

Machine-Verifiable Item Models for Geometry Diagram Generation

Mei Chen

Renaissance Philanthropy
mei.chen@renphil.org

Gordon Stein

Renaissance Philanthropy
gordon.stein@renphil.org

Abstract

Language models can draw geometry diagrams that look right but are actually wrong. We found that a more reliable way to generate diagrams is to have the model first describe the construction, then have a geometry engine compute every point and check the result against the stated geometric requirements, and only then render the diagram. Because the checklist for each prompt is written once, when the prompt is, and applied by the engine, no one grades a diagram by hand. We tested this on 600 benchmark prompts across six models and found that structured generation raised pass rates for every model that could run it. We also had two human graders check the grader. They showed it is lenient in one direction: it passed 36 of 200 diagrams they failed and failed none the human graders passed.

1 Introduction

Automatic item generation (AIG) can reduce the workload of producing new test items and their answer keys (Bejar, 2002; Gierl and Haladyna, 2013). This paper extends it to items whose response is a geometry diagram, and shows how a machine can check that diagram against the key. In template-based AIG, an author writes a single *item model*, a template from which a computer generates many items, each carrying its own answer key.

Producing a correct diagram is harder than it appears. Consider a simple request:

“Draw an acute triangle ABC with $\angle A = 60^\circ$ and $\angle B = 70^\circ$, then draw the altitude from C , meeting AB at H .”

To be correct, the drawing must satisfy several requirements *exactly*. Angle A is 60° . The point H lies on the segment AB , not past its end. The altitude meets AB at a true right angle. A language model that draws directly has no step at which the drawing is checked against these requirements, so

nothing guarantees them. When they fail, they fail silently.

The same exactness makes such diagrams hard to score. Geometry benchmarks such as Inter-GPS and MathVista pose the reverse task (Lu et al., 2021, 2024). They show the language model a finished diagram and ask a question about it. They do not ask whether a model can *produce* a correct figure. Benchmarks that do score generated pictures, such as T2I-CompBench++ and GenExam, rely on a model-based judge that inspects the output (Huang et al., 2025; Wang et al., 2026). A judge can report that a shape *resembles* a right triangle. It cannot certify that an angle is exactly 90° . For geometry diagrams, then, neither producing the item nor scoring it is straightforward.

Our approach replaces guessing with computation and makes the stated requirements machine-checkable. The language model describes the construction in words: “ H is the foot of the perpendicular from C to AB .” A geometry engine then computes the coordinates. The model decides *what* to build. The engine determines *where* every point lies. This single change addresses both problems.

- **Better drawings.** A solver performs the computation, so a right angle is exactly right rather than approximately so.
- **Automatic scoring.** The finished figure is a set of coordinates, so each encoded requirement can be checked directly. The question “is this angle 90° ?” replaces a judge’s impression of the picture.

That is this paper’s contribution: a *machine-verifiable item-model framework* for geometry diagrams. One item model emits both the prompt and a *computed* answer key. The key is a machine-checkable list of geometric requirements. The figure produced in response is verified against that key. We make two empirical claims and are delib-

erate about their strength. Structured construction improves how often models get the geometry right (Section 4). Human validation shows the automatic score is optimistic, so it is not yet a validated high-stakes score (Section 3). Here the benchmark is evidence for these two claims, not the contribution itself.

The rest of the paper explains how the language model describes a diagram (Section 2), how scoring follows from that same description (Section 3), what we found on six large models (Section 4), and why this matters for educational measurement (Section 5).

2 How the model draws a diagram

The model specifies geometric relationships, and a geometry engine computes the coordinates. A renderer then produces the diagram (Figure 1). The division of labour is the central idea: **the model decides what to build, and the engine decides where the points lie**. The model writes no coordinate for any constructed point.

Step 1: a description, not coordinates. The model fills in a simple, typed form. We call it the intermediate representation, or IR. The form offers 31 kinds of building block: points, segments, lines, circles, ellipses, polygons, and intersections. Each block names a *relationship* rather than a position. A point can be “the midpoint of A and B ,” “where these two lines cross,” or “the foot of the perpendicular from C .” In our example the model writes “ H is where line AB meets the perpendicular to AB through C .” It never writes “ $H = (3.07, 0)$.” Because the form is typed, structural mistakes are caught immediately, before any geometry is computed. A missing point or a wrong number of arguments never reaches the engine. A free point, one that nothing constrains, such as A or B in the example, is placed by the engine at random within the canvas, or at a rough position the model may suggest.

Step 2: the engine solves for the points. A geometry engine reads the description and computes the coordinates, handling the building blocks in dependency order. We use SymPy, a symbolic-mathematics library. Symbolic methods have recently solved olympiad geometry problems (Trinh et al., 2024). Here the engine has a more modest task. It determines where the points of a described figure lie. When a step is ambiguous, for example

when two circles meet at two points, the description states which one to keep. The engine, not the model, produces every coordinate. That is what makes the right angle in our example exactly 90° rather than approximately so.

Step 3: render. From the exact coordinates the system writes standard drawing code and compiles it to an image. Details an author should not have to manage, such as clipping a line at the edge of the canvas, are handled automatically.

Two ways to drive it. The model can fill in the form directly. It can also use a short “recipe” language that names a construction, such as `perp_bisector(A, B)`, which then expands into the same form. Every result in this paper comes from the direct form. The recipe language was not yet mature when the benchmark was run, and we have not compared the two at scale. Either way, the description can carry its own requirements, tested *before* the figure is drawn, so the model can catch a mistake and retry. We compare the direct form, which we call the *structured mode*, against a *by-hand mode* in which the model writes drawing code directly, with no engine and no checks. That baseline is the reference against which we measure the structured mode (Section 4). Having a model write drawing code directly is itself an active line of work (Belouadi et al., 2024a,b). That work asks how closely the output resembles a target picture. We ask whether its geometry is correct.

3 Automatic scoring from shared construction logic

Each item model writes two things at once: the prompt shown to the language model and the checklist used to score the diagram (Figure 2). Because the finished diagram is a set of coordinates, the engine can decide each check directly. “Is the angle at H a right angle?” is settled by a single calculation on the coordinates. No judge is asked whether the picture appears square. This makes scoring inspectable and inexpensive. It does not make scoring complete. The checklist’s validity still depends on whether it captures everything the prompt asks for and the visual conventions a reader expects. We examine both limits below.

The question and its answer key are written together. Each item and its answer key come from the *same* short program. Because the two are emitted together, they are far less likely to drift

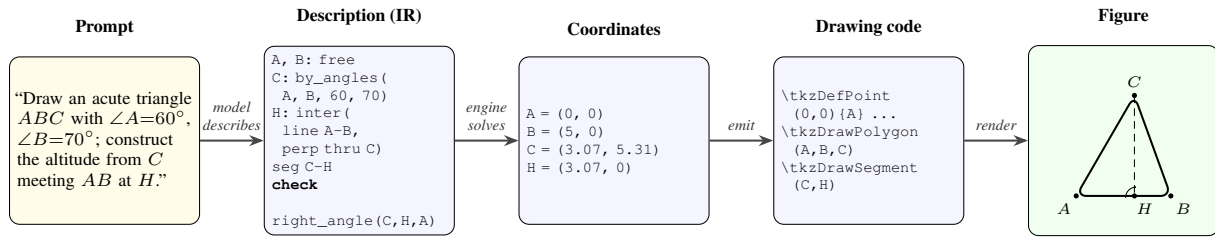


Figure 1: One request, start to finish. The model writes a description of the construction. The geometry engine solves it for exact coordinates. The system emits drawing code and renders it. The model writes relationships (“ H is the foot of the perpendicular from C ”) rather than numbers, so the geometry is computed rather than guessed. Those same coordinates are what we score against in Section 3.

Check	What it confirms
right_angle	an angle is a right angle
midpoint	a point is halfway along a segment
collinear / parallel	points lie on a line / lines never meet
point_on_segment	a point lies between two endpoints
equal_lengths	two segments have equal length
angle_equal	an angle equals a stated value

Table 1: A sample of the 15 checks in the scoring vocabulary. Each is decided on the computed coordinates. An item’s list of checks is what a passing answer must satisfy.

apart through manual editing. Errors in the item model itself remain possible. An author can still omit a predicate, encode the wrong one, or write wording whose intended constraint the checklist does not fully capture. Re-running the item model with a new random seed produces a fresh, parallel item. The checklist is never shown to the language model while it works.

A small, readable checklist. Each encoded requirement uses a small, fixed vocabulary of checks (Table 1): right angle, midpoint, parallel lines, point on segment, and so on. There are 15 such checks in the current version. The construction is computed symbolically; each check is then decided numerically on the coordinates, within a small tolerance (Section 4). A diagram passes only if every check on its list passes. The list defines what the automatic scorer evaluates. It plays the role a specification plays in formal methods: a statement of what must hold, written so that a machine can check it. The check here is numeric and applied to one figure at a time, not a proof over all figures. For that reason we say *verifiable* rather than *verified*.

Does the automatic score match a human’s?

An automatic score is useful only if it agrees with a careful human grader, so we tested that agreement. We fixed a few blind spots found on an early pilot. We then ran a pre-registered study on a later run of the benchmark.¹ We set the bar in advance: the automatic score had to match the human verdict at Cohen’s $\kappa \geq 0.70$, on a scale where 1.0 is perfect agreement and 0 is chance. Two graduate-level graders then scored 200 diagrams by hand, without seeing the automatic verdict. The 200 were drawn from that run stratified by automatic verdict: every uncertain and every rendered failing diagram, plus passes balanced across models, modes, and difficulty tiers. Each grader rated independently, and disagreements were then discussed to a single consensus verdict per diagram. We compare the automatic verdict with that consensus, both collapsed to pass or fail, with a partial human verdict counted as a fail.

We did not clear that bar. Agreement was moderate, Cohen’s $\kappa = 0.46$, well short of the pre-registered 0.70. We treat this as a central limitation. The disagreements run *one way*. In 36 of the 200 cases the automatic scorer passed a diagram the graders failed. No automatic failure received a passing human verdict. That is the more serious direction once a score is read as a certificate of correctness, because every disagreement is a false positive. We therefore state it plainly: **the pass rates in Section 4 may overestimate correctness**. Inspecting the 36 disagreements traced them to the checklist rather than to the ratings. They concentrate in items with an added construction, like our altitude, whose checklist confirms the construction but does not re-check the prompt’s stated angles.

¹Protocol: docs/human_study_protocol.md in the code repository, <https://github.com/RenaissancePhilanthropy/geometry-diagram-generator>.

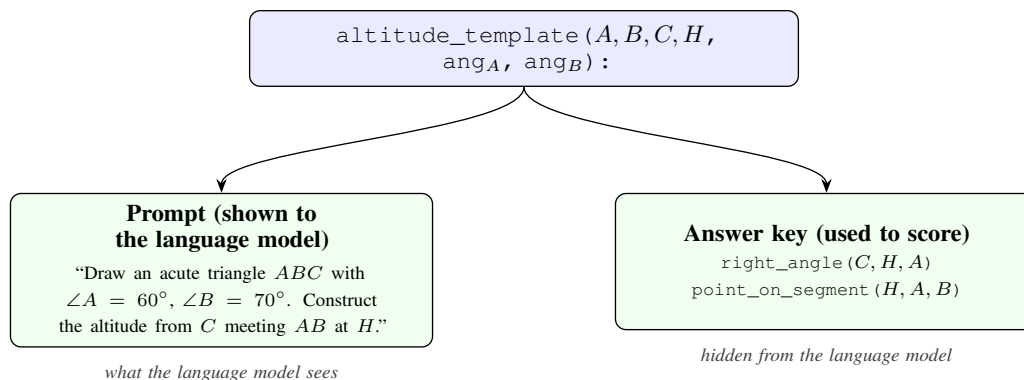


Figure 2: One item model writes both the question and its answer key. The two cannot drift apart through separate editing. An item model can still leave a check out. This key confirms the altitude but not the two stated angles. That gap is the main finding of the human study reported in this section.

The two graders agreed with *each other* at the 0.70 we had asked the scorer to reach. The fix is to add the missing numeric checks to those item models. That revision was not applied to the results reported here. Instead, Section 4 reports, alongside each automatic pass rate, the share of automatic passes the graders confirmed, as a sensitivity calculation.

What the checks cannot see. That particular gap closes with more checks. A deeper one does not. The checks read the geometry, the coordinates, and nothing else. They never look at how the figure is drawn. A diagram can satisfy every requirement and still mislead a reader. A right-angle mark can be placed outside the corner. An angle can be labelled on the wrong side. A figure can pass every check and still be laid out so poorly that a reader cannot follow it. This is the converse of the language model-based judge in Section 1. A judge sees how a picture looks but cannot check its geometry. Our checks test the geometry but cannot see how it looks. The two are complementary. The checks say whether the computed geometry is correct. Whether the drawing presents that geometry faithfully and clearly still requires visual review.

4 Benchmark results

Across the 600 item-model prompts of GEOGEN-BENCH, the structured mode raised automatic pass rates for every model that could run both modes (Table 2). GEOGENBENCH holds 801 prompts, all scored by the checker of Section 3: 600 generated from the item models described above, and 201 written from the topic scaffold of a K–12 mathematics curriculum (Carnegie Learning, Inc. and Texas Education Agency, 2024). The results be-

low cover the 600. The prompt set, item models, checker vocabulary, and rendered outputs are public, and so is the code.² We ran six large models from three companies, each in two modes. The *by-hand* mode has the language model write drawing code directly. The *structured* mode has it fill in the description of Section 2. Each prompt was run three or more times, so a cell in Table 2 holds 1,800 to 2,861 generations, and a rate is the share of those generations that passed the automatic checker. The checker overestimated human acceptance in the validation sample (Section 3), so these rates may overestimate correctness.

How we ran it. The by-hand mode asks the language model for TikZ drawing code. The structured mode asks for the typed form of Section 2. For the by-hand mode, an extractor recovers the coordinates from the TikZ source; output the extractor cannot parse counts as a failure. Both run at temperature 1.0, the sampling setting that controls randomness, with no explicit limit on output length. Both have the same budget of up to three attempts. What differs is the retry trigger. The by-hand mode retries only when its code fails to render. The structured mode also retries when its description fails to compile or the figure breaks a check it declared. Those declared checks are the language model’s own. The held-out scoring predicates generated by the item model, the ones we grade against, are never shown to the language model. The comparison is therefore between two complete workflows. The structured mode’s feedback lets a model catch its own geometric mistakes,

²Data: <https://huggingface.co/datasets/meix/geogenbench>. Code: the repository in footnote 1.

Model	By hand	Structured	\$/item
GPT-5.5	91.9%	94.1%	0.075
Claude Sonnet 4.6	81.2%	93.4%	0.088
Claude Opus 4.7	71.6%	93.3%	0.53
Claude Haiku 4.5	70.5%	75.3%	0.032
Gemini 2.5 Pro	58.4%	83.1%	0.050
Gemini 2.5 Flash	35.9%	blocked	—
<i>Confirmed by graders</i>	75%	87%	—

Table 2: Share of generations *passing the current automatic checker* over the 600 item-model prompts, each run three or more times, in the by-hand mode and the structured mode, with the average cost of one structured generation in US dollars. The checker overestimated human acceptance in the validation sample (Section 3), so these rates may overestimate correctness. The last row gives the share of automatic passes that human graders confirmed, pooled over models. Every model that could run the structured mode improves. The three strongest converge near 93–94% despite a sevenfold difference in cost. Gemini 2.5 Flash could not run the structured mode (see text). Bold marks the highest rate.

which the by-hand mode cannot, and its extra retries appear in its per-item cost. Each requirement is decided on the computed coordinates within an absolute tolerance of $\tau = 5 \times 10^{-3}$, applied to angles in radians and to lengths in coordinates normalised to $[-10, 10]$. The 600 item-model prompts divide evenly across three difficulty tiers of 200 each. They come from 30 item models covering triangles, quadrilaterals, circles, segments, and lines, with 3.2 requirements apiece on average. The 201 curriculum prompts were drafted with a language model, with answer keys written by hand. They carry more requirements, 6.0 on average. Cost per item is the input and output tokens of the prompt and its response, summed across any retries and priced at each vendor’s published rates, then averaged over every prompt, failures included. Failures are dominated by wrong geometry, a requirement that does not hold, rather than by broken drawing code. The structured mode’s remaining failures are mostly timeouts.

The structured mode helped every model that could run it (Table 2). Pass rates rose by 2 to 25 percentage points. The largest gain, 25 points, went to Gemini 2.5 Pro, the weakest by-hand drawer among the models that could run both modes. In the structured mode, the three strongest models finished within a single point of one another, despite a sevenfold difference in price.

The human study of Section 3 supports a sensitivity calculation. In that study the checker erred

in one direction only. The graders rated diagrams from a pilot run of the same benchmark. They confirmed 75% of the automatic passes drawn by hand (86 of 114) and 87% of those from the structured mode (55 of 63). If those proportions carry over to the full run, the structured rates in Table 2 fall to about seven-eighths of those shown and the by-hand rates to about three-quarters. Under that assumption the structured advantage survives and widens, which would mean the structured mode also reduces errors the checker misses. The per-model cells in that study hold 7 to 23 diagrams, too few to correct each row separately.

One practical limitation concerns vendor infrastructure. Gemini 2.5 Flash could not run the structured mode. The API rejected our output schema, with its 31 kinds of building block, as too complex before generation began, with the message that the schema “produces a constraint that has too many states for serving.” The limit is in the vendor’s constrained-decoding service, not in the model. Gemini 2.5 Pro and GPT-5.5 accepted the same schema. The results support a single point: the structured mode helps at scale, across every model that could run it.

5 Why this matters for measurement

An item model can generate related prompts and their scoring checklists together (Figure 3). Whether those variants have comparable difficulty is a calibration question we return to below.

Fixing every parameter but one and sweeping that parameter, for example an angle through $30^\circ, 40^\circ, 50^\circ$, produces a family of structurally parallel questions, and the checker confirms each variant against its encoded checks without human review. The answer key is produced *with* each item rather than written afterward. A common source of scoring error, a hand-written key that has drifted from its question, is therefore largely designed out. This is the item model of the measurement literature (Gierl and Lai, 2012), here one that emits its own key. This design also shifts where quality control happens, in the way the item-generation literature recommends. Rather than vetting every generated item, one reviews the construction logic that produces them all (Gierl and Lai, 2016), which complements rather than replaces checking the generated items.

The diagram a language model produces is a constructed response, and the same description that

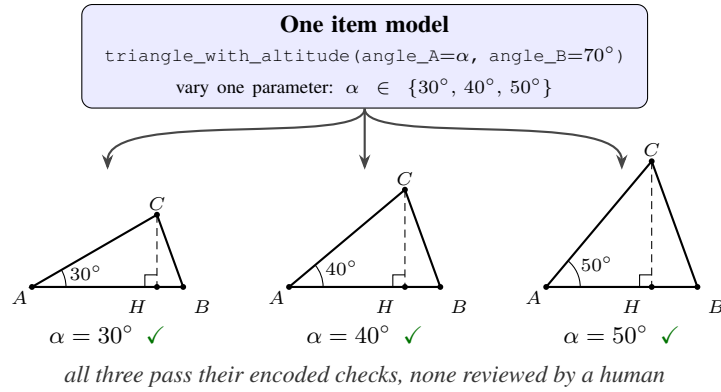


Figure 3: One item model yields a family of items that pass their encoded checks. This is automatic item generation with the answer key built in. Hold every parameter fixed but the angle at A and sweep it through $30^\circ, 40^\circ, 50^\circ$. That produces parallel variants. For each, the checker of Section 3 confirms that the altitude from C meets AB at a right angle (✓), with no human review. One item model yields a refreshable supply of items, each carrying its own key.

built the item scores it. Scoring a student’s drawing would first require a format the engine can read, which we have not built. Here a measurement audience should consider two claims separately. The *generation* claim rests on firmer ground. The key is computed with the item rather than written separately, so question and key are far less likely to drift apart. The key is also a short, readable list of exactly what must be true (Table 1), open to inspection rather than embedded in a holistic rubric. The *scoring* claim must be established empirically, and our validity study (Section 3) shows that it is not yet established. The automatic score agrees with human graders only moderately and errs in a single direction. For now it is an optimistic estimate. That is usable for benchmarking, with the sensitivity calculation of Section 4, and for flagging likely-wrong work. It is not yet fit for standalone high-stakes scoring. The next step is to add the numeric checks that Section 3 pinpoints and rerun the validation.

Two measurement questions the checklist opens.

First, isomorphism. Variants generated from one item model are parallel in construction. Parallel construction does not guarantee equal difficulty. A 30° angle may make the problem more difficult than a 50° one. Whether the variants are true isomorphs is a calibration question that only field testing with students can answer. That is future work, and the framework makes it inexpensive, because every variant comes with its key. Second, partial credit. The checker decides every requirement separately, so a score need not be all-or-nothing. A diagram that places the altitude correctly but misses the stated angle can earn credit for what it got right.

The per-requirement verdicts provide the basis for a partial-credit rubric. The weights and score categories remain to be decided. Our graders also rated diagrams requirement by requirement. They agreed with the checker on 98% of 864 individual ratings. The overall disagreement in Section 3 comes largely from requirements the checklist left out, not from the ones it contains.

Scope. The version described here covers plane geometry. The same machinery should extend to coordinate graphs and three-dimensional solids by adding new kinds of building block. We have not built those yet. Where no item model fits, an author can still write the item and its checklist by hand and use the same checker. The lesson of the paper is short. When a language model describes a construction instead of drawing it, a machine can build the diagram, check it, and score it. Human validation still decides whether the checks are the right ones. With both in place, geometry items can be generated and scored at scale.

References

- Isaac I. Bejar. 2002. Generative testing: From conception to implementation. In Sidney H. Irvine and Patrick C. Kyllonen, editors, *Item Generation for Test Development*, pages 199–217. Lawrence Erlbaum Associates.
- Jonas Belouadi, Anne Lauscher, and Steffen Eger. 2024a. AutomaTikZ: Text-guided synthesis of scientific vector graphics with TikZ. In *Proc. Int. Conf. on Learning Representations (ICLR)*.
- Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. 2024b. DeTikZify: Synthesizing graphics programs for scientific figures and sketches with TikZ.

- In *Proc. Conf. on Neural Information Processing Systems (NeurIPS)*.
- Carnegie Learning, Inc. and Texas Education Agency. 2024. Bluebonnet learning mathematics: Teacher’s edition (grades 6–8 and geometry). Texas Education Agency, <https://learn.texas.gov/>.
- Mark J. Gierl and Thomas M. Haladyna, editors. 2013. *Automatic Item Generation: Theory and Practice*. Routledge.
- Mark J. Gierl and Hollis Lai. 2012. The role of item models in automatic item generation. *International Journal of Testing*, 12(3):273–298.
- Mark J. Gierl and Hollis Lai. 2016. [A process for reviewing and evaluating generated test items](#). *Educational Measurement: Issues and Practice*, 35(4):6–20.
- Kaiyi Huang, Chengqi Duan, Kaiyue Sun, Enze Xie, Zhenguo Li, and Xihui Liu. 2025. T2I-CompBench++: An enhanced and comprehensive benchmark for compositional text-to-image generation. *IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI)*.
- Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2024. MathVista: Evaluating mathematical reasoning of foundation models in visual contexts. In *Proc. Int. Conf. on Learning Representations (ICLR)*.
- Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. 2021. Inter-GPS: Interpretable geometry problem solving with formal language and symbolic reasoning. In *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482.
- Zhaokai Wang, Penghao Yin, Xiangyu Zhao, Changyao Tian, Yu Qiao, Wenhai Wang, Jifeng Dai, and Gen Luo. 2026. GenExam: A multidisciplinary text-to-image exam. In *Proc. Int. Conf. on Machine Learning (ICML)*.