



ebook ฟรี · สรุปลงแล้ว

บอก 20 ตัว คนเดียวคุณ

สรุปหลักการ Grok Bot: วิธีให้ทีม AI ทำงาน 1,000+ ชิ้นต่อ
เดือน โดยระบบไม่พัง



เรียบเรียงโดยทีม Kruoop · ตุลาคม 2026
kruoop.com

เริ่มต้น

เรื่องนี้คืออะไร

วิศวกรคนหนึ่งตื่นขึ้นมา เจองาน 20 ชิ้นที่บอท AI ทำเสร็จและรวมเข้าระบบหลักไว้ให้แล้ว ไล่อ่านดูทีละชิ้น คุณภาพใช้ได้

คนนั้นคือ **Lauren Tan (@poteto)** วิศวกรที่เคยทำงานที่ Cursor, Meta และ Netflix (เคยเป็นผู้จัดการทีมวิศวกร 2 ปี) ปัจจุบันนำทีม Grok Bot และเปิดชุดเครื่องมือที่ใช้คุมบอทให้คนอื่นใช้ฟรี

3,000+

งานที่ส่งในเวลา 5 เดือน

1,000+

งานต่อเดือน (ทีมตั้งเป้า 2,000+)

20+

บอทที่ทำงานพร้อมกัน

10,000

ครั้งที่ทีม Cursor ใช้สกิลชุดนี้ใน 1 สัปดาห์

"PR" คืออะไร

PR (Pull Request) คืองานแก้โค้ด 1 ชิ้นที่ส่งมาให้ตรวจสอบก่อนรวมเข้าระบบหลัก ในเล่มนี้คิดง่าย ๆ ว่า **1 PR = งาน 1 ชิ้น**

ความเชื่อใจมาจากการตรวจงาน

ที่กล้าปล่อยให้บอกทำงาน ไม่ใช่เพราะ AI เก่งขึ้น แต่เพราะมีระบบตรวจงานที่เชื่อถือได้

ปัญหาจริงของ AI ไม่ใช่ "เขียนไม่เป็น" แต่คือ มันเดาโดยไม่รู้ว่ากำลังเดา

ตัวอย่างจริง

บอทยืนยันสาเหตุของบักอย่างมั่นใจ แต่พอเปิดบันทึกการทำงานดู กลับพบว่ามันไม่เคยเปิดไฟล์ที่เกี่ยวข้องเลยสักครั้ง

เส้นทางความเชื่อใจ



ความเชื่อใจโตตามระบบตรวจงาน ไม่ได้โตตามความเก่งของโมเดล และวิธีคุมบอทก็คล้ายวิธีคุมทีมคนมาก

5 หลักการ แกน

1

ตรวจสอบสำคัญกว่าเขียนพรอมต์

อย่าแก้ด้วยการสั่งให้ยาวขึ้น ให้สร้างวิธีพิสูจน์ว่างานถูกจริง

2

กฎแข็งชนะกฎอ่อน

ฝังกฎไว้ในโครงสร้างระบบ อย่าหวังให้บอททำได้เอง เพราะบอทลืมได้เสมอ

3

ทางที่ง่ายที่สุดต้องเป็นทางที่ถูก

บอทจะเลือกทางลัดเสมอ งานของเราคือออกแบบให้ทางลัดคือทางที่ถูก

4

บอก 1 ตัว ทำ 1 หน้าที่

แต่ละตัวมีความจำของตัวเอง ข้อมูลไม่ปนกัน งานไม่มั่ว

5

รับเฉพาะหลักฐานที่ทำซ้ำได้

ไม่รับคำว่า "น่าจะแก้แล้ว" ต้องเห็นว่าก่อนแก้เจอปัญหา และหลังแก้ปัญหาย่อยไป



ให้ "ตา" กับ AI

ถ้าบอทรันงานเองไม่ได้ คนต้องมานั่งตรวจทุกชิ้น และคนจะกลายเป็นคอขวดทันที

Control Glass

สกลิตที่ให้บอทเปิดแอปเอง คลิกเอง จับภาพหน้าจอ อ่าน error และวัดความเร็วได้จริง ไม่ต้องเดาจากการอ่านโค้ดอย่างเดียว

Feature Map: แผนที่ทุกฟีเจอร์

เขียนทุกฟีเจอร์จากมุมมองผู้ใช้ ให้บอทรูว่าจะไปทดสอบที่ไหน อย่างไร

หัวข้อ	ตัวอย่าง
ทางเข้า	เมนูไหน หน้าไหน
ปุ่มลัด	กดปุ่มอะไรเพื่อเปิด
ปุ่มอยู่ตรงไหน	ตำแหน่งบนหน้าจอที่บอทต้องคลิก
ใช้งานอย่างไร	ขั้นตอนที่ผู้ใช้ทำจริง
ตรวจอย่างไร	ภาพหน้าจอต้องตรง · ลื่นต่ำกว่า 55 FPS = ไม่ผ่าน

ผลที่ได้: รายงานบั๊กที่เคยมีแค่ "???" กับภาพหน้าจอ กลายเป็นบั๊กที่บอททำซ้ำและแก้ได้เอง

บอกกะดึก และการสอบแบบปิดตา

Benny · บอกกะดึก

- รับรายงานบั๊กจาก Slack
- เปิดเครื่องบนคลาวด์ แล้วทำให้บั๊กเกิดซ้ำ
- แก้ แล้วส่งงานกลับมาพร้อมหลักฐาน ไม่ใช่แค่บอกว่า "แก้แล้ว"

ทดสอบสกิลแบบปิดตา

สกิลที่เขียนให้บอทใช้ ต้องผ่านการสอบก่อนใช้จริง

- ให้บอทย่อยหลายตัวทำงานในโพลเดอร์ที่ตั้งชื่อไม่ให้รู้วากำลังทดสอบอะไร
- ให้โมเดลคนละค่ายเป็นกรรมการ กันการเข้าข้างตัวเอง
- วนแก้จนได้คะแนน 10/10



กันพลาดด้วย โครงสร้าง

ถ้ามีแค่คำสั่ง สกิล และคู่มือ ไม่เข้าก็เร็วระบบจะละ ทางแก้คือทำให้ วิธีที่ถูกเป็นวิธีเดียวที่ทำได้ ทีมเรียกสถาปัตยกรรมนี้ว่า "Dune"

ชั้น 1 · แฉงที่สดุ

โครงสร้างระบบ

แยกโฟลเดอร์และส่วนงานให้ทำผิดไม่ได้ตั้งแต่ต้น บอทชอบลอกแบบที่เห็นอยู่แล้ว



ชั้น 2 · แฉง

ระบบตรวจสอบอัตโนมัติ

CI และ lint ทำผิดเมื่อไรขึ้นสีแดง รวมงานไม่ได้



ชั้น 3 · อ่อนที่สดุ

คำสั่งและสกิล

บอทลืมได้ พลาดได้ ต้องคอยสุ่มตรวจ



ต้นทุน: การปรับระบบให้เป็นแบบนี้ใช้ไป 600+ PR ส่วนใหญ่เป็นการเคลียร์หนี้เทคนิคที่สะสมไว้

บทที่ 3 (ต่อ)

อะไรที่ AI ทำไม่ได้ ห้ามไปเลย



ห้ามใช้ useEffect

จุดที่คนเขียน React พลาดบ่อยที่สุด เขียนเมื่อไรระบบตรวจไม่ผ่านทันที



ห้ามเขียนคอมเมนต์ในโค้ด

ตามบทความ คอมเมนต์ที่ AI เขียนเกือบทั้งหมด (99%) เป็นเรื่องไม่เกี่ยวหรือแต่งขึ้นเอง



ห้ามเรียกข้ามส่วนงาน

แยกส่วนหลังบ้านกับส่วนหน้าจอไว้คนละโฟลเดอร์ ถ้าเรียกข้ามกัน ระบบตรวจไม่ผ่าน ป้องกันแอปกระตุก

กฎ 3 ครั้ง

ถ้าต้องพิมพ์ "อย่าเขียนแบบนี้" เกิน 3 ครั้ง ให้เปลี่ยนเป็นกฎที่ระบบตรวจให้อัตโนมัติ

pstack: ชุดเครื่องมือคุณบอก

ชุดสกิลที่เปิดเป็นโอเพนซอร์สในรูปแบบลึกลับของ Cursor ใช้ได้ฟรี หัวใจคือคำสั่ง `/poteto-mode` ซึ่งทำหน้าที่เป็นตัวแจกงาน

1

จับคู่งานกับคู่มือ

ดูว่างานนี้คือแก้บั๊ก ทำฟีเจอร์ หรือรีแพกเตอร์

2

คัดลอกขั้นตอนเป็นรายการงาน

ทำตามคู่มือทีละข้อ ไม่ดันสด

3

แจกงานให้โมเดลที่ถนัด

แต่ละโมเดลเก่งไม่เหมือนกัน

4

ต้องมีหลักฐานก่อนบอกว่าเสร็จ

ไม่มีหลักฐาน = ยังไม่เสร็จ

21

หลักการ

22

คู่มือ

23

สกิล

คำสั่งหลัก และการแบ่งงานโมเดล

คำสั่ง	ทำอะไร
<code>/loop</code>	วนทำงานกว่างานจะเสร็จ
<code>/goal</code>	ตั้งเป้าหมายยาว ติดตามข้ามรอบได้
<code>/swarm</code>	แบ่งงานให้หลายบอททำพร้อมกัน
<code>/arena</code>	ให้หลายโมเดลออกแบบแข่งกัน แล้วเลือกแบบที่ดีที่สุด
<code>/how</code>	ไต่ถามว่าระบบทำงานอย่างไร
<code>/why</code>	ค้นหามาว่าทำไมถึงออกแบบแบบนี้
<code>/interrogate</code>	ให้หลายโมเดลช่วยกันรีวิว
<code>/babysit</code>	เฝ้างานจนพร้อมรวมเข้าระบบ

ใช้โมเดลตามความถนัด

Sol

เขียนโค้ดที่ต้องละเอียด

Grok

งานเร็วและงานซ้ำ ๆ

Fable

งานที่ต้องใช้วิจารณ์ญาณ

Opus

เป็นทีมรีวิว

ทีมบอก ตัวหน้าที่

บอกแต่ละตัวดูผลงานด้านเดียว มีความจำของตัวเอง เหมือนทีมที่มีผู้เชี่ยวชาญประจำแต่ละสถานี

iOS

Baltata

แอป iPhone และมือถือ

PC

Shaoruru

แอปเดสก์ท็อป และระบบ CI/CD

🔧

Hogan

โครงสร้างระบบ และปัญหาจากผู้ใช้

A

Craig

แอป Android

✓

Quill

ระบบทดสอบ

☰

Jenny

ฝ่ายปฏิบัติการ: ประชุมเช้า รับคนใหม่
สรุปบทเรียน อัปเดตคู่มือ

สายพานงาน ตั้งแต่รับจนรวม

1

รับงาน

ตัวแฉ่งงานส่งให้บอทที่ถนัด

2

บอกหลายตัวทำพร้อมกัน

ระบบเบื่องหลังขยายจาก 15 เป็น 200+ เครื่อง

3

ตรวจอัตโนมัติ

ภาพหน้าจอ ความเร็ว และ log

4

รอบตรวจทุก 30 นาที

เสียงต่ำ → รวมเอง

เสียงสูง → คนดู

5

รวมเข้าระบบหลัก

ตรวจตอนกลางคืน

00:00

ดู PR ที่ค้าง

01:00

เช็คผล CI

02:00

แนวโน้มน
คุณภาพ

03:00

ออกรายงาน

07:00

สรุปเข้า Slack

เหตุฉุกเฉิน (P0): ตรวจเจอ → สร้างบอทแก้ทันที → ลองหลายทางพร้อมกัน
→ ปล่อยขึ้นระบบ → แจ้งคนเวร

วิศวกรยุคใหม่ = หัวหน้าเชฟ

ไม่ได้ลงมือผิดเองทุกงาน แต่ทำ 4 อย่างนี้



ออกแบบครัว

วางสถาปัตยกรรมให้ทำผัดยาก



จัดสถานี

บอทแต่ละตัวดูผลงานด้านเดียว



แจกงาน

ส่งงานให้คนหรือบอทที่ถนัด



ชิมก่อนเสิร์ฟ

ตัดสินใจงานไหนถูกจริง

จุดแข็งของวิศวกรเปลี่ยนจาก "เขียนโค้ดเป็น" เป็น "ตัดสินใจว่าโค้ดไหนถูก"

ผลข้างเคียงที่น่าสนใจ: PM และดีไซเนอร์ก็ส่งโค้ดเข้าระบบได้ ไม่ใช่เพราะเขียนโค้ดเป็น แต่เพราะระบบกันพลาดไว้ให้แล้ว

เริ่มต้น 7 ขั้นตอน

- 1 สร้างระบบตรวจสอบงานก่อนเขียนพรีเซนเตชันให้ข้อมูลที่ทีมงานจริงและทำซ้ำได้
- 2 บังคับด้วยโครงสร้างระบบ ไม่ใช่ด้วยเอกสาร
- 3 ทดสอบสเกลแบบปิดตา และให้โมเดลคนละค่ายเป็นกรรมการ
- 4 เริ่มจากบอทหน้าที่เดียวก่อน แล้วค่อยต่อเป็นทีม
- 5 คอมเมนต์ที่ต้องพูดซ้ำ ให้เปลี่ยนเป็นกฎอัตโนมัติ
- 6 ทำ **Feature Map** และเครื่องมือดูการทำงานจริง ก่อนเพิ่มจำนวนบอท
- 7 เพื่องานเคลียร์หนี้เทคนิคไว้เยอะ (กรณีนี้ใช้ 600+ PR)



เอา 5 หลักการ ไปใช้กับงานคุณ

หลักการ	ใช้กับ AI ในงานประจำวัน
ตรวจ > สั่ง	ให้ AI แแนบที่มาหรือลิงก์ทุกครั้ง แล้วสุมเช็คเอง
กฎแข็ง > กฎอ่อน	ใช้เทมเพลตหรือฟอร์มบังคับรูปแบบ แทนการพิมพ์ สั่งยาว ๆ ทุกครั้ง
ทางง่าย = ทางถูก	ให้ตัวอย่างงานดี ๆ ไปเลย AI ชอบเลียนแบบสิ่งที่เห็น
1 แชน 1 หน้า	แยกแชนเขียนแคปชัน ทำรูป ตอบลูกค้า ไม่ปนกัน
หลักฐานก่อนเชื่อ	ตัวเลข ราคา กฎหมาย ต้องมีที่มาก่อนเอาไปใช้จริง

กฎ 3 ครั้ง ฉบับคนทั่วไป

ถ้าต้องแก้ AI เรื่องเดิมเกิน 3 ครั้ง ให้เขียนลงคำสั่งถาวรของแชนหรือโปรเจกต์
ไปเลย

หน้านี้เป็นการศึกษาของทีม Kruoop ไม่ได้มาจากต้นฉบับโดยตรง



อยากลองทำจริง?

มาเรียน ลงมือทำ และแลกเปลี่ยนกับคนที่ใช้ AI ทำงานจริง

kruoop.com

แหล่งที่มา

- happyrock.cloud: SpaceX AI Engineer's Grok Bot in Practice (2 ก.ย. 2026) ซึ่งเรียบเรียงต่อจากบทความ 36Kr
- cursor.com: pstack, the skills behind 10,000 agent runs
- daily.dev: A deep dive into pstack

หมายเหตุ

ebook นี้เรียบเรียงใหม่ด้วยภาษาของทีม Kruoop จากแหล่งสาธารณะ รายละเอียดบางส่วน เช่น ตารางตรวจกลางคืน ชื่อโมเดลที่ใช้ และรอบตรวจ 30 นาที มาจากบทความสรุปรื้อสอง ยังไม่ได้ยืนยันกับเจ้าของเรื่องโดยตรง Kruoop ไม่มีส่วนเกี่ยวข้องกับ xAI, SpaceX หรือ Cursor