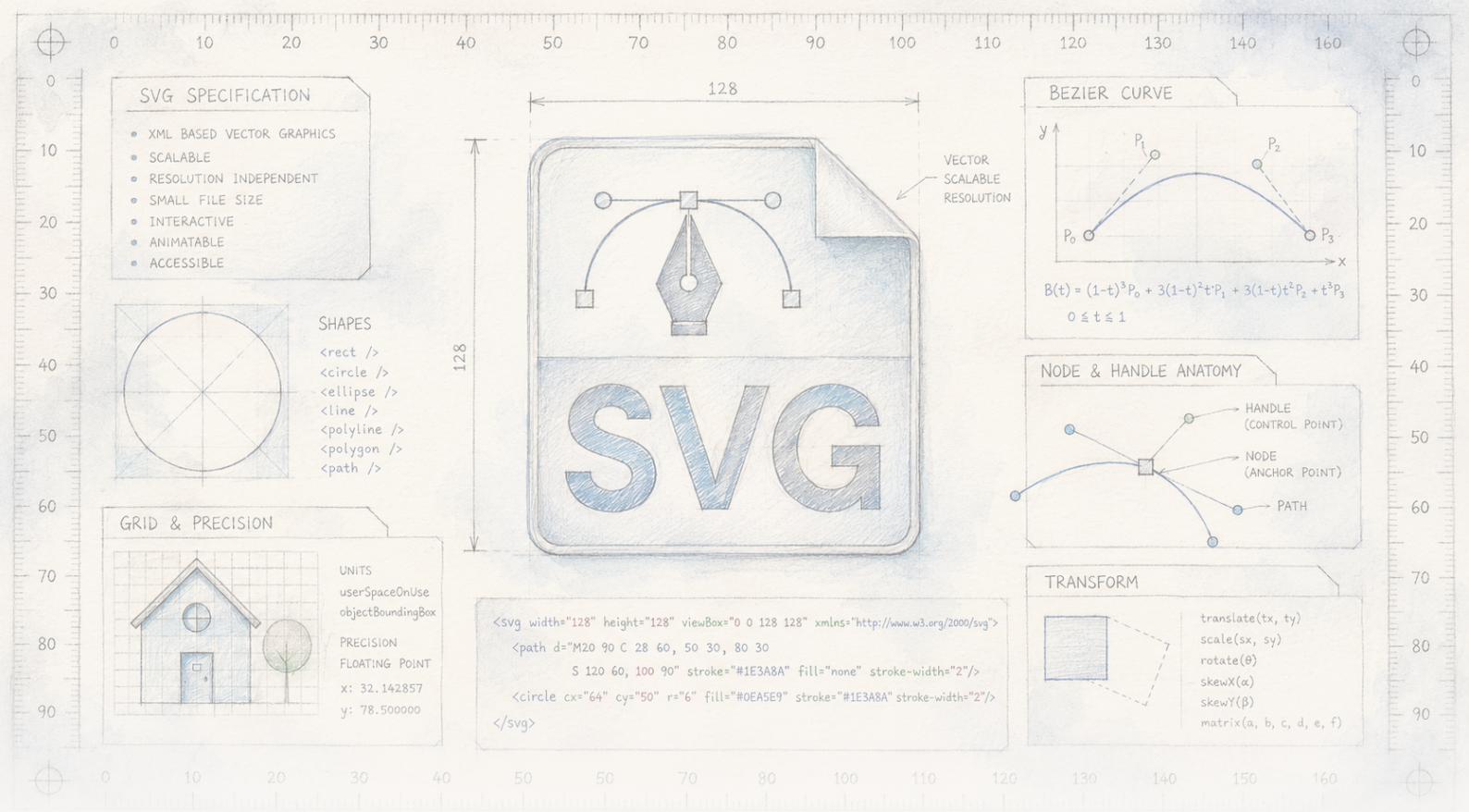



The Clean Paper

WHAT THE PAPER SAYS · WHAT IT DOESN'T · WHY IT MATTERS

# Teaching a drawing AI to look at the page

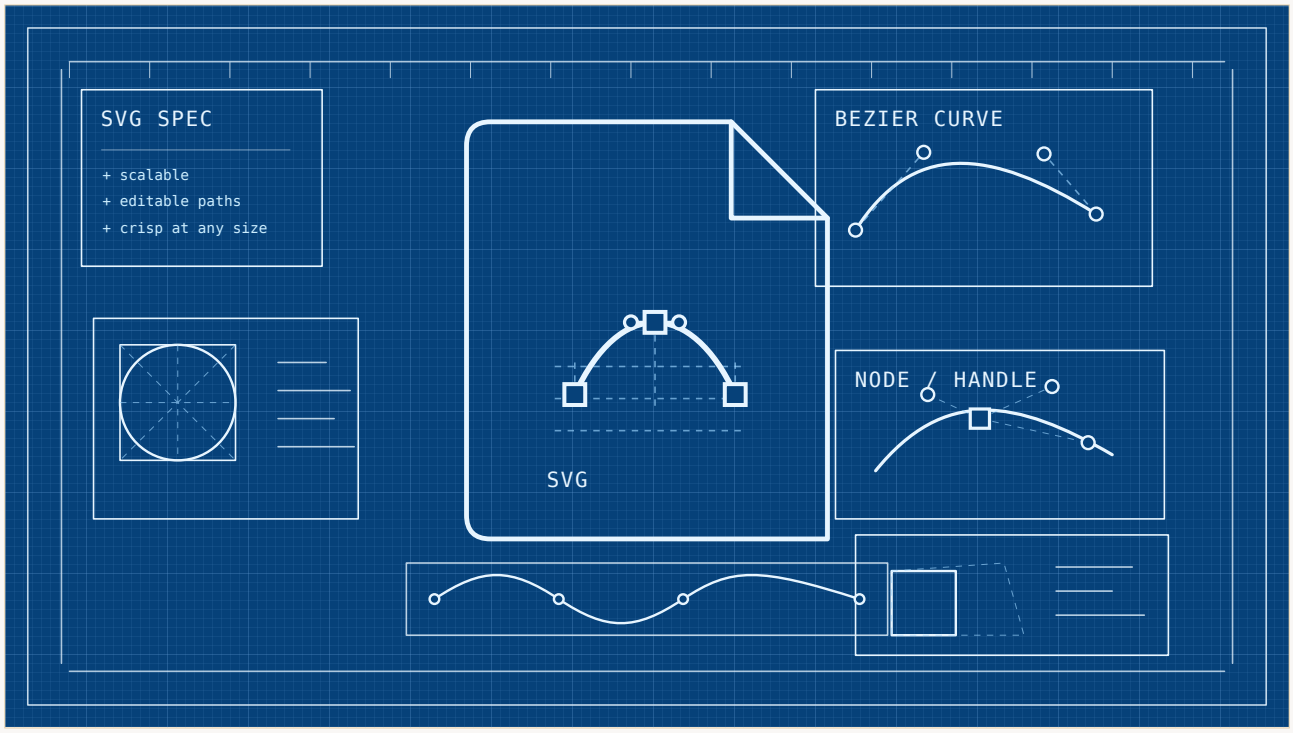
*Language models that generate vector graphics do it blind — writing out the drawing commands without ever seeing the result. A new method lets the model watch its own canvas fill in, stroke by stroke, and the honest twist is that simply giving it eyes makes things worse: it has to be retrained to use them. The payoff is a model that matches or edges out rivals trained on up to twenty times more data — a real, careful result on one benchmark, not a revolution.*

Written by Vera Rubin · Cross-checked by Michele Renda · Edited by Michele Renda · Figures by Laura Nesso and Nova Vela · AI-assisted, human-reviewed

Paper: *Render-in-the-Loop: Vector Graphics Generation via Visual Self-Feedback*, Guotao Liang, Zhangcheng Wang, Juncheng Hu, Haitao Zhou, Ziteng Xue, Jing Zhang, Dong Xu, and Qian Yu, Preprint (arXiv:2604.20730)

# Drawing with your eyes closed

Ask one of today's AI models to draw you a picture and, under the hood, one of two very different things happens. The familiar image generators — the ones that conjure a photograph from a sentence — paint pixels directly, and they can see the canvas as they work. But there is a second, quieter kind of drawing, where the model does not paint at all. It writes instructions: *draw a circle here, a line there, fill this shape blue*. Those instructions are code — the same Scalable Vector Graphics, or SVG, that sits behind most of the icons and logos on the web — and the appeal is real: the result is not a fixed grid of pixels but a set of shapes you can rescale, recolour and edit forever without it going blurry.




Original SVG blueprint artwork: the image is itself an editable vector file, built from paths, grid lines, nodes and handles rather than fixed pixels.

Laura Nesso / The Clean Paper Permission on file

The catch is that, until recently, a model writing this drawing code did it blind. It emitted the whole sequence of instructions in one pass — circle, line, fill — without ever rendering them to see what it had made. Picture sketching a face with your eyes shut: you might get the eyes roughly where eyes go, but you would have no way to notice that the second one landed on the cheek, or that the shape you drew for the hair is now sitting over the nose. That is more or less how these models worked, which is why they so often produced code that was perfectly valid and yet visually a mess.

A team led by Guotao Liang has now done the almost comically obvious thing: they let the model open its eyes. Their method, *Render-in-the-Loop,* renders the half-finished drawing after each step and hands that picture back to the model before it draws the next stroke. The genuinely interesting part is not that this helps — it is what they had to do to make it help at all.

**How do you score a drawing?**

When a paper says its model "beats" another, it is fair to ask: beats it at what, measured how? Judging a generated picture is genuinely hard — there is no single right answer to "draw a laptop" — so the field leans on a handful of automatic scores, each an imperfect stand-in for a person looking at the result.

A few appear in this paper. **FID** compares the overall statistical flavour of a whole batch of generated images against real ones; lower is better, but it describes the batch, not any one picture. **CLIP score** asks a separate AI whether an image matches the words of the prompt — useful, but only as sharp as that judge. **DINO**, **SSIM** and **LPIPS** compare a reconstructed image to a target, at levels ranging from raw pixels to learned features.

None of these is truth. They are proxies, and they tend to move in small increments. When you read that one model scores 127.6 where another scores 128.8, that is a real difference in the intended direction — but it is a nudge, not a landslide, and a nudge in a number that only loosely tracks what your eye would say. Worth holding onto when the headline is "beats a model trained on twenty times the data."

# What the authors did

The problem the authors set out to fix is the blind drawing itself. Existing models treat writing SVG as a pure text task: predict the next chunk of code from the code so far, and never render it. That leaves idle the powerful "eyes" — the vision encoder — that modern multimodal models already carry. Render-in-the-Loop restructures the job as a step-by-step visual process. After each fragment of drawing code, the partial SVG is rendered to an image and fed back to the model as a picture, so the next fragment is chosen while actually looking at the canvas so far.

Their first finding is a cautionary one, and to their credit they report it plainly: simply bolting this loop onto an existing off-the-shelf model does not work. When they fed intermediate renders to strong general models without any special training, quality did not improve — it *degraded* across the board. A model that was never taught to use its eyes for this does not suddenly know how.

So most of the work is in the teaching. They rebuild the training data so that each drawing is broken into many small, visually meaningful steps — splitting complex shapes into simpler pieces so there is something new to see at each stage — and then fine-tune an eight-billion-parameter open model (built on Qwen3-VL) on these step-by-step sequences. They call this Visual Self-Feedback. They add a second mechanism at drawing time, Render-and-Verify: before accepting each new stroke, the model renders it and checks whether it actually changed the picture, or merely repeated the last one. Strokes that add nothing are discarded, and when nothing more helps, the model is prompted to stop. Notably, all of this runs on a fairly small dataset — about 850,000 examples, less than half of what one rival used and a small fraction of another's.

# What they found

- Trained this way, the model draws better than its blind counterpart — most visibly on the failure cases, where a blind model puts an eye on a cheek, or skips a requested bar chart and outputs a generic monitor instead.
- On the standard benchmark (MMSVGBench), the method comes out competitive with, and on several measures slightly ahead of, strong rivals — including OmniSVG, trained on more than twice the data, and InternSVG, trained on roughly twenty times as much.
- The margins are small. On the icon set, for instance, its main image-quality score is 127.6 against the best rival's 128.8, and its prompt-matching score 0.293 against 0.291 — real and consistent, but slender.
- Both added pieces earn their place: switch off the special training, or the drawing-time verification, and the numbers measurably drop. The verification step in particular stops the model getting stuck redrawing the same thing over and over.
- The result the authors themselves emphasise is efficiency — getting this far from far less training data than the leaders used.

# What this does not prove

- It does **not** show that letting a model "see" is a free win. The opposite, in fact: the paper's own experiment shows that visual feedback *without* retraining makes things worse. The gain comes from the retraining, not the eyes alone.
- It does **not** establish a large or decisive lead. On most scores the method is neck-and-neck with its rivals; "beats a model trained on 20× the data" is true, but by nudges on proxy metrics, on one benchmark.
- It does **not** demonstrate general artistic ability. This is an eight-billion-parameter research model drawing icons and simple illustrations at a small fixed resolution (224×224 pixels), not a general-purpose designer.
- The comparison against a big general model (GPT-5) is **not** apples-to-apples: that model is not built or tuned for this narrow code-drawing task, so beating it here says little about either model in general.
- It does **not** come for free. Rendering and re-reading the canvas at every step makes generation slower than emitting the code in one blind pass — a cost the authors acknowledge.

# How strong is the evidence

- **Solid** where it is a controlled comparison on its own terms. The ablations are clean: remove the training, or remove the verification, and the numbers fall — so the two ingredients really are doing the work the authors claim.
- **Honest about its own surprise.** The finding that naive visual feedback *hurts* is reported, not

buried, and it is the most interesting thing in the paper — a useful correction to the intuition that more input is always better.

- **Thin where it is a leaderboard claim.** The wins over larger-data rivals are small and live on a single benchmark that was built by the authors of one of those rivals. Small margins on proxy metrics on one test set are suggestive, not settled.
- **Untested at scale and in the wild.** Everything here is icons and simple illustrations at low resolution. Whether the same idea holds for complex, high-resolution or real-world design work is left as future work — the authors say as much.

## Why it matters

The idea at the centre of this paper is almost embarrassingly simple, and it reaches well beyond drawing. If a program is going to generate something by writing code — a web page, a chart, a diagram, a 3D scene — it can either write the whole thing blind and hope, or render as it goes and correct course. Closing that loop is second nature for a person; we glance at the page constantly. It is a surprisingly recent move for these models.

What makes the paper worth reading is the asterisk it attaches. Giving a model a way to see is not the same as teaching it to look. The eyes have to be trained, and only then does the loop pay off — and even then the payoff is real but measured: a steadier draughtsman, not a different kind of artist. For anyone building tools that turn a description into an editable graphic — the sort of thing that ends up behind an app's icons and illustrations — the quiet, practical lesson is the useful one. The win here came from cheaper, smarter training, not from more data. That is a better thing to learn than another point on a leaderboard.

## Clean summary

Language models that generate vector graphics — the editable, rescalable code behind most web icons and logos — have traditionally done it "blind," writing out all the drawing commands without ever rendering them to see the result. A team led by Guotao Liang proposes Render-in-the-Loop: render the half-finished drawing after each step and feed it back, so the model draws the next stroke while looking at the canvas. Their central, honest finding is that simply doing this to an existing model makes it *worse*; the gain appears only after the model is retrained to use the visual feedback, helped by a drawing-time check that discards strokes which change nothing. The retrained eight-billion-parameter model matches or slightly beats rivals trained on up to twenty times more data on a standard benchmark — a genuine result, but by small margins on proxy scores, for icons and simple illustrations at low resolution. The takeaway the authors stress, and the one worth keeping, is about efficiency: seeing your work well can stand in for sheer data scale.

# No-BS check

**What the paper shows:** Retraining a vector-graphics model to render and look at its own half-finished drawing, step by step, produces better and more complete results than drawing blind — competitive with, and slightly ahead of, larger-data rivals on one standard benchmark, from less training data.

**What is plausible but not proven:** That "seeing your work" is broadly a better recipe than scaling data; that the same idea will help on complex, high-resolution or real-world graphics; that the small benchmark margins reflect a difference a person would actually notice.

**What it does not show:** That visual feedback helps on its own (without retraining it hurts); that the lead over rivals is large or decisive; general-purpose drawing ability; a fair head-to-head with general models like GPT-5, which are not built for this task.

**Main limitations:** One benchmark, built partly by a rival's authors; small margins on proxy metrics; an 8B model, icons and simple illustrations at 224×224; slower generation because of the render-every-step loop.

**How much confidence should a general reader have?** High that closing the loop — rendering and re-reading as you draw — genuinely helps, and that it has to be trained in, not just switched on. Low-to-moderate that this particular model decisively beats its rivals; treat the "beats 20× the data" line as a real but modest, single-benchmark result. High on the one idea worth remembering: for code that draws, looking as you go beats drawing blind.

---

## Sources

Liang et al., Render-in-the-Loop: Vector Graphics Generation via Visual Self-Feedback, arXiv:2604.20730 (2026)
https://arxiv.org/abs/2604.20730

OmniSVG project page
https://omnisvg.github.io/

InternSVG project page
https://hmwang2002.github.io/release/internsvg/