Inference של LLM על Trainium עם vLLM ו-PyTorch Native

AWS Summit Tel Aviv 2026 ,סיכום מושן — DEM305

Guy Almog, Senior Solutions Architect, AWS · Guy Ben Baruch, Principal Solutions Architect, AWS

10 בספטמבר 2026 | משך: 25 דקות | הופק מהקלטת המושן

סיכום מאת קובי אלמוג

על המסמך הזה

המסמך מסכם את הסשן DEM305 מתוך AWS Summit Tel Aviv 2026 — סשן מסלול AWS Demos ברמה 300, שרובו הדגמה חיה ולא מצגת. הוא נבנה מתמלול אוטומטי של ההקלטה המלאה ומהסליידים שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב בעשרים וחמש הדקות. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה הוצג ומה המסקנה המעשית.
- **פרקים 2–3 — החומרה:** משפחת הצ'יפים של AWS, ומה בדיוק יש בתוך Trainium2.
- **פרקים 4–5 — הסופטוור:** Neuron Developer Stack, PyTorch Native, NKI ו-Neuron Explorer.
- **פרקים 6–8 — הדמו:** ההשוואה החיה GPU מול Trainium2, ה-load testing וכוונון ה-batching, והפרופיילינג.
- **פרק 9 — מה לוקחים מכאן:** המסקנות שהדוברים ביקשו שניקח, ומה זה אומר בפועל.

הערת מקור המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך מונחים לועזיים מופיעים בו לעיתים בשיבוש פונטי (למשל "ניורון" = Neuron, "קודה" = CUDA, "באצ' סייז" = batch size) — בגוף המסמך הם נורמלו לכתיב האנגלי, ובציטוטים הושארו כפי שנאמרו. הציטוטים אומתו מול ההקלטה בחותמת הזמן המצוינת. כל מספר המופיע כאן הוא מספר שהוצג על מסך במהלך הסשן או נאמר במפורש; לא הושלמו מספרים ממקורות אחרים.

1. תקציר מנהלים

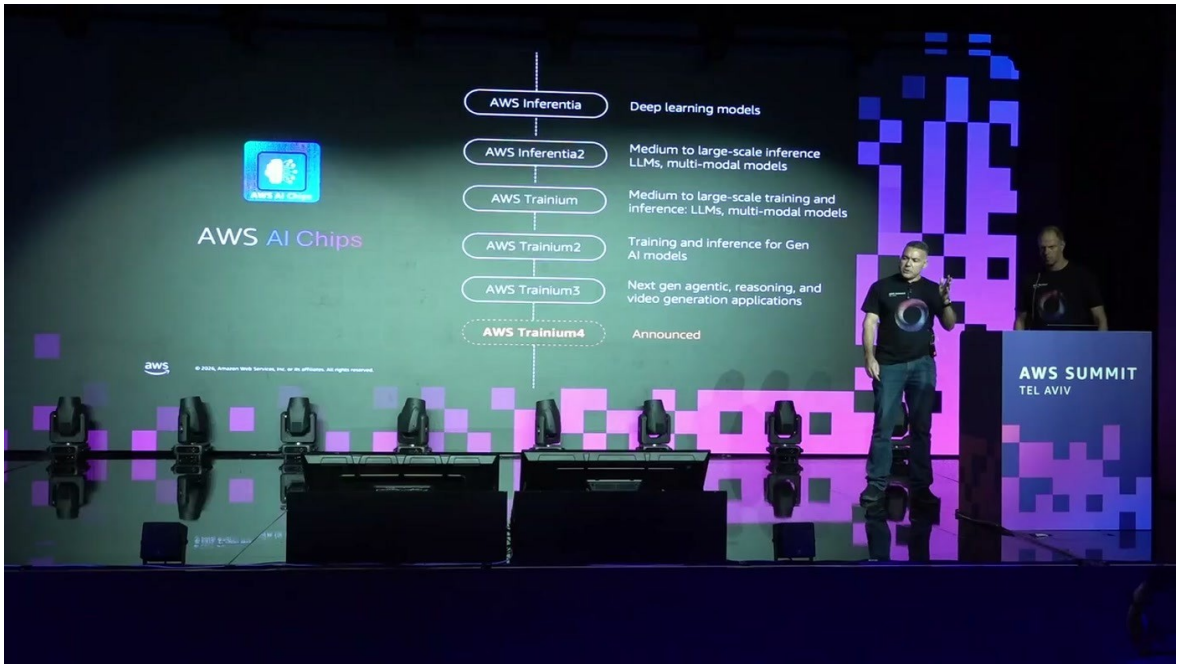
שני Solutions Architects מ-AWS — Guy Almog (מומחה Compute, עובד מול מרכזי פיתוח גלובליים בישראל) ו-Guy Ben Baruch (חטיבת Telco, וחלק מהצוות שעובד על צ'יפי Inferentia ו-Trainium) — לקחו טענה אחת והעמידו אותה במבחן חי על הבמה: inference של LLM הוא לא בהכרח עבודה ל-GPU. כל הסשן הוא דמו: אותו מודל, אותו vLLM, אותו EKS, שני נודים — אחד GPU ואחד Trainium2 — זה לצד זה על המסך.

שקופית הפתיחה: שני הדוברים, קוד הסשן DEM305, והשילוב שהסשן מוכיח — Trainium + vLLM + PyTorch Native.

- **המסר המרכזי: Inference ≠ GPU.** זו המילה הראשונה בשקופית ה-Key Takeaways שסגרה את הסשן, וגם המשפט שאיתו סיכם Guy Almog בעל פה: "יש עוד אלטרנטיבות לגי פיואים" [21:56].
- **חסם הכניסה נשבר בשכבת הקוד, לא בשכבת החומרה.** PyTorch Native מאפשר לקחת סקריפט PyTorch קיים ולהחליף שורה אחת — to('cuda'). ל-to('neuron').. עד לא מזמן, לדבריהם, "מה שפעם לפני שנה שנתיים היה עכשיו מסימה מאוד מאוד קשה לביצוע ובאמת תקע הרבה לקוחות" [23:28].
- **vLLM הוא אזרח מדרגה ראשונה ב-Neuron.** "VLLM fully supported בניורון ניורון היום זה first class citizen בתוך ה-libraries האלה" [3:22], וכך גם מאות מודלים מ-Hugging Face שרצים out of the box.
- **בדמו החי, ה-price-performance היה לטובת Trainium2.** הדשבורד חישב עלות ל-מיליון טוקנים בזמן אמת; המסקנה שנאמרה מהבמה: "הקוסט הוא משמעותית יותר נמוך" [11:01]. חשוב לסייג — זה מודל אחד (Qwen3-8B) ותצורה אחת.
- **הרווח האמיתי הגיע מכוונון ולא מהחומרה.** ברירת המחדל של vLLM ל-max_num_seqs היא 256; שינוי הפרמטר הזה ושינוי ה-concurrency העלו את ה-throughput משמעותית "בלי לפגוע ב-time to first token" [15:45]. בלי כוונון, שלושה מתוך ארבעה logical cores פשוט עמדו ב-idle.
- **Neuron Explorer הוא הכלי שמראה למה.** פרופיילר חדש שמאפשר capture בזמן אמת של ה-inference ומצביע ישירות על צוואר הבקבוק — תקשורת בין קורים, spill ל-HBM, או engine שלא מנוצל.

2. הרקע: משפחת ה-Accelerated Computing של AWS

Guy Almog פתח במיפוי מהיר של ההיצע, לא כדי למכור אלא כדי למסגר: Trainium אינו תחליף לכל דבר, הוא אופציה אחת מתוך תפריט. בצד ה-GPU הוא מנה את סדרת ה-P האחרונות — P6 עם Blackwell (B200, B300, GB200, GB300) ו-P7 שאמור להגיע, ואת סדרת ה-G לדגמים בינוניים, עם G7e החדש. בצד של AWS עצמה: Inferentia, Inferentia2, ומשפחת Trainium.



שרשרת הדורות של צ'יפי ה-AI של AWS עם ייעוד כל דור — מ-Inferentia ל-Trainium4, שמסומן Announced בלבד.

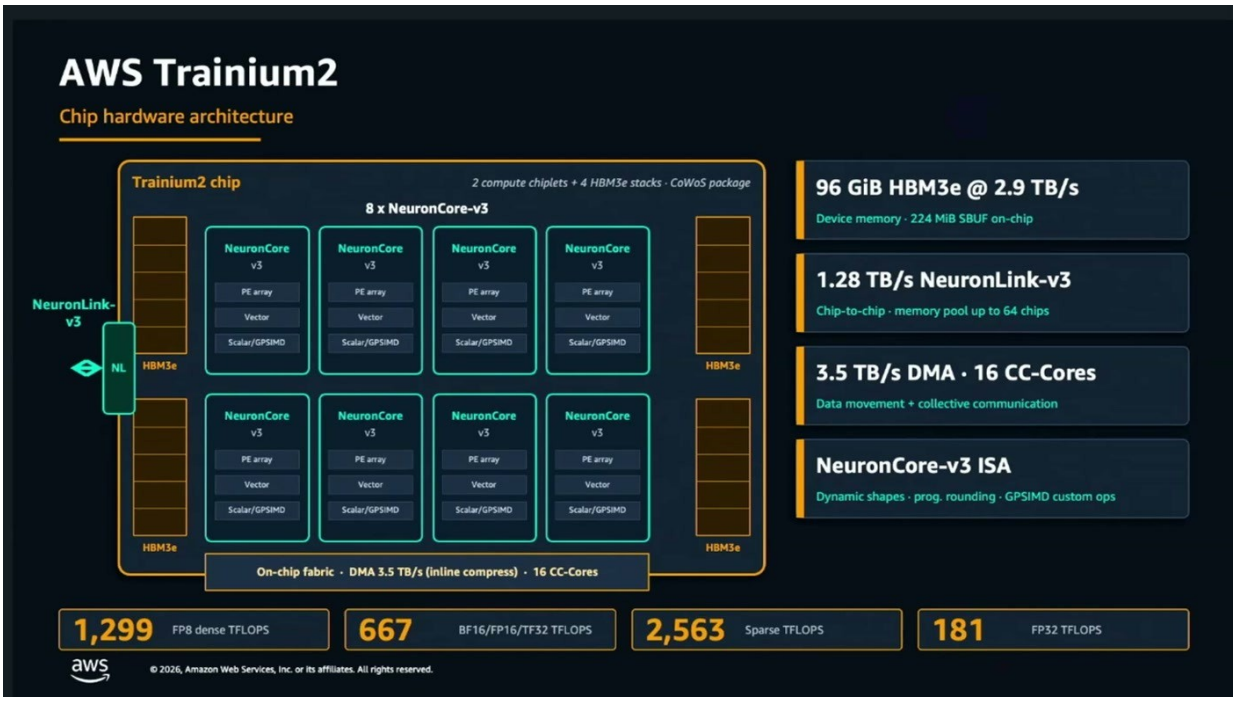
השקופית מיפתה ייעוד לכל דור: Inferentia ל-deep learning models, Inferentia2 ל-inference בינוני-גדול של LLMs ומודלים מולטי-מודליים, Trainium ל-training ו-inference בקנה מידה בינוני-גדול, Trainium2 ל-training ו-inference של מודלי GenAI, ו-Trainium3 ליישומים "next gen agentic, reasoning, and video generation". Trainium4 מופיע מקווקו, ומסומן Announced.

> **Guy Ben Baruch [1:48] —** יש לנו את טרניום 2 טרניום 3 ואפילו כבר יש את טרניום 4 שהוא הנאונס ובתהליכי פיתוח

הדמו כולו התרכז ב-Trainium2, שהוא הדור הזמין והנפוץ כיום.

3. מה יש בתוך Trainium2

החלק הזה חשוב לא בגלל המספרים אלא בגלל שהוא מסביר את כל הפרופיילינג שיבוא בהמשך. בלי להבין איפה יושב הזיכרון ומי מדבר עם מי, גרפי ה-Neuron Explorer הם רעש.



ארכיטקטורת הצ'יפ: שמונה NeuronCore-v3 בשני compute chiplets, ארבעה מחסני HBM3e סביבם, ורצועת ה-on-chip fabric שמחברת ביניהם.

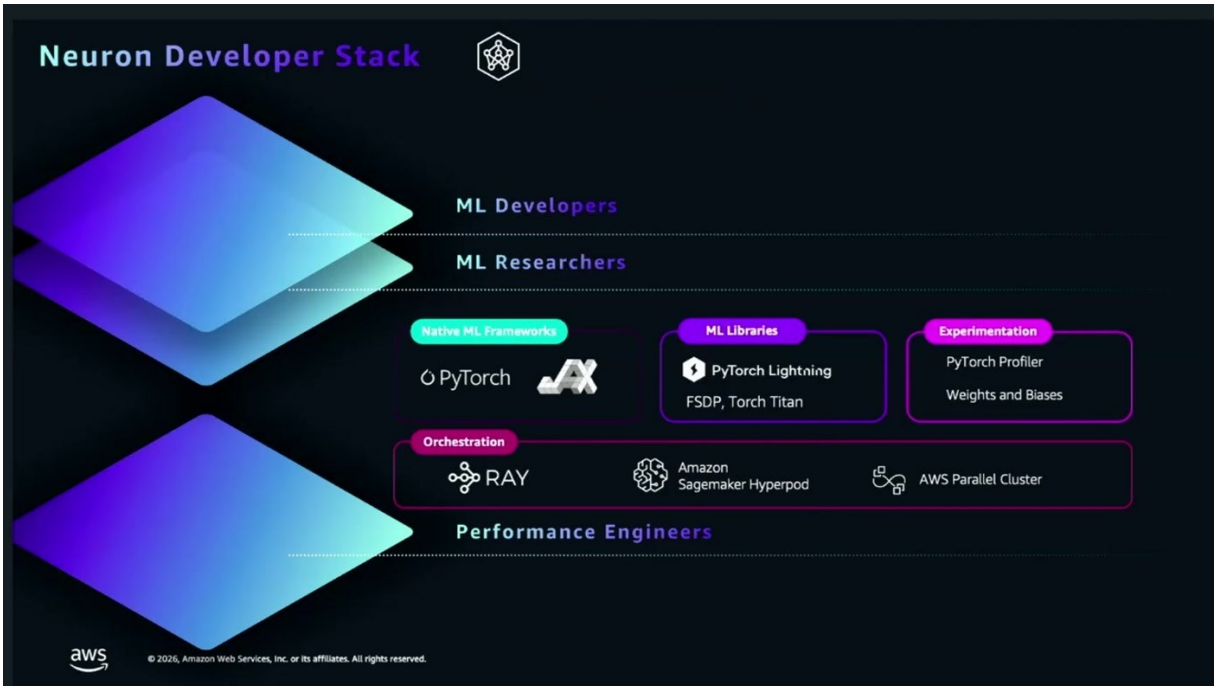
רכיב	מה שהוצג על השקופית
NeuronCore-v3	8 קורים בצ'יפ, כל אחד עם PE array, Vector ו-Scalar/GPSIMD
זיכרון מכשיר	96 GiB HBM3e בקצב 2.9 TB/s · 224 MiB SBUF על הצ'יפ
NeuronLink-v3	1.28 TB/s chip-to-chip · memory pool של עד 64 צ'יפים
תנועת נתונים	DMA בקצב 3.5 TB/s · 16 CC-Cores ל-collective communication
ISA	NeuronCore-v3: dynamic shapes, programmable rounding, GPSIMD custom ops
ביצועים (TFLOPS)	1,299 FP8 dense · 667 BF16/FP16/TF32 · 2,563 sparse · 181 FP32

Guy Ben Baruch הצביע על NeuronLink כמקבילה של NVLink בעולם NVIDIA: "יש חיבור בצד שמאל אפשר לראות של נוורון לינק, שמחבר צ'יפ-טו-צ'יפ קומיניקיישן כמו שיש להנביד את האנבילינק" [1:48].

המושג שחוזר לאורך כל הדמו Logical cores. אחת החדשות הארכיטקטוניות של Trainium2 היא היכולת לאחד שני קורים פיזיים ליחידה לוגית אחת. במקום שמונה קורים פיזיים, ה-runtime רואה ארבעה logical cores. הרווח הוא פחות תקשורת בין קורים — ובכל זאת ניצול של כל החומרה: "ברגע שאני מגדין את מספר הקורים, ומשתמש בלוג'יקל קורס, בעצם יש לי פחות קומיוניקיישן בין הקורים, אבל אני גם כמובן מנצל את כולם, כי אנדר דה הוד יש לי לוג'יקל קורס, שמנצל שני פיזיקל קורס" [15:45].

4. Neuron Developer Stack: הסופטוור שמעל החומרה

האנלוגיה שהדובר בחר היא הישירה ביותר: מה ש-CUDA הוא ל-GPU, Neuron הוא ל-Inferentia ול-Trainium. "בסוף אנחנו צריכים לכתוב לתוך החומרה דרך הסופטורסטק" [1:48]. הסטאק עצמו מוצג לא לפי שכבות טכניות אלא לפי שלוש פרסונות — ML Developers, ML Researchers ו-Performance Engineers — וזו חלוקה שימושית, כי היא אומרת לכל קורא איפה הוא נכנס.



שלוש הפרסונות ומה נתמך בכל שכבה — framework נייטיבי, ספריות ML, כלי ניסוי, ושכבת האורקסטרציה.

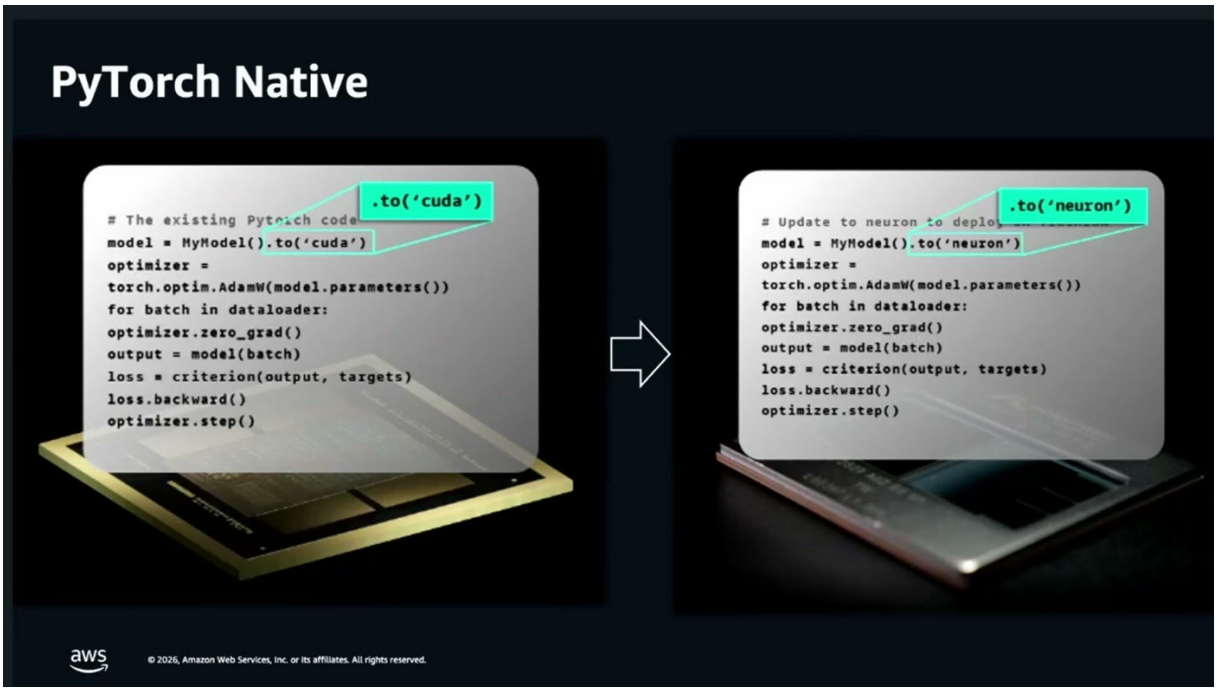
פרסונה	מה נתמך ב-Neuron
ML Developers	vLLM, Hugging Face (מאות מודלים out of the box), Ray, AWS Batch, Amazon ECS, Amazon EKS
ML Researchers	PyTorch ו-JAX כ-native frameworks; PyTorch Lightning, FSDP, Torch Titan; PyTorch Profiler ו-Weights and Biases לניסוי
Orchestration	Ray, Amazon SageMaker HyperPod, AWS Parallel Cluster
Performance Engineers	Neuron Kernel Interface (NKI), Neuron Monitor, Neuron Explorer

> **Guy Almog [3:22]** בניורון ניורון היום זה first class citizen בתוך ה-libraries האלה — VLLM fully supported ובנוסף גם Hagenface, אתם יכולים להיכנס ל-Hagenface ולראות יש מאות מודלים שאתם יכולים לקחת את ה-out of the box ולעשות להם deployment על גבי ניורון

שווה לשים לב ש-Ray, SageMaker HyperPod ו-AWS Parallel Cluster נזכרו גם בהקשר של training גדול, לא רק inference — כלומר הסטאק לא נבנה רק לתרחיש שהודגם כאן.

5. PyTorch Native: המחסום שהוסר

זהו החלק שהדוברים הציגו כשינוי המשמעותי ביותר מהשנה האחרונה, והוא גם התשובה להתנגדות הנפוצה ביותר שהם שומעים מלקוחות: "אוקיי מאוד מעניין אותנו Inferencing ו-Trainium אבל יש לנו מורכבות של לקחת את אותו פייפליין עם אותם סקריפטים ב-QDA ב-GPU לניון" [4:55]. כלומר — העניין קיים, אבל עלות ההגירה של הקוד חסמה אותו.



אותו לולאת אימון PyTorch פעמיים, זה לצד זה; ההבדל היחיד המסומן הוא הארגומנט שמועבר ל-to()..

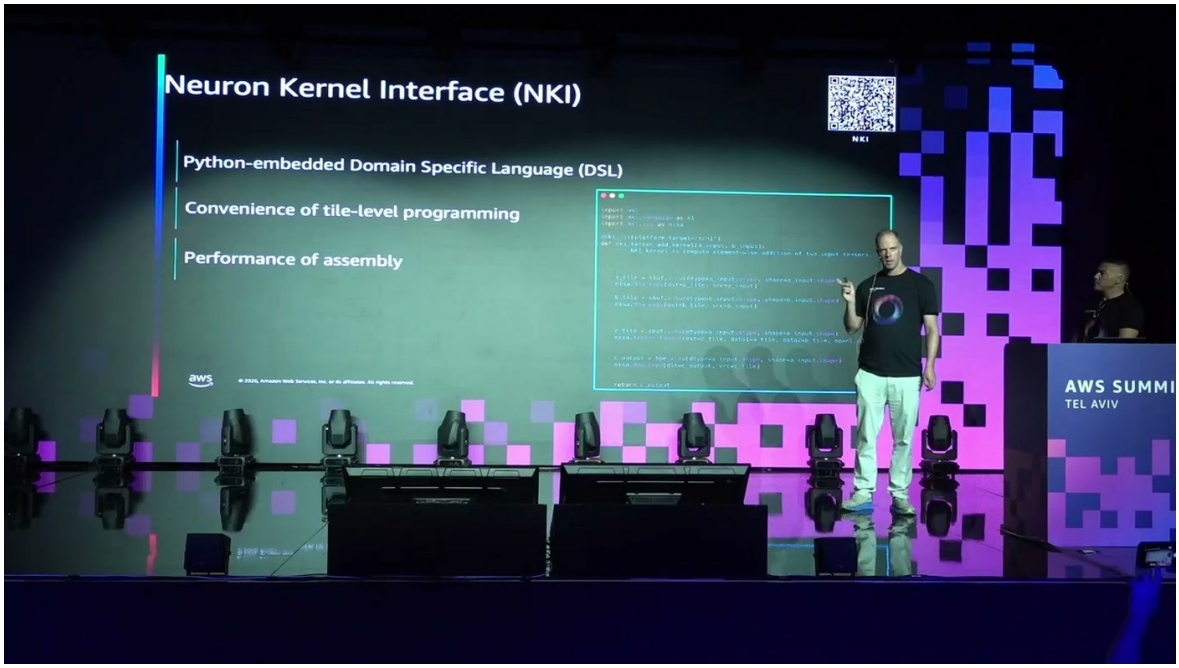
Guy Almog [4:55] — בקלות אפשר לעבור מסקריפט שיש לי היום בקודה GPU, וצריך הכל משנים מעט מאוד קוד, למשל פה אני משנה פה את סוג הדיבייס, מקודה לניורון, ואני יכול להריץ את אותו סקריפט על ניורון ב-Inferential ו-Trinium

התיאור של השקופית מדויק: שתי תמונות של אותו קוד — model, optimizer, לולאה על ה-dataloader, backward, step — כשההבדל היחיד המודגש בירוק הוא ('to('cuda. מול ('to('neuron.. השורה שהדוברים חזרו עליה בסיום: "הקשבנו ללקוחות והצוות שלנו פיתח את הדבר הזה וזה מפשט מאוד את העבודה ובמעבר" [23:28].

סייג הוגן השקופית מראה את השינוי המינימלי, לא את מלוא ההגירה. הדוברים עצמם הפנו את מי שעובד ברמה נמוכה יותר — עם kernels מותאמים — ל-NKI (פרק הבא), מה שמרמז שקוד עם kernels ייעודיים ב-CUDA עדיין דורש עבודת פורטינג אמיתית. בסשן לא הוצגה הרצה של ('to('neuron. בפועל כדמו חי.

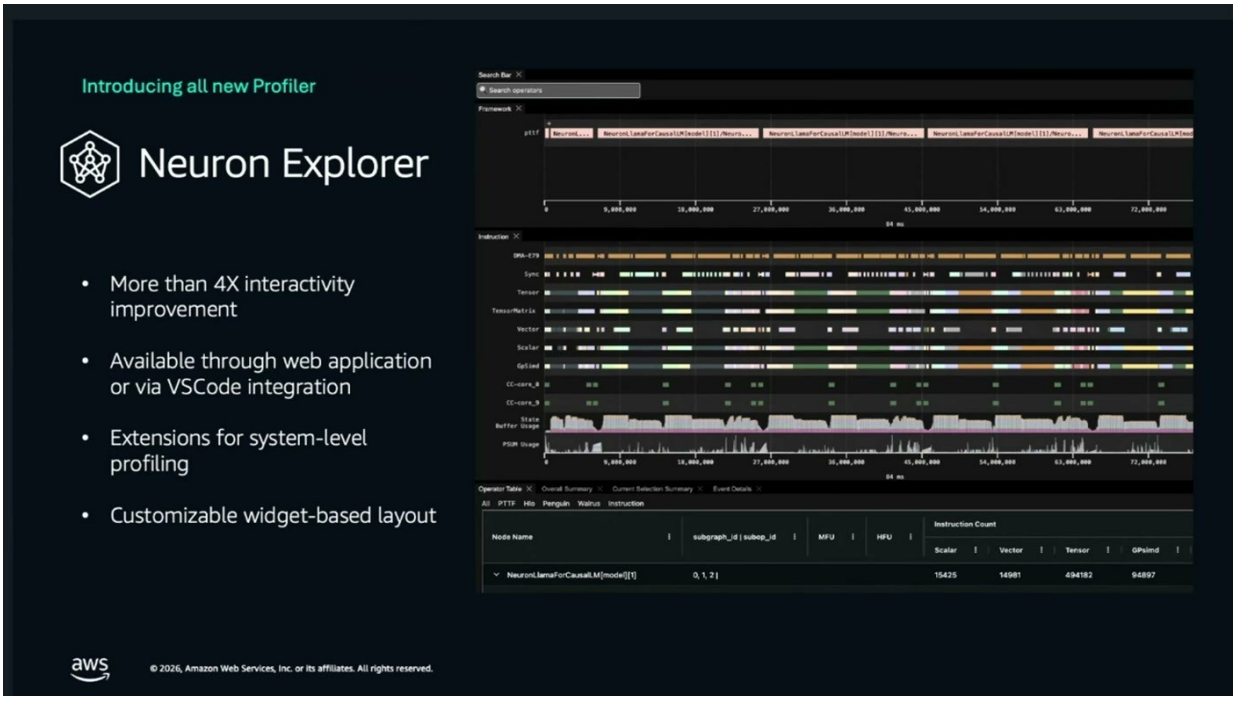
6. כלי הביצועים: NKI ו-Neuron Explorer

שני כלים הוצגו לפני הדמו, ושניהם חזרו בתוכו. הראשון הוא ה-escape hatch לרמה הנמוכה, והשני הוא העין שמסתכלת על מה שקורה בפועל.



שלוש התכונות שמגדירות את NKI, לצד דוגמת kernel שנכתב כולו ב-Python.

NKI — Neuron Kernel Interface — מוצג כ-DSL משובץ Python: "נוחות של תכנות ברמת tile" עם "ביצועים של assembly". Guy Almog מיקם אותו בדיוק בנקודה הכואבת: "אם נמחתם איזשהו קוד בקודה עם איזשהו קרנל, יש איזשהו משהו שאתם רוצים לעשות לו פורט לתוך ניורון, יש משהו שלא נתמח, אם אתם רוצים לעשות לו אופטימיזציה" [4:55]. הוא ציין גם ספריית kernels עם contributions מהקהילה, ו-agentic development skills שעוזרים להתחיל.



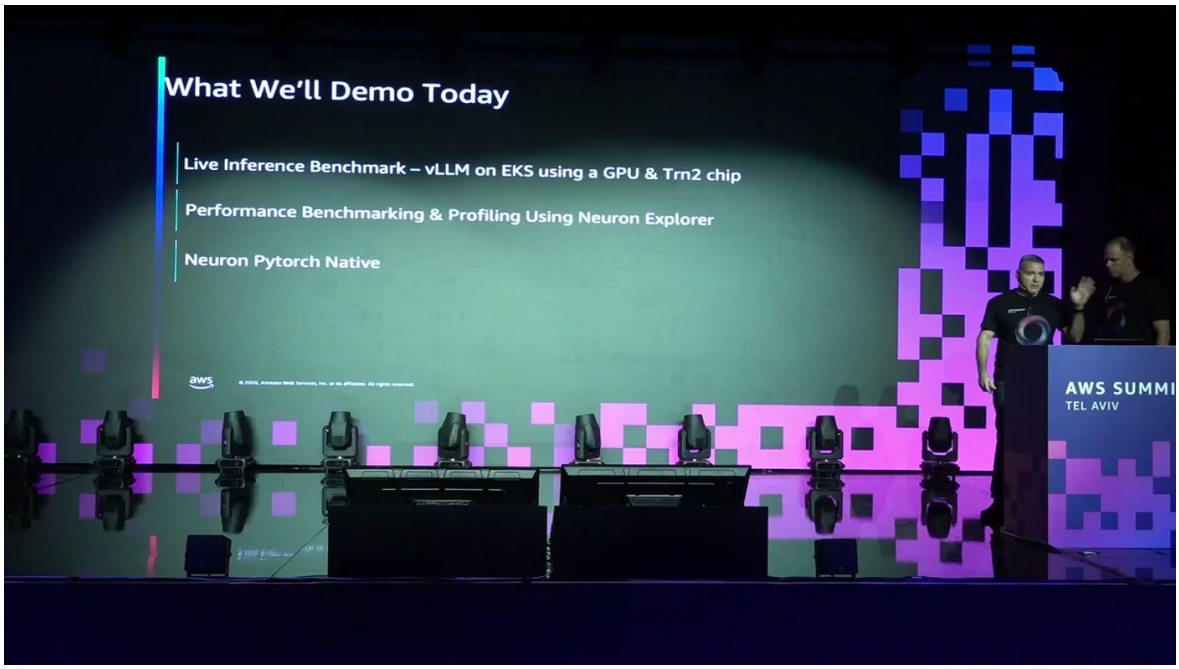
הפרופיילר החדש: ארבעת היתרונות שהוצהרו, מול צילום מסך של ה-timeline עם פירוק לפי engine.

- שיפור של יותר מפי 4 באינטראקטיביות

- זמין כ-web application או דרך אינטגרציה ל-VSCode
- הרחבות ל-profiling ברמת המערכת
- פריסה מבוססת widgets הניתנת להתאמה

> **Guy Almog על Neuron Explorer [15:45]** —
> יכולת לעשות קפצ'ר ב-real-time לאינפרנסינג, וממש לעשות דיפ טייב לתוך הפרמטרים ולראות איך החומרה מתנהגת

7. הדמו, חלק א׳: השוואה חיה GPU מול Trainium2



שלושת החלקים שהובטחו מהבמה, לפני המעבר לדשבורד החי.

הסטאפ תואר במדויק [7:58]: EKS cluster עם שני נודים. בצד אחד nodeTRN2 מסוג trn2.3xlarge — שמונה קורים פיזיים שהוגדרו כארבעה logical cores, עם pod של vLLM שרץ ב-4 tensor parallel, כלומר כל הקורים בשימוש. בצד השני node של GPU מסוג g6e עם צ'יפ NVIDIA L40S, אותו setup בדיוק, אותו vLLM — רק עם CUDA. המודל בשני הצדדים: Qwen3-8B.

> **Guy Almog [7:58] —** הנוד הראשון מצד שמאל זה TRN2, 3 extra large, יש לו 8 physical course כמו שדיברתי קודם, 4 logical course, שלמעשה אנחנו הגדרנו אותם כיחידות לוגיות שמאחדות 2 physical course

לפני ההרצה הם הראו את הקוד עצמו משני הצדדים — הרצת ה-vLLM, מניפסטי ה-EKS ושורת ההרצה — והסיכום שלהם היה לקוני: "ככה נראה הקוד של האטרניום. קצת שינויים, אבל בגדול אותו רעיון" [9:29]. זו בדיוק הטענה של PyTorch Native, רק שהפעם היא על המסך.

פלט חי של אותו prompt, עם שורת העלות המחושבת בראש המסך.

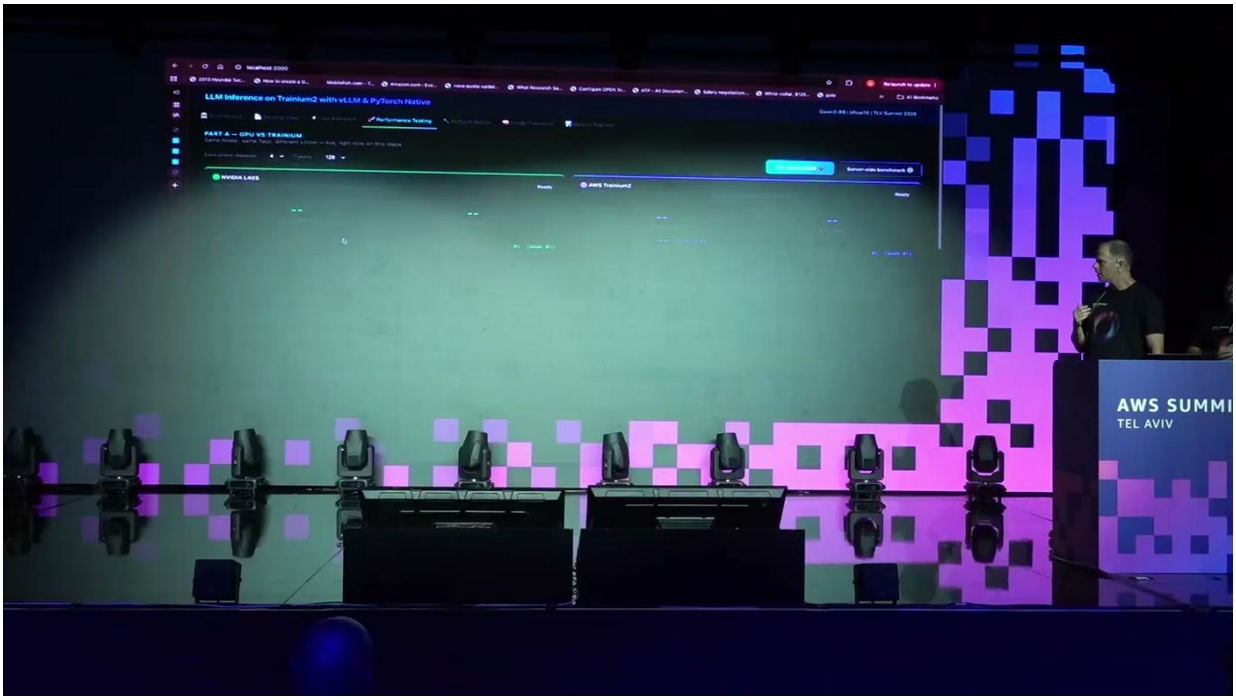
הם שלחו prompt זהה לשני הצדדים והשוו את המדדים שהדשבורד חישב בזמן אמת: time to first token, tokens per second, מספר טוקני קלט ופלט (1254 טוקנים ביציאה בהרצה שהוצגה), ועלות מחושבת.

המספר שאסור לקרוא כפשוטו ה-time to first token שהוצג בצד Trainium היה 1154 — מספר שנראה גבוה, והדוברים עצרו כדי להסביר למה הוא חסר משמעות כאן: "זה בגלל שהגי פיו רץ בארצות הברית, והטריניום רץ בכלל באוסטרליה. פשוט שם אנחנו רצים, אז יש לנו פה שעוני ב-API שיוצאים" [9:29]. במילים אחרות — ה-latency בדמו נגוע בהפרש גיאוגרפי ואינו מדד השוואה תקף. המדדים שכן היו בריי-השוואה הם tokens per second והעלות.

Guy Almog — [11:01] כמות הטוקנים פר סקרד והפרמיטים היותר חשובים והקוסט הוא משמעותית יותר נמוך. וזה אחד הדברים שאנחנו רוצים שתתחילו להכיר שהצ'יפים של AWS בסופו של דבר AWS עושה אותם כדי שלקוחות ישלמו פחות ויקבלו יותר

8. הדמו, חלק ב׳: load testing וכוונון ה-batching

כאן הסשן הפך ממכירה להנדסה. במקום prompt בודד, הם הריצו stress test עם רמות concurrency משתנות — והראו שהשאלה "איזה צ'יפ מהיר יותר" פחות מעניינת מהשאלה "האם החומרה בכלל מנוצלת".



מסך ה-load testing: NVIDIA L40S מול AWS Trainium2, עם בוררי concurrency וכמות טוקנים מעל שני הצדדים.

הפרמטר במרכז הוא מה ש-vLLM קורא לו max_num_seqs — ומה שמוכר לרובנו כ-batch size. הוא נקבע בזמן ה-compilation ל-Neuron, ולפי הדוברים ברירת המחדל שלו היא 256.

> **Guy Almog [14:08] —** ב-VLLM יש איזשהו דיפול בבאצ' סייז, איזשהו פרמטר דיפולטיבי, שהוא מגדיר את זה על 256

ההדגמה הקריטית הייתה של הקצה הנגדי. עם concurrency של 1 ו-batch size של 1, ה-request היחיד נכנס לקור אחד — ושלושה מארבעת ה-logical cores עומדים ללא עבודה:

> **Guy Almog [14:08] —** אם אנחנו עושים קונקרנסי של אחד, batch size שווה אחד, אז אם כמו שגע אמר מקוד, אם יש לנו עכשיו ארבעה logical cores, אז בעצם יש לנו שלושה cores שהם בעצם idle

העלאת ה-batch size ו-ה-concurrency העלתה את ה-utilization של הקורים ואת ה-throughput; שינוי max_num_seqs מעבר לכך העלה את הטוקנים לשנייה "בצורה משמעותית", ולפי מה שנאמר — "בלי לפגוע ב-time to first token" [15:45]. אבל הם מיד סייגו את זה, וזה הסייג הנכון:

> **Guy Almog [15:45] —** תמיד יש לי איזשהו trade-off בין כמה processing, זאת אומרת כמה requests אני רוצה לאבד ביג זמן, אבל תמיד יש לי trade-off עם ה-latency, אז הרעיינו כמובן לראות מה ה-use case שלי

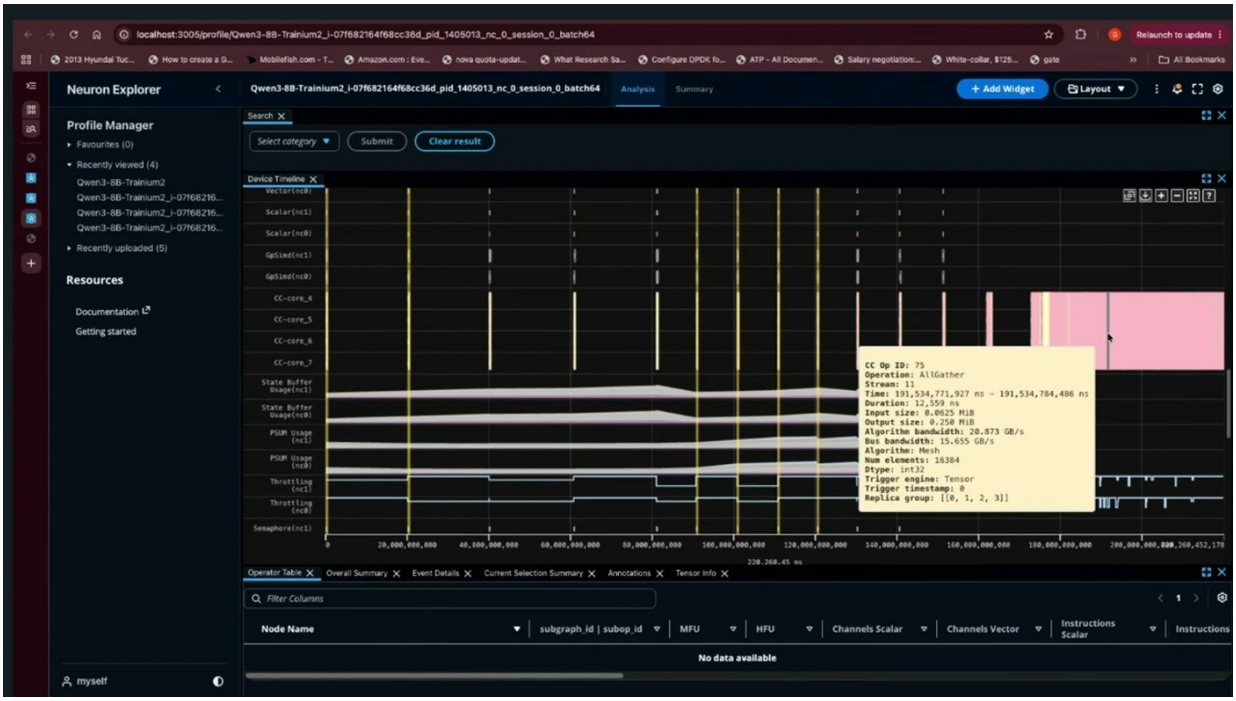
> **המסקנה המעשית של הפרק** לפני שמשווים חומרה, יש לוודא שההשוואה נעשית בנקודת עבודה שבה שני הצדדים מכוונים. ברירת מחדל של vLLM אינה נקודת עבודה מכוונת, ו-benchmark שרץ ב-concurrency 1 מודד בעיקר latency של קור בודד — לא את יכולת המערכת.

9. הדמו, חלק ג׳: פרופיילינג עם Neuron Explorer

כדי להסביר את מה שרואים בפרופיילר, הדוברים חזרו לרגע לזיכרון. ה-HBM יושב off-core — מחוצה לקורים — וכל הקורים כותבים וקוראים אליו; ה-SBUF וה-PSUM יושבים על הצ'יפ עצמו והגישה אליהם מהירה בהרבה. הפער הזה, בתוספת התקשורת בין הקורים, הוא צוואר הבקבוק שמחפשים.

> **Guy Almog [17:21] —** יש לי גם memory שיושב על הצ'יפ עצמו, מה שנקרא PISAM או ה-ASBUFF, שם הכתיבה הרבה יותר מירה, תמיד השאיפה היא לכתוב על הצ'יפ ולא לצאת החוצה

הם עשו profiling לשני תרחישים על אותו מודל Qwen3-8B: batch size 1 מול 64 requests.

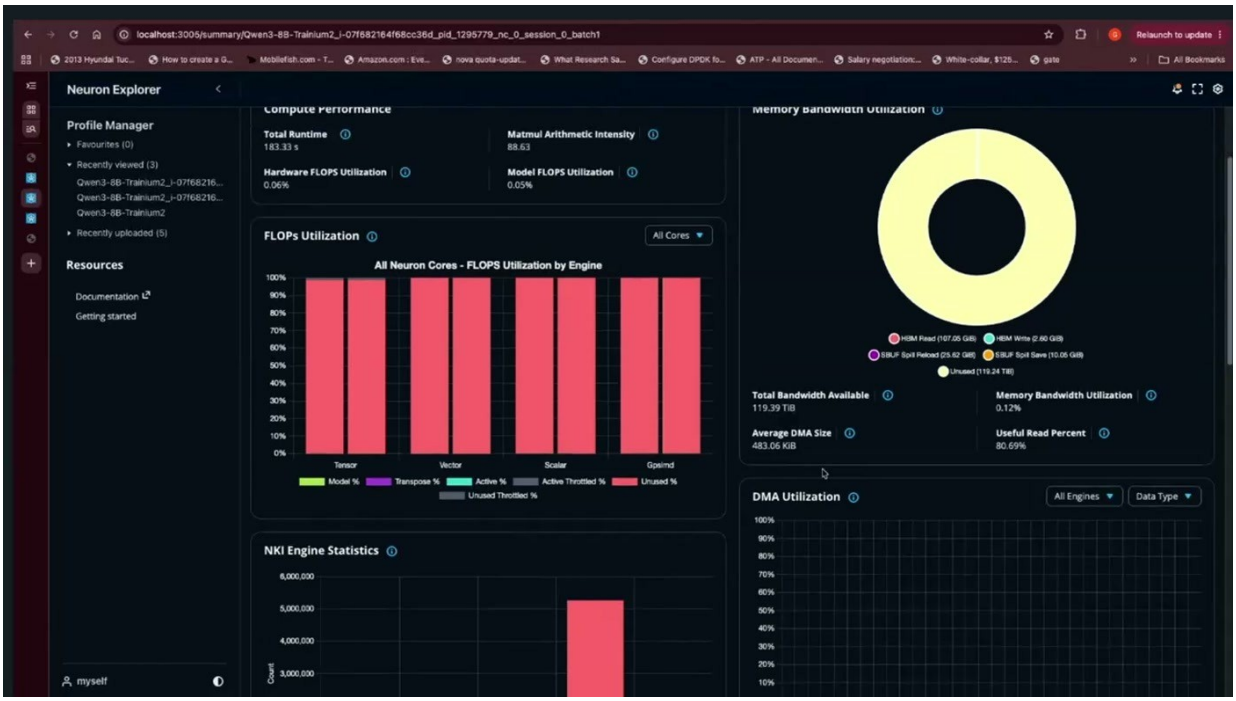


ה-Device Timeline בהרצת batch 64: הרצועות הוורודות בערוצי CC-core הן תקשורת קולקטיבית, וה-tooltip חושף פעולת AllGather בודדת עם רוחב הפס והאלגוריתם שלה.

בהרצת ה-64 בולטת פעילות כבדה בערוצי ה-CC-cores — התקשורת בין הקורים. ה-tooltip שנפתח על הבמה הראה פעולת AllGather אחת בפירוט מלא: משך, גודל קלט ופלט, algorithm bandwidth ו-bus bandwidth, אלגוריתם Mesh, ו-replica group. זה בדיוק סוג ה-drill down שהם רצו להראות.

> **Guy Almog [18:52] —** ברגע שאני מתחיל להגדיל את כמות ה-Concurrency ומתחיל להכניס הרבה מאוד request, אפשר להתחיל לראות שיש פה Bottleneck באזור ה-Communication עצמו בתוך הקורם

בהשוואה, בהרצת batch size 1 התמונה נקייה: "אפשר לראות שה קומיוניקיישן בין הקורים נראה הרבה יותר טוב זאת אומרת הרבה פחות עמוס גם מבחינת יש פה פרוטלינג אפשר לראות שהוא במצב יותר טוב והרבה פחות סייקלים שקורים בין הקורים עצמם" [20:24]. אבל "נקי" כאן אינו מחמאה — הוא סימפטום.



מסך ה-Summary של הרצת batch 1: עמודות FLOPs אדומות כמעט לחלוטין (Unused), לצד טבלת ה-Compute Performance ותרשים רוחב הפס.

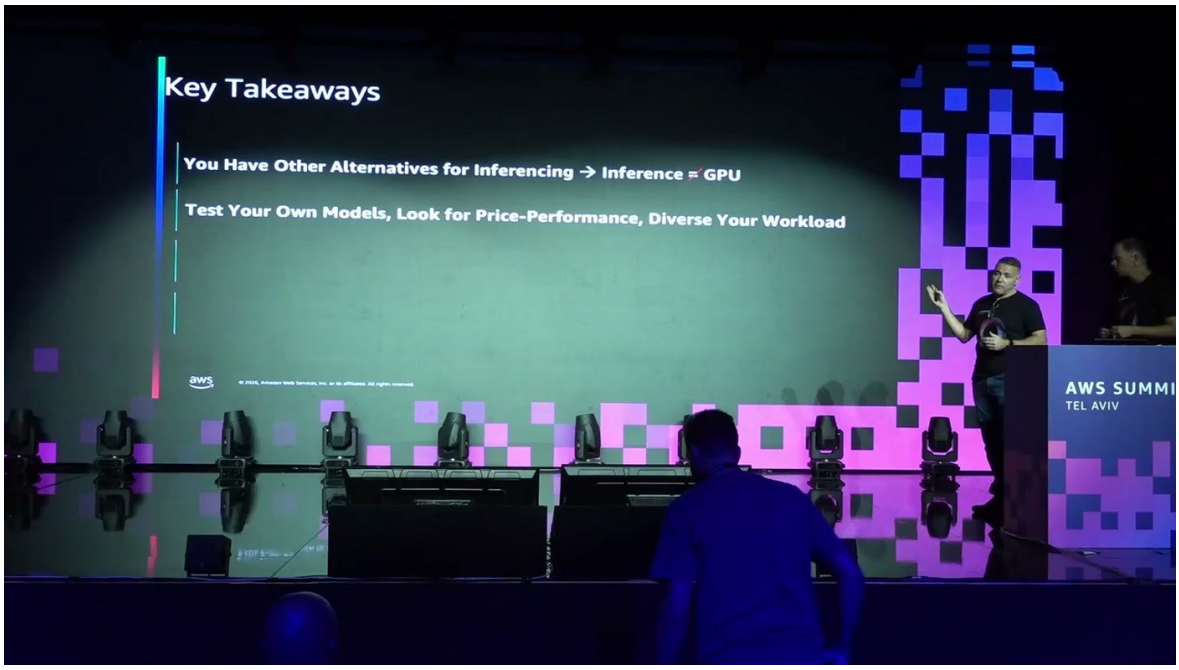
ה-Summary Stats של הרצת ה-batch 1, כפי שהופיע על המסך, מספר את הסיפור בשלושה מספרים: Total Runtime של 183.33 שניות, Hardware FLOPS Utilization של 0.06% ו-Model FLOPS Utilization של 0.05%, ו-Memory Bandwidth Utilization של 0.12%. גרף ה-FLOPs Utilization מציג את כל ארבעת ה-engines — Tensor, Vector, Scalar ו-GPsimd — כמעט כולו באדום, כלומר Unused. ה-tooltip שנפתח על אחד מהם הראה במפורש: Unused 99.9%.

איך לקרוא את המספרים האלה 0.06% ניצול FLOPS אינו ליקוי של Trainium — זו בדיוק הנקודה שהדוברים ביקשו להמחיש: הרצה ב-batch size 1 מותירה את החומרה ריקה כמעט לגמרי, וזו הסיבה שכוונון ה-batching מפרק 8 הוא שמייצר את הרווח. אותו סוג מדידה על GPU לא מכוונן היה מראה תמונה דומה באופייה.

הפרופיילר הוסיף שכבה שקשה להתעלם ממנה: summary מבוסס LLM שרץ ברקע ונותן חיווי מילולי על ה-utilization ועל הנקודות החשודות. "הוא גם יכול לתת כל מיני חיבויים לגבי, אוקיי, אני רואה שיש פה איזשהו בוטלנק ברמת הקומיוניקיישן בתוך הקורים" [20:24]. וכל זה ניתן לייצוא:

Guy Almog [21:56] — כל האינפורמציה שאתה רואים בתוך ה-Nuron Explorer, אפשר לעשות לזה export ל-JSON, להשתמש בזה בקלוט קורט קירו וכו' גם לצר תובנות

10. מה לוקחים מכאן



שקופית הסיום, עם שתי המסקנות שהדוברים ביקשו במפורש שניקח.

- **Inference ≠ GPU — אבל גם ≠ Trainium.** הטענה שהוצגה היא שקיימות אלטרנטיבות, לא ש-GPU מיותר: "זה לא אומר שתמיד זה שווה שווה גי פיואים... תכירו שיש עוד אלטרנטיבות והן לפעמים אפילו יכול להיות יותר מוצלחות ויותר טובות לכם" [21:56].
- **בדקו את המודל שלכם, לא את זה שבדמו.** הדוברים סייגו בעצמם: Qwen3-8B נבחר כי הוא קטן ומתאים לדמו. "כמובן צריך לקחת את המודל שלכם... ולדודוק אותו לראות שהכל עובד כמובן, פונקציונלית וגם כמובן להסתכל על הפרייס פרפורמנס" [21:56].
- **המדד הוא price-performance, לא throughput.** ההמלצה הייתה למדוד עלות למיליון טוקנים — "אנחנו נראים על מחיר נגיד כמה עולה לי לעשות מיליון טוקנים או [23:28] "whatever — ולא רק מהירות.
- **Profiling אינו אופציונלי.** "אנחנו נתקלים בהרבה פעמים בלקוחות שלוקחים איזשהו GPU כי נראה להם נריצים ומה שיוצא אני מרוצה ופשוט סתם מבזבזים כסף" [21:56]. בלי פרופיילר אין דרך לדעת אם ההשוואה שעשיתם הוגנת.
- **Diversification של ה-workload הוא נכס תפעולי.** היכולת להריץ גם על GPU וגם על צ'יפים אחרים היא גמישות זמינות ומחיר: "לפעמים יש זמיניות של זה יותר, זמיניות של זה יותר, ספוטים העולם נפתח לי כי אני יודע לרוץ על עוד אופציות" [23:28].

מה הייתי בודק אחרי הסשן הזה

- לקחת workload inference קיים, להריץ אותו על vLLM ב-EKS בשני הצדדים — באותו region, כדי לא לחזור על עיוות ה-latency של הדמו.
- לכוון את max_num_seqs ואת ה-concurrency בשני הצדדים לפני כל מדידה, ולדווח את נקודת העבודה יחד עם התוצאה.
- להריץ Neuron Explorer על התרחיש האמיתי ולבדוק שני דברים: ניצול ה-engines, ונפח ה-collective communication בין הקורים.
- לבדוק אם הקוד הקיים מכיל kernels ב-CUDA — זה הגורם היחיד שהוצג כדורש עבודת פורטינג של ממש (דרך NKI).

11. מילון מונחים

מונח	מה זה, כפי שהוצג בסשן
Neuron	ה-software stack של AWS לצ'יפי Inferentia ו-Trainium; המקבילה התפקודית ל-CUDA
NeuronCore	יחידת החישוב בתוך הצ'יפ; ב-Trainium2 יש שמונה NeuronCore-v3 לצ'יפ
Logical core	איחוד של שני קורים פיזיים ליחידה לוגית אחת — פחות תקשורת בין קורים, ניצול מלא של החומרה
NeuronLink	חיבור chip-to-chip בין צ'יפים; המקבילה ל-NVLink של NVIDIA
HBM	High Bandwidth Memory — זיכרון off-core; הגישה אליו יקרה ומהווה צוואר בקבוק נפוץ
SBUF / PSUM	זיכרון on-chip; הכתיבה אליו מהירה בהרבה מ-HBM. השאיפה היא להישאר בו
DMA	Direct Memory Access — כתיבה וקריאה ישירות מול הזיכרון; נמדד ב-Neuron Explorer
CC-cores	Collective Communication cores — הערוצים שמטפלים בתקשורת בין הקורים (למשל AllGather)
NKI	Neuron Kernel Interface — DSL משובץ Python לכתיבת kernels ברמה נמוכה
Neuron Explorer	הפרופיילר החדש; capture בזמן אמת, timeline לפי engine, וייצוא ל-JSON
PyTorch Native	היכולת להריץ סקריפט PyTorch קיים על Neuron בשינוי ה-device בלבד
max_num_seqs	פרמטר ה-batch size של vLLM; נקבע ב-compilation, ברירת מחדל 256
TTFT	Time To First Token — זמן עד הטוקן הראשון; מדד latency מרכזי ב-serving