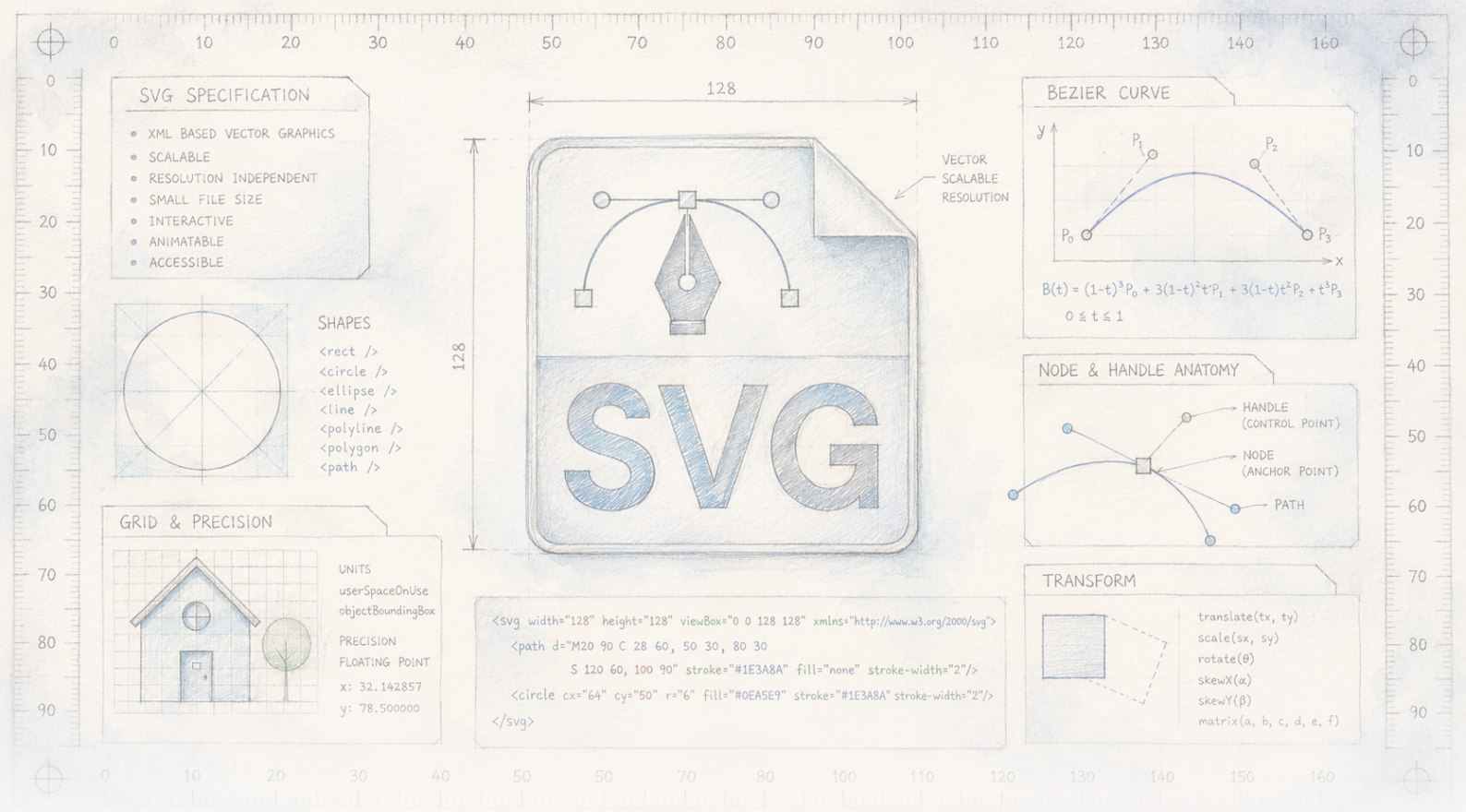


The Clean Paper

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

Ensinar uma IA de desenho a olhar para a página

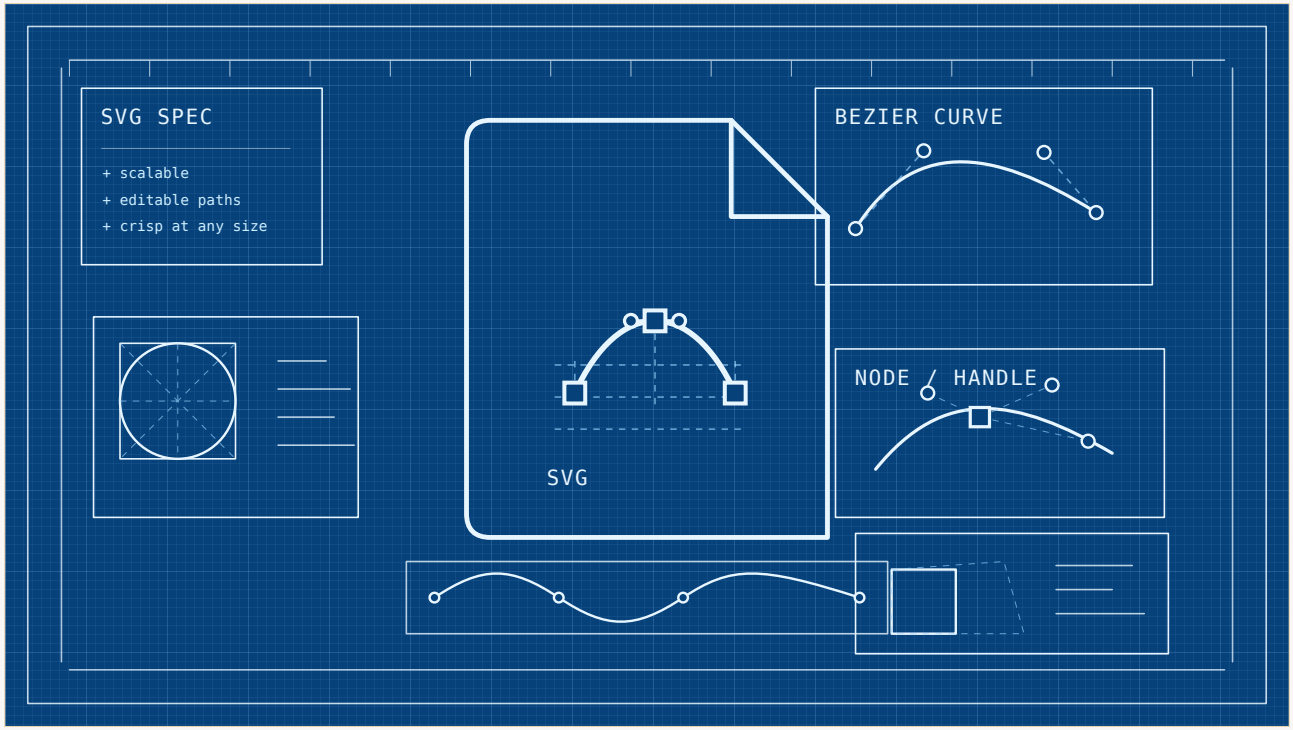
Modelos de linguagem que geram gráficos vetoriais fazem isso às cegas: escrevem os comandos de desenho sem nunca ver o resultado. Um novo método deixa o modelo observar a própria tela se preencher, traço por traço, e a virada honesta é que simplesmente dar olhos ao modelo piora as coisas: ele precisa ser retreinado para usá-los. O ganho é um modelo que iguala ou supera por pouco rivais treinados com até vinte vezes mais dados — um resultado real e cuidadoso em um benchmark, não uma revolução.

Written by Vera Rubin · Cross-checked by Michele Renda · Edited by Michele Renda · Figures by Laura Nesso · 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 modelos de IA atuais que desenhe uma imagem e, por baixo do capô, uma de duas coisas muito diferentes acontece. Os geradores de imagem familiares — aqueles que conjuram uma fotografia a partir de uma frase — pintam pixels diretamente, e conseguem ver a tela enquanto trabalham. Mas há um segundo tipo de desenho, mais silencioso, em que o modelo não pinta nada. Ele escreve instruções: *desenhe um círculo aqui, uma linha ali, preencha esta forma de azul*. Essas instruções são código — o mesmo Scalable Vector Graphics, ou SVG, que fica por trás da maioria dos ícones e logotipos na web — e o apelo é real: o resultado não é uma grade fixa de pixels, mas um conjunto de formas que você pode redimensionar, recolorir e editar para sempre sem ficar borrado.



Arte original em planta SVG: a própria imagem é um arquivo vetorial editável, construído com caminhos, linhas de grade, nós e alças, não com pixels fixos.

Laura Nesso / The Clean Paper Permission on file

O problema é que, até recentemente, um modelo escrevendo esse código de desenho fazia isso às cegas. Ele emitia toda a sequência de instruções de uma só vez — círculo, linha, preenchimento — sem jamais renderizá-las para ver o que tinha feito. Imagine desenhar um rosto de olhos fechados: talvez você coloque os olhos mais ou menos onde olhos costumam ficar, mas não teria como perceber que o segundo caiu na bochecha, ou que a forma desenhada para o cabelo agora está em cima do nariz. Era mais ou menos assim que esses modelos funcionavam, e por isso tantas vezes produziam código perfeitamente válido e visualmente bagunçado.

Uma equipe 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 meio pronto depois de cada etapa e devolve essa imagem ao modelo antes que ele desenhe o próximo traço. A parte realmente

interessante não é que isso ajuda — é o que eles precisaram fazer para que ajudasse.

Como se pontua um desenho?

Quando um artigo diz que seu modelo "bate" outro, é justo perguntar: bate em quê, medido como? Julgar uma imagem gerada é genuinamente difícil — não há uma única resposta certa para "desenhe um laptop" — então a área depende de alguns escores automáticos, cada um um substituto imperfeito para uma pessoa olhando o resultado.

Alguns aparecem neste artigo. **FID** compara o sabor estatístico geral de um lote inteiro de imagens geradas com imagens reais; menor é melhor, mas ele descreve o lote, não uma imagem específica. **CLIP score** pergunta a outra IA se uma imagem combina com as palavras do prompt — útil, mas tão afiado quanto esse juiz. **DINO**, **SSIM** e **LPIPS** comparam uma imagem reconstruída com um alvo, em níveis que vão de pixels brutos a características aprendidas.

Nada disso é verdade. São proxies, e tendem a se mover em pequenos incrementos. Quando você lê que um modelo pontua 127,6 e outro 128,8, isso é uma diferença real na direção pretendida — mas é um empurrão, não uma avalanche, e um empurrão em um número que acompanha só vagamente o que o seu olho diria. Vale guardar isso quando a manchete é "bate um modelo treinado com vinte vezes mais dados".

O que os autores fizeram

O problema que os autores tentaram corrigir é o desenho cego em si. Modelos existentes tratam escrever SVG como uma tarefa puramente textual: prever o próximo pedaço de código a partir do código até ali, sem nunca renderizá-lo. Isso deixa ocioso os poderosos "olhos" — o codificador visual — que modelos multimodais modernos já carregam. Render-in-the-Loop reestrutura o trabalho como um processo visual passo a passo. Depois de cada fragmento de código de desenho, o SVG parcial é renderizado como uma imagem e devolvido ao modelo como uma figura, de modo que o próximo fragmento é escolhido enquanto ele realmente olha para a tela até então.

A primeira descoberta deles é cautelosa, e, para crédito dos autores, relatada claramente: simplesmente aparafusar esse loop em um modelo pronto não funciona. Quando eles deram renderizações intermediárias a modelos gerais fortes sem nenhum treinamento especial, a qualidade não melhorou — ela *piorou* em todos os aspectos. Um modelo que nunca foi ensinado a usar os olhos para isso não passa a saber como fazê-lo de repente.

Assim, a maior parte do trabalho está no ensino. Eles reconstroem os dados de treinamento para que cada desenho seja dividido em muitas etapas pequenas e visualmente significativas — separando formas complexas em peças mais simples para que haja algo novo a ver em cada estágio — e então ajustam finamente um modelo aberto de oito bilhões de parâmetros (construído sobre Qwen3-VL) nessas sequências passo a passo. Eles chamam isso de Visual Self-Feedback. Acrescentam um segundo mecanismo no momento do desenho, Render-and-Verify: antes de aceitar cada novo traço, o modelo o renderiza e verifica se ele de fato mudou a imagem, ou apenas repetiu o traço anterior. Traços que não acrescentam nada são descartados e, quando nada mais ajuda, o modelo é instruído

a parar. Notavelmente, tudo isso roda com um conjunto de dados relativamente pequeno — cerca de 850.000 exemplos, menos da metade do que um rival usou e uma pequena fração do de outro.

O que eles encontraram

- Treinado desse modo, o modelo desenha melhor que sua contraparte cega — mais visivelmente nos casos de falha, em que um modelo cego põe um olho numa bochecha ou ignora um gráfico de barras pedido e entrega um monitor genérico.
- No benchmark padrão (MMSVGBench), o método fica competitivo com, e em várias medidas ligeiramente à frente de, rivais fortes — incluindo OmniSVG, treinado com mais que o dobro dos dados, e InternSVG, treinado com cerca de vinte vezes mais.
- As margens são pequenas. No conjunto de ícones, por exemplo, seu principal escore de qualidade de imagem é 127,6 contra 128,8 do melhor rival, e seu escore de correspondência ao prompt é 0,293 contra 0,291 — diferenças reais e consistentes, mas estreitas.
- As duas peças acrescentadas merecem seu lugar: desligue o treinamento especial, ou a verificação no momento do desenho, e os números caem de forma mensurável. A etapa de verificação, em particular, impede o modelo de ficar preso redesenhando a mesma coisa repetidas vezes.
- O resultado que os próprios autores enfatizam é a eficiência — chegar até aqui com muito menos dados de treinamento do que os líderes usaram.

O que isso não prova

- **Não** mostra que deixar um modelo “ver” é uma vitória gratuita. Pelo contrário: o próprio experimento do artigo mostra que feedback visual *sem* retreinamento piora as coisas. O ganho vem do retreinamento, não dos olhos sozinhos.
- **Não** estabelece uma vantagem grande ou decisiva. Na maioria dos escores, o método fica pescoço a pescoço com os rivais; “bate um modelo treinado com 20× os dados” é verdade, mas por empurrões em métricas proxy, em um benchmark.
- **Não** demonstra habilidade artística geral. É um modelo de pesquisa de oito bilhões de parâmetros desenhando ícones e ilustrações simples em baixa resolução fixa (224×224 pixels), não um designer de uso geral.
- A comparação com um modelo geral grande (GPT-5) **não** é de igual para igual: esse modelo não foi construído nem ajustado para esta tarefa estreita de desenhar por código, então vencê-lo aqui diz pouco sobre qualquer um dos modelos em geral.
- **Não** vem de graça. Renderizar e reler a tela a cada passo torna a geração mais lenta do que emitir o código em uma única passada cega — um custo que os autores reconhecem.

Quão forte é a evidência

- **Sólida** quando é uma comparação controlada em seus próprios termos. As ablações são limpas: remova o treinamento, ou remova a verificação, e os números caem — portanto os dois ingredientes realmente estão fazendo o trabalho que os autores afirmam.
- **Honesta sobre a própria surpresa.** A descoberta de que feedback visual ingênuo *prejudica* é relatada, não enterrada, e é a coisa mais interessante do artigo — uma correção útil à intuição de que mais entrada é sempre melhor.
- **Fina quando é uma afirmação de leaderboard.** As vitórias sobre rivais com mais dados são pequenas e vivem em um único benchmark construído pelos autores de um desses rivais. Margens pequenas em métricas proxy em um conjunto de teste são sugestivas, não resolvidas.
- **Não testada em escala e no mundo real.** Tudo aqui são ícones e ilustrações simples em baixa resolução. Se a mesma ideia vale para design complexo, de alta resolução ou real, fica como trabalho futuro — os autores dizem isso.

Por que importa

A ideia no centro deste artigo é quase embaraçosamente simples, e vai muito além do desenho. Se um programa vai gerar algo escrevendo código — uma página web, um gráfico, um diagrama, uma cena 3D — ele pode escrever tudo às cegas e torcer, ou renderizar enquanto avança e corrigir o rumo. Fechar esse loop é segunda natureza para uma pessoa; olhamos para a página o tempo todo. É um movimento surpreendentemente recente para esses modelos.

O que torna o artigo digno de leitura é o asterisco que ele acrescenta. Dar a um modelo uma forma de ver não é a mesma coisa que ensiná-lo a olhar. Os olhos precisam ser treinados, e só então o loop compensa — e mesmo assim o ganho é real, mas medido: um desenhista mais estável, não um tipo diferente de artista. Para quem constrói ferramentas que transformam uma descrição em um gráfico editável — o tipo de coisa que acaba por trás dos ícones e ilustrações de um aplicativo — a lição silenciosa e prática é a útil. A vitória aqui veio de treinamento mais barato e mais inteligente, não de mais dados. Isso é melhor de aprender do que mais um ponto em um leaderboard.

Resumo limpo

Modelos de linguagem que geram gráficos vetoriais — o código editável e redimensionável por trás da maioria dos ícones e logotipos da web — tradicionalmente fazem isso “às cegas”, escrevendo todos os comandos de desenho sem nunca renderizá-los para ver o resultado. Uma equipe liderada por Guotao Liang propõe Render-in-the-Loop: renderizar o desenho meio pronto depois de cada etapa e devolvê-lo ao modelo, para que ele desenhe o próximo traço olhando para a tela. A descoberta central e honesta é que simplesmente fazer isso com um modelo existente o torna *pior*; o ganho aparece apenas depois que o modelo é retreinado para usar o feedback visual, com ajuda de uma verificação

no momento do desenho que descarta traços que não mudam nada. O modelo retreinado de oito bilhões de parâmetros iguala ou supera ligeiramente rivais treinados com até vinte vezes mais dados em um benchmark padrão — um resultado genuíno, mas por pequenas margens em escores proxy, para ícones e ilustrações simples em baixa resolução. A conclusão que os autores enfatizam, e a que vale guardar, é sobre eficiência: ver bem o próprio trabalho pode substituir escala bruta de dados.

Checagem sem rodeios

O que o artigo mostra: Retreinar um modelo de gráficos vetoriais para renderizar e olhar seu desenho meio pronto, passo a passo, produz resultados melhores e mais completos do que desenhar às cegas — competitivos com, e ligeiramente à frente de, rivais com mais dados em um benchmark padrão, usando menos dados de treinamento.

O que é plausível, mas não provado: Que "ver o próprio trabalho" seja uma receita amplamente melhor do que escalar dados; que a mesma ideia ajude em gráficos complexos, de alta resolução ou do mundo real; que as pequenas margens de benchmark reflitam uma diferença que uma pessoa realmente notaria.

O que não mostra: Que feedback visual ajuda por si só (sem retreinamento, prejudica); que a vantagem sobre rivais é grande ou decisiva; habilidade geral de desenho; uma comparação justa com modelos gerais como GPT-5, que não foram construídos para esta tarefa.

Principais limitações: Um benchmark, construído em parte pelos autores de um rival; margens pequenas em métricas proxy; um modelo de 8B, ícones e ilustrações simples em 224×224; geração mais lenta por causa do loop de renderização a cada etapa.

Quanta confiança um leitor geral deve ter? Alta de que fechar o loop — renderizar e reler enquanto se desenha — realmente ajuda, e de que isso precisa ser treinado, não apenas ligado. Baixa a moderada de que este modelo específico bata decisivamente seus rivais; trate a frase "bate 20× os dados" como um resultado real, mas modesto, de um único benchmark. Alta na ideia que vale lembrar: para código que desenha, olhar enquanto se avança vence 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/