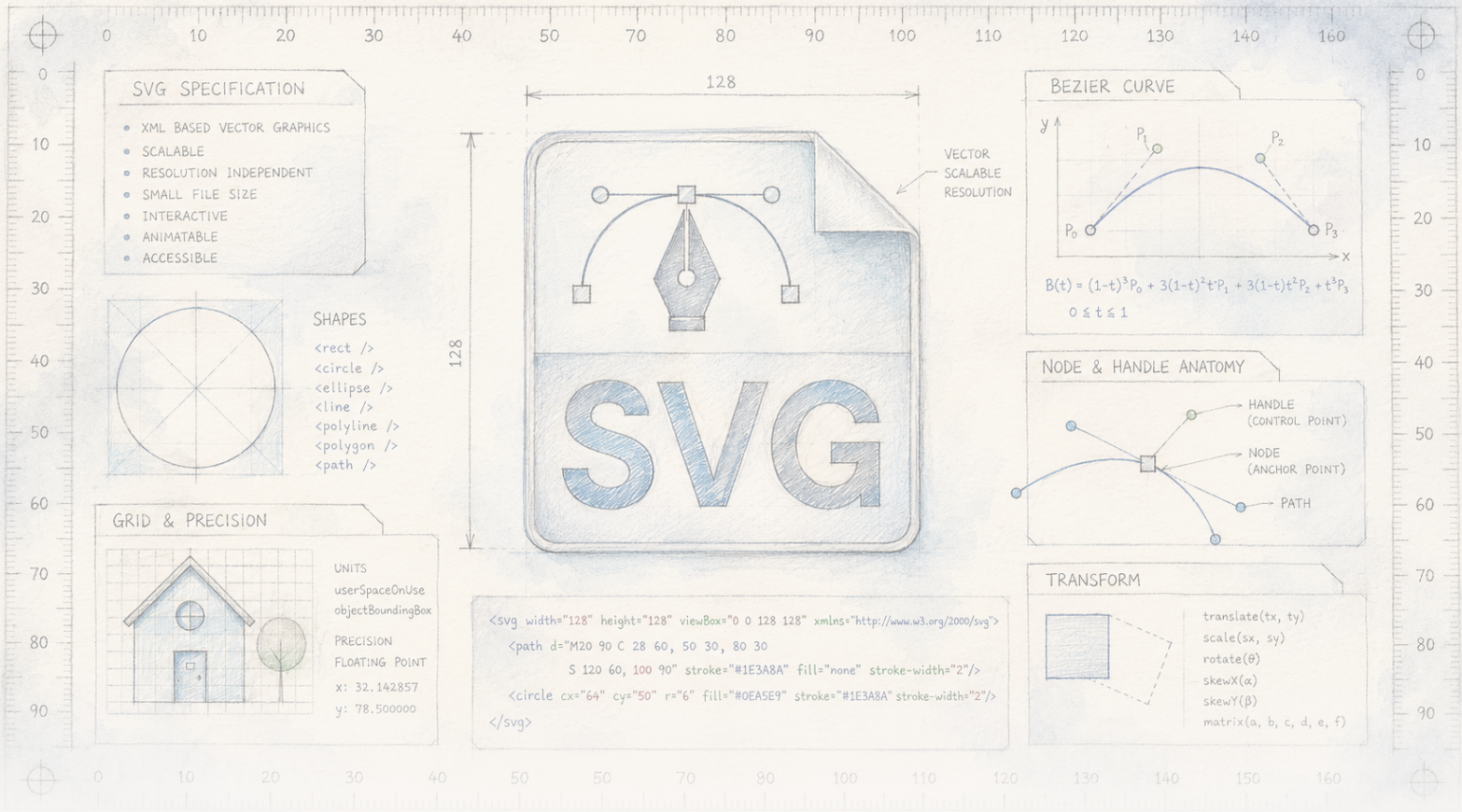


The Clean Paper

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

สอน AI วาดรูปให้มองหน้ากระดาษของตัวเอง

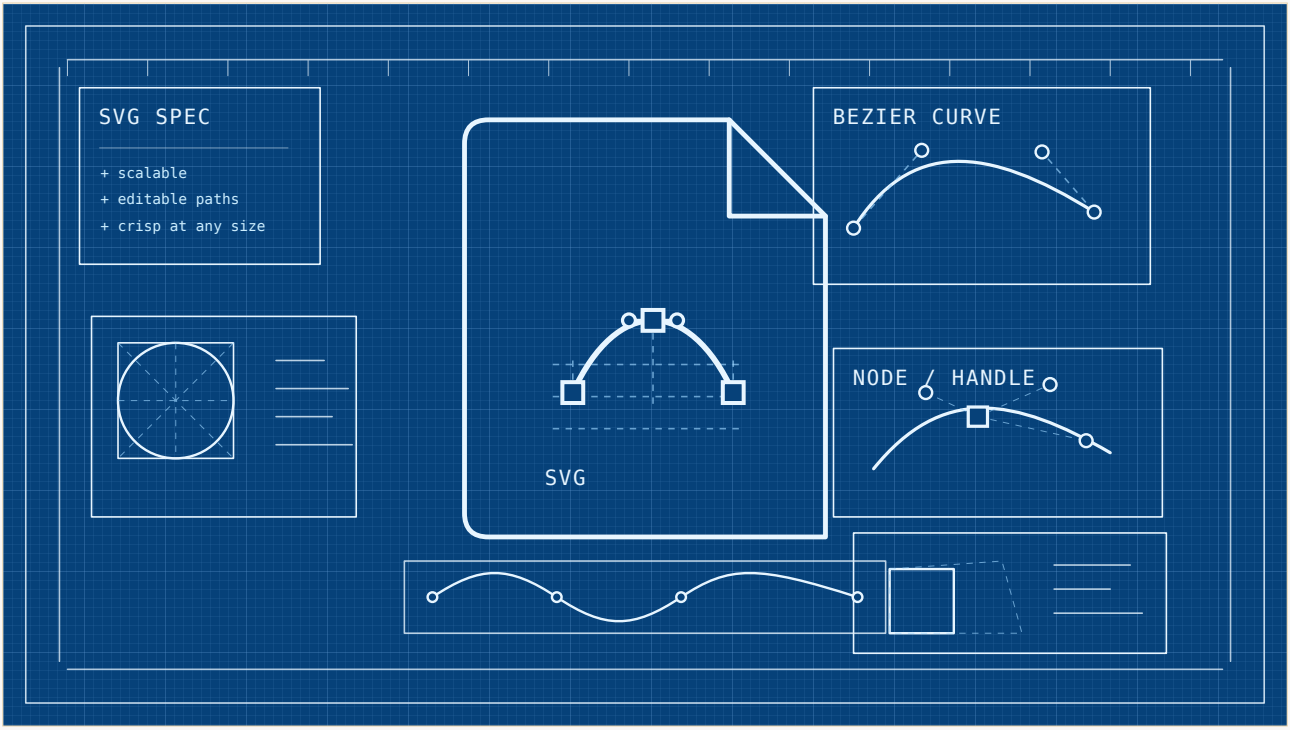
Language models ที่สร้าง vector graphics ทำแบบตาบอด — เขียน drawing commands โดยไม่เคยมองผล วิธีใหม่ให้โมเดลดู canvas ของตัวเองค่อย ๆ เต็มทีละ stroke และจุดหักที่ซื่อตรงคือแค่ให้มันมีตากลับทำให้แย่ลง: ต้อง retrain ให้รู้วิธีใช้สายตา ผลคือโมเดลที่สูสีหรือเฉือนคู่แข่งซึ่งฝึกด้วยข้อมูลมากถึง 20 เท่า — เป็นผลจริงและระวังบน benchmark หนึ่งชุด ไม่ใช่การปฏิวัติ

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)

วาดรูปทั้งที่หลับตา

เวลาขอให้โมเดล AI รุ่นปัจจุบันวาดภาพ เบื้องหลังอาจเกิดสิ่งที่ต่างกันมากสองแบบ เครื่องสร้างภาพที่เราคุ้นเคย — แบบที่สร้างภาพถ่ายจากข้อความ — วาดลงเป็นพิกเซลโดยตรง และมองเห็นภาพที่กำลังสร้างอยู่ตลอด แต่ยังมีการวาดอีกแบบหนึ่งที่เงียบกว่ามาก ซึ่งโมเดลไม่ได้ "ระบายสี" เลย มันเขียนคำสั่งว่า *วาดวงกลมตรงนี้ เส้นตรงตรงนั้น เติมรูปนี้ด้วยสีน้ำเงิน* คำสั่งเหล่านี้คือโค้ดแบบ **Scalable Vector Graphics (SVG)** ซึ่งเป็นรูปแบบเดียวกับที่อยู่เบื้องหลังไอคอนและโลโก้จำนวนมากบนเว็บ ข้อดีคือภาพไม่ได้เป็นตารางพิกเซลตายตัว แต่เป็นชุดรูปทรงที่ขยาย เปลี่ยนสี และแก้ไขได้โดยไม่แตก



ตัวอย่างงาน SVG ต้นฉบับ: ตัวภาพเองเป็นไฟล์เวกเตอร์ที่แก้ไขได้ สร้างจากพาธ เส้นกริด โหนด และจุดควบคุม แทนที่จะเป็นพิกเซลตายตัว

Laura Nesso / The Clean Paper Permission on file

ปัญหาคือ จนไม่นานมานี้ โมเดลที่เขียนโค้ดวาดภาพทำงานแบบ **มองไม่เห็นสิ่งที่ตัวเองวาด** มันสร้างคำสั่งทั้งชุดในครั้งเดียว — วงกลม เส้น การเติมสี — โดยไม่เรนเดอร์ภาพระหว่างทาง ลองนึกถึงการวาดหน้าโดยหลับตา คุณอาจกะตำแหน่งตาได้พอประมาณ แต่ไม่มีทางรู้ว่าตาข้างที่สองหล่นไปอยู่บนแก้ม หรือทรงผมที่วาดทับจมูกแล้ว โมเดลเหล่านี้ก็คล้ายกัน โค้ดอาจถูกต้องตามไวยากรณ์ทุกอย่าง แต่ภาพที่ได้กลับเละเทะ

ทีมที่นำโดย Guotao Liang จึงทำสิ่งที่ฟังดูแทบจะชัดเจนเกินไป: **เปิดตาให้โมเดล** วิธีของพวกเขาเรียกว่า *Render-in-the-Loop* ทุกครั้งที่โมเดลวาดเพิ่มหนึ่งขั้น ระบบจะเรนเดอร์ภาพที่ยังไม่เสร็จแล้วส่งกลับให้โมเดลดูก่อนสร้างเส้นถัดไป ส่วนที่น่าสนใจจริงไม่ใช่แค่ว่าวิธีนี้ช่วยหรือไม่ แต่คือ **ต้องทำอะไรบ้างจึงจะทำให้มันช่วยได้จริง**

ให้คะแนนภาพวาดอย่างไร

เวลางานวิจัยบอกว่าโมเดลหนึ่ง "ชนะ" อีกโมเดล คำถามที่ควรถามคือ ชนะในอะไร และวัดอย่างไร การประเมินภาพที่สร้างขึ้นยากจริง เพราะคำสั่งอย่าง "วาดแล็ปท็อป" ไม่มีคำตอบถูกเพียงภาพเดียว วงการจึงพึ่งคะแนนอัตโนมัติหลาย

> แบบ ซึ่งแต่ละแบบเป็นเพียงตัวแทนหยาบ ๆ ของการให้มนุษย์มองภาพ
> งานนี้ใช้หลายตัว **FID** เปรียบเทียบลักษณะทางสถิติโดยรวมของภาพที่สร้างทั้งชุดกับภาพจริง ค่ายิ่งต่ำยิ่งดี แต่บอกภาพรวมของชุด ไม่ใช่คุณภาพของภาพใดภาพหนึ่ง **CLIP score** ใช้ AI อีกตัวประเมินว่าภาพตรงกับข้อความคำสั่งเพียงใด ซึ่งมีประโยชน์แต่ขึ้นกับคุณภาพของผู้ประเมินนั้นด้วย ส่วน **DINO, SSIM และ LPIPS** ใช้เปรียบเทียบภาพที่สร้างใหม่กับภาพเป้าหมาย ตั้งแต่ระดับพิกเซลไปจนถึงคุณลักษณะที่โมเดลเรียนรู้
> คะแนนเหล่านี้ไม่ใช่ "ความจริง" เป็นเพียงตัวชี้แทน และความต่างมักมีขนาดเล็ก หากโมเดลหนึ่งได้ 127.6 และอีกตัวได้ 128.8 นั่นเป็นความต่างจริงในทิศที่ต้องการ แต่เป็นการขยับเล็กน้อย ไม่ใช่ชัยชนะขาดลอย และยังเป็นการขยับของตัวเลขที่สัมพันธ์กับสิ่งที่ตาคนเห็นเพียงบางส่วนเท่านั้น ข้อนี้ควรจำไว้เมื่อพาดหัวพูดว่า "ชนะโมเดลที่ใช้ข้อมูลฝึกมากกว่า 20 เท่า"

ผู้วิจัยทำอะไร

ปัญหาที่ผู้เขียนต้องการแก้คือการวาดแบบมองไม่เห็น โมเดลเดิมมักมองการเขียน SVG เป็นงานข้อความล้วน: ทำนายโค้ดส่วนถัดไปจากโค้ดที่มีอยู่ โดยไม่เคยเรนเดอร์ออกมาเป็นภาพ ทำให้ "ดวงตา" ที่โมเดลหลายสื่อยุคใหม่มีอยู่แล้ว — ตัวเข้ารหัสภาพ — แทบไม่ได้ถูกใช้

Render-in-the-Loop เปลี่ยนงานนี้ให้เป็นกระบวนการมองและวาดทีละขั้น หลังสร้างโค้ดวาดภาพแต่ละช่วง ระบบจะเรนเดอร์ SVG บางส่วนออกมาเป็นภาพ แล้วส่งภาพนั้นกลับให้โมเดล ดังนั้นคำสั่งชุดถัดไปถูกเลือกขณะที่โมเดลกำลัง **มองเห็นผืนภาพจริง ณ ตอนนั้น**

ผลแรกเป็นคำเตือนที่ดี และผู้เขียนรายงานตรง ๆ: **แค่เอาวงจรนี้ไปต่อกับโมเดลสำเร็จรูปไม่ได้ช่วย** เมื่อพวกเขาส่งภาพระหว่างทางให้โมเดลทั่วไปที่แข็งแรงโดยไม่ฝึกเพิ่ม คุณภาพไม่ได้ดีขึ้น แต่ **แย่ลงทุกด้าน** โมเดลที่ไม่เคยถูกสอนให้ใช้ภาพย้อนกลับแบบนี้ไม่ได้รู้วิธีใช้ประโยชน์จากมันโดยอัตโนมัติ ดังนั้นงานหลักอยู่ที่การฝึก ผู้เขียนสร้างข้อมูลฝึกใหม่โดยแบ่งภาพแต่ละภาพออกเป็นขั้นตอนเล็ก ๆ ที่มีความหมายทางภาพ แยกรูปทรงซับซ้อนเป็นส่วนง่าย ๆ เพื่อให้แต่ละขั้นมีสิ่งใหม่ให้มอง จากนั้นปรับจูนโมเดลเปิดขนาด **8 พันล้านพารามิเตอร์** ที่สร้างบน Qwen3-VL ด้วยลำดับการวาดทีละขั้นนี้ พวกเขาเรียกกลไกนี้ว่า **Visual Self-Feedback** ยังมีอีกกลไกหนึ่งตอนสร้างภาพ เรียกว่า **Render-and-Verify** ก่อนยอมรับเส้นวาดใหม่แต่ละเส้น ระบบจะเรนเดอร์แล้วตรวจว่าภาพเปลี่ยนจริงหรือไม่ หรือเพียงวาดสิ่งเดิมซ้ำ เส้นที่ไม่เพิ่มอะไรจะถูกทิ้ง และเมื่อไม่มีการเพิ่มใดช่วยแล้ว โมเดลถูกกระตุ้นให้หยุด จุดน่าสนใจคือทั้งหมดนี้ฝึกบนข้อมูลประมาณ **850,000 ตัวอย่าง** ซึ่งน้อยกว่าคู่แข่งบางรายมาก

พบอะไร

- เมื่อฝึกด้วยวิธีนี้ โมเดลวาดได้ดีกว่าเวอร์ชันที่เขียนโค้ดแบบมองไม่เห็น โดยเฉพาะกรณีผิดพลาดที่เห็นชัด เช่น วางตาบนแก้ม หรือไม่วาดกราฟแท่งที่ผู้ใช้ขอและใส่จอมอนิเตอร์ทั่วไปแทน
- บนชุดทดสอบมาตรฐาน MMSVGBench วิธีนี้ทำผลงานแข่งขันได้ และในหลายตัวชี้วัดดีกว่าคู่แข่งเล็กน้อย รวมถึง OmniSVG ซึ่งใช้ข้อมูลฝึกมากกว่าสองเท่า และ InternSVG ซึ่งใช้ข้อมูลมากกว่าราวยี่สิบเท่า
- **ช่องว่างมีขนาดเล็ก** ตัวอย่างเช่น ในชุดไอคอน คะแนนคุณภาพภาพหลักคือ 127.6 เทียบกับคู่แข่งที่ดีที่สุด 128.8 และคะแนนความสอดคล้องกับข้อความคือ 0.293 เทียบกับ 0.291 ความต่างไปในทิศเดียวกันแต่บางมาก
- ทั้งสองส่วนของวิธีมีบทบาทจริง หากตัดการฝึกเฉพาะทางออก หรือปิดขั้นตรวจภาพระหว่างการวาด คะแนนลดลงอย่างวัดได้ ขั้นตรวจภาพช่วยป้องกันไม่ให้โมเดลติดอยู่กับการวาดสิ่งเดิมซ้ำ ๆ

- ผลที่ผู้เขียนเน้นเองคือ **ประสิทธิภาพด้านข้อมูล** — ทำได้ถึงระดับนี้ด้วยข้อมูลฝึกน้อยกว่าคู่แข่งชั้นนำมาก

สิ่งที่ผลนี้ไม่ได้พิสูจน์

- **ไม่ได้แสดงว่าให้โมเดล "มองเห็น" แล้วจะดีขึ้นฟรี ๆ** ผลของงานกลับตรงกันข้าม หากให้ภาพย้อนกลับโดยไม่ฝึกใหม่ คุณภาพแย่ลง ประโยชน์มาจากการฝึกให้ใช้ภาพ ไม่ใช่จากการมีภาพเพียงอย่างเดียว
- **ไม่ได้แสดงว่าชนะคู่แข่งอย่างขาดลอย** คะแนนส่วนใหญ่ใกล้กันมาก ประโยคว่า "ชนะโมเดลที่ใช้ข้อมูลมากกว่า 20 เท่า" เป็นจริง แต่ชนะเพียงเล็กน้อยบนคะแนนตัวแทนในชุดทดสอบเดียว
- **ไม่ได้พิสูจน์ความสามารถด้านศิลปะทั่วไป** นี่เป็นโมเดลวิจัยขนาด 8B ที่วาดไอคอนและภาพประกอบง่าย ๆ ที่ความละเอียด 224×224 ไม่ใช่นักออกแบบอเนกประสงค์
- การเปรียบเทียบกับโมเดลทั่วไปขนาดใหญ่ เช่น GPT-5 **ไม่ใช่การเทียบแบบเท่าเทียมกัน** เพราะโมเดลดังกล่าวไม่ได้ถูกสร้างหรือปรับจูนสำหรับงานแคบ ๆ อย่างการเขียนโค้ดวาด SVG
- **ไม่ได้ได้มาฟรีด้านเวลา** การเรนเดอร์และอ่านผืนภาพใหม่ทุกขั้นทำให้สร้างงานช้ากว่าการเขียนโค้ดรวดเดียว ผู้เขียนยอมรับต้นทุนนี้

หลักฐานน่าเชื่อถือเพียงใด

- **ค่อนข้างแข็งแรงในส่วนที่เป็นการเปรียบเทียบควบคุมภายในงาน** การทดสอบตัดส่วนประกอบออกชัดเจน: เมื่อเอาการฝึกเฉพาะทางหรือขั้นตรวจภาพออก คะแนนลดลง แสดงว่าสองส่วนนี้ทำงานจริงตามที่ผู้เขียนอ้าง
- **ซื่อตรงกับผลที่ขัดสัญชาตญาณ** ผลที่ว่าการให้ภาพย้อนกลับแบบง่าย ๆ กลับทำให้แย่ลงถูกนำเสนออย่างชัดเจน ไม่ถูกซ่อนไว้ และอาจเป็นบทเรียนที่น่าสนใจที่สุดของงาน: ข้อมูลเพิ่มไม่ได้ช่วยเสมอไป หากโมเดลไม่ถูกฝึกให้ใช้มัน
- **อ่อนกว่ามากในส่วนที่เป็นข้ออ้างจากตารางอันดับ** ชัยชนะเหนือคู่แข่งที่ใช้ข้อมูลมากกว่ามีขนาดเล็กและเกิดบนชุดทดสอบเดียว ซึ่งบางส่วนสร้างโดยผู้พัฒนาคู่แข่ง ความต่างเล็ก ๆ บนคะแนนตัวแทนในชุดเดียวเป็นสัญญาณที่น่าสนใจ แต่ยังไม่ใช่ข้อสรุปเด็ดขาด
- **ยังไม่ได้ทดสอบในงานใหญ่และโลกจริง** งานทั้งหมดอยู่ที่ไอคอนและภาพประกอบง่าย ๆ ความละเอียดต่ำ ยังไม่รู้ว่าแนวคิดเดียวกันจะทำงานกับกราฟิกซับซ้อน ความละเอียดสูง หรือกระบวนการออกแบบจริงได้ดีเพียงใด

ทำไมจึงสำคัญ

แนวคิดกลางของงานเรียบง่ายจนแทบดูตลก แต่ใช้ได้ไกลกว่างานวาดภาพ หากโปรแกรมสร้างสิ่งใดด้วยการเขียนโค้ด — หน้าเว็บ กราฟ แผนภาพ ฉาก 3 มิติ — มันมีสองทาง: เขียนทั้งหมดโดยไม่มองแล้วหวังว่าจะถูก หรือเรนเดอร์ไปเรื่อย ๆ และแก้ทางตามสิ่งที่เห็น มนุษย์ทำอย่างหลังตลอดเวลา เรามองหน้ากระดาษซ้ำ ๆ ขณะทำงาน สำหรับโมเดล AI นี่กลับเป็นแนวคิดที่เพิ่งถูกนำมาใช้จริงจังไม่นาน

สิ่งที่ทำให้งานนี้น่าอ่านคือเครื่องหมายดอกจันที่มันเติมให้แนวคิดนั้น: **ให้โมเดลมองเห็น ไม่เท่ากับสอนให้มันดูเป็น** ดวงตาต้องถูกฝึกก่อน วงจรจึงให้ผล และแม้เมื่อให้ผลแล้ว ผลก็ยังมีขนาดพอดี ไม่ใช่การเปลี่ยนโมเดลให้กลายเป็นศิลปินชนิดใหม่ แต่เป็นการทำให้ช่างเขียนโค้ดกราฟิกทำงานมีเสถียรภาพขึ้น

สำหรับคนสร้างเครื่องมือที่แปลงคำอธิบายเป็นกราฟิกแก้ไขได้ — สิ่งที่อยู่เบื้องหลังไอคอนและภาพประกอบของแอปจำนวนมาก

— บทเรียนเงียบ ๆ นี้มีค่ามากกว่าอันดับบนตารางคะแนน: **การฝึกที่ฉลาดขึ้นอาจทดแทนการเพิ่มข้อมูลจำนวนมหาศาลได้บางส่วน**

สรุป

โมเดลภาษาที่สร้างกราฟิกเวกเตอร์ — โค้ดที่แก้ไขและขยายได้ซึ่งอยู่เบื้องหลังไอคอนและโลโก้บนเว็บ — เดิมมักทำงานแบบ "หลับตาวาด" เขียนคำสั่งวาดทั้งหมดโดยไม่เรนเดอร์ดูผลระหว่างทาง ทีมของ Guotao Liang เสนอ Render-in-the-Loop: เรนเดอร์ภาพที่ยังไม่เสร็จหลังแต่ละขั้น แล้วส่งกลับให้โมเดลดูก่อนวาดต่อ ผลสำคัญและซื่อตรงที่สุดคือ **แค่เพิ่มภาพย้อนกลับให้โมเดลเดิมไม่ได้ช่วย แต่ทำให้แย่ลง** ประโยชน์เกิดเมื่อฝึกโมเดลใหม่ให้ใช้ข้อมูลภาพนี้ และเพิ่มขั้นตรวจสอบตอนวาดเพื่อตัดเส้นที่ไม่เปลี่ยนภาพออก โมเดลขนาด 8 พันล้านพารามิเตอร์ที่ฝึกใหม่ทำคะแนนเทียบเท่าหรือดีกว่าคู่แข่งเล็กน้อยบนชุดทดสอบมาตรฐาน แม้ใช้ข้อมูลน้อยกว่าบางรายถึงราว 20 เท่า แต่ช่องว่างมีขนาดเล็กและวัดด้วยคะแนนตัวแทน บนไอคอนและภาพง่าย ๆ ความละเอียดต่ำ สิ่งที่ควรจำคือประสิทธิภาพด้านข้อมูล: เมื่อโมเดลถูกฝึกให้มองงานของตัวเองอย่างถูกวิธี การมองระหว่างทางช่วยได้จริง

ประเมินแบบไม่เกินจริง

งานวิจัยแสดงอะไร: การฝึกโมเดลกราฟิกเวกเตอร์ให้เรนเดอร์และมองภาพที่ยังไม่เสร็จทีละขั้น ทำให้ผลลัพธ์ดีและครบกว่าการเขียนโค้ดวาดภาพแบบมองไม่เห็น และทำผลงานแข่งขันได้หรือดีกว่าคู่แข่งเล็กน้อยบนชุดทดสอบมาตรฐาน ทั้งที่ใช้ข้อมูลฝึกน้อยกว่า

อะไรที่เป็นไปได้แต่ยังไม่พิสูจน์: แนวคิด "มองงานของตัวเองระหว่างทำ" อาจใช้ได้กว้างกว่าการเพิ่มข้อมูล และอาจช่วยงานกราฟิกซับซ้อนหรือความละเอียดสูงในโลกจริง แต่ยังไม่ได้ทดสอบ

มันไม่ได้แสดงอะไร: ไม่ได้แสดงว่าภาพย้อนกลับช่วยได้เองโดยไม่ฝึกใหม่ ไม่ได้แสดงชัยชนะขาดลอยเหนือคู่แข่ง ไม่ได้พิสูจน์ความสามารถวาดภาพทั่วไป และไม่ได้เป็นการเทียบอย่างยุติธรรมกับโมเดลทั่วไปขนาดใหญ่ที่ไม่ได้สร้างมาสำหรับงานนี้

ข้อจำกัดหลัก: ชุดทดสอบเดียว ช่องว่างคะแนนเล็กและเป็นคะแนนตัวแทน โมเดล 8B งานส่วนใหญ่เป็นไอคอนและภาพง่าย ๆ ที่ 224×224 และการเรนเดอร์ทุกขั้นทำให้สร้างช้าลง

ผู้อ่านทั่วไปควรมั่นใจแค่ไหน? มั่นใจสูงว่าการปิดวงจร — เรนเดอร์แล้วมองงานของตัวเองระหว่างวาด — ช่วยจริงเมื่อโมเดลถูกฝึกให้ใช้ข้อมูลนั้น และมั่นใจสูงว่าแค่เปิดภาพให้ดูโดยไม่ฝึกกลับไม่ช่วย ส่วนข้ออ้างว่าโมเดลนี้ "ชนะระบบที่ใช้ข้อมูลมากกว่า 20 เท่า" ควรอ่านเป็นผลจริงแต่มีขนาดเล็กบนชุดทดสอบเดียว บทเรียนที่น่าจำที่สุดคือ: สำหรับโค้ดที่ใช้วาดภาพ **มองระหว่างทำ ดีกว่าวาดแบบหลับตา**

แหล่งข้อมูล

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/