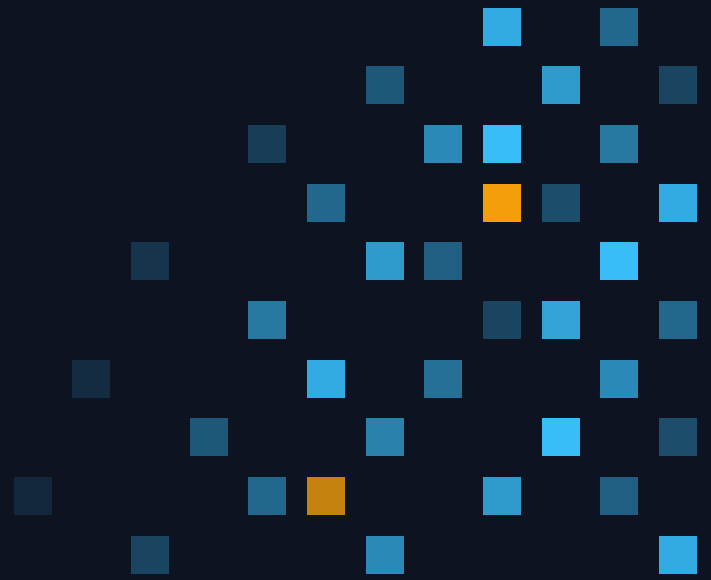
RUNTIME 2026 · TALK SUMMARY

Serving the Agentic Workload

เมื่อ AI Agent กลายเป็นงานหลัก ระบบ inference ต้องเปลี่ยนอะไรบ้าง



SPEAKER

Simon Mo

Co-founder & CEO, Inferact · vLLM

THAI SUMMARY

AGICAFET

เรียบเรียงจากวิดีโอทอล์ก 18 นาที

TL;DR

สรุปทั้งทอล์กใน 1 นาที

01

Workload เปลี่ยนไปแล้ว

งานของ AI agent มี context ยาวมาก (P50 ราว 142k token) แต่ตอบสั้น (444 token) และวนซ้ำหลายสิบ turn

02

KV cache คือหัวใจ

vLLM แก้ด้วย Hybrid KV Cache, แยก prefill กับ decode และ KV cache pool แบบกระจาย (Mooncake: throughput สูงขึ้น 3.8 เท่า)

03

ปีของ hardware หลากหลาย

NVIDIA กับ AMD ใช้ production ได้ทัดเทียมกันแล้ว ส่วน TPU ทีม Inferact ทดลองทำ megakernel ได้ 709 tok/s ต่อผู้ใช้

ยุค agent ความเร็วไม่ได้มาจาก GPU แรงอย่างเดียว แต่มาจากการ “ไม่ต้องคำนวณซ้ำ”

CONTEXT

vLLM คืออะไร และใครใช้บ้าง

Open-source engine สำหรับ inference และ serving LLM

- เริ่มจากงานวิจัย UC Berkeley Sky Computing Lab
- Prototype แรกเขียนก่อน ChatGPT เปิดตัว
- API แรกคือ Completions ยังไม่ใช่ Chat
- Contributors มากกว่า 3,000 คน
- Model labs ทำ day-0 support ร่วมกับ community

APPLICATION LAYER
Meta · Character.ai · Salesforce · Uber · Roblox · ByteDance · Databricks

MODEL LABS
Kimi · MiniMax · Thinking Machines · DeepSeek · Qwen · Mistral AI

INFERENCE CLOUD
Baseten · DigitalOcean · CoreWeave · Crusoe · Amazon Bedrock

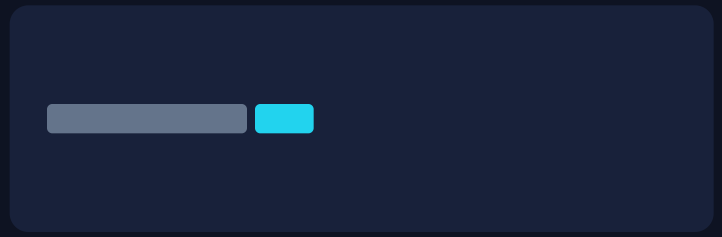
PARTNERS
NVIDIA · AMD · Google · Red Hat · PyTorch

01 · WORKLOAD SHIFT

งานของโมเดลเปลี่ยนไป 3 ยุค

2020-2022

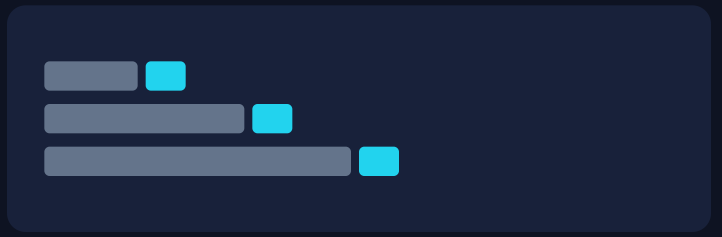
Single-turn completion



prompt เดียว ได้คำตอบเดียว เช่น autocomplete โค้ด

2022-2024

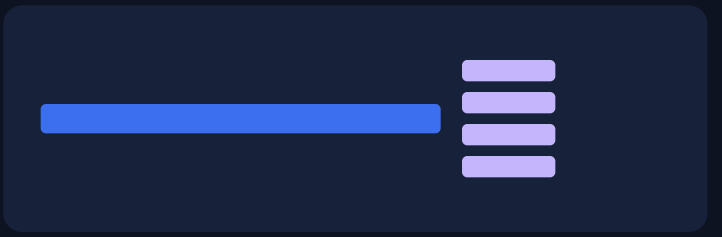
Multi-turn chat



ต้องแบกประวัติแชตไปทุกรอบ vLLM จึงเพิ่ม prefix caching, chunked prefill และแยก prefill/decode

2024-2026

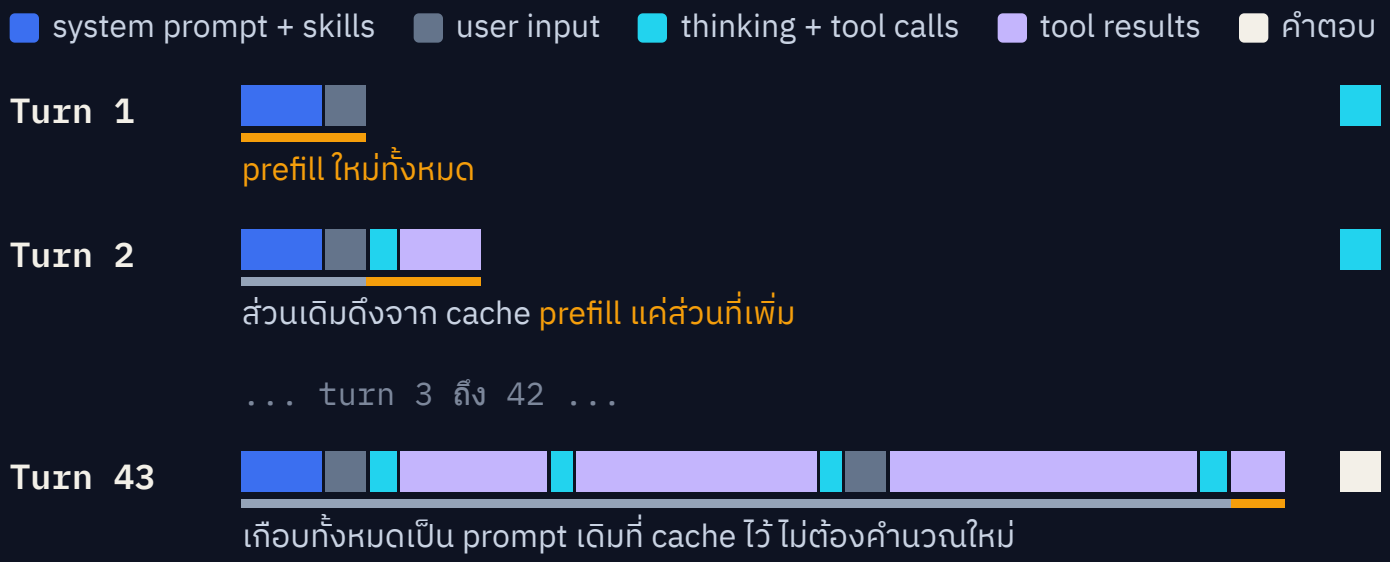
Agentic + real-time



tool call, sub-agent และ real-time ความซับซ้อนพุ่งขึ้น นี่คือโฟกัสหลักของ vLLM ปีนี้

01 · WORKLOAD SHIFT

หน้าตาของ 1 session ของ coding agent



Sub-agents tool call เดียว แตกเป็นหลาย session ขนานกันทันที

43
turns ต่อ session

142k
P50 input length (token)

444
P50 output length (token)

>95%
prefix ที่ใช้ซ้ำได้ (cache hit)

44%
ของ session มี sub-agent (พุ่งเป็นช่วง)

ตัวเลขจากสไลด์ในทอล์ก ตรงกับสถิติของ SemiAnalysis AgentX (agentic coding traces จริง) ในบล็อก vLLM × AgentX

ตัวเลขที่ต้องจำ

อ่าน 142,000 token เพื่อเขียนแค่ 444 token



ทุก turn ระบบต้อง “อ่าน” context ยาวๆ ซ้ำ แต่ “เขียน” ออกมานิดเดียว ความเร็วจริงจึงขึ้นอยู่กับว่า ใช้ cache ได้มากแค่ไหน

สัดส่วนคำนวณจากค่า P50 ในสไลด์ก่อนหน้า (142k ÷ 444)

พื้นฐานก่อนไปต่อ

Prefill, Decode และ KV Cache แบบเข้าใจง่าย

Prefill

“อ่านโจทย์ทั้งหมด”

ประมวลผล prompt, history และ tool output ทั้งก้อน หนักที่ compute มีผลต่อ TTFT (รอนานแค่ไหนกว่าจะเห็นคำแรก)

Decode

“เขียนคำตอบทีละ token”

สร้าง thinking และ action ถัดไป หนักที่ memory bandwidth มีผลต่อความลื่น (tok/s ต่อผู้ใช้)

KV Cache

“สมุดโน้ตที่จดไว้ตอนอ่าน”

เก็บผลคำนวณของ token ที่อ่านแล้ว ถ้า prefix เหมือนเดิม ก็ไม่ต้องคำนวณใหม่ เรียกว่า prefix cache hit

ปัญหาคือ “สมุดโน้ต” นี้กินหน่วยความจำ GPU มาก ยิ่ง context ยาวและผู้ใช้เยอะ ยิ่งต้องบริหารเก่ง

สไลด์อธิบายพื้นฐานเพิ่มเติมจากผู้เรียบเรียง

01 · WHERE IT GETS HARD

ยากตรงไหน: 3 โจทย์ของการ serve agent

ระบบต้องรับผู้ใช้หลักพันพร้อมกัน บาง session 10 turn บาง session 100 turn บาง tool result ยาว 100k token แล้ว multiplex ทั้งหมดให้ได้

01

Prefix cache pressure

ทุก call ส่ง context เต็มๆ ซ้ำ ภาระทั้งหมดจึงตกอยู่ที่การ reuse KV cache ให้ได้มากที่สุด

02

Execution efficiency

แต่ละ step ต้องประมวลผล token มากขึ้น แต่ยังต้องอยู่ใน latency budget และ SLA ที่ตึงเท่าเดิม

03

Global routing

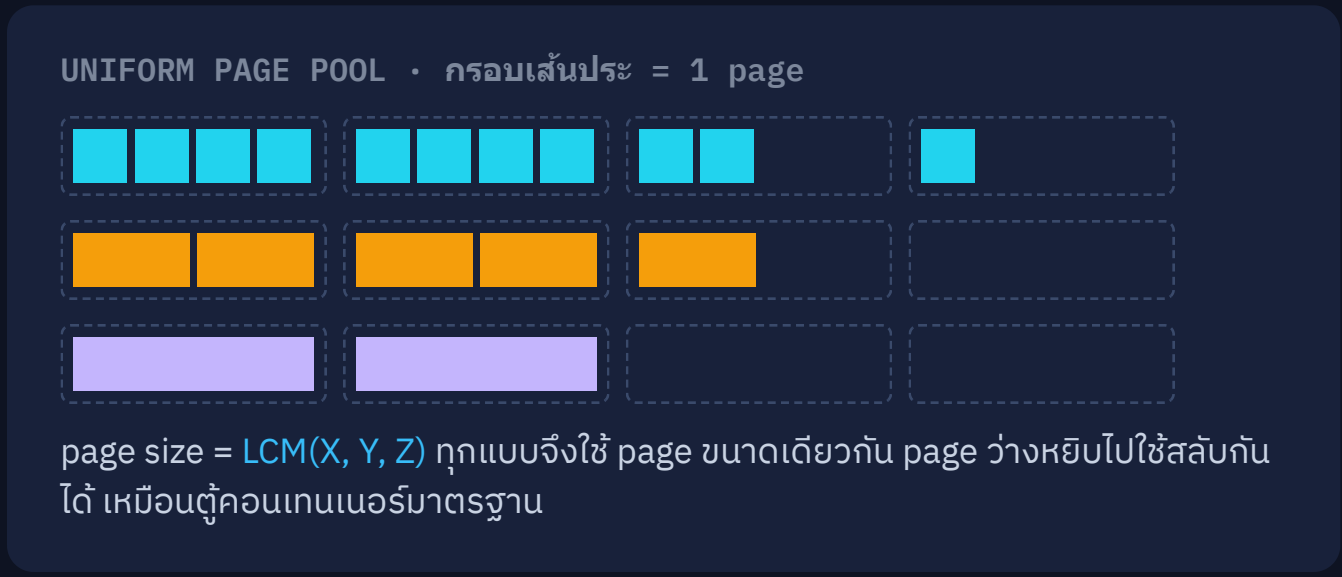
ส่ง session ไปเครื่องที่มี cache อยู่ (affinity) หรือกระจายโหลดให้เท่ากัน เมื่อ sub-agent พุ่งเป็นช่วง ต้องแลกกัน

02 · SOLUTION · KV CACHE ในเครื่องเดียว

Hybrid KV Cache: page เดียวรองรับทุก attention

โมเดลใหม่ผสม attention หลายแบบในแต่ละ layer และแต่ละแบบต้องการ KV cache ไม่เหมือนกัน

- **Full attention**
  ยิ่ง token มาก cache ยิ่งโต
- **Sliding window**
  จำแค่ช่วงล่าสุด ขนาดคงที่ต่อ request
- **Linear attention**
  state ขนาดคงที่ แต่เป็นก้อนใหญ่



page size = LCM(X, Y, Z) ทุกแบบจึงใช้ page ขนาดเดียวกัน page ว่างหยิบไปใช้สลับกันได้ เหมือนตู้คอนเทนเนอร์มาตรฐาน

2-2.5x พื้นที่ KV cache ที่ได้คืนมาใน production ชิปเดียวรับผู้ใช้พร้อมกันได้มากขึ้น ทั้ง throughput และต้นทุนดีขึ้น

02 · SOLUTION · ข้ามหลายเครื่อง

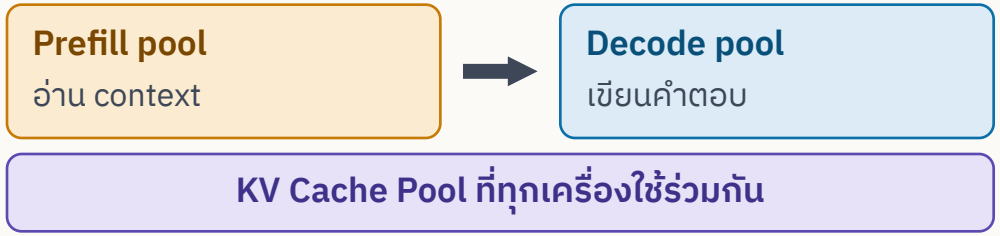
ทำไม production ต้องแยก Prefill กับ Decode

Aggregated เครื่องเดียวทำทั้งคู่



- ⚠ prefill เยอะ: thinking ที่กำลัง stream สะดุดกลางทาง
- ⚠ decode เยอะ: แทรก prefill ไม่ได้ TTFT แย่ลงทันที

Disaggregated แยกหน้าที่

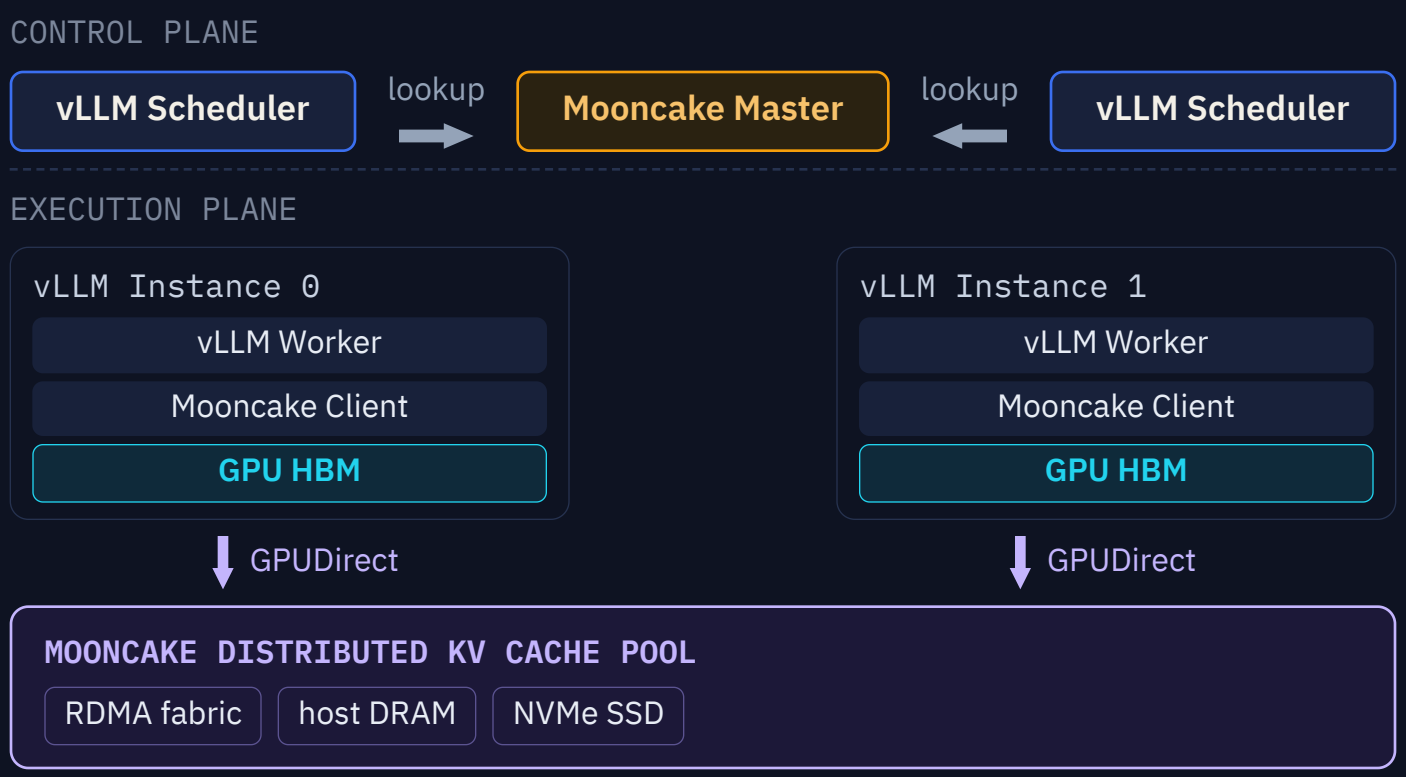


- ✓ latency สม่ำเสมอ ไม่แย่งเครื่องกัน
- ✓ ขยายแต่ละฝั่งได้อิสระตาม workload

“Nobody is serving production agentic workload on an aggregated inference setup.”

02 · SOLUTION · ข้ามหลายเครื่อง

Distributed KV Cache Pool ด้วย Mooncake



คิดแบบง่าย

KV cache ไม่ต้องติดอยู่ใน GPU เครื่องเดียว เครื่องไหนเคยคำนวณ prefix ไว้ อีกเครื่องก็ดึงไปใช้ได้

3.8x

throughput สูงขึ้น

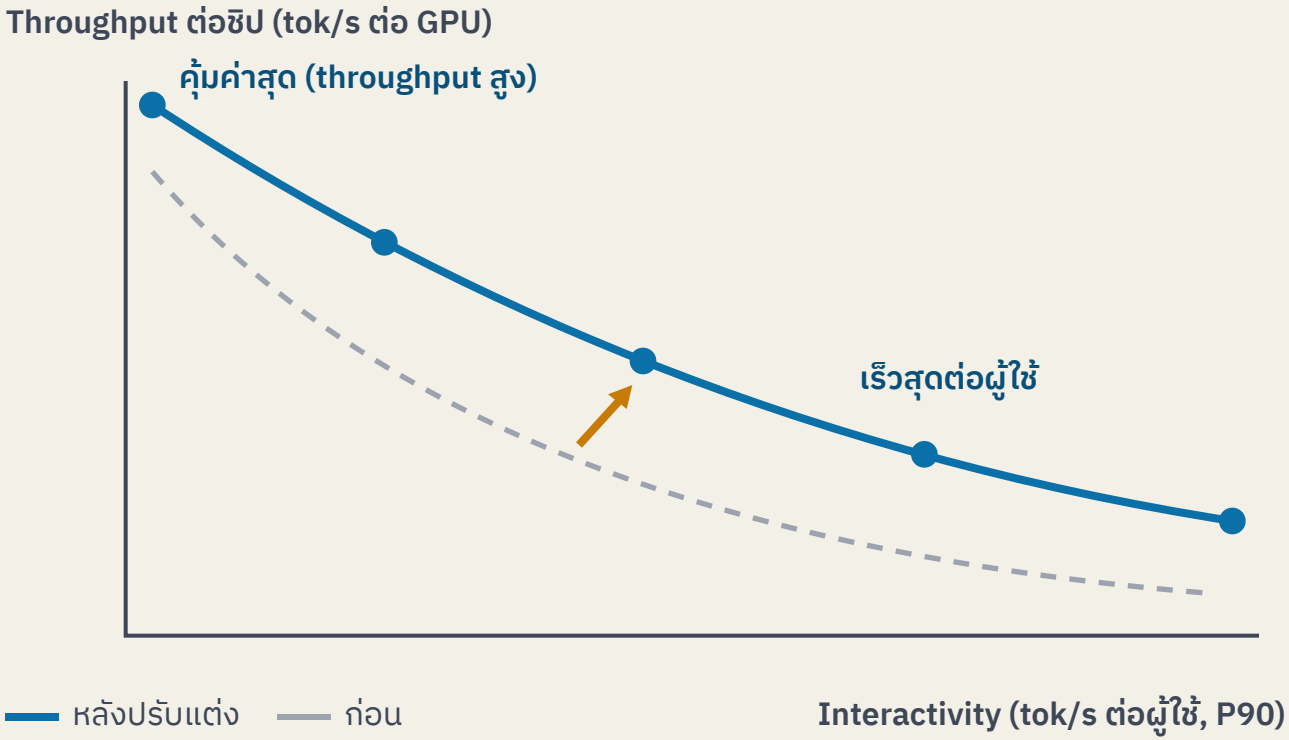
46x

P50 TTFT ต่ำลง บน agentic traces

vLLM มี native integration กับหลาย framework · ตัวอย่างนี้คือ Mooncake connector · ตัวเลข: บล็อก vLLM × Mooncake (Codex traces, 1P1D บน 12× GB200)

02 · RESULTS

วัดผลอย่างไร: กราฟ Throughput กับ Interactivity



ทุกจุด คือการตั้งค่าหนึ่งแบบ (parallelism, batch, kernel)

Pareto frontier ได้ทั้งคุ้มสุดและเร็วสุดพร้อมกันไม่ได้ ต้องเลือก

งานของ vLLM ดันเส้นไปทางขวาบน และให้เลือกจุดไหนก็ได้

BENCHMARK จริง

SemiAnalysis InferenceX · Kimi K3 2.8T · NVIDIA GB300, GB200, B200, B300 และ AMD MI355X

กราฟซ้ายเป็นภาพจำลองเพื่ออธิบายหลักการ

กราฟต้นฉบับ: Kimi K3 2.8T Agentic Token Throughput per Chip vs P90 Interactivity (อัปเดต 21 ก.ย. 2026)

03 · WHAT'S NEXT

ก้าวต่อไปของ vLLM สำหรับ agent

Agentic-first

Agent/session hints และ programmable KV cache ตั้ง TTL, priority และที่อยู่ของ cache ได้

เช่น session ที่รู้ว่าจะไม่แตะอีก 1 ชั่วโมง ควร pin ไว้ ปล่อยให้ LRU ตัดสิน หรือ evict ทิ้ง?

Agentic-efficient

Ultra-low latency (speculative decoding, engine efficiency) และ scaling, recovery ทันที

เป้าหมาย: เลือกจุดไหนบน Pareto curve ก็ได้ แล้วระบบรันให้ใน production อัตโนมัติ

Agent-friendly

Interface และ infra ที่ให้ agent ช่วยพัฒนา vLLM เอง

ทีมใช้โมเดลที่ serve ด้วย vLLM มา optimize vLLM เป็น loop ปิด พร้อม agentic evaluation

03 · HARDWARE HETEROGENEITY

ปีนี้คือปีของ hardware heterogeneity

ชิปจากหลายค่ายมี spec และหาซื้อได้ใกล้เคียงกันแล้ว (parity of chips) แต่แต่ละค่ายมีจุดแข็งต่างกัน

NVIDIA

BLACKWELL · B200 / GB200 / GB300

General purpose ออกแบบมาดี และผ่านการใช้งานหนักมาแล้ว (battle-tested)

AMD

INSTINCT MI355X

ชูขนาด HBM ที่ใหญ่ (288GB) ผู้ใช้ vLLM รายใหญ่เริ่ม deploy production บน AMD มากขึ้น

Google TPU

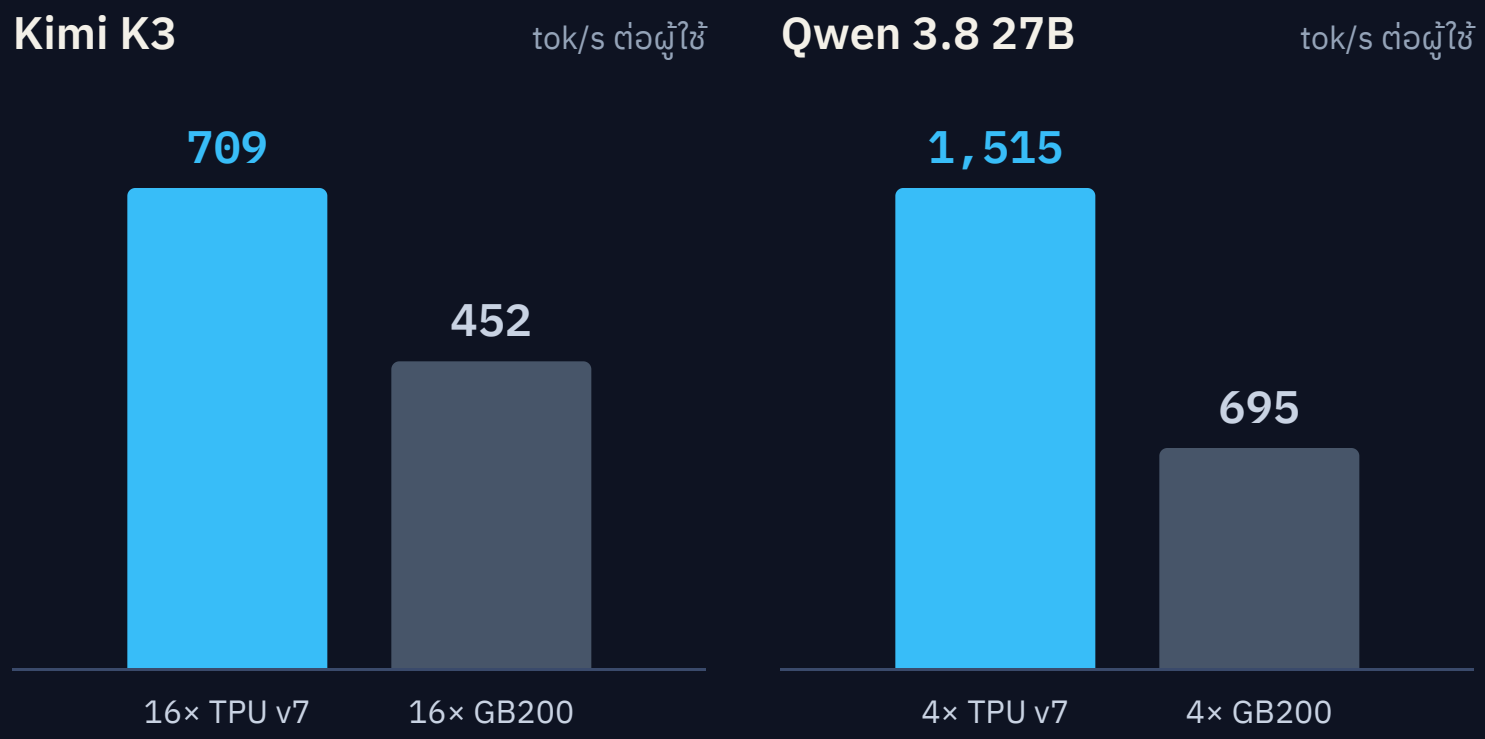
TPU v7 · IRONWOOD

Systolic array ขนาดใหญ่ และ VMEM ที่ใหญ่และโปรแกรมได้ อยู่เบื้องหลังโมเดลอย่าง Gemini

Inferact และ vLLM เป็นพาร์ทเนอร์กับทั้ง NVIDIA, AMD และ Google

03 · HARDWARE HETEROGENEITY

TPU Megakernel ตัวแรก: 709 tok/s ต่อผู้ใช้



1.6x / 2.2x

เร็วกว่า baseline (Kimi / Qwen)

ไม่ใช่การ bench-max แต่ทำเพื่อเข้าใจว่า hardware แต่ละแบบต่างกันตรงไหน

ตัวเลขของ Inferact ทั้งสองฝั่งใช้ speculative decoding (acceptance length 6) · ความสูงแท่งแต่ละกราฟใช้สเกลของตัวเอง · ตัวคูณคำนวณจากตัวเลขในกราฟ

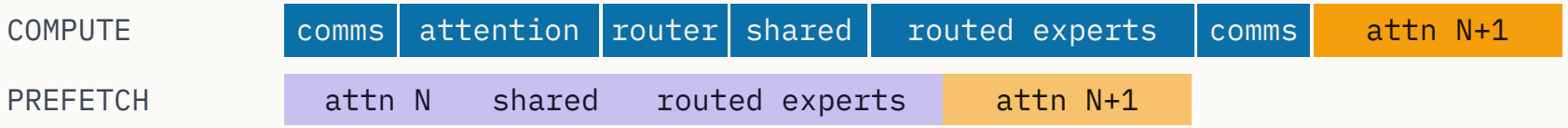
03 · HARDWARE HETEROGENEITY

ข้างใน Megakernel: ทำงานซ้อนกัน ไม่ต้องรอ

แบบเดิม: แยก kernel ทีละขั้น ต้องรอกันเป็นทอดๆ



Megakernel: ทั้ง 92 MoE layer อยู่ใน Pallas call เดียว



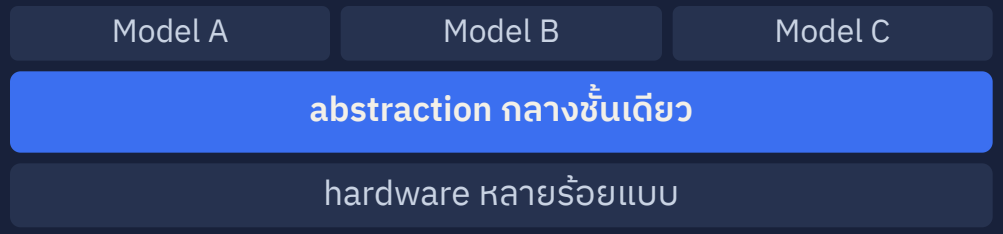
VMEM ของ TPU (คล้าย shared memory บน GPU แต่ใหญ่และโปรแกรมได้มากกว่า) จึง prefetch weight ของ layer ถัดไปขึ้นชิปไว้ก่อน ระหว่างที่ยังคำนวณ layer ปัจจุบัน เหมือนเชฟที่หั่นวัตถุดิบจานต่อไประหว่างผัดจานนี้

แผนภาพวาดใหม่จากสไลด์ “Inside the Megakernel” ของ Inferact

03 · บทเรียนจากทีม vLLM

“Premature abstraction is bad.”

2024-2025 · รวมศูนย์



abstract kernel และ model interface เพื่อรวมให้ง่าย รองรับหลายพัน architecture

2026 · แยกเพื่อความเร็ว



แต่ละโมเดลมีไฟล์ kernel และแทบเป็น inference engine ของตัวเอง เพื่อ performance สุดขีด

ควบคู่กับ: ให้ agent เขียน kernel มากขึ้น, ดึง kernel ล่าสุดจากทีม hardware และเก็บโครงสร้างโมเดลให้ flat

03 · ROADMAP

vLLM Q4 Roadmap: Agentic Production Serving

ด้าน	งานหลัก	ความหมาย
PRODUCTION	Agentic workloads · Large-scale serving	serve งาน agent ระดับ production ขนาดใหญ่
PERFORMANCE	Model performance · Speculative decoding	ให้ agent ช่วยเขียน kernel และลด latency ตอน concurrency ต่ำ
REFINEMENT	KV cache refactoring · CI & release	ปรับโครงสร้าง KV cache และดูแล CI/release ให้ vLLM เสถียรและปลอดภัย
ECOSYSTEM	Hardware support · RL & Omni	รองรับ hardware ใหม่ งาน RL และโมเดล omni-modal

ทุกหัวข้อมี working group เปิดให้ community เข้าร่วมและ contribute

TAKEAWAYS

นำไปใช้กับระบบ AI ขององค์กรเรา

วัด workload จริงก่อน

ดูสัดส่วน input/output, จำนวน turn และ prefix cache hit rate ของงาน agent ที่ใช้อยู่

ทำ prefix ให้นิ่ง

วาง system prompt, tool definition และเอกสารคงที่ไว้ต้น context เพื่อให้ cache hit สูง

เลือกจุดบน Pareto ให้ตรงงาน

งาน batch เน้น throughput ต่อชิป งาน interactive เน้น tok/s ต่อผู้ใช้ ตั้งค่า serving ต่างกัน

วางแผนขยายข้ามเครื่อง

เมื่อเกินหนึ่งโหนด พิจารณาแยก prefill/decode และ KV cache connector เช่น Mooncake

ข้อคิดเพิ่มเติมจากผู้เรียบเรียง ไม่ได้อยู่ในทอล์กต้นฉบับ

JOIN THE vLLM COMMUNITY

มาช่วยกันทำให้ open-weight model รันเร็วขึ้น

ร่วมกับ contributors มากกว่า 3,000 คนทั่วโลก

BUILD

docs.vllm.ai

LEARN

vllm.ai/blog

CONTRIBUTE

github.com/vllm-project/vllm

ที่มา: “Serving the Agentic Workload” โดย Simon Mo, Co-founder & CEO, Inferact · Runtime 2026 · สรุปภาษาไทยโดย AGICAFET