คู่มือเริ่มต้นเป็น AI Engineer อย่างเป็นระบบ

วางพื้นฐาน เข้าใจ LLM สร้าง RAG
และจัดการ Local LLM สำหรับโปรแกรมเมอร์สาย AI

Contents

บทนำ **9**

บทที่ 1 — AI Engineer คือใครในยุค LLM **10**

ทำไม role นี้เพิ่งโผล่มาในเรดาร์ของทุกคน 10

นิยามที่ใช้ได้จริงในวันทำงาน 11

ต่างจาก ML Engineer / Data Scientist / Software Engineer ยังไง 13

ความเข้าใจผิดที่ขวางคุณอยู่ตอนนี้ 15

คุณอยู่ใกล้ AI Engineer แค่ไหนแล้ว — Self-Assessment 10 ข้อ 16

กับดักที่เห็นบ่อยตอนเริ่มต้น 17

สรุปสั้น แล้วเดินต่อ 18

ก้าวต่อไปสู่บทที่ 2 18

บทที่ 2 — Skill Stack ที่ต้องมีก่อนสร้างระบบ AI จริง

ปัญหาที่ซ่อนอยู่ในคำว่า “AI roadmap”

รูปร่างของ stack: 3 Layer และเหตุผลที่ลำดับสำคัญ

10 หมวด skill ที่ AI Engineer ควรมี

T-Shape: กว้างทุกหมวด ลึกในหมวดที่ใช้จริง

Self-Assessment Worksheet: หา Gap ของตัวเอง

6-Month Roadmap

Common Traps ที่ทำให้ roadmap พัง

Quick Recap

ลองทำ: ขั้นต่อไปสำหรับสัปดาห์นี้

บทที่ 3 — Machine Learning แบบไม่จมคณิตศาสตร์

เปิดด้วยคำถามที่หลายคนกลัวจะถามออกมาดังๆ

ML ในภาษาคน — เรื่องของ pattern ไม่ใช่สูตร

Train vs Inference — เหมือนเรียนภาษาอังกฤษ vs ใช้ภาษาอังกฤษ

Overfitting — นักเรียนที่ท่องข้อสอบเก่าได้หมดแต่สอบจริงตก

ML Pipeline — เดินทางของข้อมูลจากต้นทางถึง output

Metric ที่สำคัญ — และเมื่อไหร่ที่มันโกหก

ทำไมเรื่องนี้สำคัญสำหรับงาน LLM (สะพานไปบท 4)

Try this: เปิด model card หนึ่งใบ แล้วระบุ task กับ metric

สิ่งที่ต้องจำให้ได้

Mini Glossary — 15 คำที่จะเจอตลอดเล่ม

บทที่ 4 — จาก Neural Network สู่ Transformer และ LLM

ย้อนกลับสั้นๆ ก่อนเข้าเรื่อง: neural network คืออะไรกันแน่

RNN และ LSTM: ความหวังที่ติดเพดาน

Attention: ไอเดียที่ปลดล็อกทุกอย่าง

Transformer (2017): เอา recurrence ออกทั้งหมด

Timeline: 64 ปีของชิ้นส่วนที่ค่อยๆ ต่อจนครบ

Pre-training vs Post-training: “เรียนภาษา” กับ “เรียนมารยาท”

เรื่อง “emergent abilities” ที่ต้องเล่าให้ครบทั้งสองด้าน

Try this: เปิด model card ของ Llama 3 จริง

สิ่งที่ควรเก็บกลับไป

ทางเชื่อมไปบทถัดไป

สำหรับใครที่อยากเรียนต่อจากตรงนี้

บทที่ 5 — LLM ทำงานอย่างไรในมุม Engineer

ทำไม call สองถึงแพงกว่า call แรก 26 เท่า

Token: หน่วยที่ LLM มองเห็นจริงๆ

Tokenization: ภาษาไทย vs ภาษาอื่น (ตัวเลขจริง)

Context window: โต๊ะทำงานที่มีพื้นที่จำกัด

LLM generate ทีละ token — และทำไมเรื่องนี้สำคัญต่อบิล

Temperature, Top-P, Top-K: ลูกเต๋าถ่วงน้ำหนัก

ทำไม Temperature = 0 ก็ยัง "ไม่เท่ากัน 100%"

System prompt: ของที่อยู่บนโต๊ะตลอดเวลา (และคิดเงินทุก call)

Prompt Caching: วิธีจ่ายค่า system prompt 10%

Latency: TTFT, TPOT และอะไรที่ engineer คุมได้

Worksheet: คำนวณ cost project ของคุณ

นับ token เอง: Python script ที่ใช้ได้จริง

สิ่งที่ควรเก็บกลับไป

ทางเชื่อมไปบทถัดไป

บทที่ 6 — สร้าง LLM App แรกด้วย API และ Code

50 บรรทัดแรกที่บิลเป็นเงินจริง

ก่อนเขียน code: ให้ key อยู่ในที่ของมัน

API call แรก: 15 บรรทัดที่ทำงานได้

เลือก SDK ไหน: decision tree

Streaming: ทำไม UX ต่างกันโดยที่ compute เท่าเดิม

Error ที่จะเจอจริง: rate limit, timeout, retry

Mini project: file summarizer ใน <100 บรรทัด

Logging ที่จะกลายเป็น eval set พรุ่งนี้

Quick recap

ต่อจากนี้

บทที่ 7 — Prompt Engineering สำหรับงานจริง

ทำไม prompt ที่เก่งใน ChatGPT พอใส่ API แล้วพัง

Prompt คือ code ไม่ใช่เวทมนตร์

System prompt = JD ของพนักงานใหม่

Few-shot: ใช้เมื่อจำเป็น ไม่ใช่ทุกครั้ง

Prompt Template Library: 5 patterns ที่ใช้บ่อย

Structured Output: จาก “JSON-ish” ไปสู่ JSON ที่รับประกัน

Tool Calling: ให้ model “โทรหาแผนกอื่น”

Chain-of-Thought (CoT): มุมมองที่ honest

กับดักที่พบบ่อย

Try this: Extract structured ticket ใน 20 นาที

สรุปสั้นๆ

บทที่ 8 — RAG: ให้ AI ตอบจากเอกสารและความรู้ของเรา

RAG จริงๆ คืออะไร — และสองความหมายที่ปนกัน

Pipeline 7 ขั้น: สิ่งที่เกิดขึ้นจริงใต้ฝา

8 จุดที่ RAG พังบ่อย — และวิธีคิดเรื่อง debug

Chunking strategies: ตัดยังไงให้ความหมายไม่หาย

เมื่อ “ดึงเสร็จแล้ว” ยังไม่พอ: Reranking และ Citation

Prompt vs RAG vs Fine-tuning — เลือกอะไรเมื่อไหร่

เมื่อไหร่ที่ ไม่ ควรใช้ RAG

Minimum Viable RAG: 75 บรรทัดที่ไม่มี framework

Try this: เริ่ม RAG แรกของคุณภายใน 1 ชั่วโมง

สรุปสั้น

บทที่ 9 — Embeddings, Vector Database และ Retrieval Pipeline

Embedding คือพิกัด GPS ของความหมาย

ตลาด embedding model: API vs Open-weight

ทำไม embedding model ส่วนใหญ่อ่อนภาษาไทย (และตัวที่ไม่อ่อน)

Similarity metric: cosine, dot product, L2

Vector Database Landscape

Embedding Model Selection Checklist

Hybrid Search: ทำไม semantic อย่างเดียวไม่พอ

Reranking: cross-encoder เป็น quality gate

Decision Tree: Debug retrieval ที่ออกมาแย่

ลองทำ: เทียบ embedding model 2 ตัวบนภาษาไทย

สรุปสั้น

บทต่อไป

บทที่ 10 — Evals: วัดคุณภาพ AI App อย่างไรให้รู้ว่าดีขึ้นจริง

ทำไม unit test ที่คุณคุ้นเคยใช้กับ LLM ไม่ได้

Eval Workflow ที่ทำได้จริงในบ่ายเดียว

เริ่ม eval set ของคุณด้วย 10 cases

สามวิธีให้คะแนน: เลือกอันที่เหมาะกับสถานการณ์

Regression test: จุดที่ engineer ต่างจาก hobbyist

เขียน eval harness ของคุณเองในไม่ถึง 60 บรรทัด

RAG eval = retrieval eval + generation eval

Agent eval — pass@k vs pass^k

กับดักที่เห็นบ่อยและวิธีหลีก

Quick recap

บทที่ 11 — Local LLM: รันโมเดลบนเครื่องตัวเอง

ทำไม Local LLM ถึงสำคัญในปี 2026

ความจริงที่ไม่มีใครอยากบอก: local ≠ frontier

Cloud vs Local vs Hybrid: เลือกอย่างไร

Hardware: ทำไม RAM คือ “ที่จอดรถของโมเดล”

GGUF: format มาตรฐานของ local LLM

Quantization: บีบโมเดลให้รันได้บน hardware ของเรา

เลือก runtime: Ollama vs llama.cpp vs LM Studio

ลงมือทำ: ติดตั้ง Ollama แล้วรันโมเดลตัวแรก

ภาษาไทย: เลือกโมเดลไหนดี

Ollama Command Cheat Sheet

ข้อจำกัดที่ต้องรู้ก่อนคาดหวังเกิน

Try this: pull, run, เทียบ 3 prompt

Quick recap

บทที่ 12 — Local LLM Operations: อัปเดต เปลี่ยนโมเดล ทดสอบ และดูแลระบบให้เสถียร

Local LLM = Software + Model (ต้องดูแลทั้งสองอย่าง)

Model Lifecycle: เลือก → ทดสอบ → เปรียบเทียบ → Deploy → Monitor → Update → Rollback/Replace

Model Inventory = ตู้เครื่องมือช่าง

Naming + Versioning Convention

Update Runtime — Pin Versions, Read Release Notes

Update Models — Mini Eval Set ก่อน Blind Pull

3 เรื่องจริงที่ engineer ทุกคนควรรู้

Rollback Strategy — Ollama, llama.cpp, HuggingFace

Checklist: Rollback Local LLM

Multiple Model Strategies — Per-task Aliases และ Hybrid Routing

Troubleshooting Checklists — 6 อาการที่เจอบ่อย

Cloud / Local / Hybrid Decision Matrix v2

Cleanup และ Reproducibility — ทำเพื่ออนาคตของตัวเอง

สรุปสั้น

Try this — ขยายของ Ch.11

บทที่ 13. Final Project: สร้าง Local RAG Assistant ของตัวเอง

ภาพรวมก่อนลงรายละเอียด: 7-step build journey
ทำไมเลือก stack นี้ (และคำตอบสำหรับ interviewer)
Project structure — 12 ไฟล์ที่จะสร้าง
Step 1 — Setup: เริ่มต้นด้วย 5 คำสั่ง
Step 2 — Ingest: load → chunk → embed → store
Step 3 — Retrieve: similarity search ที่เห็นภาพ
Step 4 — Generate: prompt ที่บังคับให้อ้างอิง context
Step 5 — Eval: 10 cases ที่บอกได้ว่าระบบนี้ดีขึ้นหรือแย่ลง
Step 6 — API: FastAPI + auth + structured log
Step 7 — Containerize: Dockerfile ที่สั้น 8 บรรทัด
Security & Privacy Checklist — mapped to OWASP LLM Top 10 (2025)
Deployment Checklist — ก่อน push ขึ้น GitHub
Portfolio Readiness Checklist — ทำให้ "ขายได้"
What's next — roadmap 6 เดือนหลังจากเล่มจบ
ปิดท้าย

Bibliography

บทนำ

ไม่กี่ปีก่อน การจะทำให้ซอฟต์แวร์ “เข้าใจภาษา” คน ตอบคำถามจากเอกสาร หรือสรุปข้อมูลมหาศาลได้ ยังเป็นงานของทีมวิจัยที่มี GPU เป็นฟาร์มและนักคณิตศาสตร์เต็มห้อง วันนี้สิ่งเดียวกันเริ่มต้นได้ด้วย API key หนึ่งตัวกับโค้ดไม่กี่สิบบรรทัด จุดเปลี่ยนนี้ทำให้คุณค่าของงานย้ายที่ — จาก “การฝึกโมเดล” ไปสู่ “การประกอบโมเดลสำเร็จรูปเข้ากับระบบจริงให้ใช้งานได้” และคนที่ทำงานตรงรอยต่อนี้คือสิ่งที่เราเรียกว่า **AI Engineer**

หนังสือเล่มนี้เขียนขึ้นสำหรับโปรแกรมเมอร์และคนสาย tech ในไทยที่อยากก้าวเข้าสู่สาย AI อย่างเป็นระบบ ไม่ใช่การไล่ตาม buzzword หรือท่องชื่อเครื่องมือใหม่ทุกสัปดาห์ แต่เป็นการสร้าง mental model ที่อยู่ทนกว่าเครื่องมือ เพื่อให้เมื่อ framework เปลี่ยน โมเดลเปลี่ยน หรือราคา token เปลี่ยน คุณยังตัดสินใจถูกได้ด้วยตัวเอง แนวทางของเล่มนี้คือ **concept-first** — เข้าใจหลักการก่อน แล้วใช้โค้ด Python เป็นเพียงตัวอย่างประกอบ ไม่ใช่ตัวเอกของเรื่อง

เส้นทางใน 13 บทถูกออกแบบให้เดินต่อกันเป็นขั้น เริ่มจากการทำความเข้าใจว่า AI Engineer ต่างจาก ML Engineer, Data Scientist และ Software Engineer อย่างไร และต้องมี skill stack แบบไหน จากนั้นปูพื้น Machine Learning แบบไม่จมคณิตศาสตร์ ไล่จาก neural network สู่ Transformer และ LLM เพื่อให้เห็นว่าโมเดลที่เราใช้ทุกวัน “คิด” อย่างไรในมุมของวิศวกร เมื่อพื้นฐานแน่นแล้ว เล่มนี้จะพาลงมือจริง — สร้าง LLM app แรกด้วย API, ฝึก prompt engineering สำหรับงาน production, ต่อด้วย RAG เพื่อให้ AI ตอบจากเอกสารและความรู้ของเราเอง พร้อมเรื่อง embeddings และ vector database ที่อยู่เบื้องหลัง

ครึ่งหลังของหนังสือเน้นสิ่งที่แยกของเล่นออกจากของจริง นั่นคือการวัดผลและการดูแลระบบ บท Evals จะสอนวิธีรู้ว่า AI app ของเรา “ดีขึ้นจริง” หรือแค่รู้สึกว่าดีขึ้น ส่วนบท Local LLM และ Local LLM Operations จะพาไปรันโมเดลบนเครื่องตัวเอง วางแผนฮาร์ดแวร์ จัดการ quantization และดูแลโมเดลให้เสถียรในระยะยาว และปิดท้ายด้วย Final Project ที่รวมทุกอย่างเข้าด้วยกันเป็น Local RAG Assistant ที่คุณนำไปต่อยอดเป็นผลงานใน portfolio ได้ทันที

ไม่จำเป็นต้องอ่านรวดเดียวจบ แต่ละบทมี checklist, ตาราง และภาพประกอบให้กลับมาใช้ซ้ำได้เหมือนคู่มือทำงาน ขอเพียงคุณมีพื้นฐานการเขียนโปรแกรมและความอยากลงมือทำ ส่วนที่เหลือ ค่อย ๆ ประกอบไปทีละชิ้น เหมือนที่ AI Engineer ที่ดีทำกับระบบของเขา — เริ่มจากของที่ใช้ได้จริง แล้วทำให้มันดีขึ้นทีละก้าว

บทที่ 1 — AI Engineer คือใครในยุค LLM

เพื่อน dev ส่ง link มาใน LINE ตอนสามทุ่ม:

> "เฮ้ ตำแหน่ง 'AI Engineer' ที่บริษัทกูเปิดรับนี่ มันคืออะไรกันแน่ ต่างจาก ML Engineer ตรงไหนวะ? requirement มันก็ Python, LLM API, RAG, evals, FastAPI — ไม่เห็นมี TensorFlow ไม่เห็นมีอะไรที่ดูเหมือน ML เลย"

คุณกดเปิด job posting ดู เห็นข้อความที่ดูเหมือนรายการที่ตัวเองเคยได้ยินมาทั้งหมด แต่ไม่เคยรวมอยู่ในตำแหน่งเดียวกัน แล้วก็นึกขึ้นมาได้ว่า... คุณตอบไม่ออก

ถ้าจะตอบให้เคลียร์ คุณต้องบอกได้ว่า AI Engineer ทำอะไรในวันทำงานจริง ต่างจาก ML Engineer ที่ฝึก model ยังไง ต่างจาก Data Scientist ที่วิเคราะห์ข้อมูลยังไง และต่างจาก SWE ที่ใช้ Copilot ทุกวันยังไง ไม่ใช่นิยามแบบ marketing แต่เป็นนิยามแบบที่ใช้ตอบเพื่อนได้

บทนี้คือคำตอบนั้น — และคำตอบสำหรับคำถามที่คุณอาจจะเคยถามตัวเองเงียบๆ ด้วยว่า "แล้วเราอยู่ใกล้บทบาทไหนที่สุด?"

ทำไม role นี้เพิ่งโผล่มาในเรดาร์ของทุกคน

ถ้าคุณลองนึกย้อนกลับไปแค่ 3 ปีก่อน — ปลายปี 2022 — คำว่า "AI Engineer" แทบไม่เป็น job title ที่ใครเห็นใน LinkedIn เลย

ไม่ใช่เพราะคนไม่ทำงานกับ AI งานสาย ML มีมานานแล้ว แต่ถ้าคุณจะ "สร้าง AI product" ในตอนนั้น คุณต้องมีของพร้อม 3 อย่าง: ทีม ML, GPU จำนวนหนึ่ง, และเวลาฝึก model หลายเดือน คนทั่วไปที่เขียน backend อยู่ดีๆ ไม่ได้เป็นส่วนหนึ่งของการสนทนานี้

แล้วในเวลา 8 เดือน ทุกอย่างก็เปลี่ยน

- **30 พฤศจิกายน 2022** — OpenAI เปิดตัว ChatGPT ผู้ใช้ทะลุ 1 ล้านคนใน 5 วัน ทะลุ 100 ล้านคนใน 2 เดือน
- **1 มีนาคม 2023** — OpenAI เปิด ChatGPT API ให้ developer ทั่วไปเรียกได้ ราคา $0.002 ต่อ 1,000 tokens. ตรงนี้คือ moment ที่ "build AI" หยุดเป็นกิจกรรมของทีม ML — มันกลายเป็น HTTP request ที่ใครก็ตอบกลับมาได้ในบ่ายเดียว
- **30 มิถุนายน 2023** — Shawn "swyx" Wang เผยแพร่เรียง The Rise of the AI Engineer บน Latent Space เขาสังเกตว่ามี practitioner กลุ่มหนึ่งที่ build product ด้วย LLM API + open-source tools มาสักพักแล้ว แต่ยังไม่มีชื่อเรียก เลยตั้งให้
- **ตุลาคม 2023** — AI Engineer Summit ครั้งแรกจัดที่ San Francisco ขายหมดทันทีด้วย applicant ratio 10:1 (คนสมัคร 10 คนต่อที่นั่ง 1 ที่)

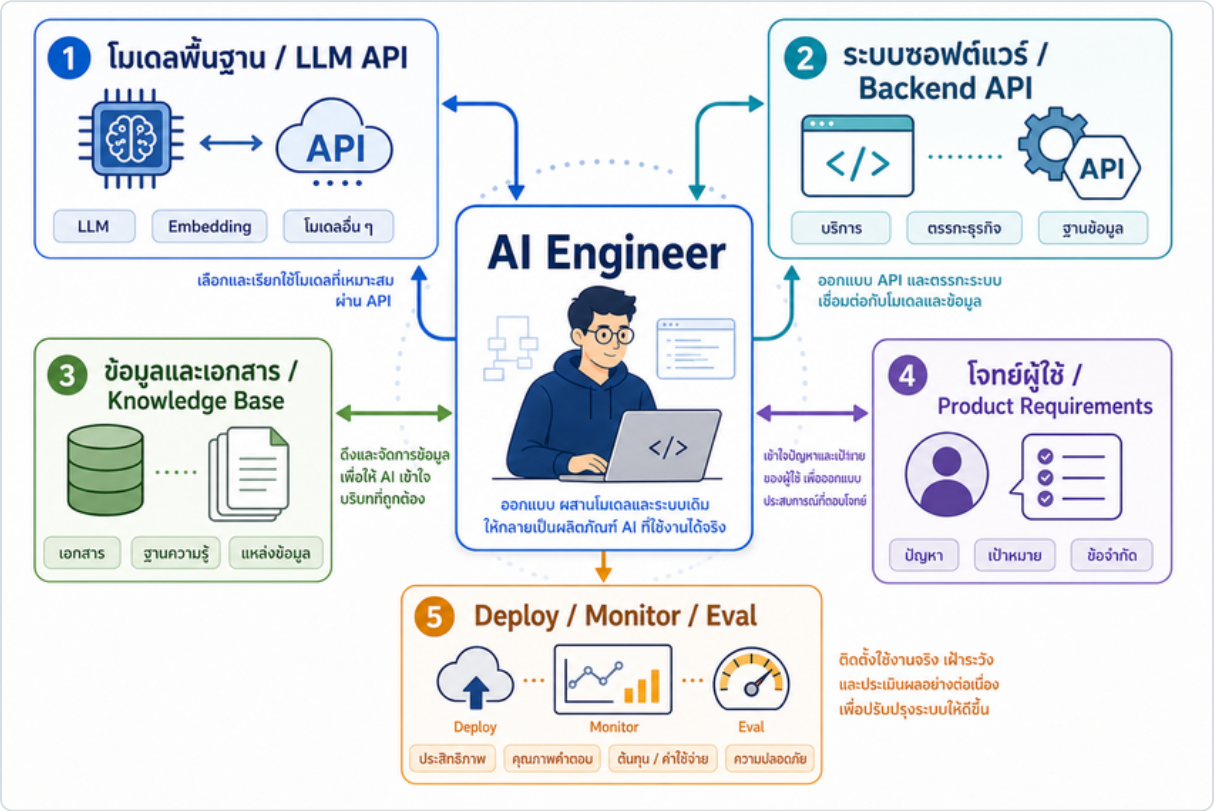
- **มกราคม 2026** — LinkedIn ออกรายงาน Jobs on the Rise 2026 (วิเคราะห์งานช่วง Jan 2023 – Jul 2025) ระบุว่า AI Engineer คืออาชีพที่เติบโตเร็วที่สุดอันดับ 1 ในสหรัฐอเมริกา มี 75,000 ตำแหน่งเปิดในช่วงนั้น

ข้อสังเกตที่ควรเก็บไว้: ตัวเลขจาก LinkedIn เป็นตลาด **US เท่านั้น** ไม่ใช่ไทย ตลาดไทยยังไม่มีรายงาน salary หรือ demand ที่ผ่านการยืนยัน — เวลาคนพูดถึง "demand ของ AI Engineer ในไทย" ส่วนใหญ่ยังเป็นความรู้สึก ไม่ใช่ data

แต่ pattern ใหญ่ที่เกิดในรอบ 3 ปีก็ชัดมาก: **foundation model พร้อมใช้แล้ว** คุณค่าทางวิศวกรรมเลยย้ายจาก "คนสร้าง model" ไปอยู่ที่ "คนประยุกต์ใช้ model" และ role ที่ชื่อ AI Engineer ก็เกิดขึ้นเพื่อรองรับงานในชั้นนั้น

> **Key idea:** AI Engineer ไม่ได้เกิดเพราะมีใครตั้งชื่อก่อน แต่เกิดเพราะมีคนทำงานในแบบใหม่ก่อนแล้วชื่อค่อยมาทีหลัง

นิยามที่ใช้ได้จริงในวันทำงาน



ถ้าจะให้นิยามแบบที่ตอบเพื่อนได้ใน 30 วินาที — AI Engineer คือ:

> วิศวกรที่เอา foundation model ที่มีอยู่แล้ว (ผ่าน API หรือ open-source) มาประกอบเข้ากับ system จริง ให้แก้ปัญหาที่เฉพาะเจาะจงของผู้ใช้หรือองค์กรได้

Chip Huyen อธิบายในหนังสือ AI Engineering (O'Reilly, 2025) ว่า AI engineering เริ่มต้นจาก product idea แล้วค่อยเดินไปหา data และ model — ต่างจาก ML engineering แบบเดิมที่เริ่มจาก data และ training ก่อน. ในการสำรวจของเธอ ผู้สร้าง LLM app ส่วนใหญ่บอกว่า "AI Engineering" คือคำที่ตรงกับสิ่งที่พวกเขาทำมากที่สุด

ลองนึกภาพแบบนี้ — งานของ AI Engineer คล้ายงานของ backend engineer ที่ทำงานกับ PostgreSQL คุณไม่ได้เขียน database engine เอง คุณรู้จักออกแบบ schema เขียน query ที่มีประสิทธิภาพ tune index เลือก connection pool ให้ตรงกับ load และต่อ database เข้ากับ application ที่เหลือ ทักษะที่มีค่าคือการใช้ของให้ดี ไม่ใช่การสร้างของขึ้นมาใหม่

AI Engineer ก็แบบเดียวกัน แต่ทำกับ LLM แทน

หรือถ้าชอบ analogy นอกโลก software — AI Engineer เหมือนเชฟที่ไม่ปลูกผักเอง ไม่เลี้ยงไก่เอง ไม่บดเครื่องเทศเอง แต่รู้จักเลือกวัตถุดิบให้ตรงกับเมนู รู้จักใช้เตาแต่ละชนิด รู้ขั้นตอนที่ทำให้ทุกอย่างเข้ากัน แล้วส่งจานออกไปที่ลูกค้ากินได้จริง คุณค่าทั้งหมดอยู่ที่การ **compose** ไม่ใช่ที่การปลูกวัตถุดิบ

บางคนเรียก role นี้ว่า "full-stack developer ของยุค LLM" ก็ไม่ผิด — เพราะมันครอบคลุมตั้งแต่ฝั่ง backend (เรียก LLM API, จัด RAG pipeline, ออกแบบ eval), ฝั่ง data (chunking, embedding, retrieval), ไปจนถึงฝั่ง product (ออกแบบ interaction ที่ใช้งานได้จริง)

งานจริง — "วันอังคาร" ของ AI Engineer คนหนึ่ง

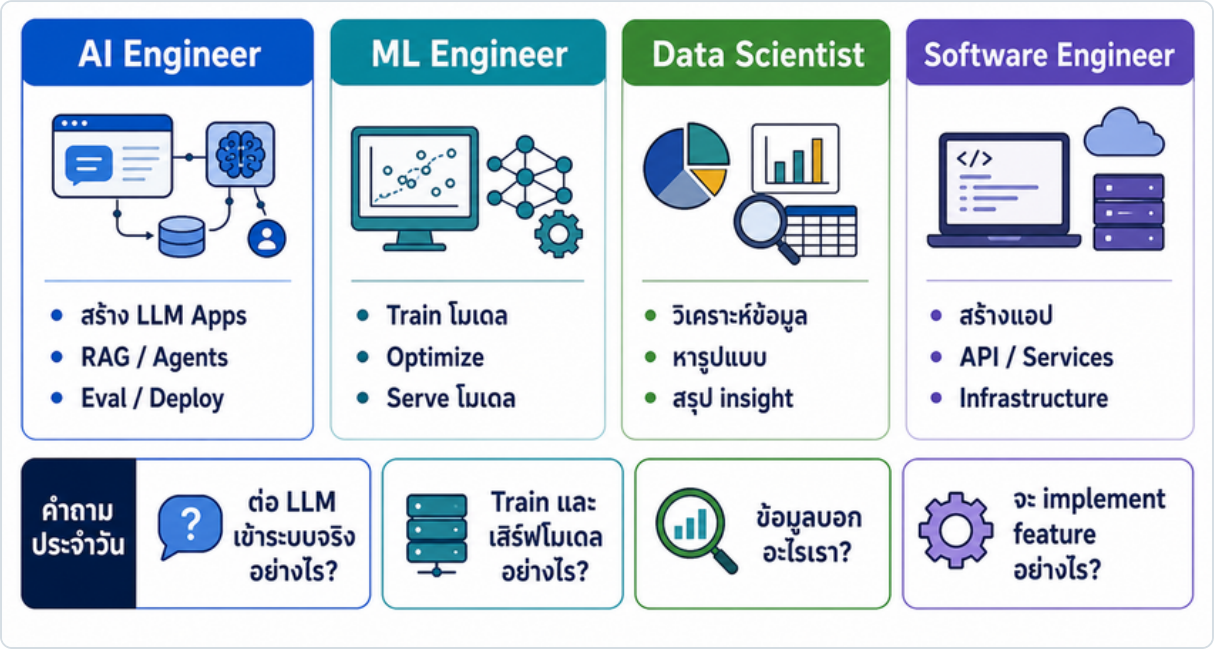
ตัวอย่างต่อไปนี้เป็น **สถานการณ์สมมุติประกอบจากงานจริงในวงการ** (composite scenario) ไม่ใช่บันทึกของบุคคลใดบุคคลหนึ่ง

> **9:00 น.** — ทีม customer support ส่ง alert มาว่า chatbot ตอบผิดบ่อยขึ้นตั้งแต่วันเสาร์ คุณเปิด monitoring dashboard เห็น answer quality score ลดลง 18% เริ่ม debug
>
> **9:30 น.** — ตรวจ RAG pipeline พบว่าทีม content อัปโหลด FAQ ฉบับใหม่เมื่อวันศุกร์ ขนาด document เพิ่มจาก 12 หน้าเป็น 47 หน้า chunking strategy เดิมที่ตั้งไว้ 500 tokens/chunk ตอนนี้ทำให้ chunk หนึ่งครอบหลาย topic — retriever เลยดึงมาผิด section
>
> **11:00 น.** — ปรับ chunking ใหม่ให้เป็น section-aware (split ตามหัวข้อ H2) re-index รัน eval set 20 cases ที่เก็บไว้ ผ่านทุก case Deploy
>
> **13:00 น.** — มีตั๋วใหม่ใน Jira: PM อยากให้ chatbot ดึง order status ได้ คุณเริ่ม prototype ด้วย tool calling — ให้ LLM เรียก `get_order_status(order_id)` แทนที่จะตอบเอง
>
> **15:30 น.** — เขียน 10 eval cases สำหรับ feature ใหม่ (case ปกติ, order ไม่พบ, order ID format ผิด, ผู้ใช้ไม่มีสิทธิ์ดู, ฯลฯ) deploy ไป staging แปะ link ใน PR ให้ทีมรีวิว

สังเกตอะไรไหม? ทั้งวัน — debugging, building, evaluating, deploying ไม่มี gradient descent, ไม่มี hyperparameter sweep, ไม่มี training loop ที่รันค้างคืน งานนี้ใกล้ software engineering มากกว่า ML research

นี่คือ shape ของงาน AI Engineer

ต่างจาก ML Engineer / Data Scientist / Software Engineer ยังไง



นี่คือส่วนที่คนสับสนกันมากที่สุด — และเป็นส่วนที่คุณจะใช้ตอบเพื่อนได้

มิติ	AI Engineer	ML Engineer	Data Scientist	Software Engineer (ทั่วไป)
คำถามหลักในแต่ละวัน	"จะต่อ LLM เข้ากับ system นี้ยังไงให้ใช้งานได้?"	"จะสร้างและ serve model นี้ให้ scale และ reliable ได้ยังไง?"	"data บอกอะไรเรา? pattern ไหนสำคัญ?"	"จะ implement feature นี้ยังไง?"
เครื่องมือหลัก	OpenAI/ Anthropic SDK, LangChain/ LlamaIndex, vector DB (Qdrant/Chroma/ pgvector), FastAPI, eval framework	PyTorch/ TensorFlow, Ray, MLflow, SageMaker/ Vertex, Kubernetes	Jupyter, pandas, SQL, sklearn, Tableau, dbt	IDE, Git, framework ของ project, CI/CD
ผลลัพธ์ที่ส่งมอบ	LLM app, RAG system, AI agent, eval pipeline, AI-powered feature	Training pipeline, custom model, inference API, model monitoring	Dashboard, report, insight, statistical model	Application, API, service, infrastructure

มิติ	AI Engineer	ML Engineer	Data Scientist	Software Engineer (ทั่วไป)
ลึก math แค่ไหน	ต่ำ–กลาง (เข้าใจ concept พอ ไม่ต้อง derive สมการ)	สูง (linear algebra, calculus, optimization)	กลาง–สูง (statistics, probability, causal inference)	ต่ำ (algorithm design พื้นฐาน)
ลึก ML แค่ไหน	กลาง (LLM literacy, RAG, evals — แต่ไม่ต้อง train model เอง)	สูงมาก (train, optimize, deploy custom model)	กลาง (classification/ prediction model)	ต่ำ (ใช้ AI tool ช่วยเขียนโค้ด)
วันทำงานเริ่มจาก	User problem / product requirement	Model performance metric	Business question	Feature request / bug report

ตารางนี้สรุปกรอบใหญ่ได้ — แต่มีอีกหลายเรื่องที่ต้องเสริม

AI Researcher ไม่ได้อยู่ในตารางนี้เพราะเป็น role ที่อยู่คนละ layer ลงไปอีก คนกลุ่มนี้ทำงานที่ research lab เช่น Anthropic Research, OpenAI Research, Google DeepMind, FAIR สิ่งที่ส่งมอบคือ research paper, model architecture ใหม่, หรือ algorithm ใหม่ ส่วนใหญ่จบ PhD หรืออยู่ใน research track ตั้งแต่ต้น ถ้าคุณถามว่า "AI Researcher ต่างจาก AI Engineer ยังไง?" คำตอบสั้นๆ คือ — Researcher ถามว่า "สิ่งนี้เป็นไปได้ไหม? ทำไม?" ส่วน Engineer ถามว่า "model ที่มีอยู่นี้ใช้แก้ปัญหานี้ได้ยังไง?"

SWE ที่ใช้ AI tools (Copilot, Cursor, Claude Code) เริ่มจะ overlap กับ AI Engineer ในช่วงนี้ ขอบเขตระหว่าง "dev ที่ใช้ AI ทำงาน" กับ "dev ที่สร้าง AI feature" กำลังเลือนลาง โดยเฉพาะกับสิ่งที่ swyx เรียกว่า "vibe coding" วิธีที่ผมมองคือ ถ้างานหลักของคุณคือ **ส่ง feature ที่มี LLM อยู่ใน critical path** (ไม่ใช่แค่ใช้ AI ช่วย refactor) คุณก็เริ่มเข้าใกล้ AI Engineer แล้ว swyx เองพูดประโยคหนึ่งที่ผมชอบ: "Someone will pay you to AI Engineer, no one will pay you to Vibe code."

วิธีที่เร็วที่สุดในการอ่าน job posting ว่ามันคือ role ไหนจริงๆ — อย่าดูที่ชื่อตำแหน่ง ให้ดูที่ **responsibilities** ถ้ามันบอกว่า train model, optimize inference, distributed training นั่นคือ ML Engineer ถ้ามันบอกว่า build LLM app, RAG, agents, evals, deploy นั่นคือ AI Engineer ถ้ามันบอกว่า analyze, find pattern, present insight นั่นคือ Data Scientist

ตัวอย่างจริง 2 ตำแหน่งที่ verify ได้

ไม่ต้องเชื่อตารางอย่างเดียว มาดู job posting จริงที่เปิดอยู่จริง

Anthropic – Applied AI Engineer (London)

ที่น่าสนใจคือ Anthropic เป็นบริษัทที่ **สร้าง** Claude เอง — ใช้ compute มหาศาลในการ pre-train และ post-train แต่เวลาที่พวกเขาจ้าง Applied AI Engineer requirement กลับเป็น:

- ประสบการณ์ 4+ ปีในฐานะ Software Engineer หรือ Forward Deployed Engineer
- ประสบการณ์สร้าง LLM application ในระดับ production (prompting, agents, evals, deployment)

- Python proficiency
- ทำงานกับลูกค้าด้าน technical ได้

ไม่มีข้อไหนระบุว่าต้องเคย train model มาก่อนเลย งานคือไปนั่ง pair programming กับลูกค้าให้ build บน Claude API ได้, สร้าง eval suite, เขียน tutorial และ sample code, จัด hackathon

แม้แต่บริษัทที่สร้าง foundation model เอง ก็ยังมี role แยกสำหรับ "application layer" — และนั่นคือ AI Engineer

Vercel – AI Engineer

Vercel คือบริษัทที่ทำ Next.js และ hosting platform เวลาพวกเขาจ้าง AI Engineer งานคือ design prototype interaction paradigms ใหม่สำหรับ UI generation และ code synthesis ต้องการประสบการณ์กับ LLM (OpenAI, HuggingFace) **บวก** frontend stack (React, Tailwind)

ตัวอย่างนี้บอกว่า AI Engineer ไม่จำเป็นต้องเป็น backend/ML specialist เสมอ — บางตำแหน่งอยู่ตรง intersection ของ product + LLM + frontend ความหมายของ role ขึ้นกับว่าบริษัทเอาไปวางไว้ที่ไหนใน stack

> **Key idea:** อย่าตัดสิน role จากชื่อตำแหน่ง — อ่าน responsibilities แล้วถามว่า "วันทำงานจริงต้องส่งมอบอะไร?"

ความเข้าใจผิดที่ขวางคุณอยู่ตอนนี้

ก่อนจะเดินต่อ มีความเชื่อ 3 อย่างที่ผมเจอบ่อยมาก — และทุกอันชะลอคนเก่งๆ ที่ "พร้อมจะเริ่ม" ไว้นานเกินจำเป็น

ความเชื่อ 1: "ต้องรู้คณิตศาสตร์ลึกถึงจะเป็น AI Engineer ได้"

ความเชื่อนี้มาจากการที่หลายคนเริ่มศึกษา AI ผ่านหนังสือ ML แบบ academic ที่เต็มไปด้วยสมการ พอเห็น gradient descent แล้วก็ปิดหนังสือ

ในทางปฏิบัติ AI Engineer ต้องการ **quantitative thinking** ไม่ใช่ **mathematical derivation** สิ่งที่ต้องเข้าใจคือ — token คืออะไร, embedding space ทำงานยังไงในระดับ intuition, accuracy/precision/recall ใช้ตอนไหน, ทำไม benchmark X ดีกว่า Y ในบริบทนี้ คุณต้อง interpret metric ได้ ไม่ใช่พิสูจน์ว่า metric นั้นถูกต้องทาง stat

อันนี้ไม่ได้แปลว่าไม่ต้องรู้ math เลย — มันแปลว่า bar เปลี่ยน ถ้าคุณ debug Python ได้ คำนวณ percentile ได้ในหัว เข้าใจ probability ระดับมัธยม คุณมี foundation พอแล้ว ML researcher คือคนที่ต้องลงลึกกว่านั้น และนั่นไม่ใช่งานที่คุณกำลังสมัคร

ความเชื่อ 2: "AI Engineer กับ ML Engineer คือสิ่งเดียวกัน"

job posting บางที่ใช้คำสองคำนี้สลับกัน — บางบริษัทเรียก ML Engineer แต่จริงๆ ทำงาน AI Engineering บางบริษัทเรียก AI Engineer แต่ทำงาน ML แบบเดิม

นี่ไม่ใช่ปัญหาของคุณ มันคือปัญหาของตลาด

วิธีดูที่ใช้ได้คือกลับไปดูที่ตารางข้างบน — ถามจาก primary question และ output ที่ส่งมอบ ถ้าใน posting พูดถึง “train”, “fine-tune”, “distributed training”, “model architecture”, “loss function” เป็นใจกลาง คือ ML Engineer ถ้าพูดถึง “RAG”, “prompt”, “evals”, “agents”, “tool calling”, “LLM API” เป็นใจกลาง คือ AI Engineer

ไม่ใช่ทุก posting จะชัด แต่ส่วนใหญ่ pattern ออก

ความเชื่อ 3: “AI Engineer = Prompt Engineer ที่เปลี่ยนชื่อ”

ในช่วงต้นปี 2023 “Prompt Engineer” เป็น buzzword ที่ฮือฮามาก มีคนเสนอเงินเดือนหลักแสน USD เพื่อให้คนมาเขียน prompt อย่างเดียว แต่ในปี 2026 ที่เราอยู่นี้ ความเข้าใจเปลี่ยนไปแล้ว

Prompt engineering เป็น **subset** หนึ่งของ skill — สำคัญแน่นอน แต่ไม่ใช่ทั้งหมด AI Engineer ต้องรู้จัก RAG, evals, deployment, tool calling, vector database, observability, cost optimization, ไปจนถึง local model management ถ้าคุณรู้แค่ prompt — คุณคือคนที่ใช้ ChatGPT เก่ง ไม่ใช่ AI Engineer

วิธีที่ผมชอบบอกคือ: prompt engineering เป็น **จุดเริ่มต้น** ไม่ใช่ **จุดสิ้นสุด** ของสายอาชีพนี้

คุณอยู่ใกล้ AI Engineer แค่ไหนแล้ว — Self-Assessment 10 ข้อ

ตอบ Yes/No 10 ข้อข้างล่างนี้ ไม่ต้องคิดมาก ตอบจากสิ่งที่คุณ **เคยทำเองจริงๆ** ไม่นับว่าเคยอ่าน เคยดู YouTube หรือเคยรู้จัก

Checklist: Self-Assessment — คุณใกล้ AI Engineer แค่ไหน?

- [ ] **1.** เคยเรียก LLM API (OpenAI, Anthropic, Gemini หรือตัวอื่น) จาก code ของตัวเอง แล้วเอา response ไปใช้ต่อในโปรแกรม
- [ ] **2.** รู้ว่า token คืออะไร และเข้าใจว่าทำไม context window ถึงสำคัญต่อ cost และ quality ของคำตอบ
- [ ] **3.** เคยออกแบบ system prompt สำหรับใช้งานในระบบจริง (ไม่ใช่แค่พิมพ์ใน ChatGPT ส่วนตัว) — เคยคิดเรื่อง role, constraint, output format
- [ ] **4.** รู้ว่า RAG (Retrieval-Augmented Generation) คืออะไร และเข้าใจว่า pipeline หลักประกอบด้วยอะไรบ้าง
- [ ] **5.** รู้จัก embeddings คืออะไร และเข้าใจความสัมพันธ์กับ vector database
- [ ] **6.** สามารถ deploy web application หรือ API ของตัวเอง (เช่น ด้วย FastAPI + Docker หรือ serverless) ได้โดยไม่ต้องให้คนอื่นช่วย
- [ ] **7.** เคยคิดเรื่องวิธีวัดคุณภาพ LLM output อย่างเป็นระบบ — มากกว่า “อ่านแล้วรู้สึกว่าใช้ได้”
- [ ] **8.** รู้ว่า tool calling / function calling คืออะไร และ enable อะไรในระบบ LLM
- [ ] **9.** อ่าน error จาก LLM API call (rate limit, context overflow, invalid request) แล้วแก้ปัญหาเองได้
- [ ] **10.** เคยคิด tradeoff ระหว่าง cost, latency, quality ในระบบที่ตัวเองสร้าง — และเลือก config โดยมีเหตุผล

วิธีอ่านผล:

คะแนน Yes	คุณอยู่ตรงไหน	ก้าวต่อไป
0–3	**AI User Zone** — ใช้ AI tools เป็น แต่ยังไม่ได้ build ระบบเอง	เริ่มจาก foundation: API call แรก (บท 6), เข้าใจ LLM ในมุม engineer (บท 5)
4–6	**Transitioning Zone** — มีฐาน software แล้ว เริ่มแตะ LLM stack ในบางจุด	หนังสือเล่มนี้คือสิ่งที่ตรงกับสถานการณ์คุณมากที่สุด — อ่านเรียงบทได้เลย
7–10	**On Track Zone** — อยู่ในเส้นทาง AI Engineer แล้ว	หนังสือจะช่วย structure ความรู้ที่กระจัดกระจาย และเติมจุดที่ยังขาด (โดยเฉพาะ evals, local LLM, ops)

อย่าตกใจถ้าได้คะแนนน้อย — checklist นี้ถูกออกแบบให้สะท้อนสิ่งที่ AI Engineer ต้องรู้ ไม่ใช่สิ่งที่คุณต้องรู้ "วันนี้" คนที่อ่านหนังสือนี้จบควรตอบ Yes ได้ทุกข้ออย่างมีความหมาย ไม่ใช่ตอบเพราะเคยได้ยินผ่านๆ

กับดักที่เห็นบ่อยตอนเริ่มต้น

ก่อนจะปิดบทนี้ มี mistake pattern 4 อย่างที่ผมเจอบ่อยในคนที่กำลังเปลี่ยนสายมา AI Engineer — ถ้าหลีกเลี่ยงได้จะประหยัดเวลาเป็นเดือน

1. กระโดดไป fine-tuning ก่อนที่จะ build อะไรด้วย API หลายคนพอได้ยินคำว่า fine-tuning ก็คิดว่านั่นคือ "ของจริง" แล้วใช้เวลา 2 สัปดาห์ทำ LoRA tutorial — ทั้งที่ปัญหาที่ตัวเองอยากแก้ใช้ system prompt + RAG ก็จบ ลำดับที่ใช้ได้คือ prompt → RAG → fine-tune (และ 80% ของงานจริงไม่ต้องไปถึงข้อสุดท้าย)

2. ติดอยู่ที่ "เรียน Python ML ให้แน่นก่อน" ถ้าคุณเขียน Python ได้ระดับ build CLI tool และเรียก HTTP API ได้ — คุณ "พอ" สำหรับเริ่มแล้ว ไม่ต้องรอจน master pandas และ NumPy ก่อน

3. ใช้ ChatGPT ผ่าน web UI แทนการเรียก API สองอย่างนี้คนละโลกในมุม engineer prompt ที่ทำงานสวยใน ChatGPT มักพังเมื่อย้ายไป API (เพราะ system message ต่างกัน, default parameter ต่างกัน, ไม่มี hidden context จาก previous session) ถ้ายังไม่ได้เรียก API ด้วย code — เริ่มจากตรงนั้น

4. มองข้าม evals "ไม่มี test ก็ดูเอาว่ามันใช้ได้" คือทางลัดที่จบไม่สวย เพราะ LLM ไม่ deterministic เปลี่ยน prompt วันนี้ แล้วพรุ่งนี้ไม่รู้ว่ามันดีขึ้นหรือแย่ลง สิ่งที่แยก hobbyist ออกจาก engineer คือ "วัดได้ว่าดีขึ้นจริง"

> **Key idea:** AI Engineer มืออาชีพไม่ได้ต่างจาก hobbyist เพราะใช้ prompt เก่งกว่า — ต่างเพราะ **วัดผลได้**

สรุปสั้น แล้วเดินต่อ

- **AI Engineer** คือคนที่ประกอบ foundation model เข้ากับ system จริง ไม่ใช่คนที่สร้าง model จาก scratch
- **Role นี้เพิ่งเป็นรูปเป็นร่าง** ในช่วง 3 ปีล่าสุด เพราะ LLM พร้อมใช้ผ่าน API แล้ว — คุณค่าย้ายจาก model layer มาที่ application layer
- **ต่างจาก ML Engineer** ที่ฝึก/optimize model, ต่างจาก **Data Scientist** ที่ส่งมอบ insight, ต่างจาก **AI Researcher** ที่ advance the science, และเริ่ม overlap กับ **SWE ที่ใช้ AI tools** แต่ลึกกว่าเพราะ LLM อยู่ใน critical path ของ product
- **ไม่ต้องเก่ง math เชิงพิสูจน์** — ต้องเข้าใจ concept พอที่จะตัดสินใจ
- **อ่าน job posting ที่ responsibilities** ไม่ใช่ที่ชื่อตำแหน่ง

Try this: คุย role กับเพื่อน dev คนเดิม

ก่อนเริ่มบทที่ 2 ลองทำสิ่งนี้ — มันจะทดสอบว่าบทนี้ติดอยู่ในหัวคุณจริงไหม

1. เปิด job board (LinkedIn, JobsDB, JobThai หรือ Wellfound) ค้นด้วยคำว่า “AI Engineer”, “Machine Learning Engineer”, “Data Scientist” สามตำแหน่ง
2. อ่าน **responsibilities** ของแต่ละตำแหน่ง — ไม่ต้องอ่าน requirements
3. ตอบในใจว่า “ตำแหน่งนี้ถูก label ตรงกับงานจริงไหม?” ถ้าไม่ตรง — มันควรเป็น role ไหน?
4. ส่ง link ที่ confuse ที่สุดให้เพื่อน dev คนเดิม แล้วลองอธิบายให้ฟังด้วยภาษาที่ใช้ในบทนี้

ถ้าทำได้ — แปลว่า mental model ติดแล้ว

ก้าวต่อไปสู่บทที่ 2

ตอนนี้คุณรู้แล้วว่า AI Engineer คือใคร และอยู่ใกล้ role นี้แค่ไหน คำถามถัดมาคือ — **ต้องมี skill อะไรบ้างถึงจะทำงานนี้ได้จริง?**

บทถัดไปจะแตก skill stack ของ AI Engineer ออกเป็น 10 หมวด แบ่งเป็น layer (foundation / LLM-specific / ops) อันไหนต้องมีก่อนเริ่ม อันไหนเรียนระหว่างทาง อันไหนทิ้งไว้ทีหลังก็ได้ พร้อม roadmap 6 เดือนแรกที่ทำตามได้จริง — ไม่ใช่ list 100 ทักษะที่ทำให้ท้อก่อนเริ่ม

ถ้า self-assessment ข้างบนทำให้คุณรู้สึกว่ามี gap หลายจุด — บทที่ 2 คือแผนที่ที่จะอุดมันทีละขั้น