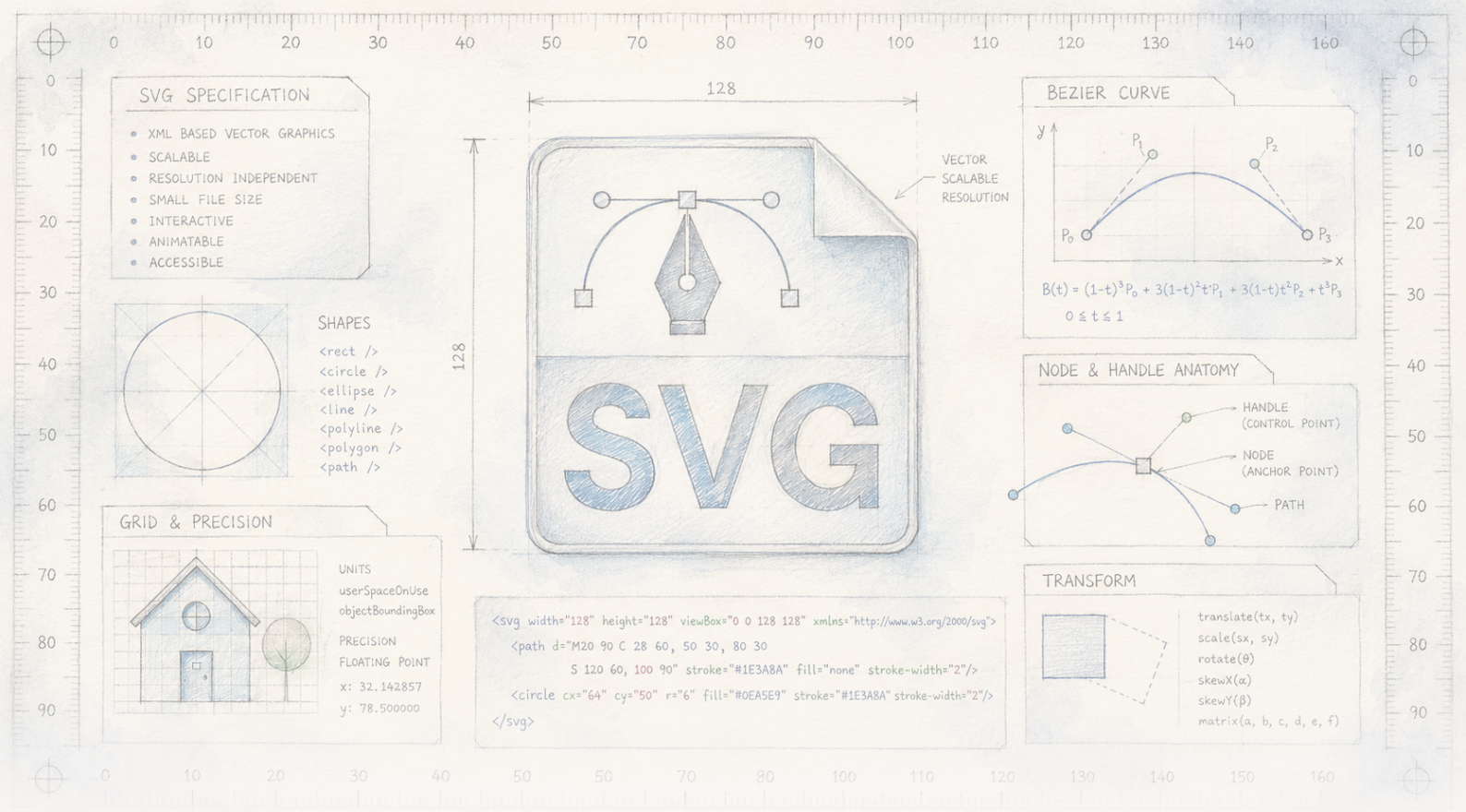

The Clean Paper

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

Ensinar uma IA que desenha a olhar para a página

Os modelos de linguagem que geram gráficos vetoriais fazem-no às cegas — escrevem os comandos de desenho sem nunca ver o resultado. Um novo método deixa o modelo observar a própria ecrã a preencher-se, traço a traço, e a reviravolta honesta é que dar-lhe simplesmente olhos piora o resultado: é preciso treiná-lo novamente para os usar. O ganho é um modelo que iguala ou supera ligeiramente rivais treinados com até vinte vezes mais dados — um resultado real e cuidadoso num benchmark, não uma revolução.

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)

Desenhar de olhos fechados

Peça a um dos atuais modelos de IA para desenhar uma imagem e, por baixo da superfície, pode acontecer uma de duas coisas muito diferentes. Os geradores de imagem mais familiares — os que transformam uma frase numa fotografia — pintam píxeis diretamente e conseguem ver a ecrã enquanto trabalham. Mas existe uma segunda forma, mais discreta, de desenhar, em que o modelo não pinta nada. Escreve instruções: *desenha um círculo aqui, uma linha ali, preenche esta forma a azul*. Essas instruções são código — o mesmo Scalable Vector Graphics, ou SVG, que está por trás da maioria dos ícones e logótipos na Web — e a vantagem é real: o resultado não é uma grelha fixa de píxeis, mas um conjunto de formas que se podem redimensionar, recolorir e editar indefinidamente sem ficarem desfocadas.

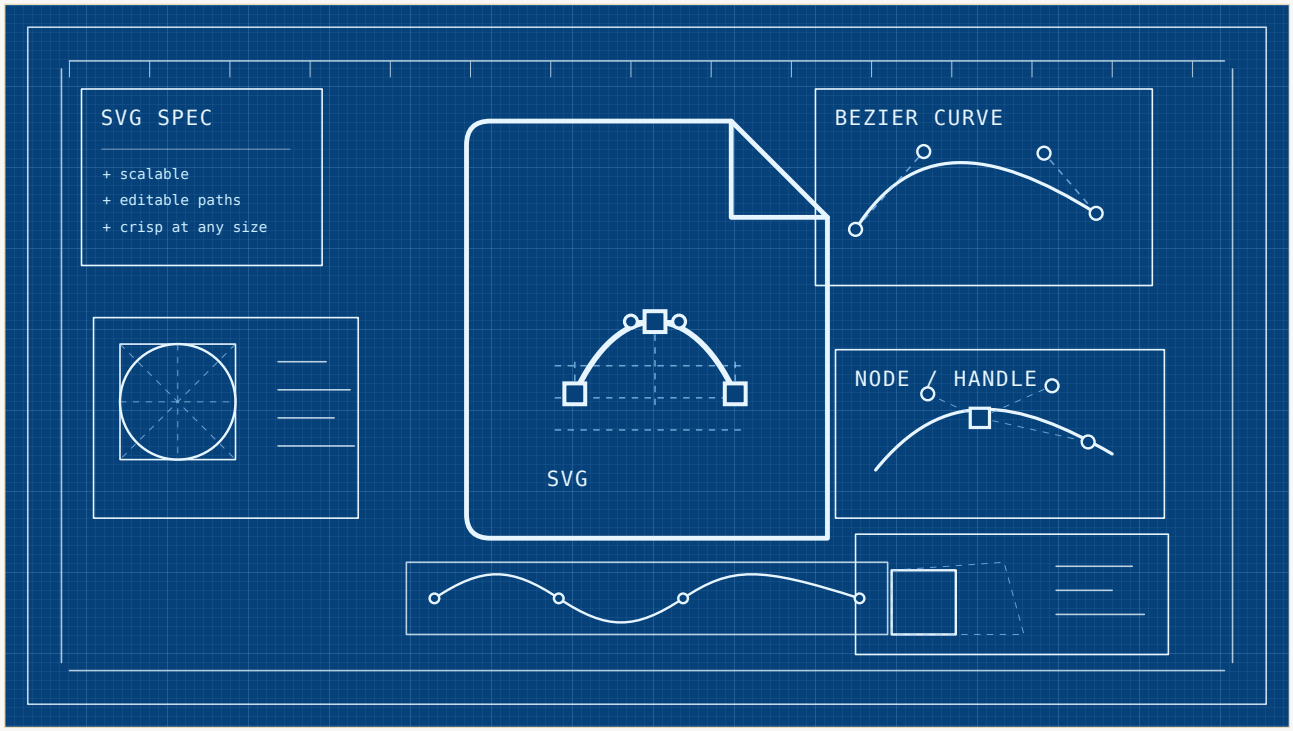

Ilustração original em SVG, ao estilo de uma planta técnica: a própria imagem é um ficheiro vetorial editável, construído com caminhos, linhas de grelha, nós e controlos, e não com píxeis fixos.

Laura Nesso / The Clean Paper Permission on file

O problema é que, até há pouco tempo, um modelo que escrevia este código de desenho fazia-o às cegas. Emitia toda a sequência de instruções de uma só vez — círculo, linha, preenchimento — sem nunca as renderizar para ver o que tinha criado. Imagine desenhar um rosto de olhos fechados: talvez coloque os olhos aproximadamente onde devem estar, mas não consegue reparar que o segundo foi parar à face, ou que a forma desenhada para o cabelo está agora sobre o nariz. Era mais ou menos assim que estes modelos funcionavam, e por isso produziam tantas vezes código perfeitamente válido que, visualmente, era uma confusão.

Uma equipa liderada por Guotao Liang fez agora a coisa quase comicamente óbvia: deixou o modelo abrir os olhos. O método, *Render-in-the-Loop*, renderiza o desenho incompleto depois de cada passo

e devolve essa imagem ao modelo antes de ele traçar o próximo elemento. A parte genuinamente interessante não é que isto ajude — é o que foi necessário fazer para que ajudasse de facto.

Como se pontua um desenho?

Quando um artigo diz que um modelo «supera» outro, é justo perguntar: supera-o em quê, e como foi medido? Avaliar uma imagem gerada é realmente difícil — não existe uma única resposta certa para «desenha um portátil» — por isso a área apoia-se num pequeno conjunto de métricas automáticas, cada uma delas um substituto imperfeito de uma pessoa a olhar para o resultado.

Algumas aparecem neste artigo. **FID** compara o perfil estatístico global de um conjunto inteiro de imagens geradas com o de imagens reais; mais baixo é melhor, mas descreve o conjunto, não uma imagem individual. A **pontuação CLIP** pergunta a outra IA se a imagem corresponde às palavras do prompt — útil, mas tão boa quanto o juiz que a produz. **DINO**, **SSIM** e **LPIPS** comparam uma imagem reconstruída com um alvo, em níveis que vão dos píxeis brutos a características aprendidas.

Nenhuma destas métricas é a verdade. São proxies e tendem a variar em pequenos incrementos. Quando lê que um modelo marca 127,6 e outro 128,8, isso é uma diferença real na direção pretendida — mas é um pequeno avanço, não uma avalanche, e num número que apenas acompanha de forma aproximada aquilo que o seu olho diria. Vale a pena lembrar isto quando a manchete é «supera um modelo treinado com vinte vezes mais dados».

O que fizeram os autores

O problema que os autores quiseram resolver é precisamente o desenho às cegas. Os modelos existentes tratam a escrita de SVG como uma tarefa puramente textual: prever o próximo bloco de código a partir do código anterior, sem nunca o renderizar. Isso deixa por utilizar os poderosos «olhos» — o codificador visual — que os modelos multimodais modernos já possuem. O Render-in-the-Loop reorganiza a tarefa como um processo visual passo a passo. Depois de cada fragmento de código de desenho, o SVG parcial é renderizado numa imagem e devolvido ao modelo como imagem, de modo que o fragmento seguinte seja escolhido olhando efetivamente para a ecrã até então.

A primeira descoberta é uma advertência — e, para mérito dos autores, é apresentada sem rodeios: simplesmente acrescentar este ciclo a um modelo existente, tal como vem de fábrica, não funciona. Quando forneceram renderizações intermédias a modelos generalistas fortes sem treino específico, a qualidade não melhorou — **piorou em todos os casos**. Um modelo que nunca foi ensinado a usar os olhos desta maneira não aprende subitamente a fazê-lo.

Por isso, a maior parte do trabalho está no treino. Os autores reconstruíram os dados de treino de modo que cada desenho fosse dividido em muitos passos pequenos e visualmente significativos — separando formas complexas em componentes mais simples para que haja algo novo a observar em cada etapa — e depois fizeram *fine-tuning* de um modelo aberto com oito mil milhões de parâmetros (baseado no Qwen3-VL) sobre estas sequências passo a passo. Chamam a isto Visual Self-Feedback.

Acrescentam um segundo mecanismo durante o desenho, Render-and-Verify: antes de aceitar cada novo traço, o modelo renderiza-o e verifica se realmente alterou a imagem ou se apenas repetiu o anterior. Traços que não acrescentam nada são descartados e, quando nada mais ajuda, o modelo recebe a instrução para parar. É importante notar que tudo isto funciona sobre um conjunto de dados relativamente pequeno — cerca de 850 000 exemplos, menos de metade do utilizado por um concorrente e uma pequena fração do utilizado por outro.

O que encontraram

- Treinado desta forma, o modelo desenha melhor do que a versão cega — sobretudo nos casos de falha mais visíveis, em que um modelo cego coloca um olho numa face ou ignora um gráfico de barras pedido e devolve um monitor genérico.
- No benchmark padrão (MMSVGBench), o método é competitivo e, em várias métricas, ligeiramente superior a concorrentes fortes — incluindo o OmniSVG, treinado com mais do dobro dos dados, e o InternSVG, treinado com cerca de vinte vezes mais.
- As margens são pequenas. No conjunto de ícones, por exemplo, a principal métrica de qualidade de imagem é 127,6 contra 128,8 do melhor concorrente, e a pontuação de correspondência ao prompt é 0,293 contra 0,291 — diferenças reais e consistentes, mas estreitas.
- Os dois componentes acrescentados justificam a sua presença: desligar o treino especial ou a verificação durante o desenho faz os números cair de forma mensurável. A etapa de verificação, em particular, impede o modelo de ficar preso a redesenhar repetidamente a mesma coisa.
- O resultado que os próprios autores salientam é a eficiência — chegar até aqui com muito menos dados de treino do que os líderes utilizaram.

O que isto não demonstra

- **Não** mostra que deixar um modelo «ver» seja um ganho gratuito. Pelo contrário: a própria experiência do artigo mostra que feedback visual *sem* novo treino piora o resultado. O ganho vem do treino, não apenas dos olhos.
- **Não** estabelece uma vantagem grande ou decisiva. Na maioria das métricas, o método está muito próximo dos concorrentes; «supera um modelo treinado com 20× mais dados» é verdade, mas por pequenos incrementos em métricas proxy, num único benchmark.
- **Não** demonstra capacidade artística geral. Trata-se de um modelo de investigação com oito mil milhões de parâmetros a desenhar ícones e ilustrações simples numa pequena resolução fixa (224×224 píxeis), não de um designer generalista.
- A comparação com um grande modelo generalista (GPT-5) **não** é uma comparação equivalente: esse modelo não foi construído nem afinado para esta tarefa estreita de desenho por código, por isso superá-lo aqui diz pouco sobre qualquer dos modelos em geral.
- **Não** é gratuito em termos computacionais. Renderizar e voltar a ler a ecrã a cada passo torna a geração mais lenta do que emitir todo o código numa única passagem às cegas — um custo que os autores reconhecem.

Quão forte é a evidência

- **Sólida** quando se trata de uma comparação controlada nos próprios termos do estudo. As ablações são claras: retire-se o treino ou a verificação e os números baixam — portanto, os dois ingredientes estão realmente a fazer o trabalho que os autores afirmam.
- **Honesta quanto à própria surpresa.** O resultado de que feedback visual ingénuo *prejudica* é apresentado, não escondido, e é uma das partes mais interessantes do artigo — uma correção útil à intuição de que mais informação é sempre melhor.
- **Fraca enquanto afirmação de liderança.** As vantagens sobre rivais treinados com mais dados são pequenas e vivem num único benchmark construído pelos autores de um desses rivais. Margens pequenas em métricas proxy num único conjunto de teste são sugestivas, não conclusivas.
- **Não testada à escala nem no mundo real.** Tudo aqui são ícones e ilustrações simples a baixa resolução. Se a mesma ideia funciona em gráficos complexos, alta resolução ou trabalho de design real fica para estudos futuros — os próprios autores o dizem.

Porque importa

A ideia no centro deste artigo é quase embaraçosamente simples e vai muito além do desenho. Se um programa vai gerar alguma coisa escrevendo código — uma página Web, um gráfico, um diagrama, uma cena 3D — pode escrever tudo às cegas e esperar pelo melhor, ou pode renderizar à medida que avança e corrigir o rumo. Fechar esse ciclo é instintivo para uma pessoa; olhamos constantemente para a página. Para estes modelos, é uma mudança surpreendentemente recente.

O que torna o artigo digno de leitura é o asterisco que acrescenta a essa ideia. Dar a um modelo uma forma de ver não é o mesmo que ensiná-lo a olhar. Os olhos têm de ser treinados e só então o ciclo compensa — e, mesmo assim, o ganho é real mas medido: um desenhador mais consistente, não um tipo diferente de artista. Para quem constrói ferramentas que transformam uma descrição num gráfico editável — precisamente o tipo de coisa que acaba por estar por trás dos ícones e ilustrações de uma aplicação — a lição discreta e prática é a útil. O ganho aqui veio de treino mais barato e inteligente, não de mais dados. É uma aprendizagem melhor do que mais um ponto numa tabela classificativa.

Resumo claro

Os modelos de linguagem que geram gráficos vetoriais — o código editável e redimensionável por trás da maioria dos ícones e logótipos na Web — têm tradicionalmente trabalhado «às cegas», escrevendo todos os comandos de desenho sem nunca renderizá-los para ver o resultado. Uma equipa liderada por Guotao Liang propõe Render-in-the-Loop: renderizar o desenho incompleto após cada passo e devolvê-lo ao modelo, para que este trace o elemento seguinte olhando para a ecrã. A descoberta central e honesta é que fazer simplesmente isto a um modelo existente torna-o *pior*; o ganho só

aparece depois de o modelo ser treinado de novo para utilizar o feedback visual, ajudado por uma verificação durante o desenho que descarta traços que não alteram nada. O modelo retreinado, com oito mil milhões de parâmetros, iguala ou supera ligeiramente concorrentes treinados com até vinte vezes mais dados num benchmark padrão — um resultado genuíno, mas com margens pequenas em métricas proxy, para ícones e ilustrações simples a baixa resolução. A conclusão que os autores salientam — e que vale a pena guardar — é sobre eficiência: saber olhar para o próprio trabalho pode compensar parte da escala bruta de dados.

Verificação sem exageros

O que o artigo mostra: Treinar novamente um modelo de gráficos vetoriais para renderizar e observar o seu próprio desenho incompleto, passo a passo, produz resultados melhores e mais completos do que desenhar às cegas — competitivos e ligeiramente superiores aos de rivais com mais dados num benchmark padrão, usando menos dados de treino.

O que é plausível, mas não está demonstrado: Que «olhar para o próprio trabalho» seja, de forma geral, uma estratégia melhor do que aumentar a quantidade de dados; que a mesma ideia ajude em gráficos complexos, de alta resolução ou usados no mundo real; que as pequenas margens do benchmark correspondam a uma diferença que uma pessoa realmente notaria.

O que não mostra: Que o feedback visual ajude sozinho (sem novo treino, prejudica); que a vantagem sobre os rivais seja grande ou decisiva; capacidade de desenho generalista; uma comparação justa com modelos generalistas como GPT-5, que não foram concebidos para esta tarefa.

Principais limitações: Um benchmark, construído em parte pelos autores de um concorrente; pequenas margens em métricas proxy; um modelo de 8B, ícones e ilustrações simples a 224×224; geração mais lenta devido ao ciclo de renderização a cada passo.

Quanta confiança deve ter um leitor geral? Elevada de que fechar o ciclo — renderizar e voltar a olhar à medida que se desenha — ajuda realmente, e de que essa capacidade tem de ser treinada, não apenas ativada. Baixa a moderada de que este modelo específico supere decisivamente os concorrentes; a frase «supera com 20× menos dados» deve ser lida como um resultado real mas modesto, num único benchmark. Elevada quanto à ideia que vale a pena lembrar: para código que desenha, olhar enquanto se trabalha é melhor do que desenhar às cegas.

Fontes

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/