vLLM על AWS: מבדיקה ועד פרודקשן

AWS Summit Tel Aviv 2026 ,סיכום סשן — TLV-SVS305

צחי, Solutions Architect (AI Infrastructure), AWS | ואומרי, Senior Open Source Engineer, AWS

10 בספטמבר 2026 | משך: כ-41 דקות | הופק מהקלטת הסשן

מסלול: AWS for Containers and Platform Engineering | רמה: 300

על המסמך הזה

המסמך מסכם את הסשן TLV-SVS305 מתוך AWS Summit Tel Aviv 2026. הסשן עוסק בשאלה מעשית אחת: מה צריך לעשות כדי להריץ מודל שפה פתוח בעצמך על AWS — מהניסוי הראשון ועד פרודקשן — ואיפה הכסף והביצועים בורחים בדרך. המסמך נבנה מתמלול אוטומטי של ההקלטה ומהסליידים שחולצו מהווידאו, כך שאפשר לחזור לתוכן בלי לצפות שוב בארבעים הדקות. חותמות הזמן בגוף המסמך מפנות לנקודה בהקלטה.

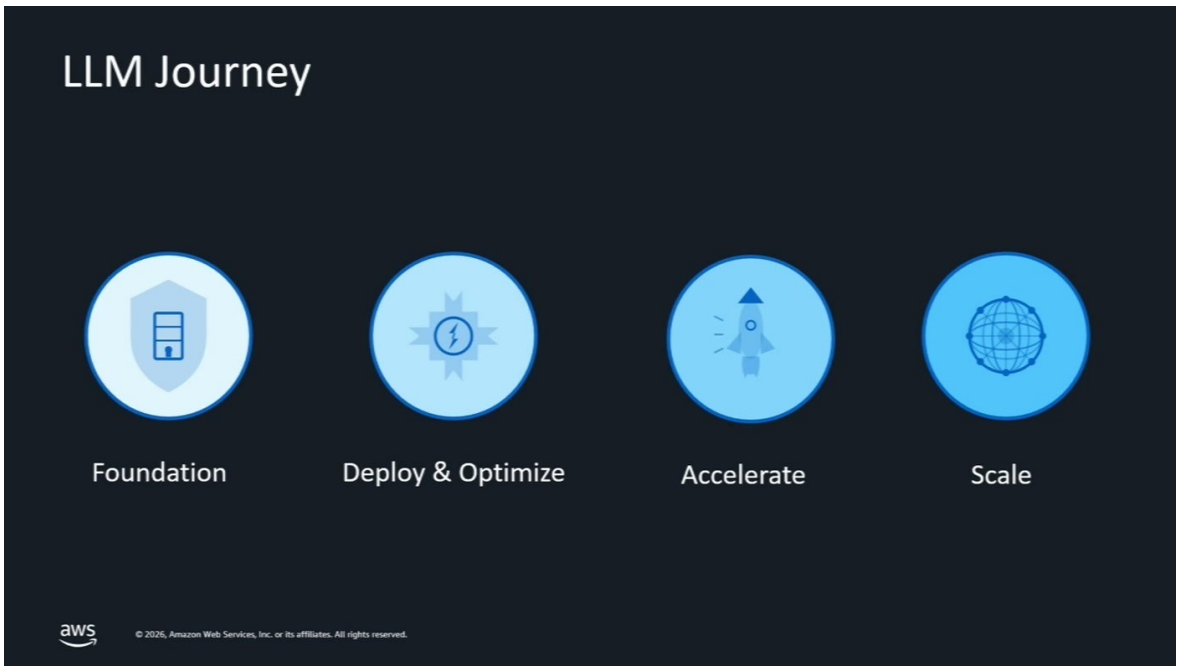
איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה נאמר, ומה שווה לקחת הביתה.
- **פרקים 2–3 — הבסיס:** למה בכלל להריץ מודל בעצמך, ולמה שמים Gateway לפני המודלים.
- **פרקים 4–6 — Deploy & Optimize:** שלושת שלבי ההסקה, חלון ההקשר, ומה vLLM עושה כדי לנצל את ה-GPU.
- **פרקים 7–8 — Accelerate:** אסטרטגיות KV-Cache, ראוטינג חכם וארכיטקטורה מפוצלת.
- **פרק 9 — Scale:** ארבעת ממדי המקביליות ומתי כל אחד מהם רלוונטי.
- **פרקים 10–14 — AI on EKS:** החלק המעשי — פרויקט האופן-סורס, בנצ'מרקינג, Cold Start, Gateways ו-AIBrix.
- **פרק 15 — מה לוקחים מכאן:** מסקנות, נקודות להמשך ומונחון.

> **הערת מקור** המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך מונחים לועזיים ושמות מוצרים מופיעים בו בשיבוש פונטי (למשל "ולעניין" = vLLM, "קרפנטר" = Karpenter, "סוצ'י" = SOCI) — הם נורמלו לכתיב האנגלי. הציטוטים מובאים כלשונם בתמלול ואומתו מול חותמת הזמן המצוינת. מספרים שמופיעים על הסליידים צוינו ככאלה, ומספרים שנאמרו בעל פה בלבד סומנו במפורש.

1. תקציר מנהלים

- **הסשן לא מנסה לשכנע אתכם להריץ מודלים בעצמכם — הוא מגדיר מתי זה מתחיל להיות מוצדק.** הפתיחה מצהירה שלקוחות מתחילים מ-API מנוהל כמו Bedrock, ורק כשה-use case נעשה ספציפי מספיק (coding agents, אופטימיזציות ייעודיות) מגיעים להרצה עצמית של מודלים בעלי משקולות פתוחות [0:00].
- **המסלול חולק לארבעה שלבים, וזו גם המפה של כל ההרצאה:** Foundation, Deploy & Optimize, Accelerate, Scale [1:31]. כל שלב פותר בעיה אחרת, וכמעט כל ארגון נמצא באחד מהם.
- **הרצה נאיבית שורפת GPU.** חיבור מודל מ-Hugging Face למנוע הסקה בלי לגעת בפרמטרים מוביל ל-utilization נמוך של הזיכרון ושל ה-GPU — משאב שהוגדר "מאוד מאוד סקארס" ויקר [3:05]. הטכניקות של vLLM (PagedAttention, Continuous Batching, Quantization) הן מה שמחזיר את הניצולת.
- **ראוטינג סטייטלס הוא באג, לא פשרה.** Round-robin שולח תור שני של אותה שיחה ל-Worker שאין לו את ה-KV Cache שחושב בתור הראשון [13:49]. הפתרון הוא ראוטר מודע ל-Session, ל-Prefix ול-Capacity.
- **Cold Start הוא מספר שאפשר לחתוך בשלוש.** בבנצ'מרק שהוצג על המסך, זמן ההעלאה הכולל על הנוד הזול ירד מ-8:05 דקות ל-2:58 בעזרת SOCI ו-Model Streaming — והנוד הזול עקף בזכות זה את הנוד היקר שהתחילו איתו [35:38].
- **הכל ארוז בפרויקט אופן-סורס אחד: AI on EKS.** שלוש שכבות — Infrastructure (Terraform), Deployments (Helm charts) ו-Guidance — שמאפשרות לשחזר כל דפוס בהרצאה בפקודה או שתיים [21:27, 23:04].



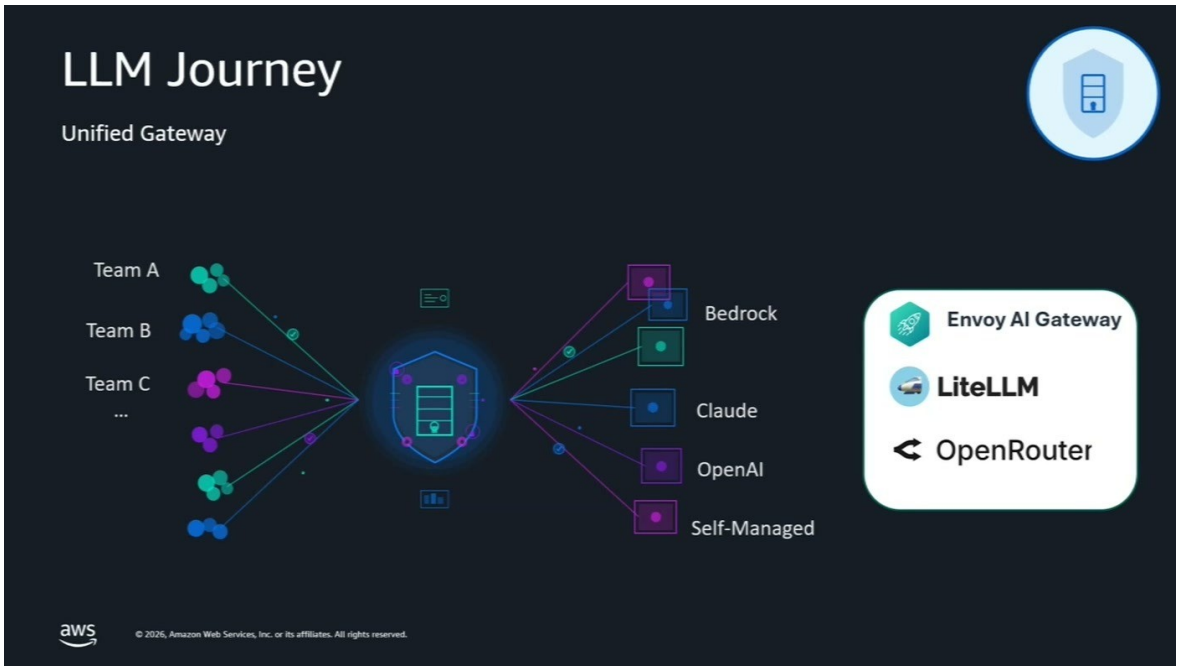
ארבעת השלבים שהגדירו את מבנה ההרצאה כולה

ההרצאה ניתנה בשני חלקים: החלק התיאורטי (עד [21:27]) על ידי Solutions Architect בתחום תשתיות AI, והחלק המעשי על ידי מהנדס אופן-סורס בכיר שהציג את הפרויקט AI on EKS. המעבר מסומן במפורש: "ומכאן אני אעביר את זה לאומרי, שיראה לנו איך לשים את כל זה באמת ב-[21:27] "AI on EKS.

2. Foundation — מה מכריח ארגון לשים Gateway

נקודת הפתיחה שתוארה היא לא בעיה טכנית אלא בעיה ארגונית. מספר צוותים, מספר מודלים, וכל צוות עם הספק וההעדפה שלו. המחיר שלוש בעיות שנמנו במפורש [1:31]: קושי להבין מה העלות cross-the-board, היעדר אכיפת מדיניות — "איך אנחנו מוודאים שצוות מסוים באמת יודע להשתמש במודל שאישרנו לו ברמת הארגון" — וקומפליאנס, כלומר חוסר יכולת לוודא שצוות משתמש רק במודל שמתאים ל-use case שלו.

התשובה שהתעשייה התכנסה אליה היא דפוס ותיק: "לשים איזשהו גייטווי, איזשהו ראוטר, לפני כל המודלים" [3:05]. במקום שכל צוות יקרא ישירות לספק, הכל עובר דרך נקודה אחת.



כל הצוותים מתנקזים לראוטר אחד; המודל שמנוהל עצמית הוא פשוט עוד יעד ברשימה

שלוש הגישות שהוצגו כדוגמאות לראוטר כזה, כל אחת בנקודה אחרת על הרצף: Envoy AI Gateway (כ-sidecar), LiteLLM (כ-deployment שרץ בתוך הקלאסטר שלכם), ו-OpenRouter (כשירות מנוהל) [3:05]. הנקודה החשובה לענייננו היא הקוביה בפינה — Self-Managed: "סך הכל עוד מודל שאתם מחברים, וככה אפשר לצרוך אותו בארגון עצמו" [3:05]. כלומר, ה-Gateway הוא מה שמאפשר להכניס מודל עצמי לארגון בלי שכל צרכן ידע על כך.

> **למה זה רלוונטי** עבור ארגון מפוקח, ה-Gateway הוא לא אופטימיזציה — הוא התנאי המקדים. בלעדיו אין auditing, אין אכיפת מדיניות מודלים ואין תמונת עלות אחת. הנושא חוזר בהרחבה בפרק 13.

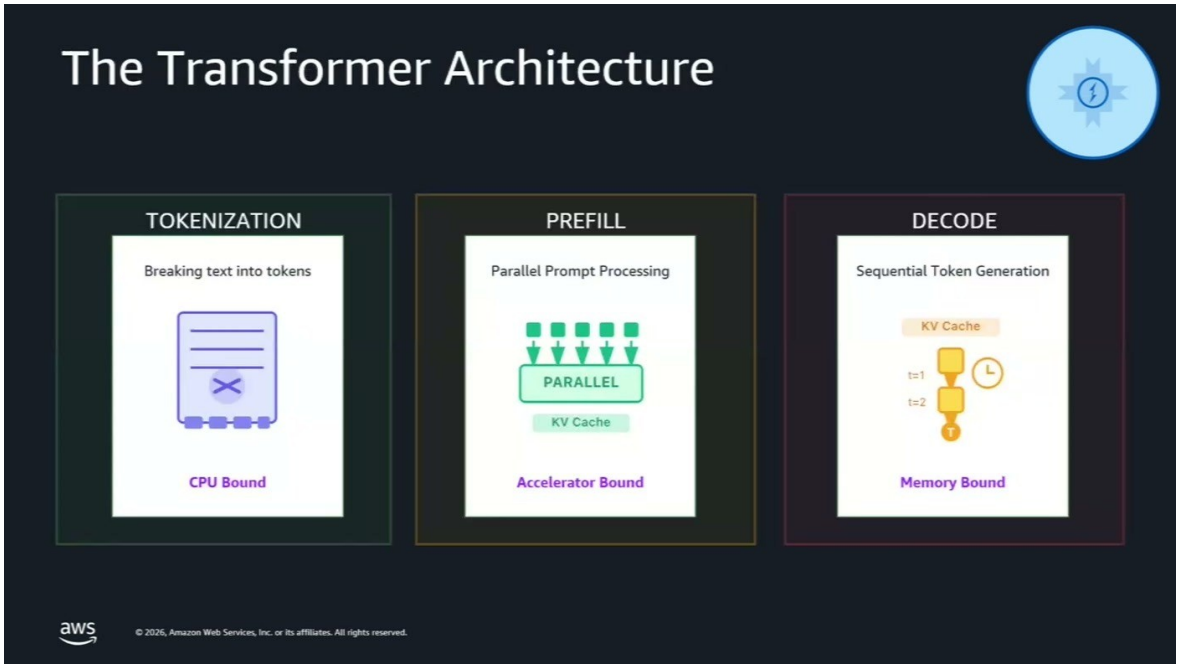
3. Deploy — הדרך הנאיבית ולמה היא יקרה

הדרך הבסיסית ביותר תוארה בשלושה צעדים: לקחת מודל בעל משקולות פתוחות מ-Hugging Face, לחבר אותו ל-inference engine, ולהתחיל להשתמש. ההערה על המינוח: "אינפרנס בקן ופריים או אינפרנס אנג'ין, כולם בערך זה בעצם אותו שם" [3:05].

הבעיה: בלי לשים לב לפרמטרים ולקונפיגורציה, "אנחנו נגיע ל-utilization מאוד נמוך, בין עם זה של הממורי, בין עם זה של ה-GPU" [3:05]. וזה כואב במיוחד כי המשאב עצמו תואר כ"מאוד מאוד סקארס", מוגבל בקיבולת, ו"יחסית יקר לכל שאר הדברים שאנחנו מכירים עד היום" [4:35].

שלושת השלבים של ההסקה

לפני האופטימיזציות הוצג רקע קצר, בהסתייגות מפורשת — "וזה לא crash course ל-LLM" [4:35] — אבל הוא מה שמסביר כל החלטת ארכיטקטורה בהמשך.



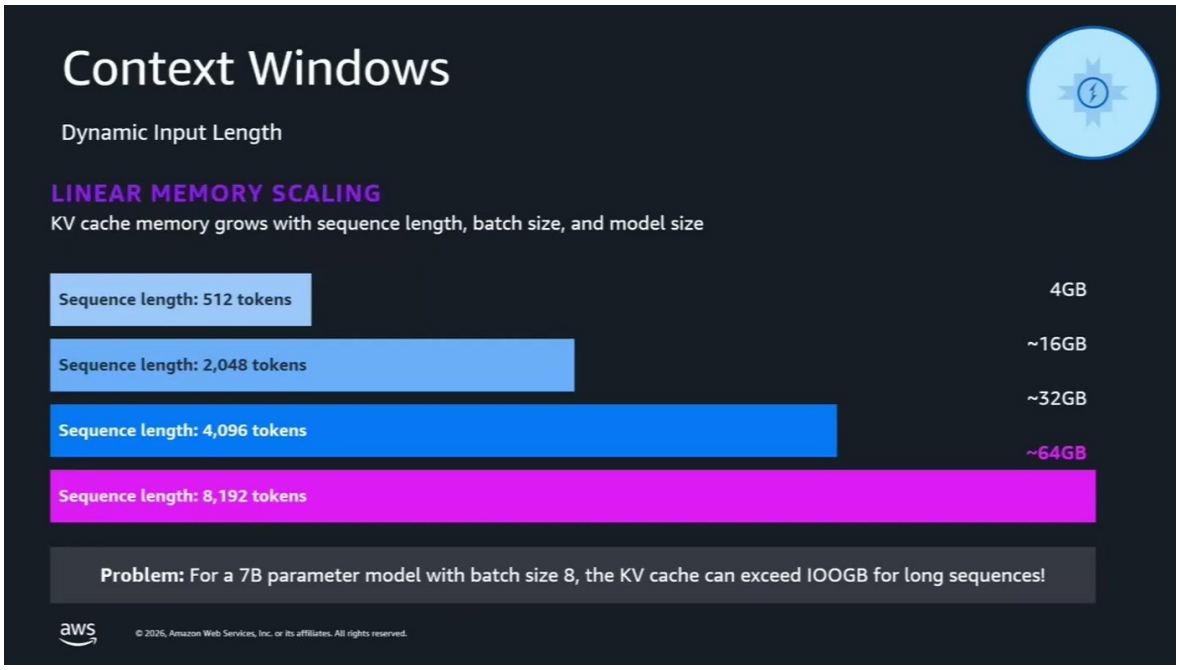
שלושת השלבים, וכל אחד עם צוואר הבקבוק שלו: CPU Bound, Accelerator Bound, Memory Bound

שלב	מה קורה בו	צוואר הבקבוק
Tokenization	תרגום הטקסט הנכנס לייצוג וקטורי של מספרים שהמודל יודע לעבוד איתו	CPU Bound — "לרוב הוא לא מהווה שום bottleneck"
Prefill	מעבר מקבילי על כל ה-input, וייצור ה-state (KV Cache) שממנו ייגזרו הטוקנים הבאים	Accelerator Bound
Decode	לולאה שמייצרת טוקן אחר טוקן עד End-of-Sequence	Memory Bound

ההבחנה הזו היא הציר של כל ההרצאה. Prefill מעמיס על יחידת העיבוד, Decode על הזיכרון: "ה-Decode הוא יותר משתמש בממורי, ובגלל זה אנחנו אומרים שהוא memory bound" [6:08]. מכאן ייגזרו בהמשך גם ה-KV Caching (פרק 7), גם הראוטינג (פרק 8) וגם ה-Disaggregated Architecture (פרק 8).

4. חלון ההקשר — הפרמטר ששולט בחשבון הזיכרון

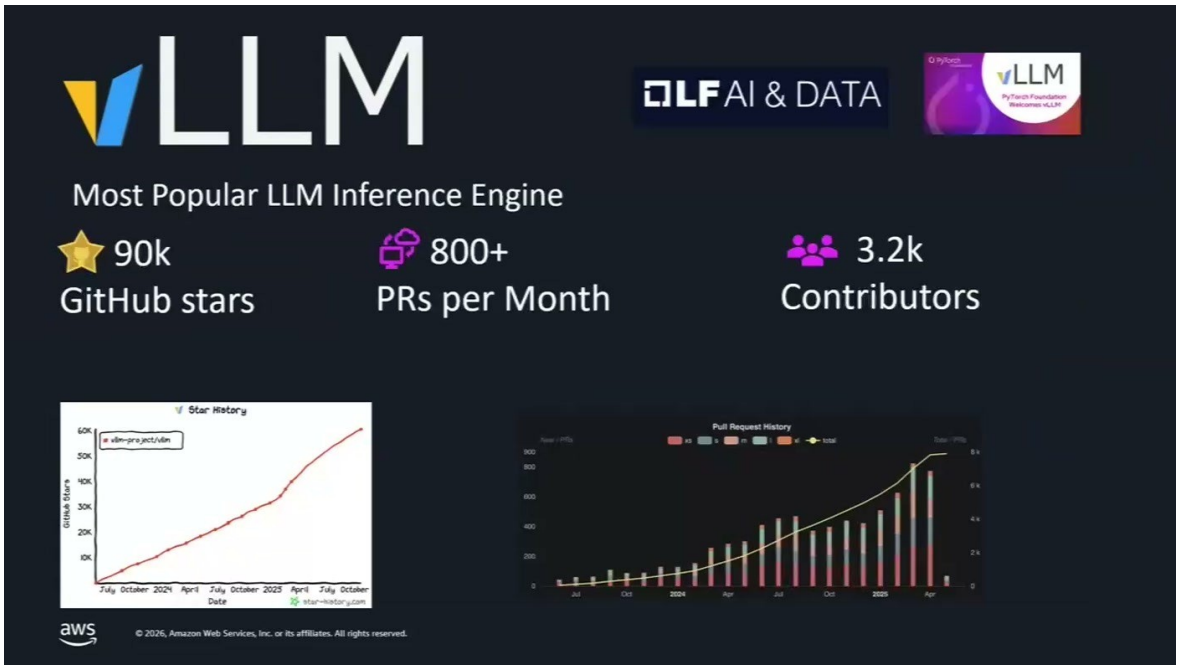
הנקודה שהודגשה: כשעובדים מול LLM מנוהל, חלון ההקשר הגדול הוא מובן מאליו והתמחור הוא לפי טוקני קלט ופלט. כשמריצים בעצמכם, לעומת זאת, "חשוב שאני מבין גם מה השייפ, מה הצורה של הדאטה שנכנס לנו" [6:08], כי "לגודל של הדאטה יש השפעה על כמות ה-GPU memory שאנחנו צריכים" [7:39].



גידול לינארי של זיכרון ה-KV Cache באורך הסדרה, כפי שהוצג על המסך

המספרים שעל הסלייד: זיכרון KV Cache של כ-4GB באורך סדרה של 512 טוקנים, כ-16GB ב-2,048, כ-32GB ב-4,096 וכ-64GB ב-8,192. השורה התחתונה שם: עבור מודל של 7 מיליארד פרמטרים עם batch size 8, ה-KV Cache יכול לחרוג מ-100GB בסדרות ארוכות. זו נקודת ההחלטה על סוג ה-GPU ועל כמות הזיכרון שמקצים לו.

5. vLLM — למה דווקא הוא, ומה הוא עושה



הסטטיסטיקות שהוצגו לפרויקט, לצד גרפי הכוכבים וה-Pull Requests

המספרים שעל הסלייד: 90 אלף כוכבים ב-GitHub, מעל 800 Pull Requests בחודש, ו-3.2 אלף תורמים. בעל פה נאמר "90 אלף סטארים בגיטהאב, ו-3 אלף עם קונטריביוטרס, נכון לפעם האחרונה שהכנו את המצגת הזאת" [7:39].

מעבר לפופולריות, הנימוק שניתן הוא חומרה: מנוע הסקה לא רק מפשט צריכה של מודלים מארכיטקטורות שונות, אלא "מאפשר לנו לבוא ולנצל בצורה יעילה סוגים שונים של אינסטנסים" — גם NVIDIA GPUs וגם מאיצים של AWS, Trainium ו-Inferentia [7:39]. זו טענה מעשית: אותו chart, חומרה אחרת.

שלוש טכניקות האופטימיזציה שהוצגו

- **PagedAttention —** במקום לשמור את ה-KV ברצף בזיכרון ולייצר פרגמנטציה, מחלקים אותו לדפים עם טבלה שמצביעה על כל חלק, בדיוק כמו מערכת הפעלה. התוצאה שנמסרה: המעבר היה ממצב של "60% ווייסט" ל"בזבוז של אחוזים בודדים" [9:09].
- **Continuous Batching —** איגוד דינמי של בקשות ממשתמשים שונים. "אנחנו לא מחכים עד שקריאה אחת של LLM תסתיים עד סוף הסיקוונס" — אלא מצרפים בקשות שמגיעות תוך כדי ושולחים אותן ל-GPU יחד, וכך חוסכים מחזורי GPU [9:09].
- **Quantization —** הורדת רמת הדיוק של משקולות המודל (FP16 ל-FP8 ומטה), "לחתוך אותם בחצי, לחתוך אותם ברבע" [9:09]. דורש התאמה במודל עצמו, ובתמורה מצמצם את ה-memory footprint.

מספרים שנאמרו בעל פה "הם יכולים להגיע לניצולת GPU של 80%, 5x Better Throughput ו-savings על כמות הטוקנים שהמודל בעצם מייצר" [10:42]. המספרים נאמרו כהערכת סדר גודל לשילוב הטכניקות ולא כתוצאת מדידה של תרחיש ספציפי.

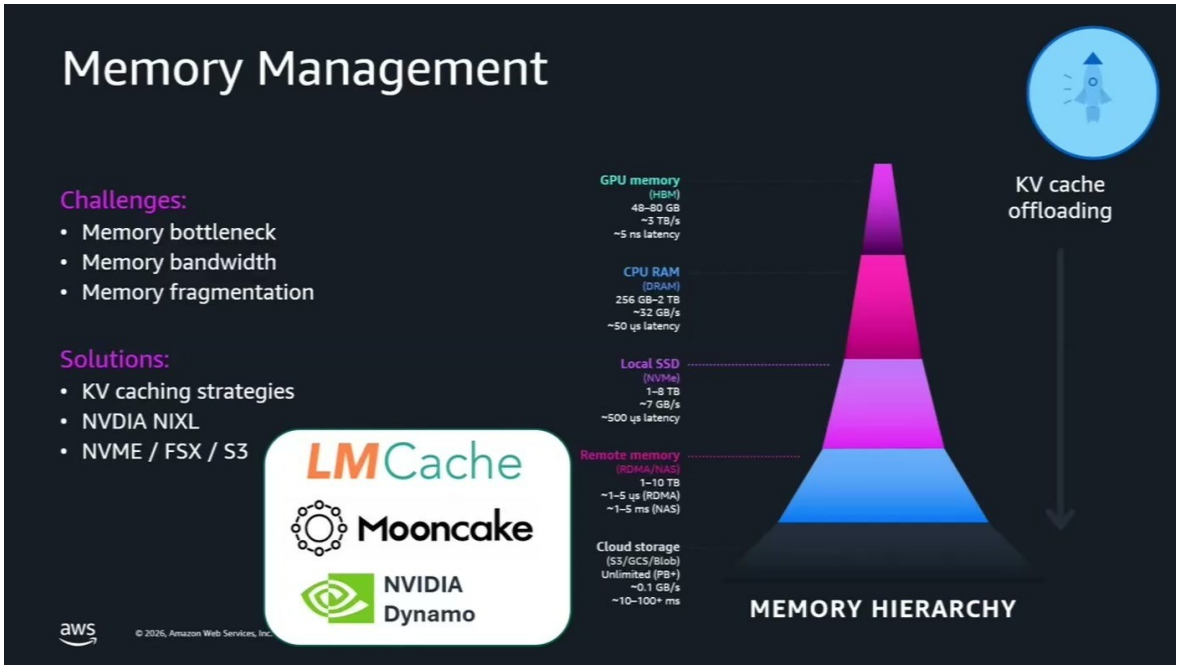
והמתחרים

שלושת מנועי ההסקה שנמנו כשכיחים ביותר: vLLM, SGLang ו-NVIDIA Triton [10:42]. ההמלצה הייתה לבנצ'מרק ביניהם בעצמכם — ואיתה אילוץ אחד קשיח שכדאי לשים לב אליו מראש: "NVIDIA Triton כמובן יעבוד אך ורק עם NVIDIA GPU ולא עם AWS Inferentia" [10:42].

6. Accelerate — הבעיה של ה-Prefix המשותף

התרחיש שהוצג: מערכת RAG שעובדת. יש system prompt משותף לבקשות דומות, ויכול להיות שגם ההקשר שנשלף מביא את אותם מסמכים [10:42]. שתי בקשות — "What is the revenue of Q4" ו-"Show Q4 earnings" — יובילו לחישוב של אותו state, אותו KV, עבור החצי הראשון של הבקשה. "הדבר הזה בעצם מוביל לאיזשהו בזבוז של הזיכרון" [12:16].

מכאן שתי אסטרטגיות שהוגדרו במפורש: KV Caching ו-KV Prefix Caching. ומעליהן עיקרון: לא כל זיכרון שווה. "ככל שהממורי שלנו קרוב יותר ל-GPU כך הוא באמת יותר מהיר... אבל הוא גם מוגבל יותר" [12:16] — כ-40 עד 80 ג'יגה-בייט ל-GPU יחיד.



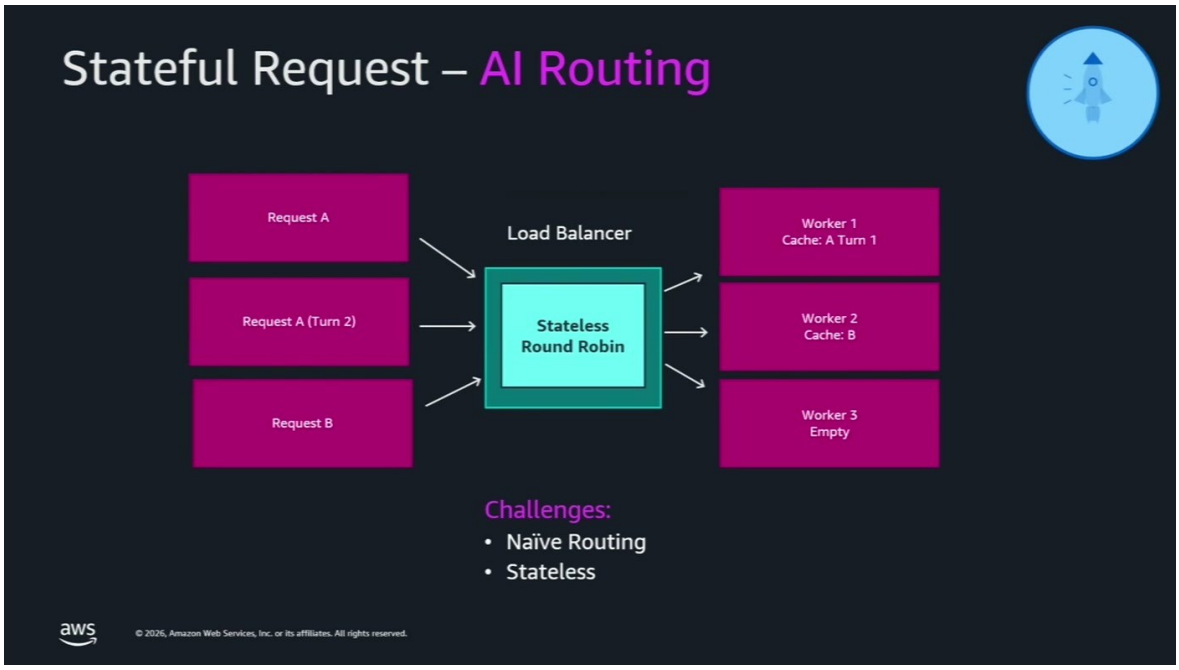
היררכיית הזיכרון, מזיכרון ה-GPU ועד אחסון ענן, לצד הפרויקטים שמנהלים את ההורדה בין השכבות

כשרוצים להחזיק כמה בקשות במקביל צריך להתחיל להוריד את ה-KV בין ה-tiers: CPU memory, NVMe, ו-all the way down עד אחסון ענן — "שבעצם אנחנו יכולים להגיע לטרות ופטות" [12:16]. שלושת הפרויקטים שנמנו כמה שנראה בשטח: LMCache, Mooncake ו-NVIDIA Dynamo, שכולם משתלבים ב-vLLM [13:49].

> **המסר של הפרק** "אתם צריכים לקחת ולהתייחס לאסטרטגיה של ה-KV Caching שלכם וגם של ה-KV Cache Offloading — מתי אתם מוציאים אותו מהזיכרון ואיך אתם עושים את זה" [13:49]. זו לא הגדרה שמגיעה עם ברירת מחדל טובה.

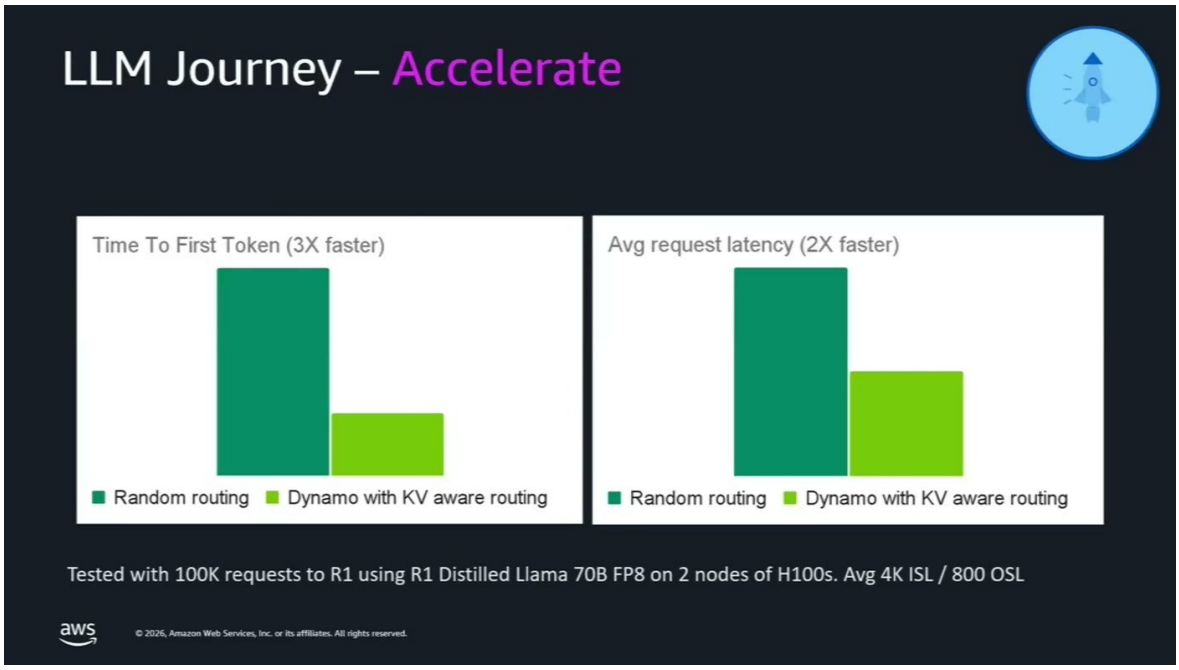
7. ראוטינג — למה Round-Robin שובר את ה-Cache

זו אחת הנקודות החדות בהרצאה: "סטייטלס ראוטינג או ראונד רובינג לא באמת עובד" [13:49]. הסיבה חוזרת לשלב ה-Prefill. אם Worker 1 כבר חישב את ה-KV Cache של בקשה A, והתור השני של אותה שיחה מנותב ב-Round-Robin ל-Worker 2 — ל-Worker 2 יש cache של בקשה אחרת לגמרי, וכל העבודה מתבצעת מחדש.



מה קורה כשה-Load Balancer לא יודע מי מחזיק את ה-Cache: Worker 2 עם cache זר, Worker 3 ריק

הדרישה שנגזרת: לשלוח בקשות מאותו session, בקשות של אותו מודל, או בקשות עם אותו prefix KV Cache, אל ה-Worker המתאים [13:49]. זה מצריך Intelligent AI Router — והדוגמה שניתנה היא NVIDIA Dynamo, עם שלושה סוגי מודעות: Session, Prefix ו-[15:19] Capacity Aware Routing.

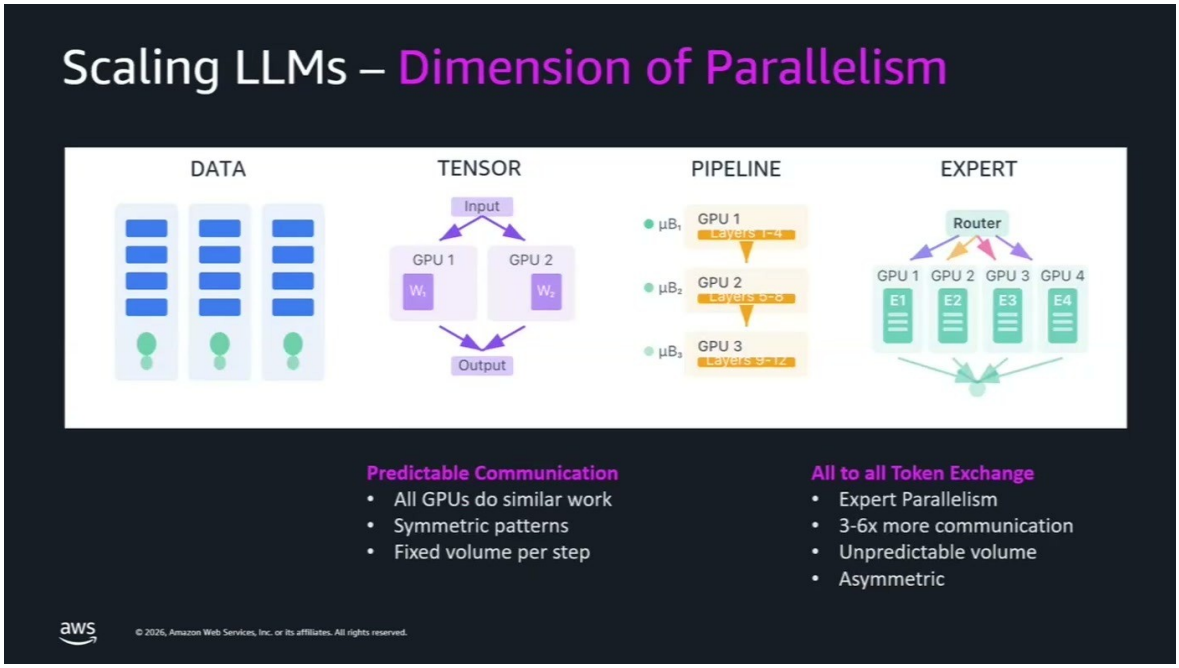


בנצ'מרק מאתר Dynamo: ראוטינג אקראי מול ראוטינג מודע ל-KV

המספרים שעל הסלייד: שיפור פי 3 ב-Time To First Token ופי 2 ב-Average Request Latency. תנאי הבדיקה מצוינים על הסלייד עצמו — 100 אלף בקשות ל-R1 Distilled Llama 70B ב-FP8, על שני nodes של H100, בממוצע 4K ISL / 800 OSL. הבנצ'מרק צוין במפורש כלקוח מאתר NVIDIA Dynamo ולא כמדידה של AWS [15:19].

8. Scale — ארבעה ממדי מקביליות

החשבון שפותח את הפרק: A100 מגיע עם 80 ג'יגה-בייט זיכרון. מודל של 70 מיליארד פרמטרים ב-FP16 — "רק המודל עצמו הוא 140 ג'יגבייט של משקולות" [15:19]. זה לא נכנס, ולכן צריך טכניקות sharding.



ארבעת הממדים: Data, Tensor, Pipeline ו-Expert, ומה כל אחד דורש מהתקשורת בין ה-GPUs

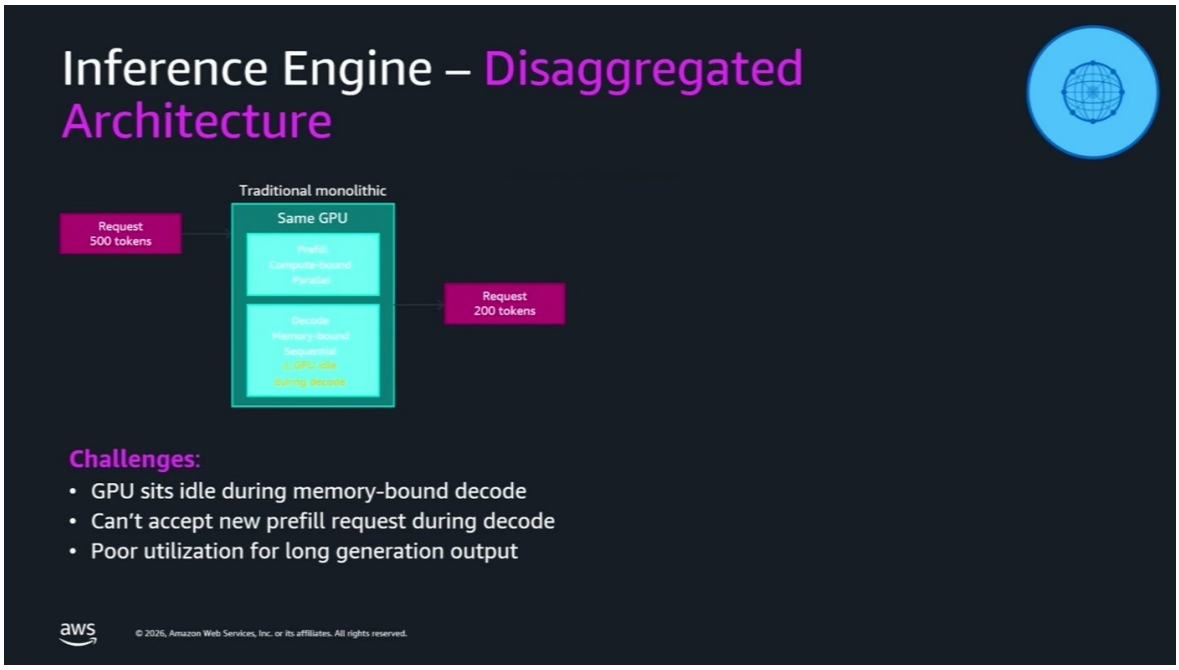
ממד	מה הוא עושה	מתי משתמשים
Data Parallelism	אין sharding כלל — המודל מפורס בכמה רפליקות והבקשות מגיעות לנקודות קצה שונות	כשהמודל נכנס ב-GPU יחיד
Tensor Parallelism	לוקח את המשקולות מהשכבות ומחלק אותן על פני מספר GPUs, ואוסף את התוצאות אחרי כל טוקן	כשהמודל לא נכנס ב-GPU אחד
Pipeline Parallelism	מחלק את שכבות המודל בין GPUs; כל node מחשב את השכבות שלו ומעביר את ה-state הלאה בצינור	כשגם 8 GPUs בשרת אחד לא מספיקים
Expert Parallelism	ראוטר בתוך המודל מחליט אילו חלקים פעילים לכל בקשה ומנתב ל-GPU המתאים	מודלים מודרניים מסוג MoE

הצירוף השכיח שצוין: "אנחנו נעשה pipeline בין נודים ו-tensor בתוך הנוד עצמו" [16:51], מה שמאפשר להריץ dense models עם מאות מיליארדי פרמטרים ולעשות להם scale על פני מספר אינסטנסים.

מה שלא הוזכר בהרחבה אבל חייב תשומת לב "חשוב לשים לב לנטוורק קונקטיביטי ביניהם. ב-AWS זה נקרא EFA, Elastic Fabric Adapter" [18:22] — הרשת בעלת הביצועים הגבוהים שמאפשרת תקשורת node-to-node. ההרצאה לא נכנסה לעומק, אבל זו התלות שמכריעה אם Pipeline Parallelism יעבוד בכלל.

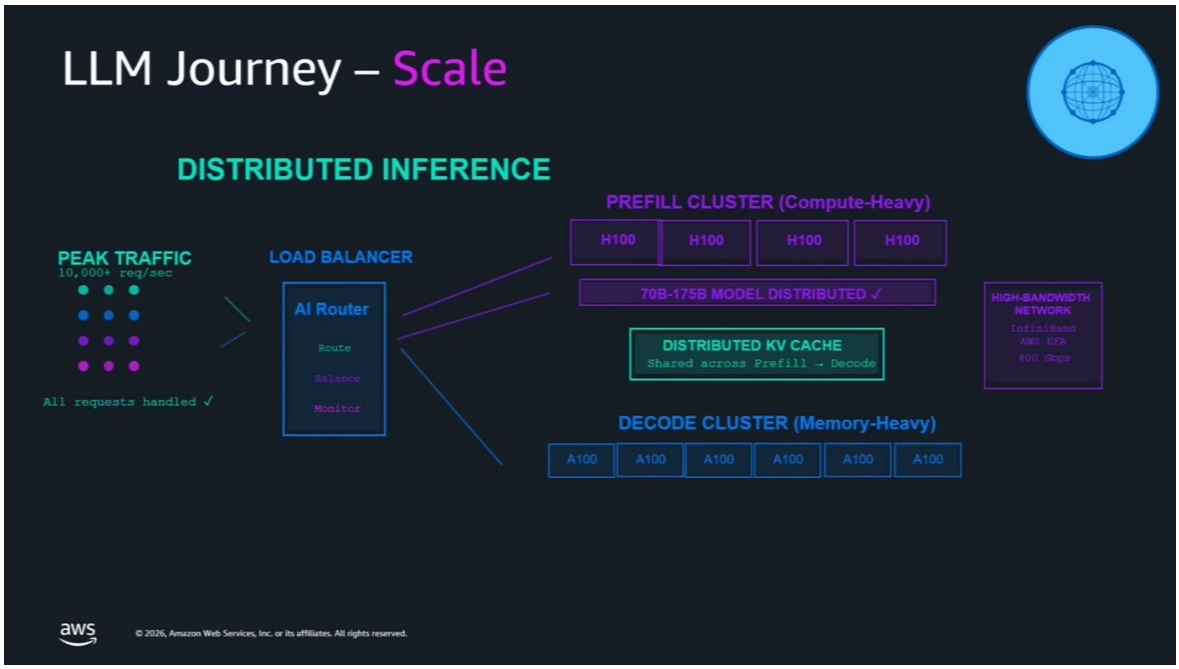
ארכיטקטורה מפוצלת (Disaggregated)

החזרה לפרק 3 סוגרת מעגל: במצב רגיל, Prefill ו-Decode רצים על אותו GPU — "אתם יכולים להבין ששניהם לא מנצלים את ה-GPU בצורה שווה. אחד מנצל את ה-GPU ברמת האקסלרטור... והשני מנצל את ה-GPU ברמת הממורי" [18:22].



האתגרים שמצדיקים את הפיצול: GPU מובטל בזמן Decode, חוסר יכולת לקבל Prefill חדש, וניצולת ירודה בפלט ארוך

הפתרון: "אנחנו מחלקים את העבודה שלנו לשני קלאסטרים שונים, קלאסטר של Prefill, קלאסטר של Decode, ובעצם מעבירים את אותו state, את אותו KV Cache, מ-node ל-node" [19:56]. הרווח שתואר הוא כפול-שלושה: מכונות מסוגים שונים לכל תפקיד, מה שפותר גם בעיות capacity, גם בעיות מחיר וגם ניצולת זיכרון.

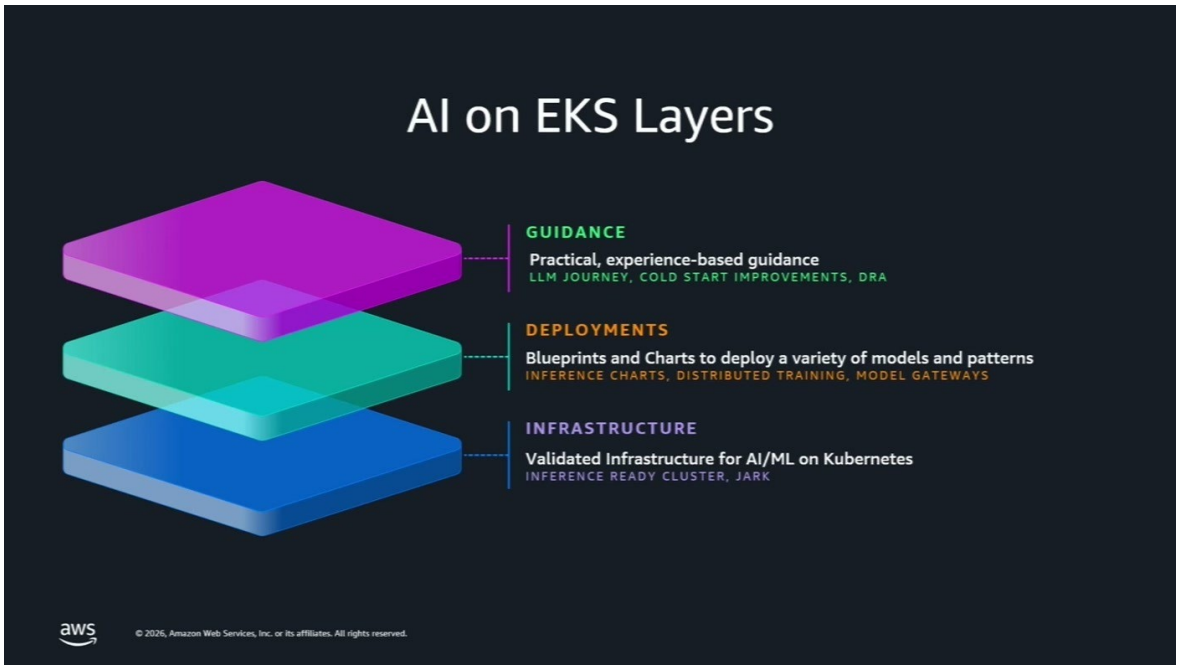


התמונה המלאה: ראוטר מודע ל-KV מנתב לקלאסטר Prefill או Decode, עם KV Cache מבוזר ביניהם

הסתייגות חשובה מהסשן לפי הבנצ'מרק שהוצג מאתר Dynamo, יתרון הביצועים של הארכיטקטורה המפוצלת "לא יגיע מיד מנוד אחד, אלא ככל שאתם תעשו scale גם ברמת הבקשות, גם ברמת הנודים וגם ברמת הקונקרנסי" [21:27]. במילים אחרות — זו לא אופטימיזציה שמשתלמת ביום הראשון.

9. AI on EKS — מהתיאוריה לקוד

החלק השני של הסשן נפתח בהצהרת כוונות: "אז למדנו עכשיו הרבה תיאוריה על ה-LLMs, ואנחנו רוצים לדעת איך לשים את כל התיאוריה הזאת בפרקטיקה" [21:27]. הכלי הוא AI on EKS — פרויקט אופן-סורס שמתוחזק על ידי AWS ומשמש להדגים את כל הדפוסים שנסקרו.



שלוש שכבות הפרויקט, מלמטה למעלה

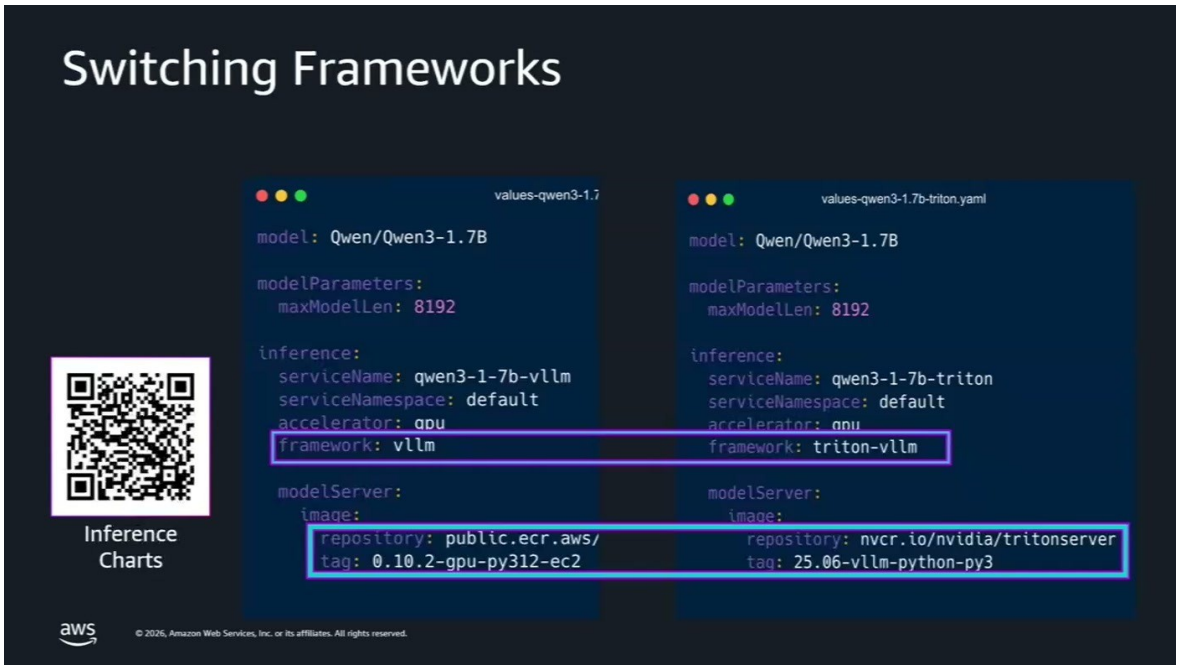
שכבה	מה יש בה
Infrastructure	קוד Terraform שנותן VPC, EKS וכל האופרטורים שצריך כדי להריץ LLM — ולא רק LLM, גם תשתית לסוכנים
Deployments	הדפוסים עצמם: Blueprints ו-Charts לפריסת מודלים, אימון מבוזר ו-Model Gateways
Guidance	דיפ-דייב מבוסס ניסיון: Cold Start, בנצ'מרקינג, בניית סוכנים — "למה בחרנו לעשות משהו בדרך שבחרנו את זה ומה אלטרנטיבות" [23:04]

נקודת ההתחלה המעשית נקראת Inference Ready Cluster. ההתקנה היא git clone, מעבר לתיקייה והרצת install.sh, עם קובץ ערכים שמאפשר לסמן בדגלים מה מותקן — kuberay operator, observability stack, AIBrix, LeaderWorkerSet, SOCI snapshotter וכן הלאה. ההערה המעשית היחידה שנמסרה על הקובץ: "אני רק מציע, תבדקו את ה-region, אני לא בטוח שאתם רוצים ב-[23:04] "us-west-2.

> **טיפ מעשי מהבמה** "יהיו הרבה QR-ים, הם כולם מגיעים לאותו אתר, אז אם אתם מעוניינים כמובן תסרקו, אבל אפשר לסרוק אחד ולהגיע לכל החומר" [21:27].

10. Inference Charts — החלפת מנוע בשורה אחת

מעל התשתית יושבים Inference Charts. הם מכסים לא רק LLMs אלא גם מודלי Diffusion ל-Text to Image, ותומכים ב-vLLM, llama.cpp, Ray, Triton ועוד [23:04]. עיקרון העיצוב שהוצהר: "ניסינו רק להראות לכם את הפרמטרים שאתם צריכים, ולהסתיר את היתר" [24:37] — בוחרים מודל, משנים Max Model Length, וכל מה שאפשר להעביר ל-vLLM נגיש.



אותו מודל, שני values files: השורות המסומנות הן כל ההבדל בין vLLM ל-Triton

זו הנקודה הפרקטית של הפרק: "אתם לא צריכים לשנות מודל, אתם לא צריכים לשנות פרמטרים, אתם פשוט אומרים אנחנו רוצים עכשיו Triton" — ומשנים את ה-image [27:46]. מה שהופך את ההשוואה בין מנועי הסקה, שבפרק 5 הומלצה בעל פה, לדבר שאפשר לעשות בפועל.

מה בעצם קורה כשמתקינים chart

הרצף שתואר [26:12]: הקלאסטר בנוי על Karpenter — "אנחנו בנינו את כל הקלסטר הזה על קרפנטר" — ולכן במצב שאין LLM, אין בכלל GPU בקלאסטר, וזו החלטת חיסכון מודעת. התקנת ה-chart מבקשת pod, Karpenter מביא node, מושכים image מ-Docker Hub, הקונטיינר עולה, שולפים את המשקולות, ואז אפשר לשלוח בקשות בתוך הקלאסטר (או החוצה, דרך port-forward או Ingress).

11. בנצ'מרקינג — ושתי אזהרות

למה בכלל: "זה לא מספיק לדעת... מה התוצאה של הבקשה. אנחנו בעצם רוצים לדעת כמה זמן הבקשות לוקחות ב-load, ולדעת איך אנחנו מוצאים את הפרמטרים הכי טובים בשביל המודל שלנו" [27:46]. בפועל מתקינים chart נוסף שמריץ Inference Perf ומחזיר מסך מטריקות — Inter-Token Latency, Time to First Token וכל השאר [29:19]. הוזכרו גם GuideLLM, שיודע לשמור את התוצאות, ו-AIPerf של NVIDIA.

אזהרה ראשונה: דאטה סינתטי משקר

"אנחנו מאוד ממליצים לכם להשתמש בדאטה שאתם תראו בפרודקשן, בגלל שהצורה של הטוקנים היא משפיעה על הבנצ'מרק" [27:46]. הדוגמה שניתנה חדה: agentic coding משתמש ב-caching כל הזמן כי הוא שולח את אותן בקשות שוב, בעוד עומס של הרבה בקשות קטנות עם שאלות הוא "בנצ'מרק שונה לגמרי". המסקנה נאמרה כבקשה: "אז בבקשה תשתמשו בדאטה שלכם" [29:19].

אזהרה שנייה: מהירות היא חצי מדידה

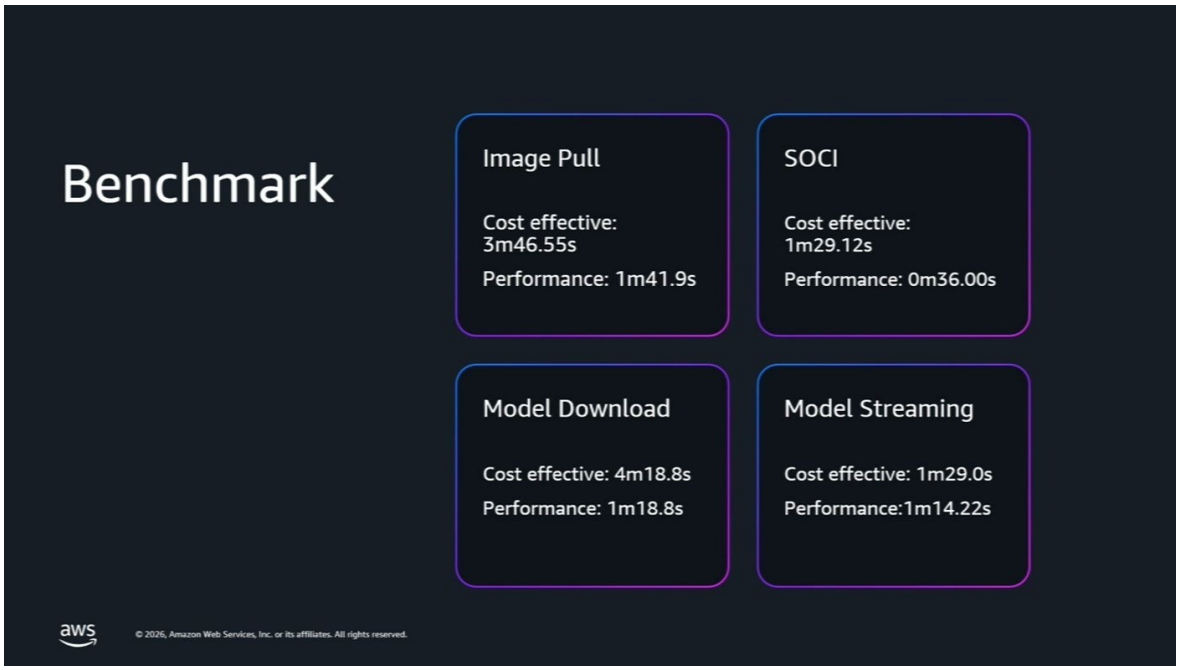
"זה יכול להיות מודל מהיר מאוד, אבל אם זה לא מביא תשובות נכונות בחזרה, זה לא ממש מודל שהוא יהיה יעיל ל-use case" [29:19]. מדידת איכות התשובות — evals — הוזכרה כשלב חיוני של הבנצ'מרקינג, אך נדחתה מפאת חוסר זמן [29:19].

> **פער מהשטח** ה-evals הן החלק היחיד במסלול שהוצג כחשוב אך לא הודגם. מי שלוקח את הפרויקט הזה לארגון צריך להשלים אותו בעצמו — בנצ'מרק latency בלי בנצ'מרק איכות הוא מדד חלקי בלבד.

12. Scaling ו-Cold Start — המספרים הכי שימושיים בסשן

לאחר שמצאנו מודל ופרמטרים והעלינו לפרודקשן, מגיעה התנועה. הדגמת ה-autoscaling השתמשה באותו Inference Chart עם ray-vllm והגדרות autoscaling: שולחים יותר תנועה ממה שהמודל יכול לשרת, Ray מזהה ומוסיף pod [30:54]. אלא שאז מתחיל כל התהליך מחדש — בקשת node, העלאת קונטיינר, שליפת משקולות.

וזה בדיוק ה-Cold Start: "השאלה של כמה זמן זה לוקח בעצם להוריד עשרות ג'יגה-בייט קונטיינר, ואז את ה-weights" [32:32]. שתי החלופות שנשקלו ונדחו כדורשות תחזוקה: הטמעת הקונטיינר ב-AMI של Bottlerocket ("אם אתם מוסיפים קונטיינר חדש... אתם צריכים להתקין את זה בחזרה, וזה הרבה עבודה") ושמירת המשקולות ב-FSx [32:32]. הפתרון שהוצג: SOCI לקונטיינר, ו-Model Streaming מ-S3 למשקולות — ECR במקום Docker Hub, S3 במקום Hugging Face [34:07].



הבנצ'מרק שהוצג: בכל תא — למעלה הנוד הזול, למטה הנוד החזק

שלב	נוד זול	נוד חזק	השיפור
משיכת Image (רגיל)	3:46.55	1:41.9	—
משיכת Image עם SOCI	1:29.12	0:36.00	פי 2.5 / פי 2.8
הורדת מודל (רגיל)	4:18.8	1:18.8	—
מודל עם Model Streaming	1:29.0	1:14.22	פי 2.9 / זניח
סה"כ	2:58 → 8:05	1:50 → 3:00	—

המספרים בטבלה הם מהסלייד; הסיכומים נאמרו בעל פה [35:38]. והמסקנה המעניינת היא לא השיפור עצמו אלא ההיפוך שהוא יוצר: "הנוד הקטן שהיה יותר חיסכוני בסוף יצא יותר מהר מהנוד הגדול שהתחלנו איתו" [35:38]. כלומר, האופטימיזציה לא רק מקצרת את ה-Cold Start — היא משנה את החלטת בחירת האינסטנס.

13. Distributed Inference ו-LLM Gateways

מודלים כמו GLM ו-Kimi החדשים פשוט לא נכנסים בשרת אחד. מעבר לזה, נאמרה בעיה מעשית שפחות מדברים עליה: גם כשקיים אינסטנס שמסוגל להחזיק את המודל, לפעמים הוא פשוט לא זמין [35:38]. Distributed Inference פותר את שתי הבעיות — גם מודלים גדולים מדי, וגם מודל קטן על נודים זולים יותר.

המימוש ב-Charts הוא LeaderWorkerSet עם vLLM ו-Pipeline Parallelism, כשהמספר בקונפיגורציה הוא מספר ה-nodes [37:08]. וההמלצה התפעולית: "תשאירו את זה באותו AZ אם אתם יכולים, זה יהיה יותר חיסכוני כי זה לא צריך ללכת סביב AZ, וזה יהיה יותר מהר" [35:38].

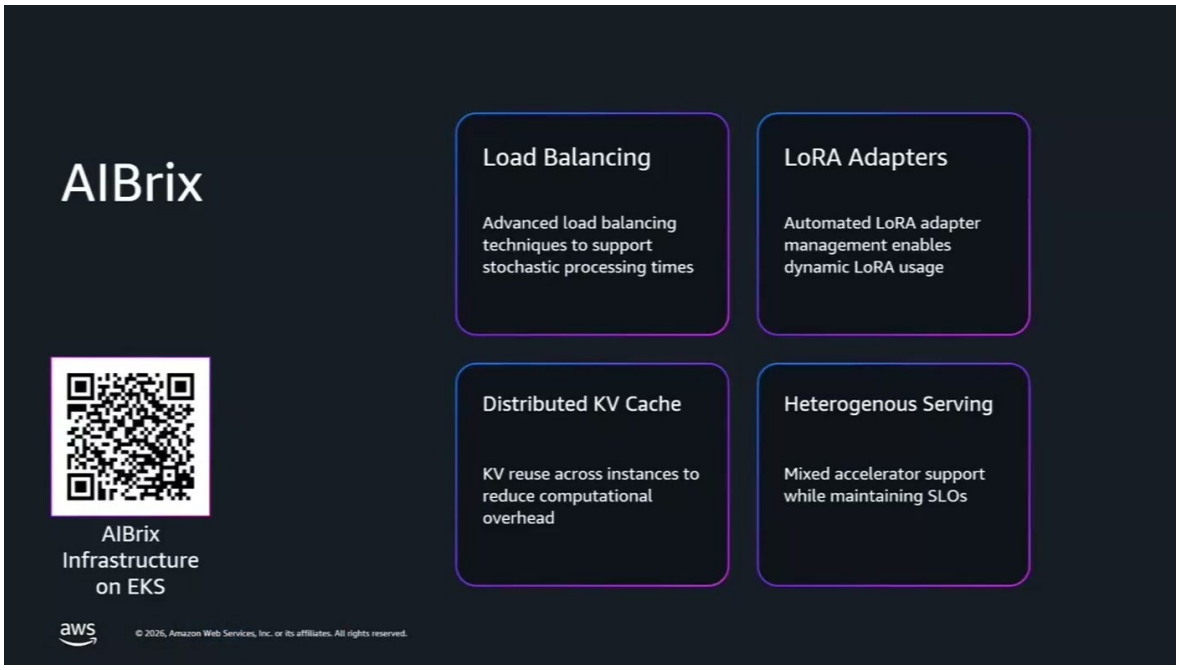
מכאן עבר הסשן מ-token maxing ל-token economics [37:08]: "אנחנו מנסים לא לשלוח כל בקשות למודל הפרונטיר הכי יקר" — אלא לזהות איזה מודל מספיק כדי לענות על הבקשה.

ארבע הפונקציות שה-Gateway מרכז: ראוטינג עם fallback, שקיפות, guardrails וניהול מפתחות

הדמו הראה שלוש התנהגויות: בקשה גדולה מנותבת ל-Bedrock, בקשה קטנה למודל קטן, וכשמופיעה תקלה — "במקום להחזיר את התקלה בחזרה למפתח, ללקוח, אנחנו שולחים את זה למודל אחר שלנו" [37:08]. בנוסף, ה-Gateway הוא מה שנותן לצוותי הפלטפורמה לראות לאן נשלחות בקשות ומה נשלח, ולבצע auditing [37:08] — מה שסוגר את הלולאה מול שלוש הבעיות שפתחו את ההרצאה בפרק 2.

14. AIBrix — האופטימיזציות המתוחכמות

הנושא האחרון, שהוגדר מראש כ"מהר מאוד" [38:40], הוא AIBrix — לצד Dynamo ו-llm-d שהוזכרו כפרויקטים מאותה משפחה. שלוש היכולות שנמנו: KV Cache Distribution, replica-aware load balancing ("שאנחנו לא סתם עושים round robin"), ו-LoRA adapter switching.



ארבע היכולות של AIBrix כפי שהוצגו, עם ה-QR לחומר המלא

מבחינת חוויית המשתמש בפרויקט, הטענה היא שהמעבר שקוף: "רק משנים מ-vLLM ל-AIBrix, וזה נותן לנו את כל המערכת של AIBrix" [38:40]. וספציפית ל-LoRA: המערכת "יודעת דינמית להחליט איזו רפליקה לשים את ה-weights" [38:40] — כלומר טעינת אדפטרים בלי הקצאת רפליקה ייעודית לכל אחד.

לשים לב AIBrix הופיע בתיאור הסשן כ"deep dive", ובפועל קיבל פחות משתי דקות. מי שזה הנושא שלו — החומר נמצא בשכבת ה-Guidance של הפרויקט, לא בהקלטה.

15. מה לוקחים מכאן

- **ההחלטה הראשונה היא Gateway, לא GPU.** עלות, אכיפת מדיניות וקומפליאנס נפתרים בשכבת הראוטר, והם התנאי שמאפשר להכניס מודל עצמי לארגון בלי לשנות כלום אצל הצרכנים [3:05, 37:08].
- **Prefill מול Decode הוא הידע שמסביר את כל השאר.** שלב אחד עמוס על המאיץ והשני על הזיכרון — וזה מה שמצדיק KV Caching, ראוטינג מודע-prefix, וארכיטקטורה מפוצלת [4:35, 18:22].
- **Round-robin לפני מודלי שפה הוא באג ביצועים.** כל ראוטר שלא מודע ל-session ול-prefix מחייב חישוב מחדש של ה-Prefill; הבנצ'מרק שהוצג מדבר על פי 3 ב-TTFT [13:49, 15:19].
- **Cold Start הוא נקודת המינוף הזולה ביותר.** SOCI ו-Model Streaming מ-S3 חתכו את זמן העלייה של הנוד הזול מ-8:05 ל-2:58, עד כדי כך שהוא עקף את הנוד היקר [35:38]. זו אופטימיזציה שלא דורשת לשנות שום דבר במודל.
- **בנצ'מרק על הדאטה שלכם, ולא רק על מהירות.** צורת הטוקנים משנה את התוצאה לגמרי, ומודל מהיר שעונה לא נכון הוא לא מודל שימושי [27:46, 29:19].
- **אל תבנו את התשתית מאפס.** AI on EKS מספק Terraform, Charts ו-Guidance לכל דפוס שהוצג — נקודת ההתחלה היא Inference Ready Cluster [23:04].

נקודות להמשך

נושא	למה זה חשוב	חותמת זמן
Evals לאיכות תשובות	הוצג כשלב חיוני בבנצ'מרקינג אך לא הודגם מפאת זמן	29:19
EFA ורשת בין nodes	התלות שמכריעה אם Pipeline Parallelism יעבוד; לא נכנסו לעומק	18:22
AIBrix	הוגדר כ-deep dive בתיאור הסשן וקיבל פחות משתי דקות	38:40
בחירת מנוע הסקה	ההמלצה הייתה לבנצ'מרק בעצמכם; Triton לא רץ על Inferentia	10:42
מתי Disaggregated משתלם	היתרון מתגלה רק בסקייל של בקשות, nodes וקונקרנסי	21:27

— סיום הסשן [38:40] אז כל ג'רני מתחיל עם צעד ראשון, אז אנחנו מאוד מקווים שהפרויקט הזה, שמה שלמדתם היום, יעזור לכם להתחיל עם הצעד הזה, או אם אתם איפשהו אחר בתהליך, אם נתנו לכם כלים בשביל להגיע לצעד הבא.

16. מונחון

מונח	פירוש כפי שהוצג בסשן
vLLM	מנוע הסקה אופן-סורס, הפופולרי ביותר להרצת LLMs; 90 אלף כוכבים ו-3.2 אלף תורמים לפי הסלייד
KV Cache	ה-state (Key/Value) שנוצר בשלב ה-Prefill ומשמש לייצור כל טוקן הבא
Prefill / Decode	שני שלבי ההסקה: הראשון Accelerator Bound, השני Memory Bound
PagedAttention	שמירת ה-KV Cache בדפים עם טבלת הצבעה, בסגנון מערכת הפעלה, במקום ברצף
Continuous Batching	איגוד דינמי של בקשות ממשתמשים שונים לאותה קריאת GPU
Quantization	הורדת דיוק המשקולות (למשל FP16 ל-FP8) לצמצום ה-memory footprint
Tensor / Pipeline Parallelism	חלוקת משקולות בין GPUs מול חלוקת שכבות בין nodes
Expert Parallelism	מודלי MoE — ראוטר פנימי מפעיל רק את החלקים הרלוונטיים לבקשה
Disaggregated Architecture	הפרדת Prefill ו-Decode לשני קלאסטרים עם חומרה שונה לכל אחד
EFA	Elastic Fabric Adapter — רשת ביצועים גבוהים ב-AWS לתקשורת node-to-node
SOCI	טכניקה להאצת משיכת image של קונטיינר ל-node
Model Streaming	העלאת משקולות המודל מ-S3 במקום מ-Hugging Face, לקיצור Cold Start
LeaderWorkerSet	המנגנון ב-Kubernetes שמשמש לפריסת Distributed Inference רב-nodes
Karpenter	מנגנון הקצאת nodes דינמית; בקלאסטר שהוצג אין GPU כשאין LLM פעיל
LoRA	אדפטר שמשנה את משקולות המודל; AIBrix טוען אותו דינמית לרפליקה מתאימה
AIBrix / Dynamo / llm-d	פרויקטים לתזמור vLLM בסקייל: KV מבוזר, ראוטינג מודע-רפליקה, LoRA
Inference Perf / GuideLLM / AIPerf	כלי בנצ'מרקינג שהוזכרו למדידת TTFT ו-Inter-Token Latency