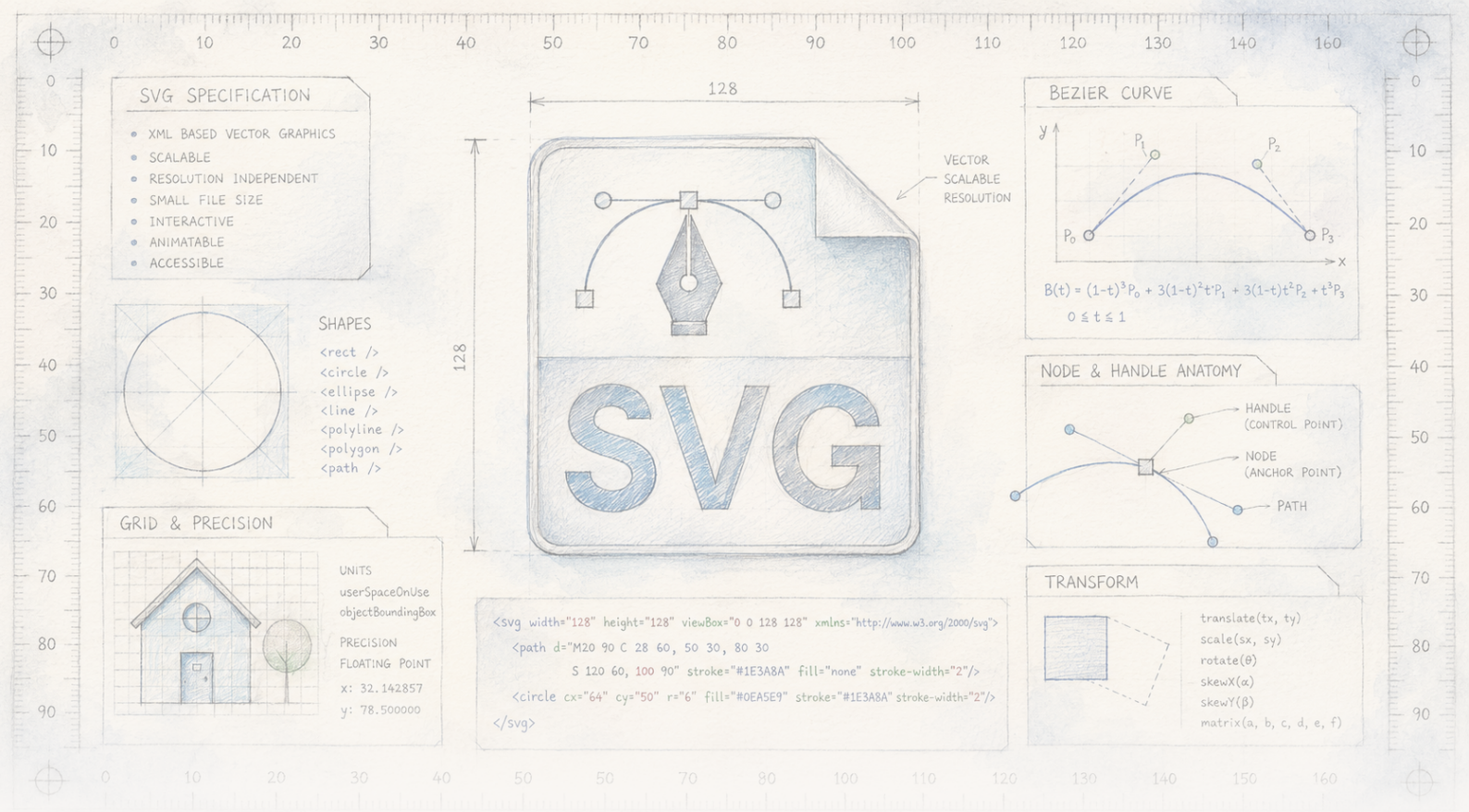


The Clean Paper

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

Jak nauczyć rysującą AI patrzeć na obraz

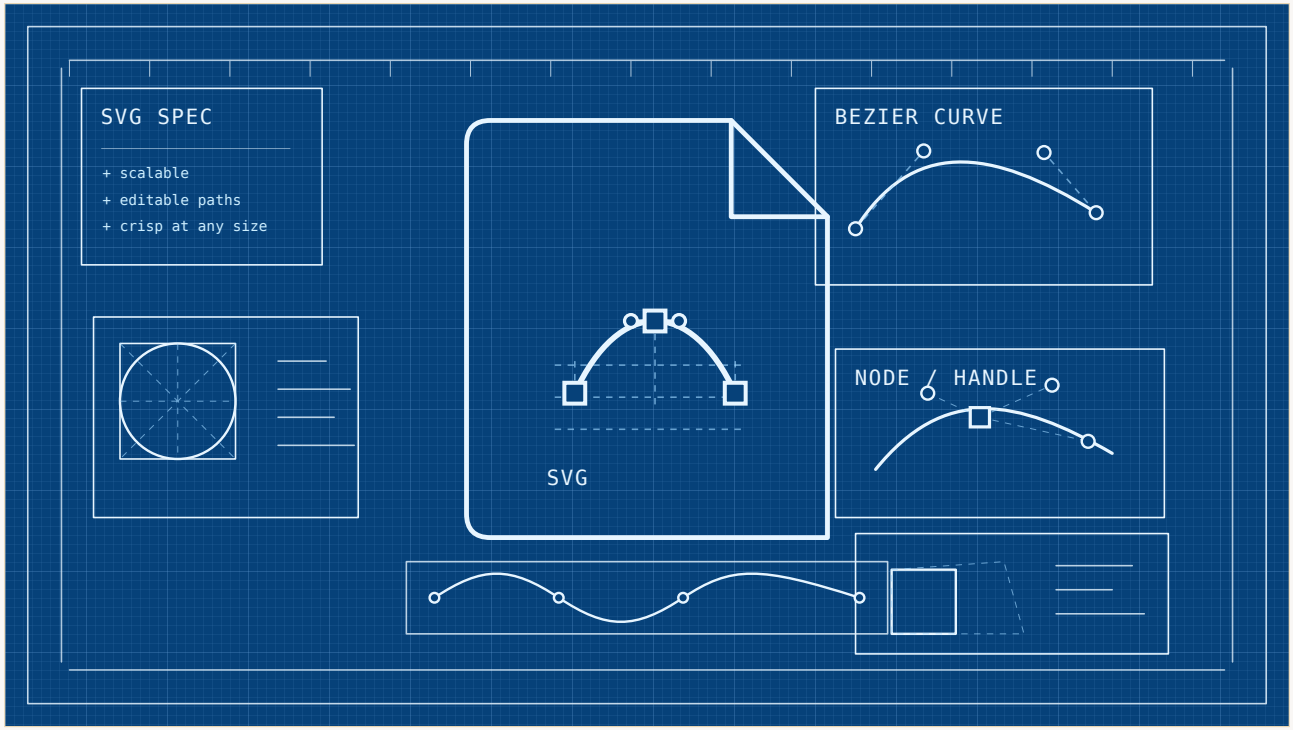
Modele językowe generujące grafikę wektorową robią to na ślepo — zapisując polecenia rysowania bez zobaczenia efektu. Nowa metoda pozwala modelowi obserwować, jak jego własne płótno wypełnia się pociągnięcie po pociągnięciu, a szczerze mówiąc, samo dawanie mu oczu pogarsza sytuację: trzeba go przetrenować, by z nich korzystać. Efektem jest model, który dorównuje lub przewyższa rywali trenowanych na nawet dwudziestokrotnie większej liczbie danych — prawdziwy, staranny wynik na jednym benchmarku, a nie rewolucja.

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)

Rysowanie z zamkniętymi oczami

Gdy poprosimy jeden z dzisiejszych modeli AI o narysowanie obrazu, pod powierzchnią dzieje się jedna z dwóch zupełnie różnych rzeczy. Znane generatory obrazów — te, które z jednego zdania wyczarowują fotografię — bezpośrednio malują piksele i widzą płótno w trakcie pracy. Istnieje jednak drugi, mniej widowiskowy rodzaj rysowania, w którym model wcale nie maluje. Pisze instrukcje: *narysuj tutaj okrąg, tam linię, wypełnij ten kształt na niebiesko*. Instrukcje te są kodem — tym samym formatem Scalable Vector Graphics, czyli SVG, który stoi za większością ikon i logo w internecie — a zaleta jest rzeczywista: wynikiem nie jest stała siatka pikseli, lecz zestaw kształtów, które można bez końca skalować, zmieniać ich kolory i edytować bez utraty ostrości.



Oryginalny rysunek techniczny SVG: sam obraz jest edytowalnym plikiem wektorowym zbudowanym ze ścieżek, linii siatki, węzłów i uchwytów, a nie ze stałych pikseli.

Laura Nesso / The Clean Paper Permission on file

Problem polegał na tym, że do niedawna model piszący taki kod rysunkowy robił to na ślepo. Generował całą sekwencję instrukcji za jednym razem — okrąg, linia, wypełnienie — nigdy ich nie renderując, by zobaczyć rezultat. Wyobraźmy sobie szkicowanie twarzy z zamkniętymi oczami: pierwsze oko może trafić mniej więcej na swoje miejsce, ale nie da się zauważyć, że drugie wylądowało na policzku albo że kształt włosów zasłonił nos. Mniej więcej tak działały te modele, dlatego tak często tworzyły kod całkowicie poprawny formalnie, lecz wizualnie chaotyczny.

Zespół kierowany przez Guotao Lianga zrobił teraz rzecz niemal komicznie oczywistą: pozwolił modelowi otworzyć oczy. Metoda *Render-in-the-Loop* renderuje niedokończony rysunek po każdym kroku i przekazuje obraz z powrotem modelowi, zanim ten wykona kolejne pociągnięcie. Naprawdę interesujące nie jest to, że rozwiązanie pomaga, lecz co trzeba było zrobić, aby w ogóle zaczęło pomagać.

Jak ocenia się rysunek?

Gdy praca mówi, że jej model „pokonuje” inny, uczciwie jest zapytać: w czym dokładnie i jak to zmierzono? Ocena wygenerowanego obrazu jest naprawdę trudna — nie istnieje jedna poprawna odpowiedź na polecenie „narysuj laptop” — dlatego dziedzina opiera się na kilku automatycznych wskaźnikach, z których każdy jest niedoskonałym zastępstwem spojrzenia człowieka.
Kilka z nich występuje w tej pracy. **FID** porównuje ogólne właściwości statystyczne całej partii wygenerowanych obrazów z obrazami rzeczywistymi; niższa wartość jest lepsza, ale wskaźnik opisuje zbiór, nie pojedynczy obraz. **CLIP score** pyta osobny model AI, czy obraz odpowiada słowom polecenia — to użyteczne, lecz tylko na tyle dobre, na ile dobry jest sam sędzia. **DINO**, **SSIM** i **LPIPS** porównują odtworzony obraz z celem na poziomach od surowych pikseli po wyuczone cechy.
Żaden z tych wskaźników nie jest prawdą samą w sobie. To przybliżenia, a ich wartości zwykle zmieniają się niewiele. Gdy czytamy, że jeden model osiąga 127,6, a drugi 128,8, jest to rzeczywista różnica we właściwym kierunku — ale raczej lekkie przesunięcie niż miażdżąca przewaga, i to w liczbie tylko luźno odpowiadającej temu, co powiedziałoby ludzkie oko. Warto o tym pamiętać przy nagłówku „pokonuje model trenowany na dwudziestokrotnie większej ilości danych”.

Co zrobili autorzy

Problemem, który autorzy chcieli rozwiązać, było właśnie ślepe rysowanie. Istniejące modele traktują pisanie SVG jako zadanie czysto tekstowe: przewidują kolejny fragment kodu na podstawie wcześniejszego kodu i nigdy go nie renderują. Potężne „oczy” — koder obrazu — które współczesne modele multimodalne już posiadają, pozostają bezczynne. Render-in-the-Loop przekształca zadanie w wizualny proces krok po kroku. Po każdym fragmencie kodu częściowy SVG jest renderowany do obrazu i przekazywany modelowi, dzięki czemu następny fragment zostaje wybrany z uwzględnieniem tego, co rzeczywiście znajduje się już na płótnie.

Pierwszy wynik jest ostrzeżeniem i autorzy uczciwie go przedstawiają: samo dołączenie tej pętli do gotowego istniejącego modelu nie działa. Gdy podawali pośrednie rendery silnym modelom ogólnym bez specjalnego treningu, jakość nie rosła — *spadała* we wszystkich przypadkach. Model, którego nie nauczono używać wzroku do tego zadania, nie zaczyna nagle wiedzieć, jak to robić.

Większość pracy dotyczy więc nauczania. Autorzy przebudowują dane treningowe tak, by każdy rysunek dzielił się na wiele małych, wizualnie znaczących kroków — rozbijają złożone kształty na prostsze części, aby na każdym etapie pojawiało się coś nowego do zobaczenia — a następnie dostrajają otwarty model o ośmiu miliardach parametrów (oparty na Qwen3-VL) na takich sekwencjach krok po kroku. Nazywają to Visual Self-Feedback. Dodają drugi mechanizm używany podczas rysowania, Render-and-Verify: przed zaakceptowaniem każdego nowego pociągnięcia model renderuje je i sprawdza, czy rzeczywiście zmieniło obraz, czy tylko powtórzyło poprzednie. Pociągnięcia niczego niewnoszące są odrzucane, a gdy nic więcej nie pomaga, model otrzymuje polecenie zakończenia.

Co istotne, wszystko to działa na dość małym zbiorze — około 850 000 przykładów, mniej niż połowie danych jednego rywala i niewielkim ułamku danych drugiego.

Co odkryli

- Tak wytrenowany model rysuje lepiej niż jego ślepy odpowiednik — najbardziej widać to w przypadkach błędów, gdy ślepy model umieszcza oko na policzku albo pomija żądany wykres słupkowy i zamiast niego tworzy zwykły monitor.
- W standardowym benchmarku MMSVGBench metoda jest konkurencyjna wobec silnych rywali, a pod kilkoma względami nieznacznie ich przewyższa — w tym OmniSVG, trenowany na ponad dwukrotnie większej ilości danych, i InternSVG, trenowany na około dwudziestokrotnie większej ilości.
- Marginesy są niewielkie. Na przykład w zbiorze ikon główny wskaźnik jakości obrazu wynosi 127,6 wobec 128,8 u najlepszego rywala, a wskaźnik zgodności z poleceniem 0,293 wobec 0,291 — różnice są rzeczywiste i konsekwentne, lecz małe.
- Oba dodatki okazują się potrzebne: wyłączenie specjalnego treningu albo weryfikacji w czasie rysowania powoduje mierzalny spadek wyników. Zwłaszcza etap weryfikacji zapobiega utknięciu modelu w ciągłym ponownym rysowaniu tego samego elementu.
- Sami autorzy podkreślają przede wszystkim efektywność — osiągnięcie takiego poziomu przy znacznie mniejszej ilości danych treningowych niż u liderów.

Czego to nie dowodzi

- Badanie **nie pokazuje**, że pozwolenie modelowi „widzieć” jest darmowym ulepszeniem. Wprost przeciwnie: własny eksperyment autorów pokazuje, że wizualna informacja zwrotna *bez* ponownego treningu pogarsza wyniki. Korzyść pochodzi z treningu, nie z samych oczu.
- **Nie ustanawia** dużej ani rozstrzygającej przewagi. W większości wskaźników metoda idzie łeb w łeb z rywalami; zdanie „pokonuje model trenowany na 20 razy większej ilości danych” jest prawdziwe, ale chodzi o niewielkie przesunięcia wskaźników zastępczych w jednym benchmarku.
- **Nie demonstruje** ogólnej zdolności artystycznej. To ośmiomiliardowy model badawczy rysujący ikony i proste ilustracje w małej, stałej rozdzielczości 224×224 pikseli, nie uniwersalny projektant.
- Porównanie z dużym modelem ogólnym (GPT-5) **nie jest równorzędne**: model ten nie został zbudowany ani dostrojony do wąskiego zadania rysowania kodem, więc pokonanie go tutaj niewiele mówi o którymkolwiek z modeli w ujęciu ogólnym.
- Rozwiązanie **nie jest bezkosztowe**. Renderowanie i ponowne odczytywanie płótna na każdym etapie spowalnia generowanie w porównaniu z jednorazowym, ślepym wypisaniem kodu — autorzy uznają ten koszt.

Jak mocne są dowody

- **Solidne** tam, gdzie chodzi o kontrolowane porównanie na własnych warunkach. Ablacje są przejrzyste: po usunięciu treningu albo weryfikacji wartości spadają, więc oba składniki rzeczywiście wykonują pracę przypisywaną im przez autorów.
- **Uczciwe wobec własnego zaskoczenia.** Wynik, że naiwna wizualna informacja zwrotna *szkodzi*, jest przedstawiony, a nie ukryty, i stanowi najciekawszy element pracy — użyteczną korektę intuicji, że więcej danych wejściowych zawsze pomaga.
- **Cienkie tam, gdzie mowa o rankingu.** Przewagi nad rywalami trenowanymi na większych zbiorach są małe i występują w jednym benchmarku stworzonym przez autorów jednego z tych rywali. Niewielkie marginesy w zastępczych wskaźnikach na pojedynczym zbiorze testowym są sugestywne, nie rozstrzygające.
- **Niesprawdzone w dużej skali i w praktyce.** Wszystko dotyczy ikon i prostych ilustracji w niskiej rozdzielczości. To, czy pomysł sprawdzi się w złożonych, wysokorozdzielczych lub rzeczywistych projektach, pozostaje zadaniem na przyszłość — autorzy mówią to wprost.

Dlaczego to ma znaczenie

Pomysł stojący w centrum pracy jest niemal zawstydzająco prosty i wykracza daleko poza rysowanie. Jeśli program ma coś wygenerować przez pisanie kodu — stronę internetową, wykres, diagram, scenę 3D — może napisać całość na ślepo i liczyć na szczęście albo renderować po drodze i korygować kierunek. Dla człowieka zamknięcie tej pętli jest drugą naturą; stale zerkamy na stronę. W przypadku tych modeli to zaskakująco nowy krok.

Pracę warto przeczytać ze względu na zastrzeżenie, które dołącza do tej idei. Zapewnienie modelowi możliwości widzenia nie jest tym samym co nauczenie go patrzenia. Oczy trzeba wytrenować i dopiero wtedy pętla przynosi korzyść — a nawet wówczas jest ona rzeczywista, lecz umiarkowana: bardziej niezawodny rysownik, nie nowy rodzaj artysty. Dla osób budujących narzędzia zmieniające opis w edytowalną grafikę — taką jak ikony i ilustracje aplikacji — najcenniejsza jest cicha, praktyczna lekcja. Zwycięstwo wynikało z tańszego, mądrzejszego treningu, nie z większej ilości danych. To lepsza wiedza niż kolejny punkt w rankingu.

Zwięzłe podsumowanie

Modele językowe generujące grafikę wektorową — edytowalny i skalowalny kod stojący za większością internetowych ikon i logo — tradycyjnie robiły to „na ślepo", wypisując wszystkie polecenia rysowania bez renderowania wyniku. Zespół Guotao Lianga proponuje Render-in-the-Loop: po każdym etapie renderować niedokończony rysunek i przekazywać go modelowi, aby kolejne pociągnięcie wykonywał, patrząc na płótno. Główny, uczciwie przedstawiony wynik jest taki, że samo dołączenie tej procedury do istniejącego modelu czyni go *gorszym*; korzyść pojawia się

dopiero po ponownym treningu modelu w używaniu wizualnej informacji zwrotnej, wspomaganym przez kontrolę w czasie rysowania, która odrzuca pociągnięcia niczego niezmieniające. Ponownie wytrenowany model o ośmiu miliardach parametrów dorównuje lub nieznacznie przewyższa w standardowym benchmarku rywali trenowanych na nawet dwudziestokrotnie większej ilości danych — to rzeczywisty wynik, ale osiągnięty małymi marginesami w zastępczych wskaźnikach, dla ikon i prostych ilustracji w niskiej rozdzielczości. Wniosek podkreślany przez autorów i wart zapamiętania dotyczy efektywności: dobre obserwowanie własnej pracy może zastąpić samą skalę danych.

Kontrola bez upiększeń

Co pokazuje praca: Ponowne wytrenowanie modelu grafiki wektorowej tak, by krok po kroku renderował i oglądał własny niedokończony rysunek, daje lepsze i pełniejsze rezultaty niż rysowanie na ślepo — konkurencyjne, a w jednym standardowym benchmarku nieznacznie lepsze od rywali trenowanych na większej ilości danych, mimo mniejszego zbioru treningowego.

Co jest prawdopodobne, ale nieudowodnione: Że „oglądanie własnej pracy" jest szeroko lepszą receptą niż zwiększanie ilości danych; że ten sam pomysł pomoże przy złożonej, wysokorozdzielczej lub rzeczywistej grafice; że małe marginesy benchmarku odpowiadają różnicy dostrzegalnej przez człowieka.

Czego praca nie pokazuje: Że wizualna informacja zwrotna pomaga sama w sobie (bez ponownego treningu szkodzi); że przewaga nad rywalami jest duża lub rozstrzygająca; ogólnej zdolności rysowania; sprawiedliwego bezpośredniego porównania z modelami ogólnymi takimi jak GPT-5, których nie zbudowano do tego zadania.

Główne ograniczenia: Jeden benchmark, częściowo stworzony przez autorów konkurencyjnego modelu; małe marginesy w zastępczych wskaźnikach; model 8B, ikony i proste ilustracje w rozdzielczości 224×224; wolniejsze generowanie przez pętlę renderującą każdy krok.

Jak duże zaufanie powinien mieć czytelnik niespecjalista? Wysokie do tego, że zamknięcie pętli — renderowanie i ponowne oglądanie w trakcie rysowania — rzeczywiście pomaga i musi zostać wytrenowane, nie tylko włączone. Niskie do umiarkowanego do twierdzenia, że ten konkretny model zdecydowanie pokonuje rywali; zdanie o „pokonaniu 20 razy większej ilości danych" należy traktować jako rzeczywisty, lecz skromny wynik z jednego benchmarku. Wysokie wobec jednej idei wartej zapamiętania: przy kodzie, który rysuje, patrzenie po drodze jest lepsze niż rysowanie na ślepo.

Źródła

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/