 1 (Global):** Let $L(\mathbf{r}) = (\sum_k r_k^p)^{1/p}$. By the Chain Rule, $\frac{\partial \mathcal{R}_G}{\partial r_i} = f'(L) \cdot \frac{\partial L}{\partial r_i}$. The derivative of the $p$-norm is $\frac{\partial L}{\partial r_i} = \frac{1}{p}(\sum r_k^p)^{\frac{1}{p}-1} \cdot p r_i^{p-1} = L^{1-p} r_i^{p-1} = (\frac{r_i}{L})^{p-1}$. Substituting this back yields $\frac{\partial \mathcal{R}_G}{\partial r_i} = f'(L)(\frac{r_i}{L})^{p-1}$.

**Part 2 (Additive):** We apply the sum rule to $\mathcal{R}_{SAR}$. Since $\mathcal{R}_A = \phi(r_i) + \sum_{j \neq i} \phi(r_j)$, the derivative of the sum of other terms with respect to $r_i$ is zero. The gradient is simply the derivative of the local term: $\phi'(r_i)$. □

**Corollary A.4** (Suppression of Partial Solutions)**.** *When the global configuration is disordered ($L \gg r_i$), whether due to a single divergent outlier ($r_j \to \infty$) or aggregate error from a worsening subset, the global-norm signal for the successful constraint $i$ vanishes, $\|\nabla_{r_i} \mathcal{R}_G\| \to 0$, while the SAR signal $\|\nabla_{r_i} \mathcal{R}_{SAR}\| = |\phi'(r_i)|$ remains non-zero and independent of $L$.*

# B PyGeoX Engine

The PyGeoX engine is architected as a Symbolic-to-Differentiable pipeline that lowers high-level geometric intent into an optimized numerical kernel. This architecture bridges the gap between discrete geometric logic and continuous optimization by maintaining a symbolic intermediate representation throughout the translation process.

## B.1 Object Model and API Structure

PyGeoX represents a geometric scene as a graph $G = (V, E)$, where vertices $V$ are geometric objects and edges $E$ are constraints. The library is organized into three distinct namespaces that facilitate this construction:

1. **Object Instantiation (`scene.add`):** This namespace handles the creation of geometric primitives. The library supports over 30 distinct object types, ranging from fundamental primitives (Points, Rays, Arcs) to a rich hierarchy of polygons (e.g., `RightTrapezoid`, `RegularOctagon`). To ensure logical consistency, PyGeoX utilizes a strict type hierarchy where specific properties are inherited automatically; for example, a `RegularPentagon` inherits from `RegularPolygon`, which in turn inherits from `Polygon`. This structure allows specific properties (e.g., apothem, internal angle) to be exposed automatically while preventing invalid operations on incompatible types.
2. **Geometric Relationships (`scene.relate`):** These methods establish topological dependencies between objects. PyGeoX provides over 30 geometric relationships, covering incidence (e.g., `point_lies_on`, `collinear`), construction (e.g., `is_circumcircle`, `is_orthocenter`), and rigid body transformations (e.g., `rotation_around_point`, `mirror_across_line`). Internally, these high-level semantic relationships are decomposed into their constituent algebraic equations.
3. **Property Constraints (`scene.constraint`):** This namespace provides the interface for defining arbitrary algebraic constraints (equalities and inequalities) on object properties.

A distinguishing feature of PyGeoX is the ability to impose equality or inequality constraints on *any* derived scalar property of a geometric object. While prior geometric reasoning systems such as AlphaGeometry (Trinh et al., 2024) or FormalGeo (Zhang et al., 2024c) typically rely on a fixed set of geometric predicates (e.g., static relationships like `same_length`), they lack the expressivity to handle arbitrary algebraic relationships between different object properties, for instance, to express that "line 1 length is larger than the circle 1 area".

In contrast, PyGeoX allows the definition of constraints that bridge different geometric domains. For instance, a user can enforce a relationship between two entirely different shapes, such as requiring the perimeter of a pentagon to equal the area of a circle (`scene.constraint.eq(pentagon.perimeter, circle.area)`). This flexibility drastically expands the scope of solvable problems beyond standard textbook Euclidean problems.

## B.2 Symbolic Translation and Solving

As illustrated in Figure 3, PyGeoX employs a lazy evaluation strategy, mapping declarative relationships to a symbolic buffer of SymPy equations rather than computing them instantly. This global

view supports a diverse set of algebraic conditions, including strict ($S_k > 0$) and non-strict ($G_j \geq 0$) inequalities, enabling the expression of complex topological constraints.

Prior to evaluation, the symbolic stack undergoes automated simplification. The engine prunes redundant constraints and re-parameterizes primitives (reducing free variables by 20–40%) while performing radical elimination to transform non-linear square roots into smooth polynomial forms. These optimized expressions are aggregated into a differentiable loss $\mathcal{E}(\mathbf{u})$ and compiled via Numba's JIT compiler.

Crucially, this compiled loss serves two distinct roles depending on the pipeline phase:

- Data Generation: The engine actively guarantees geometric validity of the randomly generated diagrams. We optimize $\mathbf{u}$ against $\mathcal{E}(\mathbf{u})$ using Basin-hopping; if the solver fails to converge to near-zero error, the diagram is flagged as potentially contradictory or over-constrained and is automatically discarded.
- RL Training: The engine shifts to a passive role. The problem is extracted as a pair consisting of the natural language description of the diagram and the executable residuals $\{r_i\}$. The RL agent bears full responsibility for GCS, while the engine merely evaluates these residuals to compute the reward signal.

### B.3 Objective Function Compilation

To provide the low-latency feedback required for Reinforcement Learning (RL), the symbolic residual stack is lowered into a scalar error function $\mathcal{E}(\mathbf{u})$ via Numba's Just-In-Time (JIT) compilation. This generates optimized machine code that utilizes efficient NumPy array operations, bypassing Python interpreter overhead and yielding 10–50$\times$ speedups. The objective function aggregates four distinct constraint classes into a single scalar loss:

$$\mathcal{E}(\mathbf{u}) = \alpha \sum_i E_i(\mathbf{u})^2 + \beta \sum_j \max(0, -G_j(\mathbf{u}))^2 + \beta \sum_k \max(0, \epsilon - S_k(\mathbf{u}))^2 + \gamma \sum_\ell \mathbb{1}_{|N_\ell(\mathbf{u})| < \epsilon} \quad (3)$$

where $E$, $G$, $S$, and $N$ represent equality ($= 0$), non-strict inequality ($\geq 0$), strict inequality ($> 0$), and not-equal ($\neq 0$) constraints, respectively. The default weights are $\alpha = \beta = \gamma = 1.0$, with strictness tolerance $\epsilon = 10^{-4}$.

### B.4 Hybrid Global Optimization and Degeneracy Handling

The resulting differentiable landscape is explored using a hybrid global optimization strategy. PyGeoX defaults to Basin-hopping with 1000 iterations, which alternates between stochastic coordinate perturbations (step size 0.5, temperature $T = 2$) and local L-BFGS-B refinement to escape non-convex local minima. Alternative methods include direct L-BFGS-B optimization (for well-conditioned problems) and dual annealing (for highly nonlinear systems).

To ensure structural validity and prevent coordinate collapse, where points converge to a single location to trivially satisfy distance constraints, the solver applies an optional separation penalty:

$$\mathcal{L}_{sep} = \delta \cdot \sum_{i \neq j} \max(0, d_{min}^2 - \|p_i - p_j\|^2) \quad (4)$$

where $d_{min}$ is automatically computed as $1/200$ of the domain span (e.g., $d_{min} = 1.0$ for a $[-50, 50]^2$ workspace) and $\delta = 1.0$ controls penalty strength. This ensures the engine provides the continuous, non-degenerate diagnostic signal required for model alignment.

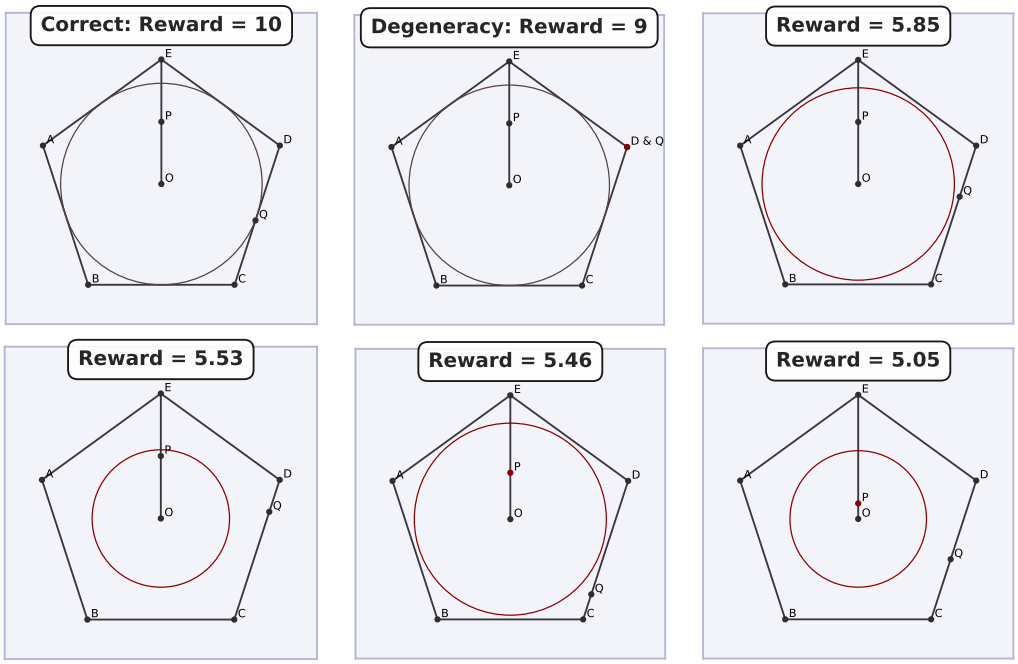



Figure 4: How the reward responds as the construction of Figure 3 degrades. Each panel is annotated with the reward it receives. The first panel is the correct construction and scores the maximum. The second is geometrically valid except that two points have collapsed onto each other, which triggers the degeneracy penalty. The remaining panels break the incircle constraint by increasing amounts, drawn in red. The reward falls smoothly rather than collapsing to zero, which is the behaviour the per-constraint decomposition is designed to preserve.

### B.5 Scope and comparison with other tools

Tables 3, 4, 5 and 6 provide a comprehensive catalog of the PyGeoX DSL. This vocabulary encompasses a wide array of geometric primitives and relational predicates that allow for the declarative specification of scene topology. Additionally, the constraint operators listed in Table 5 enable fine-grained control over metric properties, allowing users to impose precise algebraic or inequality-based conditions on the geometry.

To contextualize the technical positioning of PyGeoX, we compare its capabilities against a spectrum of existing geometric and symbolic frameworks in Table 7, namely AlphaGeometry (Trinh et al., 2024), FormalGeo (Zhang et al., 2024c), SolveSpace (Westhues & SolveSpace Contributors, 2022), GeoGebra (Ziatdinov & Valles, 2022), Sympy (Meurer et al., 2017), Z3 (SMT) (De Moura & Bjørner, 2008) and Geometry Expert (Chou et al., 1996). The current landscape is generally bifurcated into two distinct paradigms. On one side are deductive reasoning engines such as *AlphaGeometry* and *FormalGeo*, which are primarily architected for discrete theorem proving. These systems excel at rigor but function as “binary filters," offering sparse Valid/Invalid feedback, making them ill-suited for geometric synthesis tasks. They typically lack support for inequalities and continuous optimization, limiting their utility in synthesis tasks where finding *a* valid configuration is the priority.

On the other side are numerical and interactive solvers utilized in CAD and education, such as *SolveSpace* and *GeoGebra*. While these tools handle metric constraints and visual rendering effectively, they often rely on “black-box" numerical methods or manual user interaction. As shown in Table 7, they generally lack a differentiable backend, which prevents them from being controlled programmatically by optimization algorithms in a robust manner. Other symbolic tools like *SymPy Geometry* or general-purpose SMT solvers like *Z3* offer strong logical foundations but lack the granular feedback required to guide a solver toward valid geometric manifolds efficiently. As detailed in the comparison table, PyGeoX is unique in its ability to combine declarative input with a differentiable loss landscape, robust inequality handling, and automated degeneracy penalization, features that are typically fragmented across existing deductive and numerical frameworks.

Table 3: PyGeoX Object Library Taxonomy

| Category | Object Types |
|---|---|
| **Basic Primitives** | Point, Line, LineSegment, Ray, Circle, MinorArc, MajorArc, Angle |
| **Polygons** | Polygon, CyclicPolygon |
| **Regular Polygons** | RegularPolygon, RegularPentagon, RegularHexagon, RegularHeptagon, RegularOctagon |
| **Triangles** | Triangle, EquilateralTriangle, IsoscelesTriangle, ScaleneTriangle, RightTriangle, RightIsoscelesTriangle, AcuteTriangle, ObtuseTriangle |
| **Quadrilaterals** | Quadrilateral, Parallelogram, Rectangle, Square, Rhombus, Trapezoid, IsoscelesTrapezoid, RightTrapezoid, Kite, Rhomboid |

Table 4: PyGeoX Relationships

| **Line Relationships** | **Incidence** | **Circle & Arcs** |
|---|---|---|
| `parallel` | `collinear` | `tangent_to_circle` |
| `perpendicular` | `point_lies_on` | `is_chord` |
| `lines_intersect_at` | `points_lie_on` | `line_intersects_circle_at` |
| `line_extensions_intersect_at` | | |
| **Special Lines/Points** | **Polygon-Specific** | **Angles** |
| `perpendicular_bisector_at` | `is_circumcircle` | `acute_angle` |
| `angle_bisector` | `is_incircle` | `right_angle` |
| `is_midpoint` | `is_orthocenter` | `obtuse_angle` |
| `is_radius` | `is_centroid` | |
| `is_diameter` | `is_median` | |
| | `is_altitude` | |
| **Containment** | **Congruence & Similarity** | **Transformations** |
| `inside` | `congruent` | `translation` |
| `outside` | `similar` | `scale` |
| | | `rotation_around_point` |
| | | `mirror_across_line` |

Table 5: PyGeoX Constraint Types

| Category | Constraint | Mathematical Form |
|---|---|---|
| **Equality** | `eq(a, b)` | $a = b$ |
| **Inequality** | `neq(a, b)` | $a \neq b$ (not-equal constraint) |
| | `gt(a, b)` | $a > b$ (strict greater-than) |
| | `lt(a, b)` | $a < b$ (strict less-than) |
| | `geq(a, b)` | $a \geq b$ (greater-than-or-equal) |
| | `leq(a, b)` | $a \leq b$ (less-than-or-equal) |

**Note:** Arguments $a$ and $b$ can be any symbolic expression derived from object properties (e.g., coordinates, distances, angles, areas).

Table 6: PyGeoX Object Properties and Scalar Attributes

| Base Class | Common Properties and Derived Scalars |
|---|---|
| **Point** | $x, y$, `distance(P)`, `distance_squared(P)`, `angle(A, B)` |
| **Linear**<br>*(Line, Segment, Ray)* | `slope`, `intercept`, `direction_vector`, `equation` ($Ax + By + C = 0$)<br>*Segments only:* `length`, `midpoint` |
| **Curvilinear**<br>*(Circle, Arcs)* | `center`, `radius`, `area`, `perimeter` (circumference), `diameter`<br>*Arcs only:* `central_angle`, `length`, `midpoint`, `inscribed_angle` |
| **Polygonal**<br>*(General $n$-gon)* | `points`, `sides`, `num_sides`, `area`, `perimeter`, `centroid`, `internal_angles`<br>*Cyclic:* `circumcenter`, `circumradius` |
| **Regular $n$-gon** | `center`, `circumradius`, `inradius` (apothem), `side_length`, `internal_angle`, `orientation` (rotation in radians) |
| **Triangle** | `circumcenter`, `circumradius`, `incenter`, `inradius`, `orthocenter`, `medians`, `altitudes`, `midsegments`, `angle_bisectors`, `heights` |
| **Quadrilateral**<br>*(Rect, Square, Rhombus)* | `midsegments`, `diagonals`<br>*Specific solvers:* `incenter/radius` (Square/Rhombus), `circumcenter` (Rect) |

| Feature | PyGeoX | AlphaGeometry | FormalGeo | SolveSpace | GeoGebra |
|---|---|---|---|---|---|
| Primary Goal | Synthesis / Opt. | Olympiad Proofs | Theorem Proving | CAD Modeling | Education |
| Logic Type | Neuro-Symbolic | Neuro-Symbolic | Symbolic (Formal) | Numeric | Numeric/Sym |
| Solving Logic | Global Opt. | Deductive Search | Logic Deduction | Newton-Raphson | Dynamic |
| Differentiable | **Yes** | No | No | No | No |
| Inequalities | **Yes** | No | No | Limited | Partial |
| Degeneracy | **Automated** | N/A | Rules | User-Managed | N/A |
| Input Style | Declarative | NLP + Formal | Formal Lang | GUI / Script | GUI |

Table 7: Comparison of PyGeoX against geometry-specific frameworks: deductive theorem-proving systems (AlphaGeometry, FormalGeo) and numerical CAD/educational solvers (SolveSpace, GeoGebra). Comparisons against general-purpose symbolic frameworks are given in Table 8.

| Feature | PyGeoX | SymPy | Z3 (SMT) | Geom. Exp. |
|---|---|---|---|---|
| Primary Goal | Synthesis / Opt. | Symbolic Math | Logic / Algebra | Symbolic Engine |
| Logic Type | Neuro-Symbolic | Symbolic | Symbolic | Symbolic |
| Solving Logic | Global Opt. | N/A | SMT Solver | Symbolic |
| Differentiable | **Yes** | No | No | Partial |
| Inequalities | **Yes** | Yes | Full | Yes |
| Degeneracy | **Automated** | N/A | N/A | N/A |
| Input Style | Declarative | Imperative | Formula | GUI / Sym |

Table 8: Comparison of PyGeoX against general-purpose symbolic frameworks (SymPy, Z3 SMT solver, Geometry Expert). Together with Table 7, this shows that PyGeoX is unique in combining a differentiable backend, native inequality support, and automated degeneracy handling.

# C RL TRAINING DETAILS AND EXPERIMENTAL RESULTS

## C.1 BENCHMARK PROVENANCE

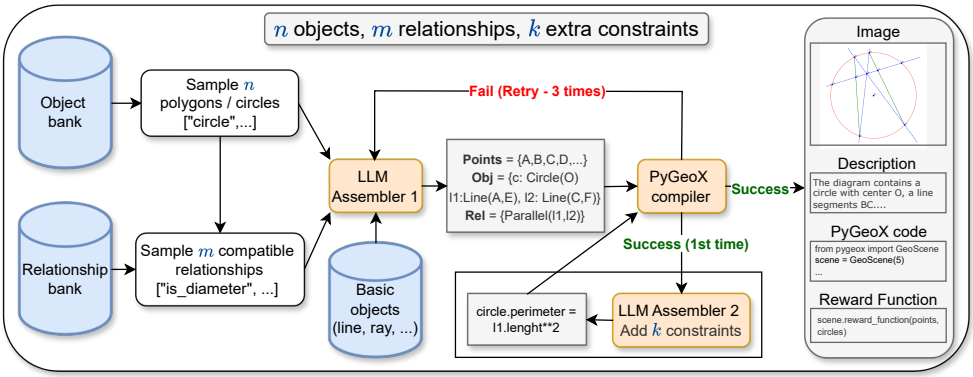



Figure 5: Data generation pipeline.

This appendix documents how each evaluation suite was built, how it relates to the training distribution, and what was checked before a problem was admitted.

The training-data pipeline of Section 4.2 is illustrated in Figure 5. These training samples are stratified into three difficulty tiers: Easy (single-polygon, $\sim$13 objects, 8 constraints), Medium (two-polygon, $\sim$15 objects, 10 constraints), and Hard (three-polygon, $\sim$23 objects, 16 constraints). Examples are found in Appendix C.2. This generation pipeline was also employed to create our automated evaluation benchmark, PyGeoX-Bench, containing 150 problems for each difficulty tier.

**PyGeoX-Bench (450).** Produced by the procedural pipeline of Section 4.2, 150 problems at each of Easy, Medium and Hard, disjoint from the problems used for SFT and RL training. Because the pipeline and the training corpus share a generator, PyGeoX-Bench is best understood as a held-out sample of the training distribution. It is the right instrument for the question the reward ablation asks, which is comparative: holding the base model, the problem distribution and the training budget fixed, which reward drives the policy furthest on the distribution it was trained to optimize. Evaluating on held-out samples from a procedural generator is the established protocol in the reinforcement-learning generalization literature, for the reason given in Section 6: when the space of generable instances is combinatorially vast, solving held-out instances cannot reduce to recalling the training ones.

**PyGeoX-Wild (150).** Human-written problems adapted from middle-school examination diagrams and from the public ZhongKaoGeo (Dai et al., 2026), MathVerse (Zhang et al., 2024b) and MathVista (Lu et al., 2024) benchmarks. We keep only the diagram description and remove the question that followed it, for example "compute angle $ABC$", so that the task becomes construction rather than question answering. The PyGeoX formalization was written afterwards, so the phrasing was never shaped by our DSL. The stylistic gap is visible in the text itself. A typical PyGeoX-Bench item enumerates every object up front and averages 454 characters, occasionally naming objects that carry no constraint at all. The Wild counterpart reads, in full, "in rectangle $ABCD$, $AB$=20, $BC$=10. $P$ is a moving point on $AB$. $DP$ intersects $AC$ at $Q$", against a Wild mean of 138 characters. Comparing the set of DSL primitives each problem's ground-truth code uses, 99.0% of Wild problems exercise a primitive combination that occurs nowhere in PyGeoX-Bench, and 37.4% use at least one primitive PyGeoX-Bench never uses. PyGeoX-Bench also contains no inequality constraints, so the training distribution has no way to express which side of a line a point falls on.

**PyGeoX-CAD and PyGeoX-Mech (150 each).** Hand-authored from parameterized templates, with every problem a manually verified natural-language and PyGeoX pair. Their vocabulary was chosen to avoid classical-geometry terminology. PyGeoX-CAD states tangency through its physical consequence rather than by name. PyGeoX-Mech avoids difficult kinematics vocabulary such as coupler, rocker, Grashof and degrees of freedom, so that a model is scored on geometry rather than on whether it recognizes technical words.

Example problems from every suite, together with reference constructions, are collected in Appendix C.2.

### C.2 EXAMPLE PROBLEMS ACROSS ALL FOUR SUITES

This section collects representative problems from all four evaluation suites. The three PyGeoX-Bench examples below are shown with the full reasoning trace and code generated by Qwen3-8B-RL, so they double as qualitative results: Figure 6 is a successful step-by-step construction on an easy problem, while Figures 7 and 8 are failure cases showing where the model breaks down on the medium and hard tiers. The PyGeoX-Wild, PyGeoX-CAD and PyGeoX-Mech examples that follow show the problem and the reference construction only, to illustrate how each suite differs from the training distribution.

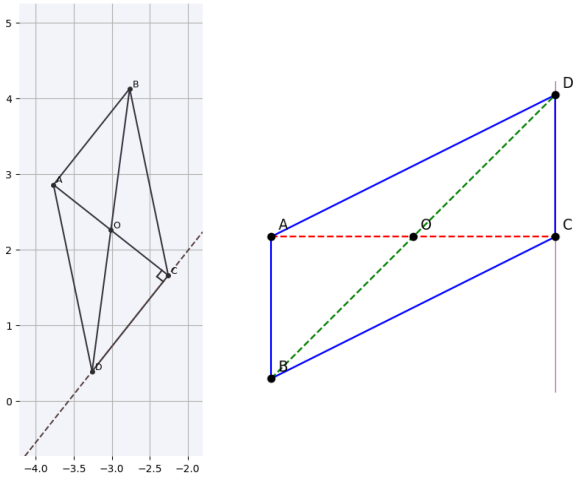



Figure 6: Ground truth diagram image for the easy difficulty example generated by PyGeoX (left) and Qwen3-8B-RL (right). The reward was 10.

**Easy problem example: 1obj_2rel_2extra_gen0067**

**Problem:** The diagram contains points $A, B, C, D, O$. There is an isosceles trapezoid $ABCD$. There is a line segment $AC$. There is a line segment $BD$. There is a line $CD$. Line $AC$ and line $BD$ intersect at point $O$. Line $CD$ is perpendicular to line $AC$.

**<think>**

[...] Let me assume point $O$ is the intersection of the diagonals $AC$ and $BD$. Since $CD$ is perpendicular to $AC$, then the intersection point $O$ lies on both diagonals.

[...] Let me set $AC$ as the x-axis passing through the origin $O$. Let $A = (-a, 0)$ and $C = (a, 0)$.

[...] Since $CD \perp AC$, $CD$ must be vertical. So $D$ is at $(a, b)$.

[...] For $ABCD$ to be an isosceles trapezoid with bases $AB$ and $CD$, $AB$ must also be vertical (parallel to $CD$).

[...] Diagonals of an isosceles trapezoid intersect on the axis of symmetry. Since $O$ is the origin, $B$ must be symmetric to $D$ through the origin.

[...] If $AC$ and $BD$ are diagonals intersecting at the origin, and $AB \parallel CD$, then $B$ must be $(-a, -b)$ to make $BD$ a straight line through the origin.

[...] Let's check leg lengths: $BC^2 = (2a)^2 + b^2$ and $DA^2 = (2a)^2 + b^2$. They are equal. This works.

[...] Assign $a = 1, b = 1$. Points: $A(-1, 0), B(-1, -1), C(1, 0), D(1, 1), O(0, 0)$.

**</think>**

```
import numpy as np

# Define the points with coordinates based on the symmetric
construction
```

```
points = {
    "A": [-1.0, 0.0],
    "B": [-1.0, -1.0],
    "C": [1.0, 0.0],
    "D": [1.0, 1.0],
    "O": [0.0, 0.0]
}

# No specific circles are mentioned in the problem, so circles is
an empty dictionary
circles = {}
```

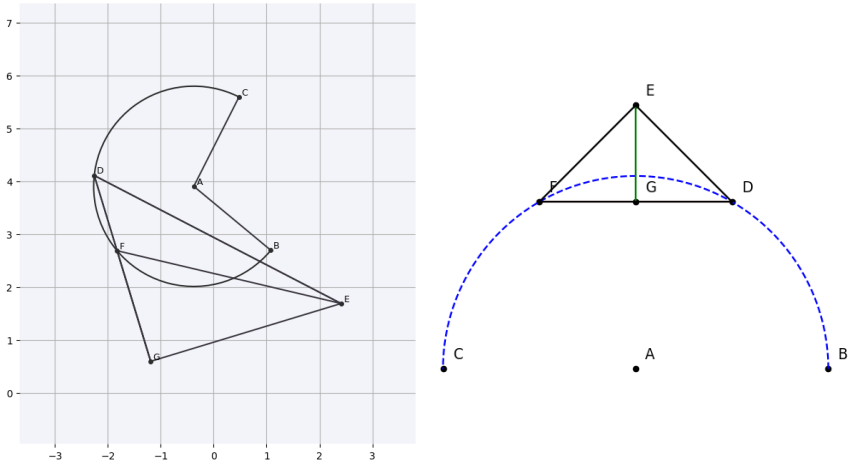



Figure 7: Ground truth diagram image for the medium difficulty example generated by PyGeoX (left) and Qwen3-8B-RL (right). The reward was 3.0.

**Medium problem example: 2obj_4rel_2extra_gen0170**

**Problem:** The diagram contains points $A, B, C, D, E, F, G$. There is a major arc with center $A$ and endpoints $B$ and $C$. There is a quadrilateral with vertices $D, E, F, G$. There is a line segment $DF$. There is a line segment $EG$. There is a line segment $DE$. There is a line segment $FG$. Point $D$ lies on the major arc with center $A$ and endpoints $B$ and $C$. Point $F$ lies on the major arc with center $A$ and endpoints $B$ and $C$. Points $D$, $F$, and $G$ are collinear. Line segment $DF$ is perpendicular to line segment $EG$.

**<think>**

[...] Let me start by placing $A$ at the origin $(0, 0)$ for simplicity. Let's set the radius to 2, so $B$ is at $(2, 0)$ and $C$ is at $(-2, 0)$.

[...] Points $D$ and $F$ are on the major arc. I need $DF$ to be perpendicular to $EG$.

[...] Let me pick $D$ and $F$ such that line $DF$ is horizontal. If $D$ is at $(1, \sqrt{3})$ and $F$ is at $(-1, \sqrt{3})$, both are on the circle $(1^2 + (\sqrt{3})^2 = 4)$.

[...] Line $DF$ is horizontal $(y = \sqrt{3})$. Since $D, F, G$ are collinear, $G$ must be on this line. Let me choose $G$ as the midpoint $(0, \sqrt{3})$.

[...] $EG$ must be perpendicular to $DF$. Since $DF$ is horizontal, $EG$ is vertical.

[...] $E$ must be on the vertical line $x = 0$. Let's place $E$ at $(0, \sqrt{3} + 1)$.

[...] Check: $DF$ slope is 0, $EG$ slope is infinite. They are perpendicular.

**</think>**

```
import numpy as np

# Define points
A = [0.0, 0.0]
B = [2.0, 0.0]
```

```
C = [-2.0, 0.0]
D = [1.0, np.sqrt(3)]
E = [0.0, np.sqrt(3) + 1.0]
F = [-1.0, np.sqrt(3)]
G = [0.0, np.sqrt(3)]

points = {
    "A": A,
    "B": B,
    "C": C,
    "D": D,
    "E": E,
    "F": F,
    "G": G
}

circles = {}
```

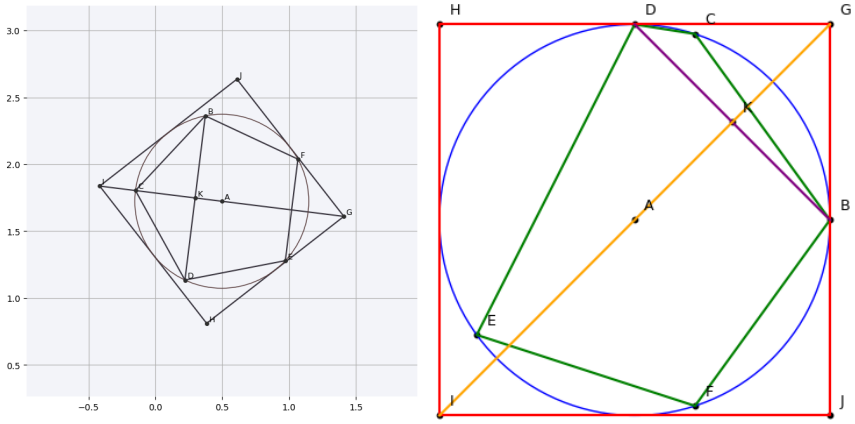



Figure 8: Ground truth diagram image for the hard difficulty example generated by PyGeoX (left) and Qwen3-8B-RL (right). The reward was 4.71.

**Hard problem example: 3obj_5rel_2extra_gen0244**

**Problem:** The diagram contains points $A, B, C, D, E, F, G, H, I, J, K$. There is a circle with center $A$. There is a regular pentagon with vertices $B, C, D, E, F$. There is a square with vertices $G, H, I, J$. There is a line segment $BD$. There is a line segment $GI$. Line $BD$ and line $GI$ intersect at point $K$. Points $B, K, D$ are collinear. Line $BD$ is perpendicular to line $GI$. The circle is the circumcircle of the regular pentagon. The circle is the incircle of the square.

**<think>**

[...] The circle centered at $A$ is the circumcircle of the pentagon and the incircle of the square. Let radius be $r$.

[...] Square vertices for incircle $r$ must be at distance $r\sqrt{2}$. I'll place the square axis-aligned: $G(r, r), H(-r, r), I(-r, -r), J(r, -r)$.

[...] Line $GI$ connects $(r, r)$ and $(-r, -r)$, so its slope is 1.

[...] $BD$ intersects $GI$ at $K$ and is perpendicular. Thus, $BD$ must have slope $-1$.

[...] I place $B$ at $(r, 0)$. For $BD$ to have slope $-1$, $D$ must be at $(0, r)$.

[...] Checking regular pentagon vertices: If $B$ is at $0°$, $D$ should be at $144°$. However, $(0, r)$ is at $90°$.

[...] The regular pentagon condition conflicts with the slope $-1$ condition in this orientation.

[...] Given the constraints, I will proceed with $D(0, r)$ to satisfy the perpendicularity and intersection constraints, even though it deviates from a standard regular pentagon vertex.

[...] Intersection $K$ of $y = x$ and $y = -x + r$ is $(r/2, r/2)$.

**</think>**

```
import numpy as np

# Define the radius
r = 1.0

# Points
A = [0.0, 0.0]

# Square vertices (incircle radius r)
G = [r, r]
H = [-r, r]
I = [-r, -r]
J = [r, -r]

# Pentagon vertices (circumcircle radius r)
# Note: D is placed at (0, r) to satisfy perpendicularity
B = [r, 0.0]
C = [np.cos(np.radians(72)), np.sin(np.radians(72))]
D = [0.0, r] # Adjusted to satisfy slope constraint
E = [np.cos(np.radians(216)), np.sin(np.radians(216))]
F = [np.cos(np.radians(288)), np.sin(np.radians(288))]

# Intersection point K
K = [r/2, r/2]

# Circles
circles = {"A": r}

# Points dictionary
points = {
    "A": A, "B": B, "C": C, "D": D, "E": E, "F": F,
    "G": G, "H": H, "I": I, "J": J, "K": K
}
```

**PyGeoX-Wild, PyGeoX-CAD and PyGeoX-Mech reference problems.** Figure 9 shows one problem from each of the three out-of-distribution suites exactly as the model receives it, together with the reference construction.

### C.3 System Prompt

We provide the full system prompt used to align the LLM for geometric reasoning and Python code generation during the GRPO training. This system prompt is used for all single-shot model evaluations shown in Section 5 and Table 10 and was also used during the RL training.

**System Prompt: The Geometry Architect**

You are an **Expert Computational Geometer and Python Algorithmist**. Your objective is to solve geometry problems by generating high-quality reasoning traces and precision Python code. You operate in two primary modes: **Constructive Algebra** and **Numerical Optimization**.
[...]

**STRATEGY EXAMPLES**

**Example 1: Numerical Optimization (Basinhopping)**
**Problem:** Construct a right-angled triangle with side lengths 3, 4, and hypotenuse 5. Find the coordinates of a point $P$ located strictly inside the triangle that is equidistant from all three sides (the incenter). Provide the coordinates of $P$ and the radius of the circle centered at $P$ that is tangent to the sides.

**<think> Phase A: Construction Strategy**

1. **Analyze:** The goal is to construct a classic **3-4-5 Right Triangle** and locate its **Incenter** P. The incenter is the unique point equidistant from all sides of the triangle.

**PyGeoX-Wild** `problem_64` (source: ZhongKaoGeo)

In isosceles triangle $ABC$, $AC = BC = 10$ and $AB = 12$. $BC$ is the diameter of circle $O$, which intersects $AB$ at $D$ and $AC$ at $G$. $DF \perp AC$ at $F$. $DF$ intersects the extension of $CB$ at $E$.

*Answer:* points $A$, $B$, $C$, $D$, $E$, $F$, $G$, $O$ and circle $O$.

**PyGeoX-CAD** `cadh_t4_cadh_ogee_blend_0021`

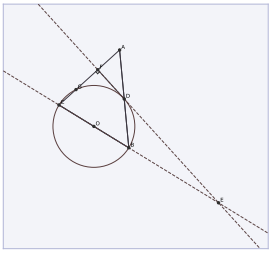
A step in a pressed panel is blended with an S-curve made of two arcs. Set up coordinates with the left-hand end of the lower face at the origin $P$. The lower face runs from $P$ along the positive $x$-direction to $Q$ and is 37.1 mm long. The upper face runs parallel to it 18.4 mm above, from $S$ (directly above $P$) to $R$ (directly above $Q$). The first arc, radius 9.5 mm and centred at $C1$, runs smoothly out of the lower face, leaving it at $T1$ which is 10.6 mm along from $P$. The second arc, radius 7.3 mm and centred at $C2$, runs smoothly into the upper face at $T2$. The two arcs meet each other at $J$, and the curve is smooth there too, with no kink where one arc becomes the other. The S-curve runs left to right, climbing from the lower face to the upper one.

*Answer:* points $C1$, $C2$, $J$, $P$, $Q$, $R$, $S$, $T1$, $T2$ and circles $C1$, $C2$. A stated fact counts as satisfied only when its residual is below $5 \times 10^{-3}$ mm.

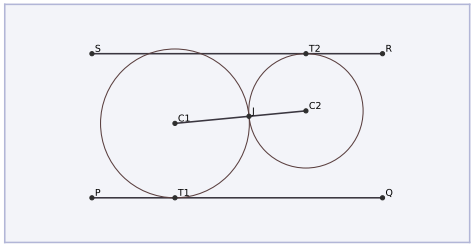
**PyGeoX-Mech** `mechh_t4_mechh_quick_return_extremes_0071`

A quick-return drive has a crank 8.7 mm long turning about a fixed centre $O$, and a long slotted lever pivoted to the bench at $S$, directly below $O$ and 20.7 mm down from it. Set up coordinates with $O$ at the origin. The crank pin slides in the lever's slot, so it always sits somewhere along the lever between the pivot $S$ and the lever's tip, and the lever measures 31.1 mm from $S$ to its tip. As the crank goes round, the lever rocks back and forth between two extreme positions, and it reverses at exactly the moments when the crank has swung as far to one side as it can take the lever, which is when the crank itself lies square to the lever. Show the mechanism at both of those turning points: the crank pin at $P1$ with the lever tip at $T1$ for the extreme where the tip has swung to the right, and the crank pin at $P2$ with the lever tip at $T2$ for the extreme where it has swung to the left.

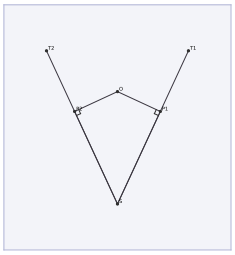
*Answer:* points $O$, $P1$, $P2$, $S$, $T1$, $T2$ and no circles.



(a) PyGeoX-Wild reference




(b) PyGeoX-CAD reference



(c) PyGeoX-Mech reference

Figure 9: One problem from each out-of-distribution suite, verbatim as presented to the model, with the reference construction rendered by PyGeoX. The Wild item is an unmodified ZhongKaoGeo exam problem, adapted only by dropping the trailing question. Neither engineering problem states a coordinate directly. The CAD item never uses the word tangent, so "runs smoothly out of" and "no kink" have to be recognised as tangency conditions, which force the two arc centres to be collinear with the junction $J$. The Mech item requires recognising that the two extremes of travel occur where the crank lies square to the lever, the perpendicularity marked at $P1$ and $P2$, before any position can be computed.

2. **Anchoring:** To simplify the search space and prevent the shape from spinning or drifting, I will anchor vertex B at the origin (0,0). I will constrain C to lie on the positive X-axis and A to lie on the positive Y-axis. This reduces the optimization from 8 variables down to 4.
3. **Derive Constraints:**
   - **Geometry:** The distances AB, BC, and AC must be 3, 4, and 5 respectively.
   - **Incenter Property:** The distance from P to line AB (the Y-axis), line BC (the X-axis), and the hypotenuse AC must be equal.
   - **Containment:** P must strictly reside within the triangle. I will use the cross-product winding method to penalize the solver if P drifts outside the boundary.

**Phase B: Implementation Mapping**

4. **Tooling:** I will use `scipy.optimize.basinhopping` for global optimization and `numba.jit` to ensure the loss function is evaluated at machine speed.
5. **Loss Function:** Squared error of side lengths and variance minimization for equidistance.
6. **Clean Output:** Ensure all NumPy types are cast to native Python lists and floats.

**</think>**

```
import numpy as np
from scipy.optimize import basinhopping
from numba import jit

@jit(nopython=True)
def geometry_loss(x):
    # Anchor B at (0,0), A on Y-axis, C on X-axis
    A = np.array([0.0, x[0]]); B = np.array([0.0, 0.0])
    C = np.array([x[1], 0.0]); P = np.array([x[2], x[3]])
    loss = 0.0
    def d2(p1, p2): return np.sum((p1 - p2)**2)

    # Constraints: Side lengths 3, 4, 5
    loss += (d2(A, B) - 3.0**2)**2 * 100.0
    loss += (d2(B, C) - 4.0**2)**2 * 100.0
    loss += (d2(A, C) - 5.0**2)**2 * 100.0

    # Incenter Property: P equidistant to lines AB, BC, AC
    def dist_line_sq(p, u, v):
        num = ((v[0]-u[0])*(u[1]-p[1]) -
        (v[1]-u[1])*(u[0]-p[0]))**2
        den = d2(u, v) + 1e-12
        return num / den

    d_ab = dist_line_sq(P, A, B)
    d_bc = dist_line_sq(P, B, C)
    d_ac = dist_line_sq(P, A, C)

    # Penalize variance between distances to ensure equidistance
    loss += (d_ab - d_bc)**2 * 50.0 + (d_bc - d_ac)**2 * 50.0

    # Inequality: P must be inside (Winding order check)
    cp1 = (B[0]-A[0])*(P[1]-A[1]) - (B[1]-A[1])*(P[0]-A[0])
    cp2 = (C[0]-B[0])*(P[1]-B[1]) - (C[1]-B[1])*(P[0]-B[0])
    cp3 = (A[0]-C[0])*(P[1]-C[1]) - (A[1]-C[1])*(P[0]-C[0])
    if cp1 > 0: loss += cp1**2 * 1000.0
    if cp2 > 0: loss += cp2**2 * 1000.0
    if cp3 > 0: loss += cp3**2 * 1000.0
    return loss

x0 = np.array([3.0, 4.0, 1.0, 1.0])
res = basinhopping(geometry_loss, x0, niter=100,
minimizer_kwargs={"method": "L-BFGS-B"})
sol = res.x.tolist()
points = {"A": [0.0, sol[0]], "B": [0.0, 0.0], "C": [sol[1], 0.0],
"P": [sol[2], sol[3]]}
```

```
circles = {"P": abs(float(sol[3]))}
```

**Example 2: Constructive Geometry**
**Problem:** Construct an equilateral triangle ABC with side length 6 and its Circumcircle O.

**<think>**

1. **Anchors:** Place A at $(0, h)$, B at $(-3, 0)$, and C at $(3, 0)$.
2. **Height:** For side 6, $h = 6\sqrt{3}/2$.
3. **Circumcenter:** In an equilateral triangle, the circumcenter O is the centroid. $O = (A + B + C)/3$.
4. **Radius:** O is the distance from A to any vertex.

**</think>**

```
import numpy as np
side = 6.0
h = side * np.sqrt(3) / 2.0
B = np.array([-side/2, 0.0]); C = np.array([side/2, 0.0]); A =
np.array([0.0, h])
O = (A + B + C) / 3.0
radius_O = np.linalg.norm(A - O)
points = {"A": A.tolist(), "B": B.tolist(), "C": C.tolist(), "O":
O.tolist()}
circles = {"O": float(radius_O)}
```

**FINAL AUDIT CHECKLIST:**
[...]

### C.4 Reproducibility details

This appendix consolidates compute and hyperparameter information needed to reproduce the main-text experiments.

**Hardware.** All training was performed on a single node of $8\times$ NVIDIA H200 GPUs (141 GB each, Hopper architecture). Inference for evaluation and frontier-model comparisons used standard API endpoints or local inference on the same hardware.

**Base model.** Qwen3-8B for all main-text rows of Table 10. SFT on Llama-3.1-8B-Instruct was also attempted (LoRA, $r$=8, $\alpha$=32) but produced near-zero performance across all reward configurations, reflecting insufficient math and instruction-following capacity (see Section 6).

**SFT hyperparameters (Qwen3-8B).** Training framework: `ms-swift`. LoRA fine-tuning ($r$=8, $\alpha$=32, dropout 0.05) on all linear projections. Learning rate $1 \times 10^{-5}$, cosine decay, 10% warmup, weight decay 0.01, Adam with $(\beta_1, \beta_2) = (0.9, 0.95)$, gradient clipping at 1.0. Effective batch size: gradient accumulation of 32 steps with per-device batch size 2 across 8 GPUs. Maximum sequence length 18,196 tokens, 2 epochs, bf16 precision with FlashAttention-2. Sample weighting via `dataset_weighted` loss scaling. Checkpoints evaluated every 50 steps; the best checkpoint by validation loss was selected. Training-data volumes equalized across reward variants by adjusting the number of epochs.

**RL hyperparameters (Qwen3-8B).** Framework: `OpenRLHF`. Algorithm: GRPO (`group_norm` advantage estimator). Training data: $\sim$10k medium-difficulty problems sampled from the procedural corpus. Learning rate $5 \times 10^{-6}$, cosine decay with minimum LR, 3% warmup. KL coefficient 0.01 (k3 estimator), asymmetric PPO clipping $\epsilon \in [0.2, 0.3]$, discount factor $\gamma = 1.0$, PTX regularization 0.05, Adam with $(\beta_1, \beta_2) = (0.9, 0.95)$, gradient clipping at 1.0. $G = 8$ rollouts per problem, maximum generation length 8,192 tokens, train batch size 144, micro-train batch size 2. Reward clipped to $[-10, 10]$. Inference via vLLM with 4 engines. Composite reward (Eq. 2) with shaping weight $w = 6.0$, temperature $T = 0.1$, success-bonus $R_{\text{bonus}} = 4.0$, degeneracy penalty cap 4, and tolerance $\|\mathbf{r}\|_2^2 < 10^{-3}$. Sandbox execution timeout 90 s with `numpy`, `scipy`, `sympy`.

**Reward variants.** The five reward formulations used in the ablation are: SAR ($\sum_i \exp(-r_i/T)$ with $T$=0.1); MSE ($\exp(-\|\mathbf{r}\|_2^2/T_{\text{mse}})$ with $T_{\text{mse}}$=10, chosen to maximize reward spread on partially correct solutions); Sparse ($\mathbb{I}[\|\mathbf{r}\|_2^2 < 10^{-3}]$); SAR+S+D and MSE+S+D add the sparse success bonus and degeneracy penalty from Eq. 2 to SAR and MSE respectively.

**Seeds and runs.** All runs use seed 42. Each reward variant was trained with a single seed to isolate the effect of reward design. Evaluation is one-shot (single sample per problem) on all four suites.

**Data generation.** The 100k-problem training corpus, the four evaluation suites and the procedural pipeline are all included in the release. Appendix C.1 documents the provenance of each suite.

### C.5 Gradient informativeness analysis

To empirically validate the Outlier Gradient Masking analysis (Section 3), we evaluated all five reward formulations on 3,893 partially correct solutions from the training corpus, i.e., problems where the base model satisfies some but not all constraints. To isolate the dense component, we report the normalized reward from each formulation *without* the sparse bonus or degeneracy penalty, so all values lie in $[0, 1]$. For GRPO, absolute reward values matter less than relative gaps: the algorithm ranks rollouts within each group, so rewards must discriminate between solutions of varying quality. Table 9 reports the fraction of samples in three reward regimes.

Table 9: Reward discrimination on 3,893 partially correct solutions. All values are the normalized dense component only (no sparse bonus or degeneracy penalty), mapped to $[0, 1]$. SAR places 97% of samples in the useful range where relative differences are preserved; MSE collapses 60% to near-zero, making most partially correct solutions indistinguishable to GRPO.

| **Reward** | **Collapsed ∼0 ($< 0.1$)** | **Useful** $[0.1, 0.9]$ | **Collapsed ∼1 ($> 0.9$)** |
|---|---|---|---|
| SAR | 2.9% | **97.1%** | 0.0% |
| SAR+S+D | 4.8% | **95.2%** | 0.0% |
| MSE+S+D | 63.5% | 36.5% | 0.0% |
| MSE | 60.4% | 19.0% | 20.6% |
| Sparse | 100.0% | 0.0% | 0.0% |

SAR's per-constraint decomposition ensures that partially correct solutions receive rewards that reflect the degree of constraint satisfaction, preserving relative gaps that GRPO can exploit for ranking. MSE's global-norm aggregation collapses the majority of these solutions to near-zero, making them indistinguishable and confirming the Outlier Gradient Masking failure mode identified in Section 3.

### C.6 Full results with confidence intervals

Table 10 reports the five-way reward ablation and Table 11 repeats Table 2, each with a 95% Wilson interval on every cell. Each column of Table 10 has $n = 150$.

Table 10: Five-way reward ablation on Qwen3-8B with 95% Wilson intervals.

| **Model** | **Reward** | **Bench (E)** | **Bench (M)** | **Bench (H)** | **PyGeoX-Wild** | **PyGeoX-CAD** | **PyGeoX-Mech** |
|---|---|---|---|---|---|---|---|
| Qwen3-8B (Base) | none | 0.593 [0.51, 0.67] | 0.340 [0.27, 0.42] | 0.153 [0.10, 0.22] | 0.608 [0.53, 0.68] | 0.207 [0.15, 0.28] | 0.473 [0.40, 0.55] |
| Qwen3-8B-SFT | SAR | 0.687 [0.61, 0.76] | 0.487 [0.41, 0.57] | 0.313 [0.24, 0.39] | 0.622 [0.54, 0.70] | 0.187 [0.13, 0.26] | 0.480 [0.40, 0.56] |
| Qwen3-8B-SFT | MSE | 0.460 [0.38, 0.54] | 0.273 [0.21, 0.35] | 0.133 [0.09, 0.20] | 0.405 [0.33, 0.49] | 0.200 [0.14, 0.27] | 0.527 [0.45, 0.60] |
| Qwen3-8B-SFT | Sparse | 0.540 [0.46, 0.62] | 0.327 [0.26, 0.41] | 0.180 [0.13, 0.25] | 0.628 [0.55, 0.70] | 0.253 [0.19, 0.33] | 0.520 [0.44, 0.60] |
| Qwen3-8B-SFT | SAR+S+D | 0.613 [0.53, 0.69] | 0.433 [0.36, 0.51] | 0.287 [0.22, 0.36] | 0.689 [0.61, 0.76] | 0.160 [0.11, 0.23] | 0.520 [0.44, 0.60] |
| Qwen3-8B-SFT | MSE+S+D | 0.273 [0.21, 0.35] | 0.187 [0.13, 0.26] | 0.047 [0.02, 0.09] | 0.162 [0.11, 0.23] | 0.207 [0.15, 0.28] | 0.500 [0.42, 0.58] |
| Qwen3-8B-RL | SAR | 0.667 [0.59, 0.74] | 0.453 [0.38, 0.53] | 0.200 [0.14, 0.27] | 0.514 [0.43, 0.59] | 0.393 [0.32, 0.47] | 0.727 [0.65, 0.79] |
| Qwen3-8B-RL | MSE | 0.547 [0.47, 0.62] | 0.367 [0.29, 0.45] | 0.080 [0.05, 0.13] | 0.446 [0.37, 0.53] | 0.387 [0.31, 0.47] | 0.693 [0.62, 0.76] |
| Qwen3-8B-RL | Sparse | 0.653 [0.57, 0.72] | 0.440 [0.36, 0.52] | 0.260 [0.20, 0.34] | 0.649 [0.57, 0.72] | 0.267 [0.20, 0.34] | 0.633 [0.55, 0.71] |
| Qwen3-8B-RL | SAR+S+D | 0.693 [0.62, 0.76] | 0.473 [0.40, 0.55] | 0.360 [0.29, 0.44] | 0.709 [0.63, 0.78] | 0.367 [0.29, 0.45] | 0.740 [0.66, 0.80] |
| Qwen3-8B-RL | MSE+S+D | 0.633 [0.55, 0.71] | 0.387 [0.31, 0.47] | 0.153 [0.10, 0.22] | 0.581 [0.50, 0.66] | 0.280 [0.21, 0.36] | 0.760 [0.69, 0.82] |

### C.7 Paired significance tests

Table 12 reports exact paired McNemar tests comparing the composite reward against every alternative, in both training regimes and on all six slices. All configurations are evaluated on identical problems, so the paired test is the appropriate instrument. Holm correction is applied within each slice and stage across the five comparisons.

Table 11: Frontier systems evaluated one-shot and inside the agent harness, with 95% Wilson intervals.

| Setting | Model | Bench (E) | Bench (M) | Bench (H) | PyGeoX-Wild | PyGeoX-CAD | PyGeoX-Mech |
|---|---|---|---|---|---|---|---|
| one-shot | DeepSeek-V3.2 | 0.52 [0.42, 0.62] | 0.39 [0.30, 0.49] | 0.25 [0.18, 0.34] | 0.64 [0.53, 0.73] | 0.64 [0.56, 0.71] | 0.69 [0.61, 0.76] |
| one-shot | Claude-Sonnet-4.5 | 0.53 [0.43, 0.62] | 0.57 [0.47, 0.66] | 0.37 [0.28, 0.47] | 0.65 [0.55, 0.74] | 0.77 [0.70, 0.83] | 0.90 [0.84, 0.94] |
| one-shot | Gemini-3-Flash | 0.55 [0.45, 0.64] | 0.54 [0.44, 0.63] | 0.38 [0.29, 0.48] | 0.77 [0.67, 0.84] | 0.73 [0.66, 0.80] | 0.98 [0.94, 0.99] |
| one-shot | Gemini-3-Pro | 0.61 [0.51, 0.70] | 0.49 [0.39, 0.59] | 0.51 [0.41, 0.61] | 0.81 [0.72, 0.88] | — | — |
| one-shot | Claude-Haiku-4.5 | 0.57 [0.47, 0.66] | 0.46 [0.37, 0.56] | 0.32 [0.24, 0.42] | 0.61 [0.50, 0.71] | 0.62 [0.54, 0.69] | 0.78 [0.71, 0.84] |
| +harness | DeepSeek-V3.2 | 0.80 [0.71, 0.87] | 0.84 [0.76, 0.90] | 0.86 [0.78, 0.91] | 0.79 [0.69, 0.86] | 0.80 [0.67, 0.89] | 0.92 [0.81, 0.97] |
| +harness | Gemini-3-Flash | 0.79 [0.67, 0.88] | 0.81 [0.70, 0.89] | 0.78 [0.66, 0.87] | 0.83 [0.73, 0.90] | 0.79 [0.67, 0.88] | 0.95 [0.86, 0.98] |
| +harness | Claude-Haiku-4.5 | 0.79 [0.70, 0.86] | 0.79 [0.70, 0.86] | 0.72 [0.63, 0.80] | 0.70 [0.62, 0.77] | 0.74 [0.66, 0.80] | 0.89 [0.83, 0.93] |

Table 12: Exact paired McNemar $p$-values, SAR+S+D against each alternative, within training stage. * marks the comparisons where SAR+S+D has the lower solving rate. On PyGeoX-CAD and PyGeoX-Mech, where SAR+S+D is not the top point estimate, the difference from the leading configuration is not significant ($p$=0.64 against RL-SAR on CAD and $p$=0.75 against RL-MSE+S+D on Mech).

| vs. | Bench (E) | Bench (M) | Bench (H) | Wild | CAD | Mech |
|---|---|---|---|---|---|---|
| *RL stage, against RL-SAR+S+D* | | | | | | |
| Base | 0.0081 | 0.0017 | <0.0001 | 0.0107 | 0.0002 | <0.0001 |
| SAR | 0.60 | 0.73 | 0.0003 | <0.0001 | 0.64* | 0.88 |
| MSE | 0.0003 | 0.0195 | <0.0001 | <0.0001 | 0.77* | 0.35 |
| Sparse | 0.29 | 0.50 | 0.0167 | 0.14 | 0.0275 | 0.0139 |
| MSE+S+D | 0.18 | 0.0470 | <0.0001 | 0.0026 | 0.0660 | 0.75* |
| *SFT stage, against SFT-SAR+S+D* | | | | | | |
| Base | 0.69 | 0.0243 | 0.0012 | 0.0290 | 0.21* | 0.30 |
| SAR | 0.0433* | 0.13* | 0.61* | 0.0639 | 0.52* | 0.34 |
| MSE | 0.0004 | <0.0001 | 0.0002 | <0.0001 | 0.26* | 1.00* |
| Sparse | 0.0708 | 0.0052 | 0.0090 | 0.11 | 0.0094* | 1.00 |
| MSE+S+D | <0.0001 | <0.0001 | <0.0001 | <0.0001 | 0.19* | 0.68 |

### C.8 Cross-family replication on Gemma-3-4B

To test whether the reward-design effect is tied to one model family, we repeat the comparison on Gemma-3-4B, a model at roughly half the scale and from a different family. Four reward arms are trained with GRPO under identical prompts, hyperparameters and data order: SAR+S+D, PCT+S+D, MSE+S+D and Sparse. PCT is the zero-temperature limit of SAR discussed in Section 3.2, which turns the exponential kernel into a per-constraint indicator and therefore measures the fraction of constraints satisfied. The run uses LoRA of rank 32 on 1,200 easy-tier problems for 200 steps on a single A100, which makes it far cheaper to reproduce than the main H200 experiments.

**Training signal.** The mechanism the paper identifies replicates. GRPO advantages are $A_i = r_i - \text{mean}(r_{1:G})$, so a prompt group whose rewards are all identical contributes no gradient. Measuring the fraction of prompt groups that carry a non-zero advantage, the two per-constraint arms hold 89–96% across training, against 80–84% for the global-norm MSE reward and 38% for Sparse. The ordering is unchanged under a stricter threshold requiring the reward spread to exceed 5% of the arm's range. The two per-constraint arms are also the only ones whose in-training solving rate improves with disjoint first-to-last confidence intervals.

**Downstream solving rate.** At this scale the four rewards are not separable. Every reward produces a large gain over the untrained base model on the training tier, where easy-tier solving rises from 4.7% to between 22.0% and 24.7%, each at $p < 0.0001$. None of that extends beyond the training tier. On medium the trained checkpoints reach 23.3–26.0% against 26.0% for the base model (all $p \geq 0.34$), on hard 0.0–1.3% against 0.7% ($p = 1.00$), on PyGeoX-Wild 6.0–8.7% against 7.3% (all $p \geq 0.75$), and on the two engineering suites every configuration including the base model sits at the floor. The four rewards are statistically indistinguishable from one another everywhere, including on the training tier where the effect is largest (all pairwise $p \geq 0.54$).

**Scope.** We report this as a partial result. Half the parameters, LoRA adaptation rather than full fine-tuning, roughly a tenth of the training problems, and a single tier on which all four rewards already saturate are together enough to explain the absence of separation, and this experiment cannot

distinguish that explanation from a genuine failure to transfer. Running the full corpus at this scale would settle it and is left to future work.

### C.9 AGENT HARNESS

The harness exposes the PyGeoX verifier to an LLM agent through three calls. `describe()` prints a summary of the objects and relationships the agent has built, so it can audit its own formalization. `verify()` returns a per-constraint residual report of a geometric construction. `solve()` runs a cold (random initialization) or warm (using an LLM-suggested diagram) numerical solve. The harness is organized as modular folders, each pairing written instructions with the routines they call, so it can be adopted by other agent frameworks without depending on our evaluation code.

**Strategy ladder.** An agent first formalizes the natural-language problem into PyGeoX. It then escalates only as far as it needs to. Stage 2 is a cold numerical solve from a random start using the basin-hopping solver. Stage 3 is a verifier-informed reseed, where the agent reads which constraints failed and proposes a geometrically derived starting configuration. Stage 4 is manual construction and refinement against per-constraint residual feedback over up to ten rounds. Stage 5 decomposes the problem into subsets of constraints and reintroduces them incrementally. The agent may also extend PyGeoX with custom objects and relationships.

**Which stage produced the answer.** Table 13 reports the share of solved problems that required a model-derived starting configuration, meaning stage 3 or beyond, because a cold solve alone was not enough. Higher values mean the solver alone was insufficient and the geometry had to come from the LLM first to unstick the optimizer.

Table 13: Share of solved problems that required a model-derived starting configuration rather than a cold numerical solve.

| Model | PyGeoX-Bench | PyGeoX-Wild | PyGeoX-CAD | PyGeoX-Mech |
|---|---|---|---|---|
| Claude-Haiku-4.5 | 11% | 26% | 57% | 34% |
| DeepSeek-V3.2 | 25% | 50% | 72% | 48% |
| Gemini-3-Flash | 31% | 60% | 75% | 72% |

**Formalization error.** The failure that survives the ladder is mistranslation. Table 14 reports how often the agent's own verification passes while the oracle disagrees. The rate runs between 3.3% and 22.0%. It is structural rather than numerical: the checker can only test the constraints the agent itself wrote down, so a self-consistent but wrong formalization verifies as correct and no amount of solver feedback detects it.

Table 14: Formalization error rate, defined as the fraction of problems where the agent's own verification passes but the oracle disagrees.

| Model | PyGeoX-Bench | PyGeoX-Wild | PyGeoX-CAD | PyGeoX-Mech |
|---|---|---|---|---|
| Claude-Haiku-4.5 | 21.3% | 20.8% | 22.0% | 9.3% |
| DeepSeek-V3.2 | 16.4% | 15.5% | 20.0% | 8.0% |
| Gemini-3-Flash | 21.0% | 12.2% | 14.0% | 3.3% |

### C.10 TRAINING ON WEAKER BASE MODELS

Table 15 reports the per-tier results behind the claim in Section 6 that strong math and code capability is a prerequisite. Base Llama-3.1-8B-Instruct solves 0.05, 0.03 and 0.00 on Easy, Medium and Hard. SFT with any dense reward variant leaves it at or below that floor at every tier. Sparse-filtered SFT gives a small lift on Easy and Medium at the cost of severe output instability, with roughly a third of its generations failing to produce a code block at all, and Hard remains at or below 0.02 in every configuration.

We manually inspected 60 completions to identify the cause. Output format is not the problem, since roughly 90% produce valid Python with the correct dictionary output. About 12% crash at runtime through division by zero or undefined variables. The remainder execute but satisfy only trivial constraints: coordinates are placed by assumption rather than derived, several points are

Table 15: Llama-3.1-8B-Instruct, solving rate by tier. No reward configuration lifts the model off the floor on the hard tier.

| Configuration | Easy | Medium | Hard |
|---|---|---|---|
| Base | 0.05 | 0.03 | 0.00 |
| SFT SAR+S+D | 0.02 | 0.01 | 0.01 |
| SFT Sparse | 0.08 | 0.05 | 0.02 |

assigned identical coordinates, geometric formulas are fabricated, for instance an inradius given as $(a+b+c-s)/2$, and derivations contradict themselves within a single script. A base model that produces correct solutions with probability near zero explains why no reward design can help, because when every rollout fails, every reward scheme returns its floor and there is no gradient signal for any reward to preserve.