אג'נטים בפרודקשן: attention, קונטקסט ועלויות

AWS Summit Tel Aviv 2026 ,סיכום סשן — TLV-AIM301

שקד דולצמן, Senior Technical Account Manager, AWS | דורון בלייברג, Generative AI Solutions Architect, AWS | ירדן רון, Skai Engineering Manager

10 בספטמבר 2026 | משך: כ-40 דקות | הופק מהקלטת הסשן

מסלול: Building AI and Agentic Applications | רמה: 300

על המסמך הזה

המסמך מסכם את הסשן "TLV-AIM301, "Best Practices for Agentic Applications, מתוך AWS Summit Tel Aviv 2026. הוא נבנה מתוך ההקלטה המלאה של הסשן ומהסליידים שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב בארבעים הדקות. חותמות הזמן בסוגריים מרובעים מפנות לנקודה המדויקת בהקלטה.

הסשן מחולק לשלושה חלקים: שקד דולצמן על הנדסת קונטקסט ועל attention, דורון בלייברג על טכניקות הזרקה, דחיסה והוזלת עלויות, וירדן רון מ-Skai על מה קרה כשהעקרונות האלה הוטמעו במוצר שרץ שנה וחצי בפרודקשן.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה הוצג ומה שווה לאמץ.
- **פרקים 2–6 — הנדסת קונטקסט:** למה אג'נט מתדרדר לאורך זמן, מה עושים עם attention, גארדרייז ומדידה.
- **פרקים 7–12 — הטכניקות:** hooks, task list גלובלי, skills ודחיסה, workflow agents, prompt caching וצמצום קריאות לכלים.
- **פרק 13 — המקרה של Skai:** Celeste AI והסטודיו — מה עבד בפרודקשן ומה נדרש כדי להגיע לאוטונומיה.
- **פרק 14 — מה לוקחים מכאן:** ארבע קבוצות ה-best practices כפי שהדוברים סיכמו אותן, ונקודות להמשך.

הערת מקור המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך מונחים לועזיים הופיעו בו בשיבוש פונטי ("אג'נט קור" = AgentCore, "סטרנד" = Strands, "קש" = cache) ותוקנו מול הסליידים. הציטוטים אומתו מול ההקלטה בחותמת הזמן המצוינת ומובאים כלשונם, כולל שיבושי תמלול קלים. מספרים שמופיעים על הסליידים בלבד — צוינו ככאלה.

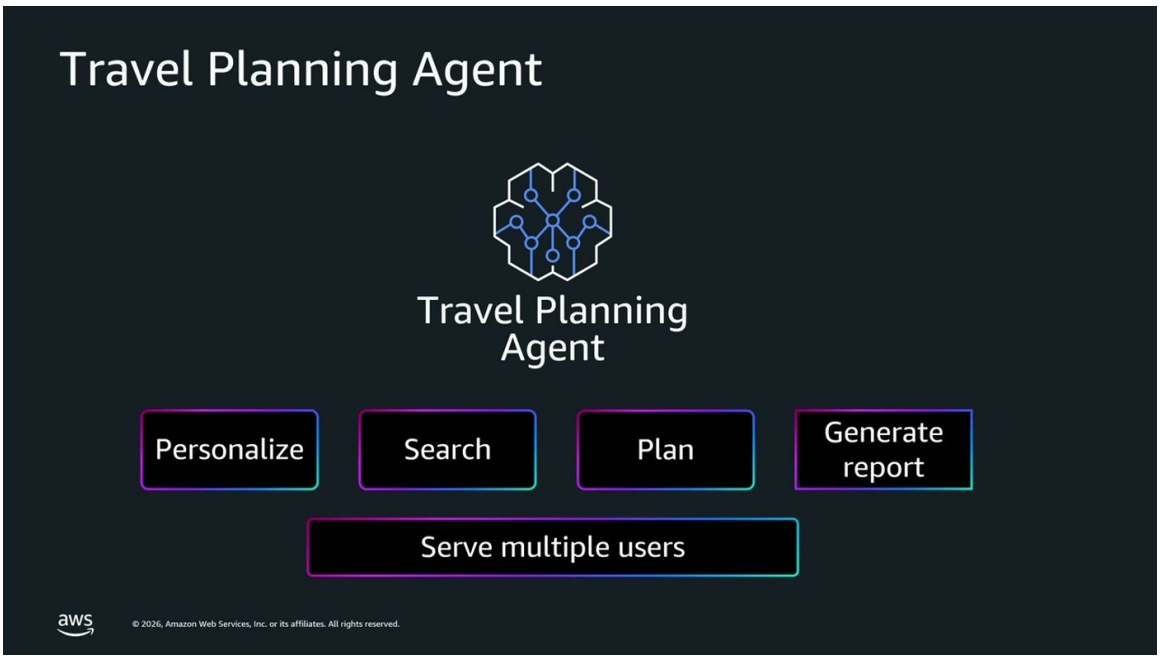
1. תקציר מנהלים

הסשן לא עסק בשאלה איך בונים אג'נט, אלא בשאלה למה אג'נט שעובד יפה בדמו מפסיק לעבוד בפרודקשן — והתשובה שחזרה בכל שלושת החלקים היא אחת: attention. כל טוקן שנכנס לקונטקסט גוזל תשומת לב ממה שכבר נמצא שם, ולכן רוב ה-best practices שהוצגו הן טכניקות להוצאת מידע מהקונטקסט, לא להכנסתו.

- **הבעיה היא לא גודל חלון הקונטקסט אלא פיזור ה-attention.** "הרבה לפני שנגיע לתקרה, הביצועים מתחילים להידרדר" [6:07]. גם עם חלון של מיליוני טוקנים, כל הוראה נוספת מוסיפה מידע וגוזלת תשומת לב ממה שכבר קיים.
- **איסורים בפרומפט עובדים הפוך.** "אל תציג טיסות עם קונקשן" מכניס את המושג connection לקונטקסט ומושך אליו attention. ההמלצה: להחליף איסורים בהוראות חיוביות, ולהגדיר מה שייך ולא מה לא שייך [7:39].
- **גארדרייז בסיסטם פרומפט הם בזבוז מובנה.** הם סטטיים, לא משתנים בין ריצה לריצה, ובכל זאת נשלחים למודל בכל איטרציה. ההמלצה היא להוציא אותם לשירות חיצוני כמו Amazon Bedrock Guardrails שנבדק פעם אחת לבקשה [9:12].
- **Hooks ו-progressive disclosure עדיפים על "לתת הכל מראש ולקוות".** הזרקת קונטקסט בנקודת הזמן שבה האג'נט צריך אותו — אחרי קריאה לכלי, למשל — "הראו שיפור מאוד מאוד משמעותי ביכולת של האג'נטים לעקוב אחר משימות" [13:55].
- **לא כל טסק חייב להיות אוטונומי.** כשגרף הריצה ידוע מראש, workflow agents מעל Step Functions או Temporal הופכים את הראוטינג והברנצ'ינג לדטרמיניסטיים, מקטינים קריאות ל-LLM ומשפרים עלויות וזמנים [18:28].
- **Prompt caching הוא מנוף העלות הגדול ביותר.** טוקנים מהמטמון זולים ב-כ-90%, ועיצוב הקונטקסט כ-append-only עם פרפיקס יציב מגיע ל-80–90% cache hit ויותר [21:36, 23:07].
- **Skai הוכיחה את זה בפרודקשן.** Celeste AI רצה שנה וחצי מעל Strands, משרתת יותר מ-8,000 מותגים וכ-10 מיליארד דולר תקציבי פרסום בשנה; ההמשך — מוצר נפרד, "הסטודיו", מעל Bedrock AgentCore, עם workflows ונקודות עצירה דטרמיניסטיות [26:15, 33:54].

2. הבעיה: האג'נט שעובד ב-80% מהמקרים

שקד דולצמן פתחה בארבע שאלות לקהל — האם האג'נט עדיין נותן תוצאות אמינות לאורך זמן, האם הקונטקסט מנוהל ביעילות, והאם יודעים לזהות מתי multi-agent עדיף על מונוליט [0:00]. משם עברה לדוגמה שליוותה את כל החלק הראשון: אג'נט לתכנון חופשה משפחתית, שמחפש טיסות, מלונות ופעילויות לפי סגנון, הרכב משפחתי ותקציב.



האג'נט לדוגמה שליווה את הסשן: ארבע יכולות — Personalize, Search, Plan, Generate report — ודרישה לשרת מספר משתמשים במקביל.

לפני הבנייה הוגדרו ארבעה KPIs: דיוק ואמינות (תוצאות נכונות, שלמות ומבוססות בכל ריצה, כולל מקרי קצה), מהירות (כמה זמן לוקח לסיים משימה, כולל לולאות החשיבה והקריאות לכלים), יעילות עלות, ותחזוקתיות — "בהמשך נרצה להוסיף לו יכולת, לשנות לו הוראה. אנחנו רוצים לעשות את זה בלי לשבור את מה שכבר עובד" [3:04].

האינסטינקט הראשון הוא ללמד את האג'נט כל מה שהוא צריך לדעת: הגדרות כלים, מדיניות ביטולים של כל חברות התעופה, skills, העדפות מושב — כי ככל שנלמד אותו יותר, כך הוא יהיה חכם יותר.

הכל נכנס לקונטקסט: Domain Knowledge, System prompt, Tools, Instructions, RAG ו-Skills — האינסטינקט שהסתיים בא לתקוף.

> **— שקד דולצמן [3:04]** בניתם אג'נט והרצתם אותו עשר פעמים. בשמונה מתוך עשר הפעמים האלה הוא מחזיר לכם תוצאות ממש טובות. אבל מה קורה עם ה-20% שנשארו?

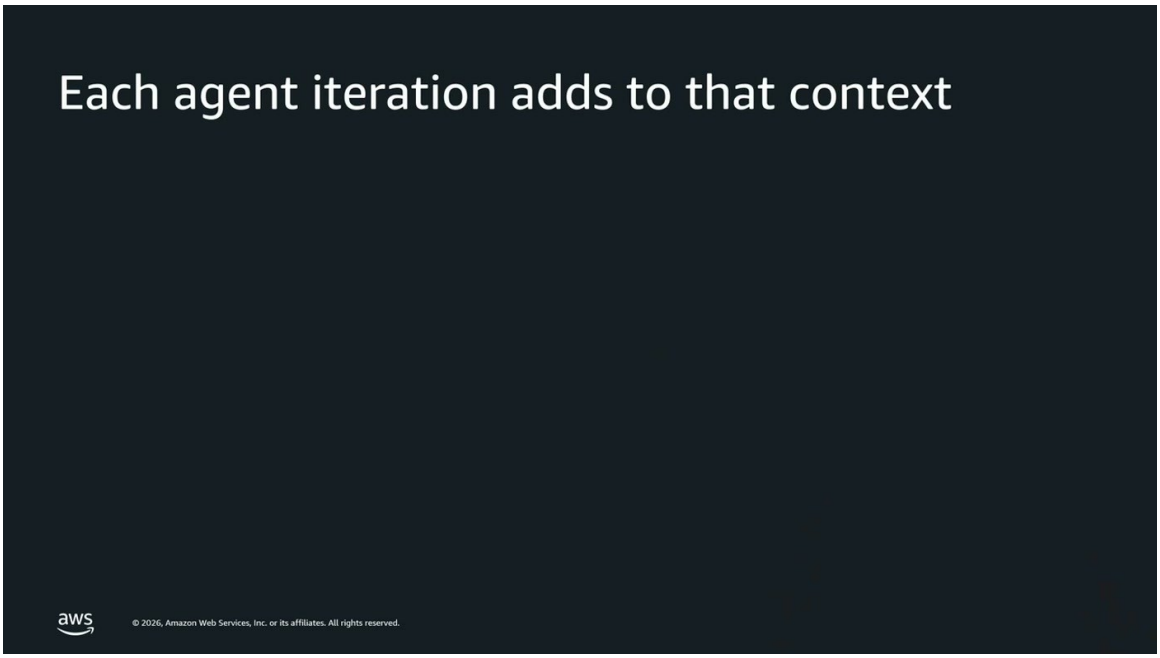
הדוגמה הקונקרטית: פרומפט של "תכנן לי טיול לברצלונה באוגוסט, תקציב של 3,000 דולר, עם ילדים, טיסה ישירה בלבד". האג'נט בוחר קודם כל בכלי חיפוש מלון — בחירה שגויה, כי עם כל המידע שבקונטקסט הוא לא זיהה שהשלב הראשון בתכנון טיול הוא הטיסה. באיטרציה הבאה הוא כבר בוחר בכלי הנכון, אבל מחזיר טיסות עם קונקשן — בדיוק מה שנאסר עליו [3:04].

3. למה זה קורה: attention והלולאה האג'נטית

ההסבר עבר דרך מנגנון ה-attention: מודלים לא קוראים טקסט ליניארית, אלא כל טוקן בקונטקסט "שם לב" לכל טוקן אחר. זה המנגנון שנותן משקל למידע רלוונטי ומאפשר להסיק מסקנות — אבל הוא משאב מוגבל.

> **— שקד דולצמן [4:35]** ככל שיש לנו יותר טוקנים בקונטקסט, כך ה-attention מתפרס ביניהם, ואז הוראות כמו טיסה ישירה בלבד הולכות לאיבוד.

הלולאה האג'נטית מחמירה את זה מבנית. באיטרציה הראשונה הקונטקסט מכיל את ה-system prompt, הפרסונה, הכללים, כל רשימת הכלים, הידע שהוזן מראש והבקשה של המשתמש. האג'נט חושב, בוחר כלי, מתבונן בתוצאה ומחליט אם סיים. אם לא — כל הקונטקסט של האיטרציה הראשונה, כולל תוצאות הכלי, נוסף כקונטקסט לאיטרציה הבאה [4:35].



כל איטרציה מוסיפה לקונטקסט: התוצאות של הסיבוב הקודם הופכות לקלט של הסיבוב הבא, וכך הלאה.

התוצאה היא מה שכונה context rot: מידע חשוב הולך לאיבוד, הדיוק בבחירת כלים יורד, האג'נט מתחיל להתעלם מהוראות ולערבב פרטים מאיטרציות שונות. השאלה המתבקשת — האם עם חלון קונטקסט של מיליוני טוקנים הגודל עדיין הבעיה? — קיבלה תשובה חד-משמעית [6:07].

— שקד דולצמן [6:07] הרבה לפני שנגיע לתקרה, הביצועים מתחילים להידרדר. לרוב אנחנו נראה שאת התוצאות הטובות ביותר אנחנו מקבלים בתחילת השיחה.

4. מה עושים במקום: ארבעה כללי הנדסת קונטקסט

העיקרון הפותח הוא הקשוח מכולם: לא כל מה שנכון צריך להיות בקונטקסט. לפני שמוסיפים הוראה — לבדוק האם המודל מפספס את הבקשה בלעדיה, ולהריץ כמה וכמה שאילתות. "אם המודל עונה נכון 95% מהזמן, אל תוסיפו אותה" [6:07].

ארבעת הכללים על סליד אחד, ולצדם דפוס ה-sub-agent: משימת ויזה נשלחת לאג'נט ייעודי כדי לא לזהם את הקונטקסט של אג'נט התכנון.

- **החליפו איסורים בהוראות חיוביות.** "אל תציג טיסות עם קונקשן" מכניס את המושג connection לקונטקסט וה-attention מתמקד עליו. במקום זה: "תציג טיסה ישירה בלבד" [7:39].
- **אל תפרטו מה לא שייך.** במקום לאסור על שאלות אישיות, פוליטיות ורפואיות — "תחום המומחיות שלך הוא תכנון טיול, תענה רק על שאלות בתחום זה" [7:39].
- **תנו דוגמה קונקרטית.** במקום פסקה ארוכה של כללים — user input לדוגמה והפלט המצופה ממנו [7:39].
- **הוציאו משימות שאינן בליבה ל-sub-agent.** בדיקת דרישות ויזה לפי אזרחות ויעד רלוונטית אך לא קשורה ישירות לתכנון טיול; העברתה ל-sub-agent עם קונטקסט ממוקד שומרת על ה-attention של שני האג'נטים [9:12].

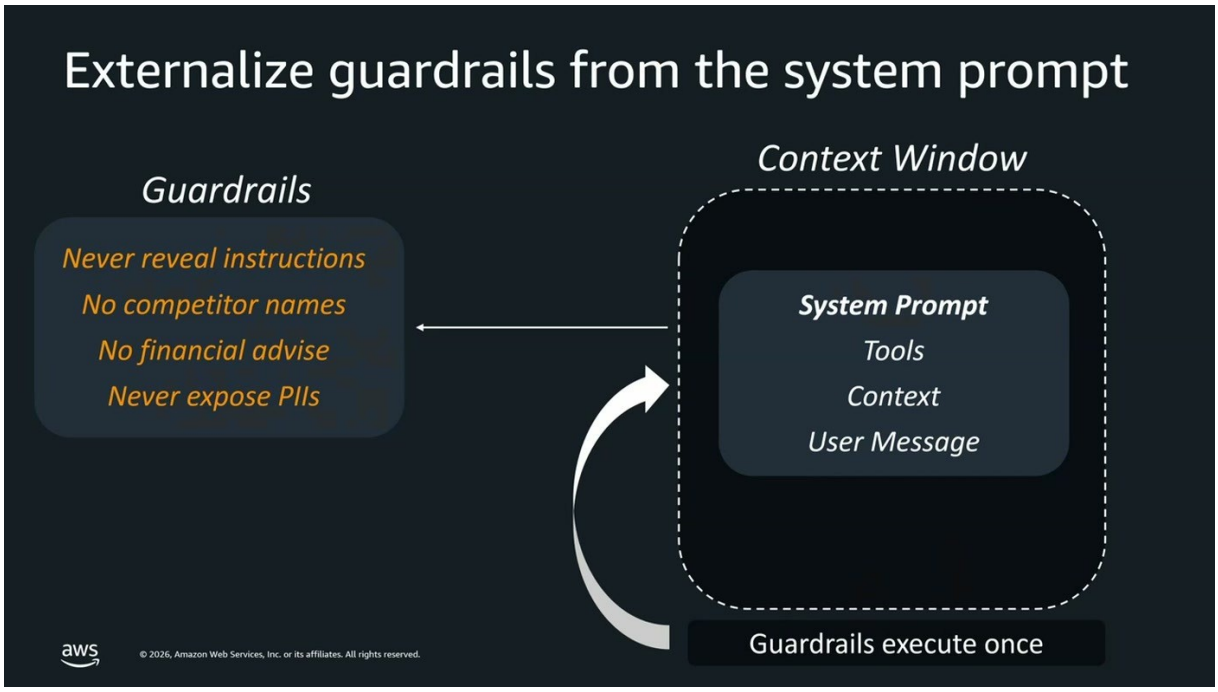
לגבי תנאיות if/then הייתה אמירה מפורשת: הן נכשלות לרוב כי הן ריאקטיביות — "עד שאנחנו מגיעים למצב שהתנאי שלנו מתקיים, ה-attention של המודל כבר מפוקס לכיוון מסוים, ועכשיו לנסות ולשנות אותו, זה כנראה לא יעבוד" [7:39].

> **— שקד דולצמן [7:39]** תחשבו על זה קצת כמו להגיד לא או אסור לפעוטות. אני לא יודעת מה איתכם, לי יש ילדה בת שנה וחצי, כשאני אומרת לה לא או אסור — זה בדיוק מה שהיא עושה.

5. גארדרייז: להוציא אותם מהסיסטם פרומפט

גארדרייז נחוצים — שהאג'נט יעשה רק את מה שתוכנן לעשות, לא יחשוף PII ולא ייצר תוכן מזיק. הבעיה היא איפה הם יושבים: רובנו שמים אותם בסיסטם פרומפט, והם הופכים לאחד המקורות הגדולים לבזבוז attention.

> **— שקד דולצמן [9:12]** הבעיה שהכללים האלה הם סטטיים, הם לא משתנים בין ריצה לריצה, ועדיין, כמו שראינו בלופ האג'נטי, הם נשלחים למודל כל פעם מחדש.



הגארדרייז יוצאים מחלון הקונטקסט לשירות שרץ פעם אחת לבקשה, מסביב לקריאה למודל.

הפתרון שהוצג: gateway חיצוני לגארדרייז — שירות כמו Amazon Bedrock Guardrails שבודק את הקלט והפלט פעם אחת לבקשה, בצורה דטרמיניסטית. ה-attention של האג'נט מתפנה למשימה, ובסיסטם פרומפט נשארת "שכבה רזה של שורה או שתיים" כקו הגנה אחרון [9:12–10:50].

6. אובזרבביליטי ואבלואציה: מה למדוד

החלק הראשון נסגר בשאלה מעשית — איך בכלל יודעים כמה לולאות חשיבה האג'נט ביצע ואיך הוא בחר כלים. התשובה: בלי אובזרבביליטי ואבלואציה אין על מה לדבר.

המדד הראשון ברשימה — ספירת איטרציות וקריאות לכלים; קפיצות בספירה מסמנות אג'נט מבולבל.

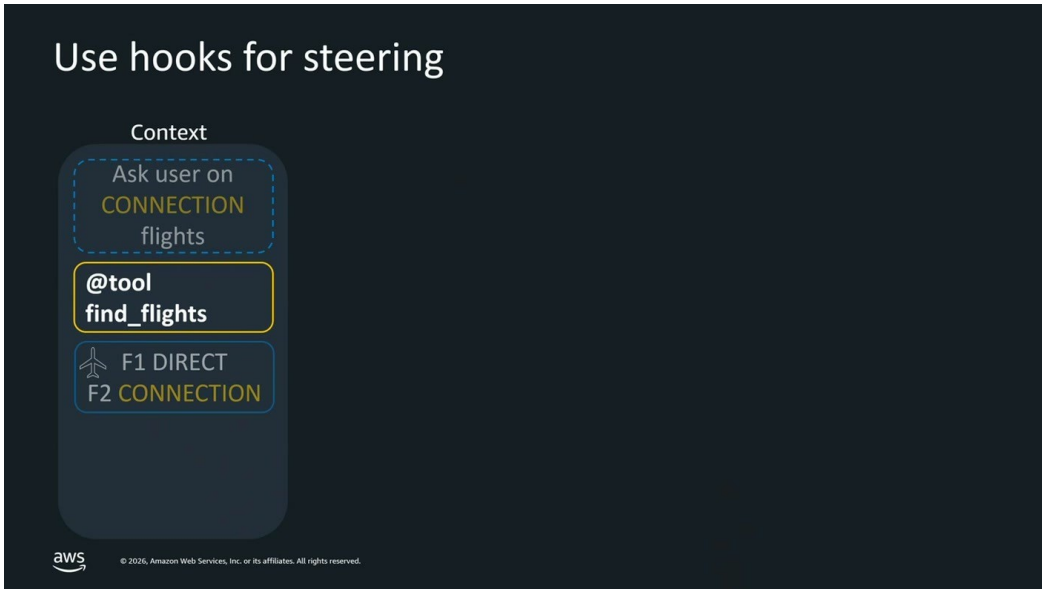
- **איטרציות וקריאות לכלים.** "תמדדו כמה איטרציות ביצע האג'נט שלכם עד שהוא החזיר לכם את התשובה והשלים את המשימה", וכמה טוקנים היו בכל אחת מהן [10:50].
- **זמן לכל איטרציה, מפולח.** להפריד בין זמן המתנה למודל לבין latency שמגיע מכלי API — ברגע שמאתרים את צוואר הבקבוק אפשר לתקן אותו [10:50].

- **דיוק בבחירת כלים.** כל בחירה שגויה היא עוד איטרציה בלולאה, עוד טוקנים מבוזבזים ועוד attention שנשרף [10:50].
- **התאמה בין ה-reasoning לפעולות בפועל.** לוודא שתהליך החשיבה של האג'נט תואם את הפעולות שהוא באמת ביצע [10:50].
- **אבלואציה כ-quality gate.** לפני כל שינוי, כדי לוודא שלא שוברים את מה שכבר עובד [10:50, 38:31].

7. Hooks: הזרקת קונטקסט בנקודת ההחלטה

דורון בלייברג פתח את החלק השני בבעיית Lost in the Middle: כשהקונטקסט מתארך והאג'נט רץ, האם הוא באמת זוכר את ההנחיה שנתנו בתחילת הקונטקסט? הכלי שהוצג הוא hooks — מנגנון שה-SDKs מספקים.

> **— דורון בלייברג [12:23]** הוקים מאפשרים להגדיר קולבקים דטרמיניסטיים בזמן הריצה של האג'נט עצמו, במקומות שונים של תהליך הריצה: תחילת ריצה של אייג'נט, סוף ריצה של אייג'נט, לפני טול, אחרי טול.



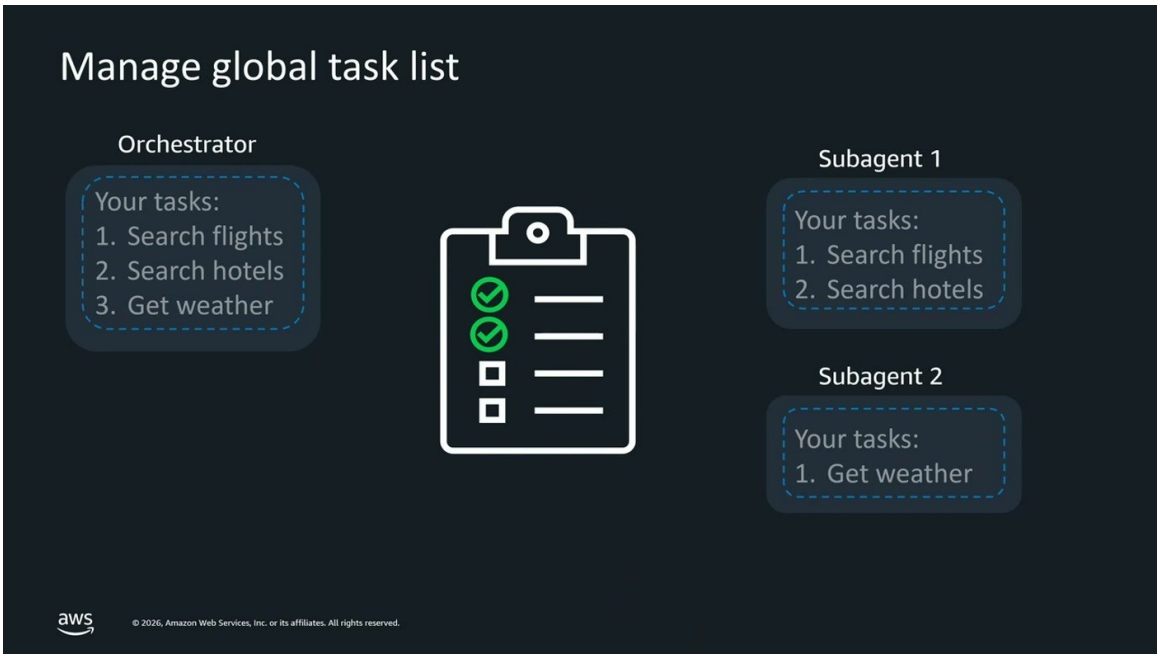
hook שרץ אחרי הכלי find_flights: הוא בוחן את התוצאות, מזהה טיסות עם קונקשן, ומזריק את ההנחיה בדיוק ברגע שבו היא רלוונטית.

הדוגמה: hook שנרשם על after-tool, עובר על רשימת הטיסות שחזרה, מזהה שיש בה טיסות עם קונקשן, ומזריק ברגע הזה attention pull ספציפי — "הנה רשימת הטיסות שחזרה, בוא תטפל בבקשה עבור הלקוח". הסיכוי שהאג'נט יעקוב בשלב הזה גבוה בהרבה [13:55].

> **— דורון בלייברג [13:55]** בחינות שאנחנו עשינו לשיטות כאלה הראו שיפור מאוד מאוד משמעותי ביכולת של האג'נטים לעקוב אחר משימות.

8. Task list גלובלי בקובץ חיצוני

הדפוס הרווח הוא לבקש מהאג'נט בתחילת הריצה לחשוב ולפרט את הצעדים הבאים. הבעיה: הרשימה יושבת אי-שם בתחילת הקונטקסט, ומעקב אחריה — אילו כלים נקראו, מה הצליח ומה נכשל — דורש reasoning כבד על פני קונטקסט שגדל כל הזמן [13:55].



רשימת משימות אחת מחוץ לקונטקסט, שהאורקסטרטור ושני sub-agents קוראים וכותבים אליה.

הפתרון: להוציא את רשימת המשימות לקובץ persistent חיצוני. שני רווחים מיידיים — הסטטוס נשמר בצורה מסודרת ועקבית במקום אחד, ו-sub-agents נוספים יכולים לשתף פעולה על אותה רשימה. את הסטטוס המעודכן אפשר להזריק חזרה לקונטקסט אחת לכמה זמן, שוב באמצעות hooks: מה בוצע, מה נכשל ומה עומד כרגע לפני האג'נט [15:27].

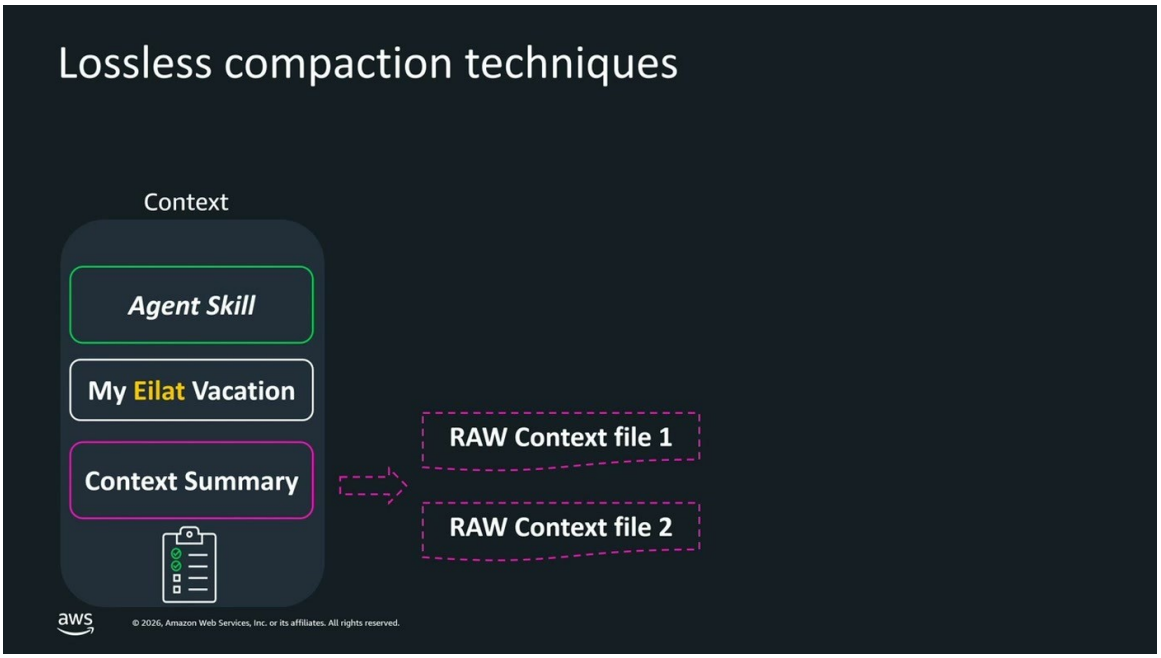
9. Skills, progressive disclosure ודחיסה ללא איבוד מידע

שתי הטכניקות הקודמות מוסיפות קונטקסט. הטכניקה הבאה מורידה אותו. Skills מאפשרים לאג'נט להחזיק skill ראשי שמפנה ל-skills נוספים — למשל Domestic Flights ו-International Flights — וכל skill מכיל הפניות לקבצי reference או לסקריפטים.

כשמבקשים תכנון טיול לאילת, האג'נט מבין סמנטית שמדובר בטיסה פנים-ארצית וטוען לקונטקסט אך ורק את ה-skill הרלוונטי, ואיתו את רשימת האתרים המומלצים לחיפוש טיסות בארץ [15:27].

> **— דורון בלייברג [16:58]** ככה אנחנו שומרים על הקונטקסט עם Just-in-Time Injection של קונטקסט, או מה שנקרא Progressive Disclosure — ולטעון רק את מה שצריך ברגע שהאג'נט צריך.

גם עם זה, אג'נט שרץ לאורך זמן ימלא את חלון הקונטקסט ויידרש ל-compaction. הבעיה המוכרת בסיכום קונטקסט היא איבוד מידע. הפתרון שהוצג משתמש באותו progressive disclosure: בתוך ה-summarization שותלים הפניות לקבצים שמחזיקים את המידע המקורי.



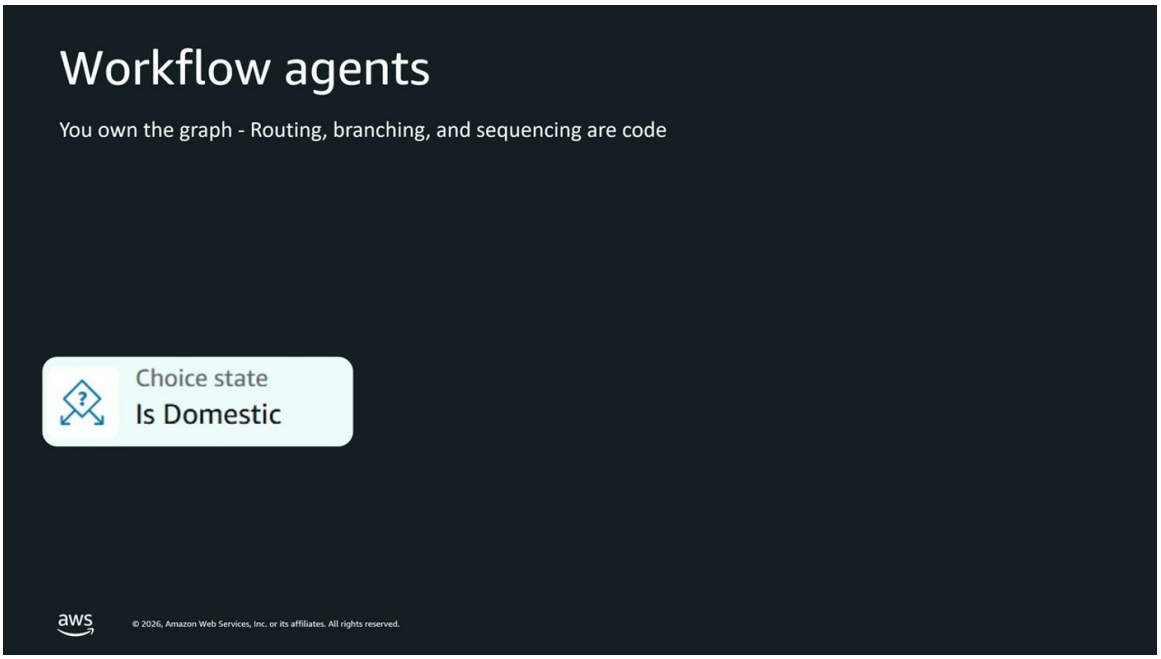
ה-Context Summary מחזיק מצביעים לקבצי RAW context — האג'נט טוען אותם רק אם הסיכום לא מספיק לו.

ברגע שהאג'נט רואה שהתוכן בסיכום לא מספיק, הוא טוען שוב את הדאטה המפורט ששמור בצד; אם הסיכום מספיק — הסתיימנו. בסיום הדחיסה מחזירים גם את ה-task list לקונטקסט, כדי שהאג'נט ידע מה בוצע ומה השלב הבא [18:28].

10. Workflow agents: להחזיר את הדטרמיניזם

אחרי כל המאמץ לשמור על אג'נט אוטונומי שרץ כמו שרוצים, הגיעה שאלת הבסיס — האם בכלל צריך אוטונומיה בכל שלב.

> **— דורון בלייברג [18:28]** אבל לא כל טסק הוא טסק שחייב להיות אוטונומי עבור האג'נט. בהרבה מאוד מקרים אנחנו יכולים לדעת מראש מה יהיה גרף הריצה שאנחנו נרצה.



"You own the graph" — ראוטינג, ברנצ'ינג ו-sequencing מוגדרים בקוד; ה-Choice state מחליט Is Domestic בלי לשאול את המודל.

הפתרון: workflow agents מעל workflow managers מבוססים — AWS Step Functions, Temporal ואחרים. גרף הריצה הופך לדטרמיניסטי ומוגדר בקוד, והיכולות האג'נטיות של ה-LLM משמשות ל-heavy lifting של

ה-reasoning וההבנה. ההחלטות על branching, sequencing ו-routing מתקבלות בקוד, לא על ידי המודל [20:02].

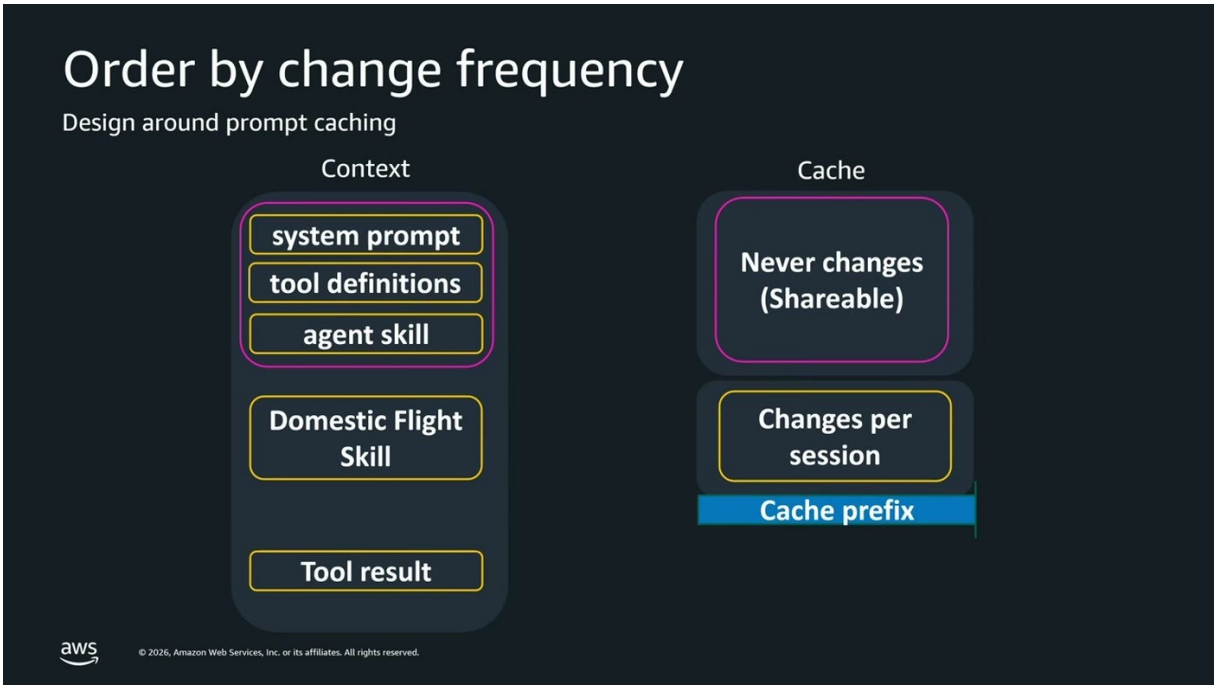
היתרון הנוסף שהודגש הוא קונסטרקטים כמו הרצה במקביל: "אני לא יודע מי מכם שניסה עם ה-SDK-ים השונים להריץ מודל בפרלל, אנחנו בדרך כלל נשענים שוב פעם על wishful thinking של instructions." [20:02] השיטה מורידה קונטקסט, משפרת דטרמיניזם, מקטינה את כמות הקריאות ל-LLM, ומשפרת עלויות וזמנים.

11. Prompt caching: איפה נמצא הכסף

כדי לרתום prompt caching צריך להבין מה הוא עושה. כששולחים קונטקסט ל-LLM, הוא מחשב את ה-attention על מה ששלחנו. החישוב הזה כבד — דורש לא מעט compute — והוא דטרמיניסטי: אותו קונטקסט יניב תמיד אותו attention. השלב הזה נקרא pre-fill, ואת תוצאתו אפשר לשמור במטמון. אחריו ממשיך ה-LLM לשלב ה-decode שמייצר את הטוקנים אחד אחרי השני [20:02].

> **— דורון בלייברג [21:36]** טוקנים שמגיעים מתוך הקאש הם בכ-90% זולים יותר מאשר טוקנים שלא מהקאש.

השימוש הקל הוא הקונטקסט ההתחלתי — system prompt, tool list, agent skill. אם הוא נשאר סטטי, הוא נשמר במטמון ונעשה בו reuse. האזהרה הייתה מפורשת: לוודא שהוא באמת לא משתנה. "שמנו tenant id כפרמטר — שינינו; שמנו date and time — שינינו" [21:36]. את שאר הקונטקסט צריך לפרוס בצורה של append only.



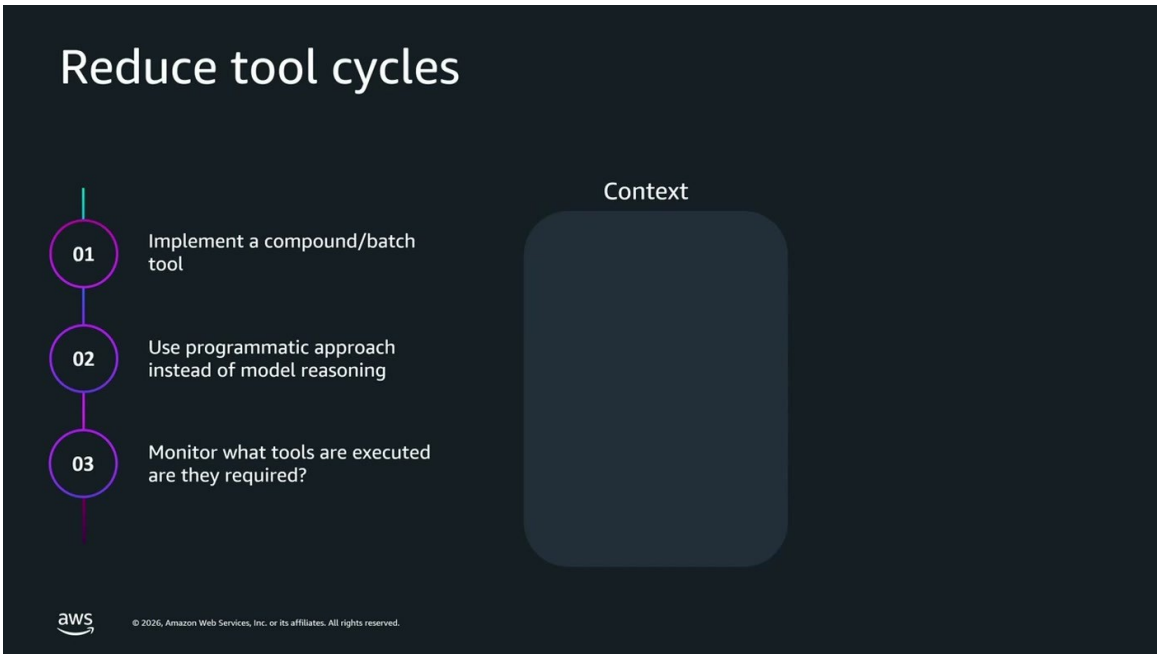
סידור הקונטקסט לפי תדירות שינוי: מה שלעולם לא משתנה (ושניתן לשיתוף בין סשנים) למעלה, מה שמשתנה פר-סשן מתחתיו, ותוצאות הכלים בסוף — עם ה-cache prefix בין השכבות.

בפועל סשנים משתנים: מי שביקש טיסה לאילת טען Domestic Flight Skill, ומי שביקש טיסה ללאס וגאס טען משהו אחר. הטריק הוא להתייחס לכל שלב שחזר מהבקשה הקודמת כסטטי ולשמור גם אותו במטמון — כך שכל איטרציה היא cache hit על החלק הקודם, עם cache miss קטן רק על החלק החדש [23:07].

> **— דורון בלייברג [23:07]** בצורה הזאת אנחנו יכולים להגיע לקאשים מאוד מאוד גבוהים באחוזים שלהם, יכולים להגיע גם ל-80-90% ואפילו יותר ולהוזיל את העלויות באופן משמעותי.

12. צמצום מחזורי כלים: compound tool במקום reasoning

הגורם השני לעלות הוא מספר הסייקלים. בדרך כלל כותבים כלים בגרנולריות נמוכה, או מייבאים אותם מ-MCP שכבר החליט עבורנו. זה נותן לאג'נט גמישות לעבוד דינמית ואוטונומית — "אבל כל החלטה כזאת היא עוד ריזנינג, היא עוד קריאה למודל, עוד קונטקסט" [24:41].

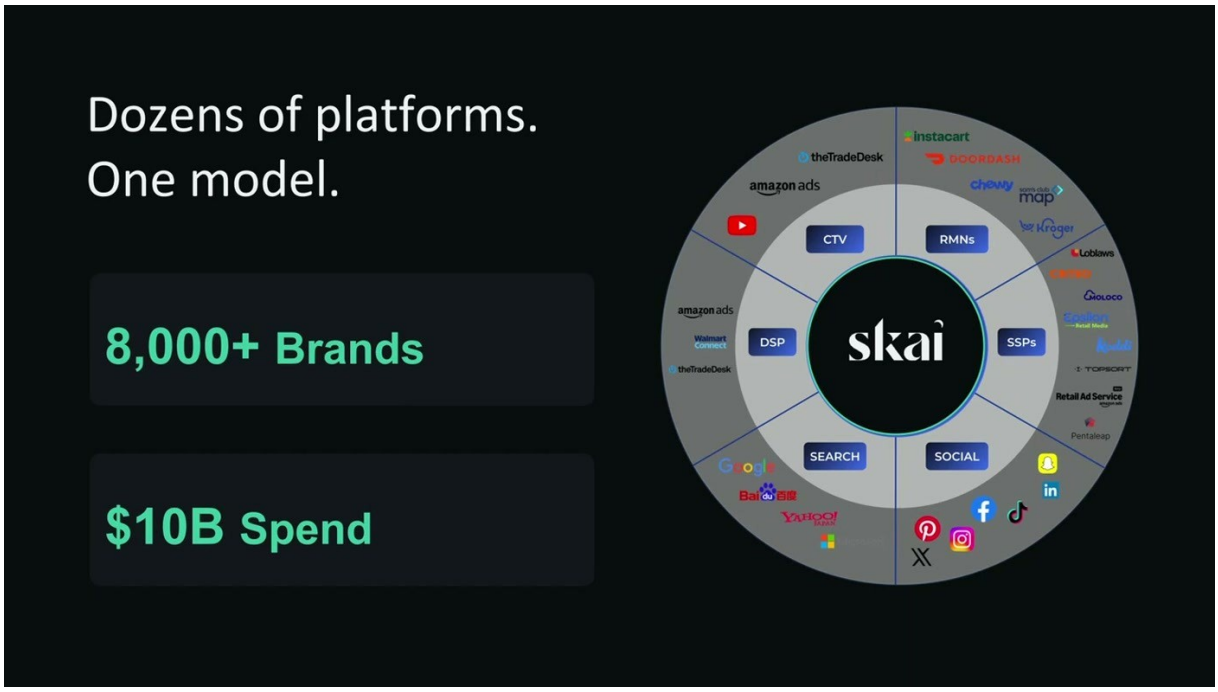


שלושה צעדים: compound/batch tool, גישה פרוגרמטית במקום reasoning של המודל, וניטור של אילו כלים באמת מורצים ואם הם נחוצים.

— דורון בלייברג [24:41] במקרה הזה אני יודע שתמיד אני רוצה לקרוא ל-Search Flight, Search Hotels ו-Get Weather. אז למה לא לכתוב כבר איזשהו Compound Tool שמייצר את השלישייה הזאת של הקריאות — כקריאות כבר פונקציה ולא קריאות כלים — ולהקטין את כמות ה-Cycling.

13. המקרה של Skai: Celeste AI והסטודיו

ירדן רון, Engineering Manager ב-Skai, מוביל כשנתיים את הצוות שבונה אג'נטים ללקוחות. הלקוחות הם מנהלי קמפיינים במותגים גדולים, עם תקציבי פרסום גדולים, שמפרסמים ביום-יום בגוגל, בפייסבוק, באמזון ובעוד רשימה ארוכה של ערוצים — לכל דומיין טרמינולוגיה שונה ומדדים שונים [24:41–26:15].



הסקייל שמאחורי הבעיה: יותר מ-8,000 מותגים ו-10 מיליארד דולר תקציב מנוהל, על פני עשרות פלטפורמות פרסום.

Skai יושבת באמצע ועונה על השאלות האלה: "למעלה משמונת אלפים ברנדים שמנהלים דרך המערכת כעשרה מיליארד דולר של תקציבי פרסום בשנה" [26:15]. ההחלטה הייתה לקחת מודלי שפה ולייצר אג'נט שממיר את הפרומפט של הלקוח לפעולות במערכת.

ההתחלה הייתה מהירה ומכוונת פידבק: מודלים מ-Bedrock, חשיפה מוקדמת ללקוחות. הפידבק הגיע מהר ואישר את הכיוון — אבל גם הבהיר שצריך לכתוב מחדש כדי לעמוד בקצב ההתפתחות הטכנולוגית. הגרסה השנייה נבנתה מעל Strands, ה-open source framework של AWS, "שאפשר לבנות גרסה מלאה, גמישה משלנו ולרוץ איתה בקצב של הטכנולוגיות" [27:45].

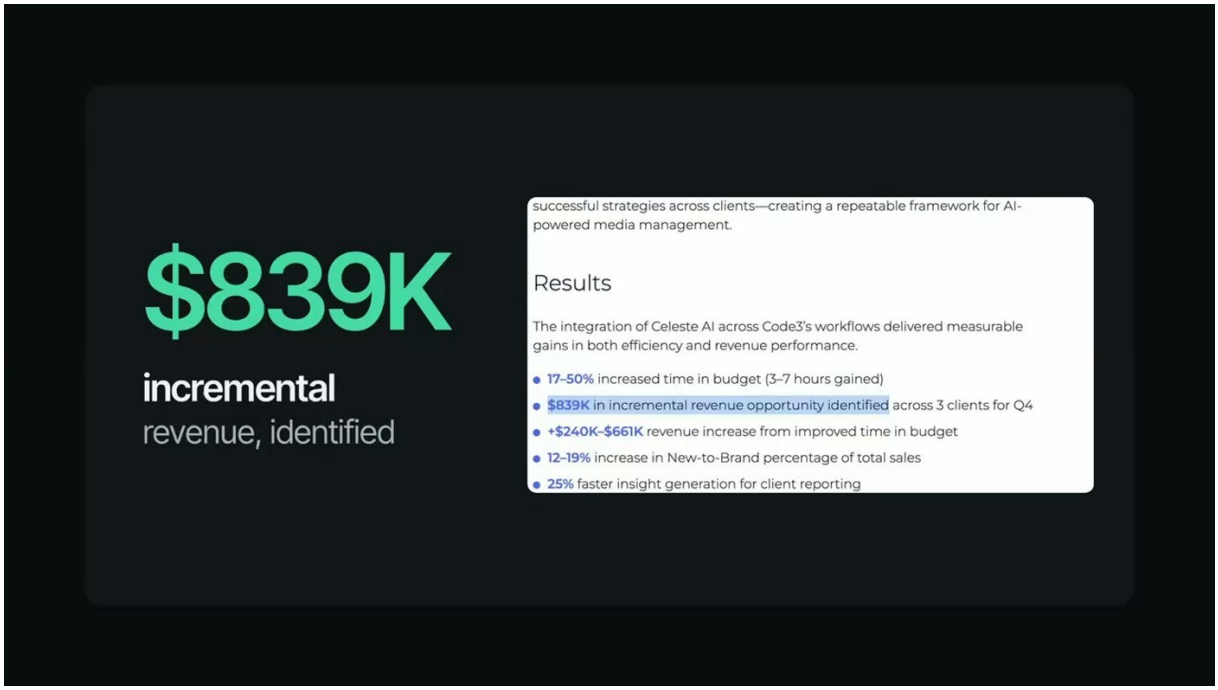
> **— ירדן רון, Skai [27:45]** אז שנה וחצי בפרודקשן עם האג'נט שלנו Celeste, למדנו הרבה דברים בדרך, ומכל מה שלמדנו לקחתי שלושה דברים שאני חושב שהביאו את האימפקט הכי גדול על הלקוחות שלנו.

שלושת השינויים שהביאו את האימפקט

- **מודל ייעודי לפענוח טרמינולוגיה.** המילה "visibility" אומרת דבר אחד בעולם האמזון ודבר אחר בעולם ה-display. Celeste נדרשה להמון איטרציות רק כדי למצוא את המטריקה הנכונה, ולכן הוצא החוצה מודל חדש שקובע מראש מהן המטריקות, הדומיינים וה-entities הרלוונטיים — "מהר מאוד הרגשנו את השיפור בקוואליטי" והחיסכון באיטרציות מיותרות [29:16].
- **למידה של הטרמינולוגיה הפנימית של כל לקוח.** מחרוזת שנראית חסרת פשר היא המספר הסידורי של דגם נעליים חדש, או prefix בשם קמפיין. נבנתה מערכת שלומדת את המשתמשים מאחורי הקלעים — גם כשהלקוח מבקש במפורש לזכור, וגם מתוך האיטרציות בשיחה עצמה [30:49].
- **ספריית טמפלטים וכפתור שמירה אחד.** כתיבת פרומפטים אינה הצד החזק של מנהלי קמפיינים, ולכן נוצרה ספריית טמפלטים שמנצלת את המודל בצורה מיטבית; ובנוסף — כפתור אחד ששומר תשובה מוצלחת לספריית הפרומפטים של הלקוח. הפיצ'ר "צבר תאוצה אדירה" ו-stickiness גבוה מרגע יציאתו [32:22].

> **— ירדן רון, Skai [30:49]** במקרה הזה דורון דיבר קודם על ההוקים — אפשרנו בדרך הוקים להזריק בדיוק בזמן הנכון לקונטקסט את מה שCeleste צריכה, ולאפשר לה להתחיל נקי עם attention מאוד ברור בדיוק למה שהיא צריכה.

התוצאות שדווחו: שיפור בחוויה ובהתייעלות היומיומית, הכנת דוחות שלקחה מעל שעה ירדה למספר דקות, ועמידה ב-KPI פנימיים — "אנחנו רואים את זה בחתימות החוזים מחדש, הם לא מוכנים לוותר על [32:22] "Celeste.



סליד תוצאות של לקוח (Code3) כפי שהוצג: $839K הזדמנות הכנסה שזוהתה על פני שלושה לקוחות ברבעון, 17–50% שיפור בזמן-בתקציב, ו-25% האצה בהפקת תובנות.

השלב הבא: הסטודיו מעל AgentCore

החלק האחרון עסק במה ש-Celeste עדיין לא עשתה. הלקוחות עדיין נדרשו לקום בבוקר ולבקש ממנה אנליזה, בעוד הפרסומות רצות כל הזמן. הדרישה שלהם, בתרגום של רון: "אני רוצה אוטונומיה מלאה שתרוץ כל היום, שCelestew תהיה שם כל הזמן בשבילי" [33:54] — אבל עם גבולות גזרה ברורים: "אני לא יכול לישון בלילה, לקום בבוקר ולגלות שהיא שינתה לי את כל המערכת בלי שאני מודע לזה".

ההחלטה הייתה לא להרחיב את Celeste אלא לבנות מוצר נפרד לצדה — הסטודיו — מעל Amazon Bedrock AgentCore. הוא חושף מערכת טריגרים שמזהה אנומליות ומדדים שזזים מעבר לצפוי (למשל שינוי קיצוני ב-ROI), והטריגרים מפעילים workflows שהלקוח מגדיר [35:24].

> **— ירדן רון, [35:24] Skai** ולכל agent יש בדיוק את הדומיין הרלוונטי, רק הסקילים שמעניינים אותו, הוא לא מקבל את כל הדסקריפשנים של כל מה שלא מעניין אותו. הוא מאוד ממוקד. ה-attention ברור, אנחנו מורידים בצורה משמעותית את מרחב הטעות.

הנימוק לבחירה ב-workflow על פני אג'נט אחד גדול היה ישיר: "אם אני אבקש מאג'נט אחד גדול ובודד לעשות את זה, אני סומך יותר מדי על הסטטיסטיקה, קונטקסט גדל, הוא עלול לטעות ואני לא מוכן לעמוד בזה. אבל כשאנחנו מדברים על workflow, אנחנו יודעים שיש פה נקודות עצירה ברורות, דטרמיניסטיות" [35:24]. השכבה האחרונה היא ביצוע מותנה: הלקוח מגדיר את התנאים שבהם הוא מוכן לתת לפעולה לרוץ גם בלילה; מה שלא עומד בקריטריונים ממתין ל-human in the loop [36:55].

שלוש שאלות שרון השאיר לקהל

- האם האג'נטים שפיתחתם עושים הרבה יותר איטרציות מאשר מה שהם באמת צריכים?
- האם יש ללקוחות שלכם flows ברורים וקבועים שרק הם מבצעים בתוך המערכת?
- האם יש לכם במערכת פעולות ברורות שמצריכות חסמים דטרמיניסטיים לפני שהן רצות?

ואת המסר החותם: "אנחנו הפסקנו לבקש מאג'נט אחד לעשות תמיד את הכל" [36:55].

14. מה לוקחים מכאן

הסיכום שהציגה דולצמן חילק את כל הסשן לארבע קבוצות של [38:31] best practices. זו הרשימה שאפשר לקחת ישר לצוות.

קבוצה	מה עושים בפועל
שמירה על attention	להעדיף הוראות חיוביות על פני איסורים; להוציא גארדרייז ל-gateway חיצוני; להוסיף הוראה רק אם באמת צריך אותה (מבחן ה-95%).
מדידה לפני שינוי	למדוד איטרציות וקריאות לכלים; להריץ אבלואציה לפני כל שינוי — זה ה-quality gate.
לוודא שהאג'נט עושה את מה שהתכוונתם	לא לתת את כל המידע מראש ולקוות שייזכר; להשתמש ב-hooks וב-progressive disclosure כדי להזריק קונטקסט בדיוק בזמן; לנהל task list גלובלי שהאג'נט יכול לחזור אליו; ואם צריך דטרמיניזם — workflow agents.
אופטימיזציה	להבין כמה קונטקסט וכמה טוקנים צורכים; לשמור על פרפיקס יציב ל-prompt caching; לצמצם קריאות לכלים באמצעות compound tool וקוד דטרמיניסטי.

> **— שקד דולצמן [38:31]** אנחנו רוצים שתהיו מסוגלים לשלם על חשיבה ולא על חזרה.

הערות להטמעה

- **הרווח המהיר ביותר הוא prompt caching.** הוא לא דורש שינוי בהתנהגות האג'נט, רק סידור מחדש של הקונטקסט לפי תדירות שינוי והקפדה על append only. שווה לבדוק תחילה שאין tenant id או timestamp ששוברים את הפרפיקס.
- **הוצאת גארדרייז ל-gateway היא שינוי ארכיטקטוני קטן עם שני רווחים.** פחות טוקנים בכל איטרציה, ובדיקה דטרמיניסטית אחת לבקשה במקום הסתמכות על ציות המודל.
- **Workflow agents הם התשובה הסבירה לתהליכים רגולטוריים.** בכל מקום שבו נדרשות נקודות עצירה, אישור אנושי ויכולת לשחזר ריצה — הגרף צריך להיות בקוד, לא בהחלטת מודל. זה בדיוק מה ש-Skai עשתה בסטודיו.
- **"מבחן ה-95%" הוא הכלי הכי זול שהוצג.** לפני כל הוראה שמוסיפים לפרומפט — להריץ כמה שאילתות בלעדיה. אם המודל עונה נכון ב-95% מהמקרים, ההוראה עולה יותר ממה שהיא מחזירה.

מונחון

מונח	הסבר כפי שהוצג בסשן
Attention	המנגנון שבו המודל נותן משקל לטוקנים רלוונטיים. משאב מוגבל — ככל שיש יותר טוקנים בקונטקסט, כך הוא מתפרס ביניהם.
Context rot	ההידרדרות שנוצרת כשקונטקסט מצטבר לאורך איטרציות: הוראות הולכות לאיבוד, דיוק בבחירת כלים יורד, פרטים מתערבבים.
Lost in the Middle	התופעה שבה הנחיות שניתנו בתחילת הקונטקסט נשכחות ככל שהוא מתארך.
Hooks	קולבקים דטרמיניסטיים שה-SDK מריץ בנקודות מוגדרות בריצת האג'נט: תחילת ריצה, סוף ריצה, לפני קריאה לכלי, אחרי קריאה לכלי.
Attention pull	הזרקת הנחיה ממוקדת ברגע שבו האג'נט צריך אותה, כדי למשוך אליה את תשומת ליבו.
Skills	יחידות ידע שהאג'נט טוען לפי הצורך; יכולות להכיל הפניות לקבצי reference או לסקריפטים.
Progressive Disclosure	טעינת קונטקסט just-in-time — רק מה שצריך, ורק כשהאג'נט צריך אותו.
Compaction	דחיסה או סיכום של הקונטקסט כשמתקרבים לגבול החלון. בגרסה "ללא איבוד מידע" שותלים בסיכום הפניות לקבצי ה-RAW המקוריים.

מונח	הסבר כפי שהוצג בסשן
Workflow agents	גרף ריצה דטרמיניסטי המוגדר בקוד מעל workflow manager (AWS Step Functions, Temporal); המודל עושה reasoning, הקוד מחליט על routing ו-branching.
Pre-fill / Decode	שני שלבי הריצה של ה-LLM: חישוב ה-attention על הקונטקסט (כבד ודטרמיניסטי — ניתן למטמון), ואז ייצור הטוקנים עד completion.
Prompt caching	שמירת תוצאת ה-pre-fill לשימוש חוזר. טוקנים מהמטמון זולים בכ-90%.
Compound tool	כלי אחד שעוטף רצף קריאות ידוע מראש כקריאות פונקציה, במקום כמה קריאות כלים נפרדות שכל אחת מהן דורשת סבב reasoning.
Guardrails gateway	שירות חיצוני (למשל Amazon Bedrock Guardrails) שבודק קלט ופלט פעם אחת לבקשה, במקום כללים סטטיים בסיסטם פרומפט.
Strands	ה-framework בקוד פתוח של AWS לבניית אג'נטים; הבסיס לגרסה השנייה של Celeste AI.
Amazon Bedrock AgentCore	הפלטפורמה שעליה נבנה "הסטודיו" של Skai — שכבת ההרצה לאג'נטים אוטונומיים עם טריגרים ו-workflows.