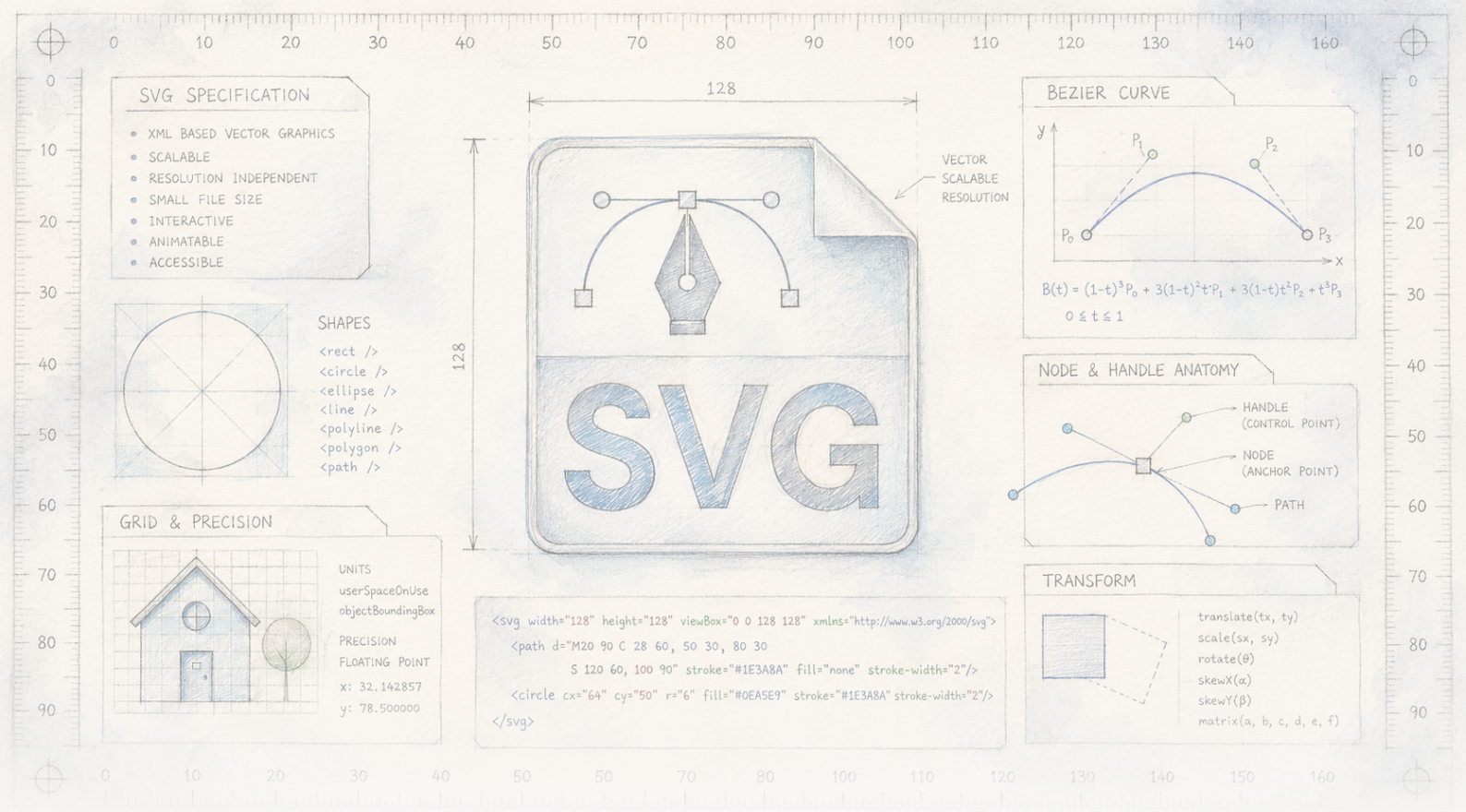

The Clean Paper

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

Enseñar a una IA de dibujo a mirar la página

Los modelos de lenguaje que generan gráficos vectoriales lo hacen a ciegas: escriben los comandos de dibujo sin ver nunca el resultado. Un nuevo método deja que el modelo observe cómo se llena su propio lienzo, trazo a trazo, y el giro honesto es que darle ojos sin más empeora las cosas: hay que reentrenarlo para usarlos. El resultado es un modelo que iguala o supera ligeramente a rivales entrenados con hasta veinte veces más datos: un resultado real y cuidadoso en un benchmark, no una revolución.

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)

Dibujar con los ojos cerrados

Pídele a uno de los modelos de IA actuales que te dibuje una imagen y, por debajo, puede ocurrir una de dos cosas muy distintas. Los generadores de imágenes conocidos —los que hacen aparecer una fotografía a partir de una frase— pintan píxeles directamente, y pueden ver el lienzo mientras trabajan. Pero existe una segunda forma de dibujo, más silenciosa, en la que el modelo no pinta en absoluto. Escribe instrucciones: *dibuja un círculo aquí, una línea allí, rellena esta forma de azul.* Esas instrucciones son código: el mismo Scalable Vector Graphics, o SVG, que está detrás de la mayoría de los iconos y logotipos de la web. Y su atractivo es real: el resultado no es una cuadrícula fija de píxeles, sino un conjunto de formas que se pueden escalar, recolorear y editar indefinidamente sin que se vuelvan borrosas.

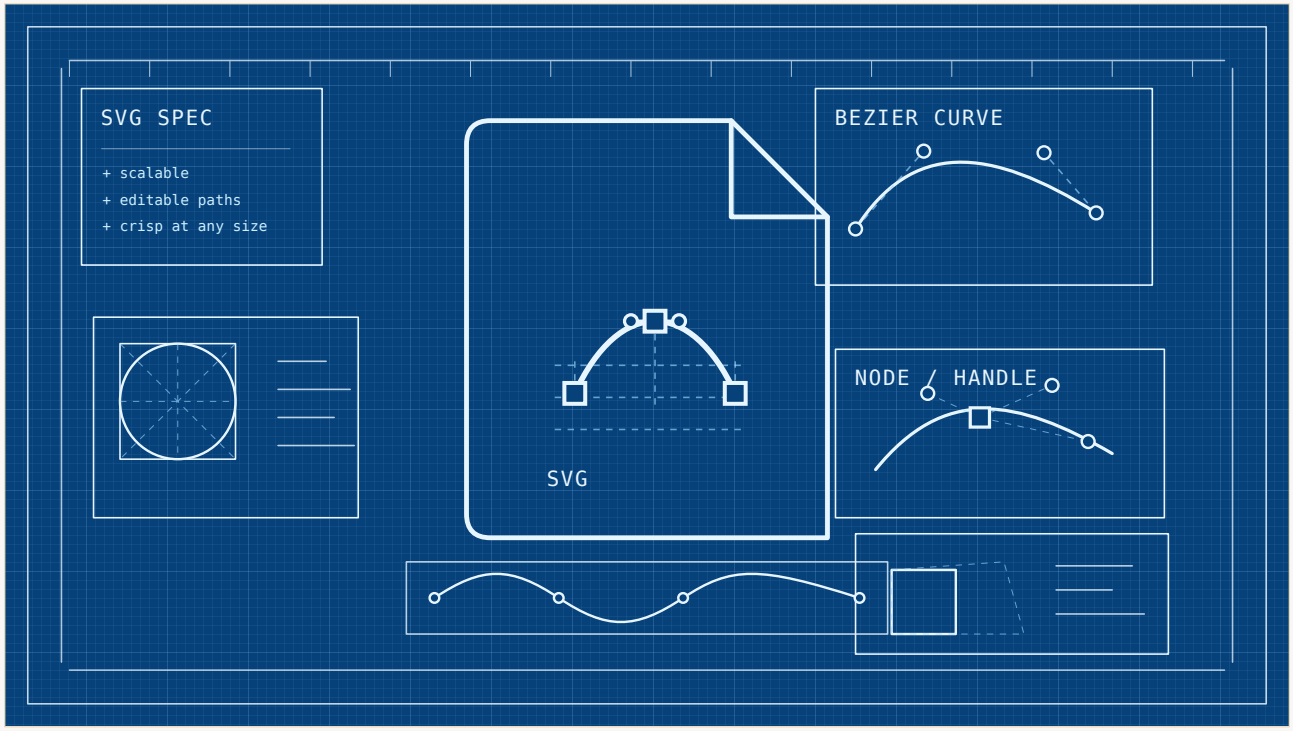

Ilustración original en SVG con aspecto de plano técnico: la imagen es en sí misma un archivo vectorial editable, construido con trazados, líneas de cuadrícula, nodos y manejadores en lugar de píxeles fijos.

Laura Nesso / The Clean Paper Permission on file

El problema es que, hasta hace poco, un modelo que escribía este código de dibujo lo hacía a ciegas. Emitía toda la secuencia de instrucciones en una sola pasada —círculo, línea, relleno— sin renderizarlas nunca para ver qué había hecho. Imagina dibujar una cara con los ojos cerrados: quizá coloques los ojos más o menos donde van los ojos, pero no tendrías forma de notar que el segundo acabó en la mejilla, o que la forma del pelo ahora tapa la nariz. Así funcionaban, más o menos, estos modelos, y por eso producían tan a menudo código perfectamente válido pero visualmente caótico.

Un equipo dirigido por Guotao Liang acaba de hacer lo casi cómicamente obvio: dejó que el modelo abriera los ojos. Su método, *Render-in-the-Loop*, renderiza el dibujo a medio hacer después de cada paso y devuelve esa imagen al modelo antes de que trace el siguiente trazo. Lo realmente interesante

no es que esto ayude, sino lo que tuvieron que hacer para que ayudara en absoluto.

¿Cómo se puntúa un dibujo?

Cuando un artículo dice que su modelo "supera" a otro, conviene preguntar: ¿lo supera en qué, medido cómo? Juzgar una imagen generada es realmente difícil —no hay una única respuesta correcta a "dibuja un portátil"—, así que el campo se apoya en un puñado de puntuaciones automáticas, cada una un sustituto imperfecto de una persona mirando el resultado.
Varias aparecen en este trabajo. **FID** compara el sabor estadístico global de todo un lote de imágenes generadas con imágenes reales; más bajo es mejor, pero describe el lote, no una imagen concreta. La **puntuación CLIP** pregunta a otra IA si una imagen coincide con las palabras del prompt: útil, pero tan precisa como ese juez. **DINO**, **SSIM** y **LPIPS** comparan una imagen reconstruida con una imagen objetivo, a niveles que van desde píxeles brutos hasta rasgos aprendidos.
Ninguna de estas métricas es la verdad. Son proxys, y suelen moverse por pequeños incrementos. Cuando lees que un modelo puntúa 127,6 donde otro puntúa 128,8, eso es una diferencia real en la dirección buscada, pero es un empujón, no un deslizamiento de tierras, y en un número que solo sigue de forma aproximada lo que diría tu ojo. Conviene recordarlo cuando el titular es "supera a un modelo entrenado con veinte veces más datos".

Qué hicieron los autores

El problema que los autores intentan arreglar es el propio dibujo a ciegas. Los modelos existentes tratan la escritura de SVG como una tarea puramente textual: predecir el siguiente fragmento de código a partir del código escrito hasta ahora, sin renderizarlo nunca. Eso deja ociosos los poderosos "ojos" —el codificador visual— que los modelos multimodales modernos ya llevan dentro. Render-in-the-Loop reorganiza el trabajo como un proceso visual paso a paso. Después de cada fragmento de código de dibujo, el SVG parcial se renderiza como imagen y se devuelve al modelo, de modo que el siguiente fragmento se elige mirando realmente el lienzo hasta ese momento.

Su primer hallazgo es una advertencia, y lo cuentan con claridad: simplemente añadir esta vuelta a un modelo existente no funciona. Cuando dieron renderizados intermedios a modelos generales fuertes sin ningún entrenamiento especial, la calidad no mejoró; se *degradó* en todos los casos. Un modelo al que nunca se le enseñó a usar los ojos para esto no sabe hacerlo de repente.

Así que casi todo el trabajo está en la enseñanza. Reconstruyen los datos de entrenamiento para que cada dibujo se divida en muchos pasos pequeños y visualmente significativos —partiendo formas complejas en piezas más simples para que haya algo nuevo que ver en cada etapa— y luego afinan un modelo abierto de ocho mil millones de parámetros (basado en Qwen3-VL) con esas secuencias paso a paso. Lo llaman Visual Self-Feedback. Añaden un segundo mecanismo durante el dibujo, Render-and-Verify: antes de aceptar cada nuevo trazo, el modelo lo renderiza y comprueba si realmente cambió la imagen o si solo repitió el anterior. Los trazos que no añaden nada se descartan, y cuando nada más ayuda se le indica al modelo que se detenga. En particular, todo esto funciona con un

conjunto de datos bastante pequeño: unos 850.000 ejemplos, menos de la mitad de lo que usó un rival y una pequeña fracción de lo que usó otro.

Qué encontraron

- Entrenado de esta forma, el modelo dibuja mejor que su equivalente ciego, sobre todo en los casos de fallo, donde un modelo ciego pone un ojo en una mejilla o se salta un gráfico de barras pedido y genera un monitor genérico.
- En el benchmark estándar (MMSVGBench), el método resulta competitivo con rivales fuertes y, en varias medidas, ligeramente por delante, incluidos OmniSVG, entrenado con más del doble de datos, e InternSVG, entrenado con unas veinte veces más.
- Los márgenes son pequeños. En el conjunto de iconos, por ejemplo, su puntuación principal de calidad de imagen es 127,6 frente a 128,8 del mejor rival, y su puntuación de coincidencia con el prompt es 0,293 frente a 0,291: diferencias reales y consistentes, pero estrechas.
- Las dos piezas añadidas se justifican: si se desactiva el entrenamiento especial o la verificación durante el dibujo, los números caen de forma medible. La verificación, en particular, evita que el modelo se quede atascado redibujando lo mismo una y otra vez.
- El resultado que los propios autores subrayan es la eficiencia: llegar hasta aquí con muchos menos datos de entrenamiento que los líderes.

Qué no demuestra

- **No** muestra que dejar que un modelo "vea" sea una ganancia gratuita. De hecho, lo contrario: el propio experimento del artículo muestra que la retroalimentación visual *sin* reentrenamiento empeora las cosas. La mejora viene del reentrenamiento, no solo de los ojos.
- **No** establece una ventaja grande o decisiva. En la mayoría de las puntuaciones, el método está muy cerca de sus rivales; "supera a un modelo entrenado con 20× más datos" es cierto, pero por pequeños empujones en métricas proxy, en un único benchmark.
- **No** demuestra capacidad artística general. Es un modelo de investigación de ocho mil millones de parámetros que dibuja iconos e ilustraciones simples a una pequeña resolución fija (224×224 píxeles), no un diseñador de propósito general.
- La comparación con un gran modelo general (GPT-5) **no** es de igual a igual: ese modelo no está construido ni ajustado para esta tarea estrecha de dibujo por código, así que superarlo aquí dice poco sobre cualquiera de los dos modelos en general.
- **No** sale gratis. Renderizar y volver a leer el lienzo en cada paso hace que la generación sea más lenta que emitir el código en una sola pasada ciega, un coste que los autores reconocen.

¿Qué solidez tienen las pruebas?

- **Sólida** cuando se trata de una comparación controlada en sus propios términos. Las ablaciones son limpias: quitar el entrenamiento o quitar la verificación hace caer los números, así que los dos ingredientes hacen realmente el trabajo que los autores afirman.
- **Honesta sobre su propia sorpresa.** El resultado de que la retroalimentación visual ingenua *perjudica* se informa, no se entierra, y es lo más interesante del artículo: una corrección útil a la intuición de que más entrada siempre es mejor.
- **Débil como afirmación de tabla clasificatoria.** Las victorias frente a rivales entrenados con más datos son pequeñas y viven en un único benchmark construido por los autores de uno de esos rivales. Pequeños márgenes en métricas proxy en un solo conjunto de prueba son sugerentes, no concluyentes.
- **No probada a escala ni en el mundo real.** Todo aquí son iconos e ilustraciones simples a baja resolución. Si la misma idea funciona para diseño complejo, de alta resolución o real queda para trabajo futuro; los autores lo dicen.

Por qué importa

La idea central del trabajo es casi vergonzosamente simple, y va mucho más allá del dibujo. Si un programa va a generar algo escribiendo código —una página web, un gráfico, un diagrama, una escena 3D— puede escribirlo todo a ciegas y esperar, o renderizar a medida que avanza y corregir el rumbo. Cerrar ese bucle es natural para una persona; miramos la página constantemente. Es un movimiento sorprendentemente reciente para estos modelos.

Lo que hace que el artículo merezca leerse es el asterisco que añade. Darle a un modelo una forma de ver no es lo mismo que enseñarle a mirar. Los ojos deben entrenarse, y solo entonces el bucle compensa; incluso entonces, el resultado es real pero medido: un dibujante más estable, no una clase nueva de artista. Para quien construya herramientas que convierten una descripción en un gráfico editable —lo que acaba detrás de iconos e ilustraciones de una aplicación—, la lección práctica y silenciosa es la útil. La mejora vino de un entrenamiento más barato e inteligente, no de más datos. Eso es más valioso que otro punto en una tabla.

En pocas palabras

Los modelos de lenguaje que generan gráficos vectoriales —el código editable y reescalable detrás de la mayoría de los iconos y logotipos de la web— tradicionalmente lo han hecho “a ciegas”, escribiendo todos los comandos de dibujo sin renderizarlos nunca para ver el resultado. Un equipo dirigido por Guotao Liang propone Render-in-the-Loop: renderizar el dibujo a medio hacer después de cada paso y devolverlo al modelo, para que trace el siguiente movimiento mirando el lienzo. Su hallazgo central y honesto es que hacer esto sin más a un modelo existente lo vuelve *peor*; la mejora

aparece solo después de reentrenar al modelo para usar la retroalimentación visual, con ayuda de una comprobación durante el dibujo que descarta trazos que no cambian nada. El modelo reentrenado de ocho mil millones de parámetros iguala o supera ligeramente a rivales entrenados con hasta veinte veces más datos en un benchmark estándar: un resultado genuino, pero por márgenes pequeños en puntuaciones proxy, para iconos e ilustraciones simples a baja resolución. La conclusión que los autores subrayan, y la que conviene conservar, trata de eficiencia: ver bien tu propio trabajo puede sustituir parte de la escala bruta de datos.

Sin rodeos

Lo que muestra el artículo: Reentrenar un modelo de gráficos vectoriales para renderizar y mirar su propio dibujo a medio hacer, paso a paso, produce resultados mejores y más completos que dibujar a ciegas: competitivos con rivales entrenados con más datos, y ligeramente por delante de ellos en un benchmark estándar, con menos datos de entrenamiento.

Lo que es plausible pero no está demostrado: Que “ver tu trabajo” sea en general una receta mejor que escalar datos; que la misma idea ayude en gráficos complejos, de alta resolución o de uso real; que los pequeños márgenes del benchmark reflejen una diferencia que una persona notaría.

Lo que no muestra: Que la retroalimentación visual ayude por sí sola (sin reentrenamiento perjudica); que la ventaja frente a rivales sea grande o decisiva; capacidad general de dibujo; una comparación justa con modelos generales como GPT-5, que no están construidos para esta tarea.

Principales limitaciones: Un benchmark, construido en parte por autores de un rival; pequeños márgenes en métricas proxy; un modelo 8B, iconos e ilustraciones simples a 224×224; generación más lenta por el bucle que renderiza en cada paso.

¿Cuánta confianza debería tener un lector general? Alta en que cerrar el bucle —renderizar y volver a mirar mientras se dibuja— ayuda de verdad, y que debe entrenarse, no solo activarse. Baja a moderada en que este modelo supere decisivamente a sus rivales; trata la frase “supera con 20× menos datos” como un resultado real pero modesto, de un solo benchmark. Alta en la idea que vale la pena recordar: para código que dibuja, mirar mientras se avanza es mejor que dibujar a ciegas.

Fuentes

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/