

סוכנים אסינכרוניים ב-Anthropic: Managed Agents, verifier loops זיכרון ו-dreaming

ארבעה עקרונות לסוכנים שעובדים שעות בלי השגחה: להפריד את ה-harness מה-sandbox, להפריד את הבדיקה מהעבודה, לתקן זיכרון offline ולבנות harness ברמת הארגון

מהנדס מ-Anthropic (שמו לא מופיע בהקלטה) · AI Engineer World's Fair (לפי השקפים), בפוסט חוזר ב-X של Movez · 22.07.2026

משך: כ-19 דקות · X

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה קצרה, כ-19 דקות, של מהנדס מ-Anthropic. הלוגו על המסך מראה שהיא ניתנה ב-AI Engineer World's Fair. שם הדובר לא נאמר בהקלטה ולא מופיע בשקפים. הוא מציג את האופן שבו Anthropic חושבת היום על סוכנים אסינכרוניים, כלומר סוכנים שרצים לבד שעות ארוכות. הנקודה המרכזית: Claude Managed Agents, API- שיצא באפריל, ושורה של עקרונות שלפי הדובר נכונים לכל סוכן שעובד לאורך זמן.

הטענה בגדול: ככל שה-task horizon של המודלים מתארך, צריך מוצר מסוג אחר. השלב הבא אחרי autocomplete ו-chat, ואחרי סוכני קוד סינכרוניים כמו Claude Code, הוא סוכנים אסינכרוניים. כדי שיעבדו צריך ארבעה דברים: לנתק את ה-harness מסביבת ההרצה, להעביר את הבדיקה ל-verifier בקונטקסט נפרד, להוסיף שלב offline שמתקן את הזיכרון (dreaming), ולבנות harness אחד לכל הארגון במקום harness אישי.

איך לקרוא את המסמך

- **פרק 1:** תקציר מנהלים, ומה לא נאמר בהרצאה.
- **פרק 2:** מטפורת האור והחלון, גרף ה-task horizon ושלושת משטחי ה-API.
- **פרק 3:** העיקרון הראשון. ה-(harness) brain מופרד מה-(sandboxes) hands, וה-session log הוא event log שאפשר רק להוסיף אליו.
- **פרק 4:** העיקרון השני. verifier בקונטקסט נפרד, הלולאה build/judge והניסוי ב-Parameter Golf.
- **פרק 5:** העיקרון השלישי. זיכרון in-band, ו-dreaming כהליך offline שמתקן אותו. הדוגמה היא Pokemon.
- **פרק 6:** העיקרון הרביעי. org-level harness, סוכן פרואקטיבי ו-multiplayer.
- **פרקים 7-8:** מה כדאי לעשות עכשיו, שאלות פתוחות ומילון מונחים.

הערות מקור המסמך נוצר אוטומטית מהתמלול של ההקלטה ומהשקפים שחולצו ממנה. שמות ומונחים עלולים להשתבש בתמלול האוטומטי (ASR): "Cloud Code" הוא "meter", "Claude Code" הוא כנראה METR, ו-"Sonic 3.5" הוא Sonnet 3.5. שמות שלא הצלחנו לאמת מסומנים במסמך כלא ודאיים. הציטוטים מועתקים מילה במילה מהתמלול ונבדקו מול חותמת הזמן שלהם. לא הוספנו שום מידע ממקורות חיצוניים. הכיתוב של הפוסט ב-X, כולל הציטוט שמיוחס בו לדובר, לא מופיע בהרצאה ולכן לא שימש כמקור.

1. תקציר מנהלים

ההרצאה היא בעיקר רשימה של עקרונות ארכיטקטורה ולא הכרזת מוצר. כל עיקרון מגובה בניסוי קטן שהדובר הריץ בעצמו או בתוצאה מ-Claude Plays Pokemon. אין בה מספרים עסקיים. מה שיש בה זה דפוס אחד שחוזר בכל העקרונות: מוציאים החוצה את מה שקודם ישב בתוך הקונטקסט של הסוכן. ה-state עובר ל-log חיצוני, הבדיקה עוברת לקונטקסט נפרד, תיקון הזיכרון עובר לתהליך offline, והקונפיגורציה עוברת מהמשתמש לארגון.

- **async** נהיה אפשרי רק כשה-**task horizon** התארך: בתקופת (Opus 3 (2024 מודל עבד 10–20 דקות לבד, ואז autocomplete ו-chat היו מספיקים. כשהגענו לכשעה של עבודה, async עדיין היה חוויה גרועה. רק עכשיו אפשר להשתמש בו באמת [0:00].

- **שלוש שכבות API: Messages API** (בונים harness לבד), **Agent SDK** (ה-harness של Claude Code), ו-**Managed Agents** מאז אפריל, שמגיע עם harness וגם עם תשתית deployment מנוהלת [1:31].

- **להפריד brain מ-hands**: כשה-harness ו-sandbox ישבו באותו container, נפילה של ה-container מחקה את ה-session. עכשיו ה-harness stateless, ה-session הוא append-only event log, וה-credentials נשמרים ב-vault נפרד ולא נכנסים ל-sandbox [3:02].

- **ה-session הוא אובייקט קונטקסט שלא נמחק**: compaction רגיל זורק את כל מה שלא נכנס לסיכום. כשה-log רק גדל, המודל יכול לחזור אליו ולשלוף ממנו. הדובר מקשר את זה ל-recursive language models [4:34].

- **verifier בקונטקסט נפרד**: כשהמודל בודק את עצמו באותו קונטקסט שבו עבד, יוצאת confabulation. ב-Claude Code הכלי לזה נקרא goal וב-Managed Agents נקרא outcomes, והעיקרון זהה: מגדירים מצב סיום מדיד ונותנים לקונטקסט עצמאי לבדוק אותו [6:07].

- **ב-Parameter Golf הדפוס עבד**: הדובר הריץ לולאת verifier עם Opus 4.7 ועם מודל frontier נוסף ("mythos class", השם לא ודאי). הלולאה רצה עד שה-outcome התקיים, 20 איטרציות בדיוק שעמדו בתנאי הבנצ'מרק. מספרי loss לא נמסרו [7:39].

- **מודלים חזקים יותר כותבים זיכרון טוב יותר**: ב-Pokemon, Sonnet 3.5, רשם הערות טקטיות ו-4.6 רשם הערות אסטרטגיות. ב-Continued Learning Bench הביצועים משתפרים מדור לדור. ההבדל הוא ביכולת לזקק הפשטה שתעזור בהמשך [10:42].

- **dreaming מתקן טעויות בזיכרון**: זיכרון שגוי אחד גרם ל-Claude לטעות במיקום שלו במשחק ולנפול ל-trap door בחמש ריצות מתוך חמש. עם שלב dreaming offline הטעות תוקנה [13:53].

- **org-level harness**: הדובר אומר ש-"Cloud Tag" (השם לא ודאי) אינו סתם Slack bot. זה harness multiplayer עם זהות משלו ועם גישה לקונטקסט של הארגון. עובד חדש מקבל harness מוכן ביום הראשון, וה-harness יכול גם לפנות לאנשים ביוזמתו [15:25].

מה לא נאמר

- **הכיתוב בפוסט ב-X לא מגובה בהרצאה**: ההרצאה לא אומרת ש-80% מהמהנדסים משתמשים ב-self-improving loops. אין בה "agentic graphs", אין תחזית של 4–6 חודשים ואין "no more prompting".
- אין מספרים בניסויים: לא ערכי loss מ-Parameter Golf, לא ציונים מ-Continued Learning Bench, ולא כמה רחוק הגיע כל מודל ב-Pokemon.

- לא הוסבר איך בנוי ה-dreaming: מה מפעיל אותו, באיזה מודל הוא רץ, כמה הוא עולה, ואיך מוודאים שהוא לא מכניס לזיכרון טעויות חדשות.

- לא נאמר מה ה-latency או העלות של ארכיטקטורה שבה ה-harness stateless ושולף כל הזמן מה-session-log.

- על ה-org-level harness לא נאמר דבר לגבי הרשאות, פרטיות בתוך הארגון, או מה קורה כשהרבה אנשים מכוונים אותו בבת אחת ומבקשים דברים סותרים.

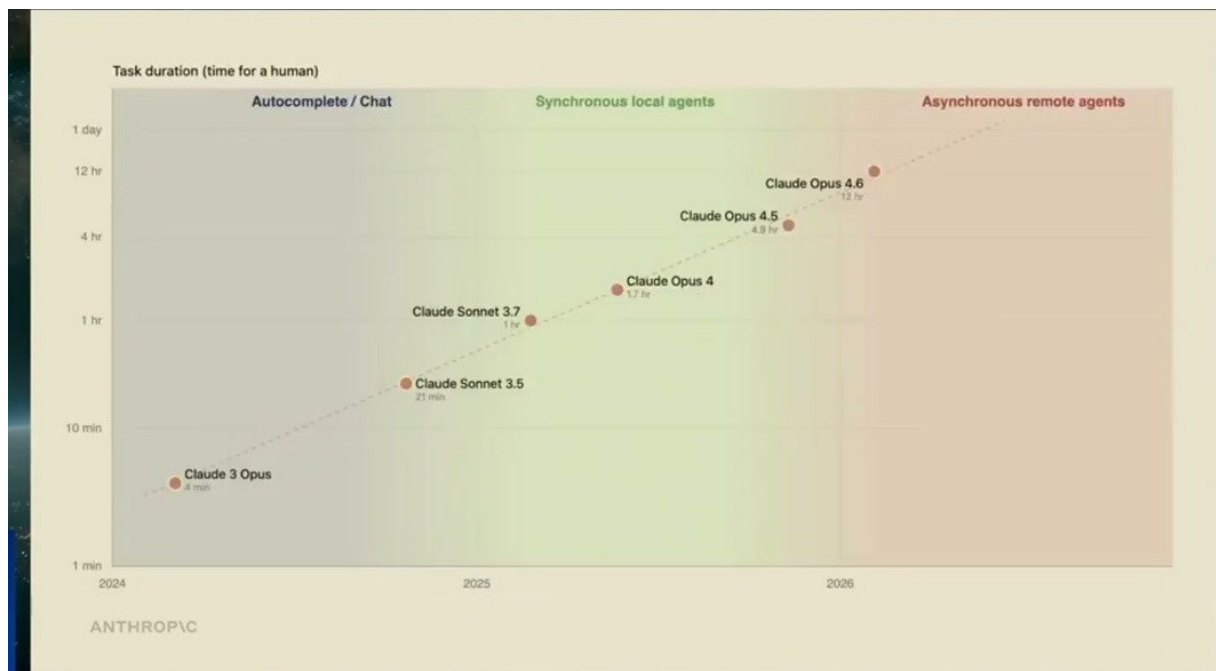
- ה-Q&A לא נכלל בהקלטה.

2. מחלון לחלון: task horizon ומשטחי ה-API



שקף הפתיחה: ציור של חלון שאור נכנס דרכו לחדר חשוך. זו המטפורה של ההרצאה, המודל הוא מקור האור והמוצר הוא החלון

המסגרת שהדובר מציע: Claude הוא מקור אור, ומוצרים הם החלונות שהאור עובר דרכם. לאורך זמן משתנה החלון שצריך כדי לראות את האור. כשמודלים עבדו לבד 10–20 דקות בלבד (Opus 3, 2024), לפי מדידה שהתמלול קורא לה "meter" (ונראה שהכוונה ל-METR), היה הגיון רק במוצרים שבהם האדם מעורב כל הזמן, כמו autocomplete ו-chat. כשהגענו לכשעה, הגיע תורם של סוכני קוד סינכרוניים כמו Claude Code, שרצים מקומית ואפשר לכוון אותם בקלות.



גרף task horizon של דורות Claude, מ-Claude 3 Opus ועד דור x.4. הציר האנכי הוא משך המשימה בזמן של אדם. הרקע מחולק לשלושה אזורים: Autocomplete/Chat, סוכנים מקומיים סינכרוניים וסוכנים מרוחקים אסינכרוניים

הדובר מודה שהוא עצמו היה מעורב בניסיונות לבנות סוכנים אסינכרוניים בשלב שבו המודלים החזיקו כשעה, ושזה לא עבד: "But when models can only do like an hour of work, async as an experience is kind of bad." [0:00]. הסוכן נתקע בשגיאה וחוזר אליך אחרי זמן קצר. רק עכשיו ה-horizon ארוך מספיק.

משטחי ה-API השתנו בהתאם. Messages API יצא לפני כשנתיים: prompt נכנס, תשובה יוצאת, אין deployment, ובונים harness לבד. Agent SDK מאפשר לקרוא ל-Claude Code מתוך קוד, כלומר Anthropic נותנת את

ה-harness. Managed Agents, מאז אפריל, נותן את ה-harness וגם את תשתית ה-deployment המנוהלת. לפי הדובר, העקרונות שבהמשך תקפים גם מעבר ל-Managed Agents, למשל ל-"CLAWS" (כך בתמלול, לא ברור למה הכוונה) ולכל סוכן שרץ לאורך זמן.

3. עיקרון ראשון: לנתק את ה-brain מה-hands

בגרסה הראשונה של Managed Agents, ה-harness וה-sandbox יישבו באותו container. היו לזה שתי בעיות. הראשונה: כשה-container או ה-harness נפלו, ה-session הלך לאיבוד, וזה הרסני לסוכן שרץ שעות. השנייה: ה-credentials יישבו באותו container עם הסוכן. לתת למודל חזק גישה לסודות ולהשאיר אותו לרוץ 10 שעות בלי השגחה זה, במילים של הדובר, "a little bit spooky" [3:02].

- **Brain = harness**: תהליך stateless שמדבר עם session.
- **Session**: event log שאפשר רק להוסיף אליו. אם ה-harness או ה-sandbox נופלים, אפשר להמשיך ממנו.
- **Hands = sandboxes**: containers שמריצים את הכלים. Harness אחד יכול לנהל הרבה hands, ולפי הדובר Claude עושה את זה בקלות וביעילות.
- **Vault**: ה-credentials לא נכנסים ל-sandbox בשום שלב. הם נשמרים ב-vault נפרד.

— **הדובר**, Anthropic [4:34] In this architecture and also more broadly with recursive language models, the idea is that the context object is persistent and is unadulterated. So it's append only. And the model can always go back and fetch old context

מזה יוצא יתרון ל-compaction. context engineering רגיל בוחר מה לשמור וזורק את כל השאר. כאן ה-session הוא אובייקט קונטקסט חיצוני שהמודל שולף ממנו לפי הצורך, ושום דבר לא נמחק. הדובר מתאר את זה כ-"a very, very nice architecture for context engineering" [4:34].

4. עיקרון שני: להשתמש ב-verifiers



ציור של כיתה צפופה וכאוטית. השקף ממחיש מודל שבדוק את העבודה של עצמו מתוך קונטקסט עמוס במידע

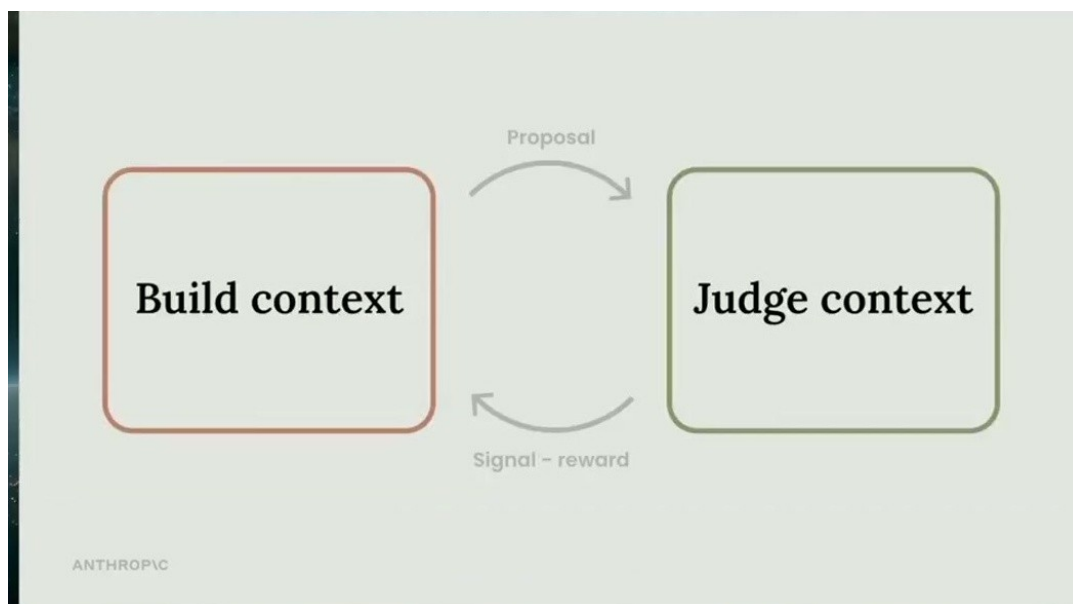
הבעיה: כשאותו context window משמש גם לעבודה וגם לבדיקה שלה, יוצאים "odd artifacts and confabulation" [6:07]. בקונטקסט כבר יש הרבה מידע, והמודל לא מכוון לבדיקה ביקורתית. הפתרון של Anthropic הוא להעביר את הבדיקה ל-context window נפרד, שאפשר לכוון במיוחד למשימת הביקורת.

However, even on tasks that do have verifiable outcomes, agents still sometimes exhibit poor judgment that impedes their performance while completing the task. Separating the agent doing the work from the agent judging it proves to be a strong lever to address this issue. The separation doesn't immediately eliminate that leniency on its own; the evaluator is still an LLM that is inclined to be generous towards LLM-generated outputs. But tuning a standalone evaluator to be skeptical turns out to be far more tractable than making a generator critical of its own work, and once that external feedback exists, the generator has something concrete to iterate against.

ANTHROPIC

[Anthropic Engineering Blog](#)

קטע מבלוג ההנדסה של Anthropic: גם במשימות עם תוצאה שאפשר לאמת, סוכנים מפגינים שיקול דעת חלש. הפרדת הסוכן שעושה את העבודה מהסוכן ששופט אותה מתוארת כמנוף חזק. לפי הקטע, evaluator נפרד עדיין נוטה להיות סלחני, אבל קל הרבה יותר לכוון אותו לספקנות



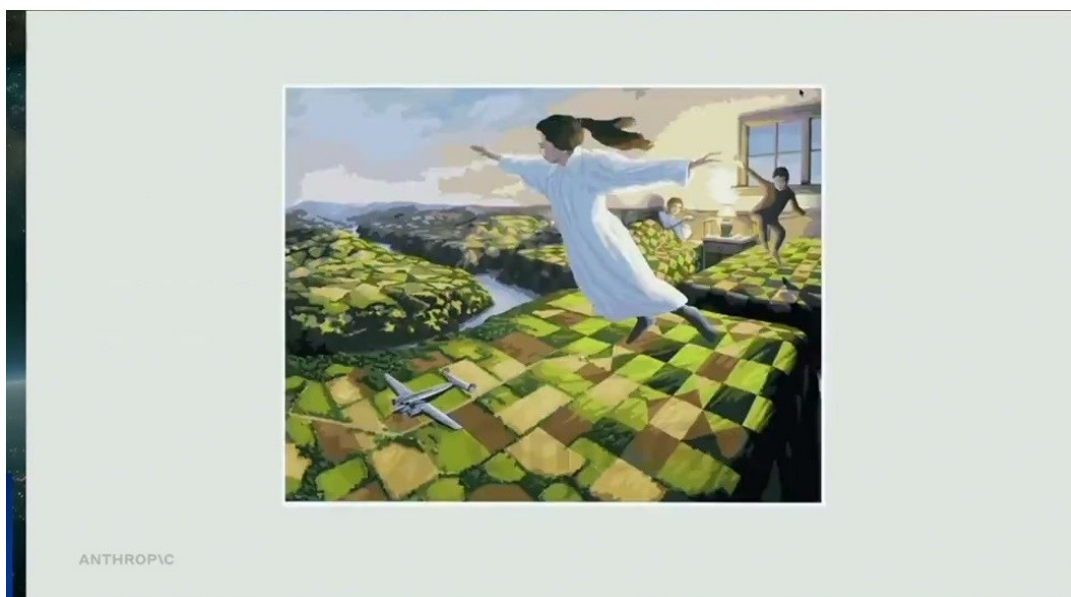
דיאגרמת הלולאה: Build context שולח Proposal ל-Judge context, וה-Judge context מחזיר signal ו-reward

בפועל: build agent ו-verifier agent עובדים בלולאה. ל-verifier יש goal או rubric, והלולאה ממשיכה עד שהוא מאשר. לפי הדובר, זה הרעיון שמאחורי כל טרנד ה-loops. הכלים: goal ב-Claude Code ו-outcomes ב-Managed Agents. בשניהם מגדירים מצב סיום מדיד, הבדיקה נעשית בקונטקסט עצמאי, ויוצאים מהלולאה רק אחרי שה-verifier אישר.

הניסוי: Parameter Golf, בנצ'מרק שפרסמה OpenAI ובודק יכולת מחקר ML. צריך לאמן מודל קטן על שמונה H100 בפחות מ-10 דקות, והציר האנכי בגרף הוא loss, כך שנמוך יותר עדיף. הדובר בנה לולאת verifier עם Managed Agents ו-outcomes והריץ אותה עם Opus 4.7 ועם מודל frontier שהוא כינה "mythos class" (השם לא ודאי). ה-outcome שהגדיר: 20 איטרציות בדיוק שעומדות בכל התנאים הניסויים של הבנצ'מרק. המסקנה שלו היא שמודלי frontier טובים מאוד בדפוס הזה. הוא לא נתן מספרים.

— הדובר, Anthropic [7:39] Because what happens is, instead of encoding steering into me as the human, you're encoding the signal into the environment

5. עיקרון שלישי: זיכרון in-band ו-dreaming



ציור סוריאליסטי של דמות שמרחפת מעל שדות. השקף פותח את הנושא של dreaming כתהליך offline שמגבש זיכרון

האנלוגיה היא למוח האנושי. במהלך היום ההיפוקמפוס רושם זיכרון קצר טווח (מה אכלת בצהריים). בלילה, בזמן חלימה, תהליך offline מעביר לזיכרון ארוך הטווח בקורטקס רק את מה שחשוב. אצל Claude, "memory tools" זה פשוט אפשרות לכתוב לתיקיית זיכרון במערכת הקבצים, "[10:42] That's really it".

שתי ראיות לכך שהכתיבה ה-in-band משתפרת מדור לדור. הראשונה, Claude Plays Pokemon: Sonnet 3.5, רשם הערות טקטיות "pretty crappy" והגיע לא רחוק, ו-4.6 רשם הערות אסטרטגיות והגיע הרבה יותר רחוק. השנייה, Continued Learning Bench, בנצ'מרק open source של הדובר (ייתכן שהשם הוא Continual Learning Bench). במשימה של מענה רציף על שאלות מול מסד SQL, עם כתיבת זיכרון בין שלב לשלב, הביצועים עולים עם כל דור.

— הדובר, Anthropic [12:22] And so basically, higher capacity models have a better sense of like what abstraction to save to memory that'll be useful later. Like, they're not just writing a specific fact. They're writing, how does this generalize to future sessions

הבעיה בכתיבה in-band: במהלך session אפשר לכתוב זיכרון שגוי, או זיכרון שעוזר למשימה הנוכחית אבל לא מתאים למשימות אחרות ("locally optimal, but not globally optimal" [12:22]). dreaming הוא תהליך out-of-band שעובר על מאגר הזיכרון ועל כל ה-traces הקודמים, מוצא טעויות ומתקן אותן.

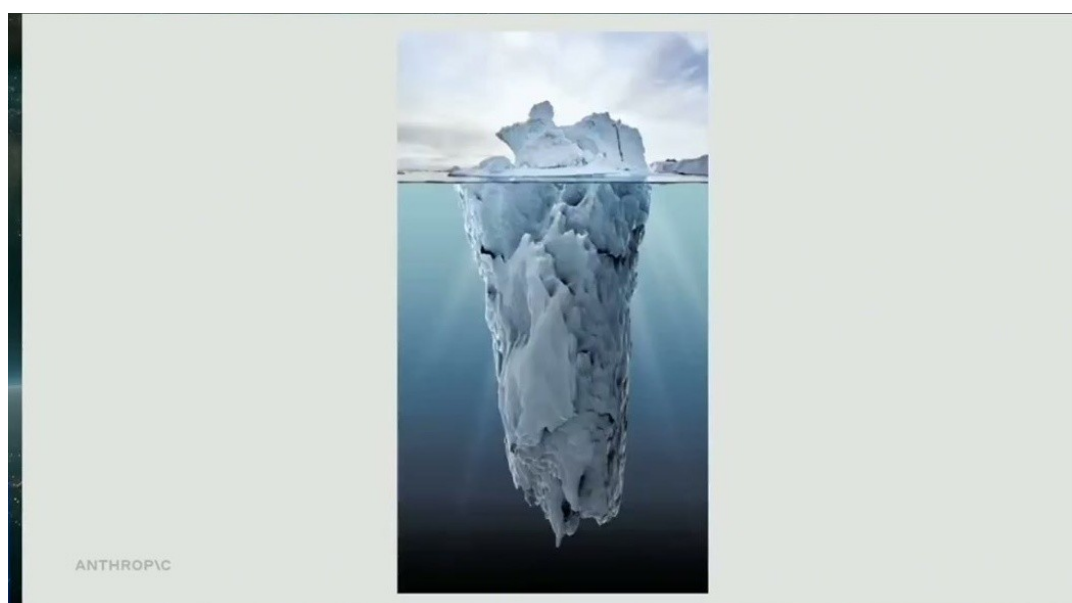


השוואה מ-Pokemon בין RAW MEMORY ל-MEMORY + DREAMING. משמאל הדמות נופלת ל-trap door בגלל זיכרון שגוי על המיקום, ומימין, אחרי dreaming, היא עוקפת אותה

— **הדובר**, Anthropic [13:53] So I saw this in five replicates, five out of five replicates with raw memory store, fell down this trap. With dreaming, this error is corrected

בגרף ההתקדמות שהוצג, ציר y עולה כשיש התקדמות לשלב הבא ויורד כשיש backtracking. ה-baseline בלי זיכרון (אפור) כמעט לא זז, כי זה שלב קשה במיוחד. הריצה עם זיכרון גולמי (כתום) נופלת שוב ושוב ל-trap door וחוזרת אחורה. ריצות ה-dreaming מתקנות את הטעות ועוברות שלב. השורה התחתונה של הדובר: "those mistakes get stuck in memory unless you have an offline process to kind of correct them" [15:25]

6. עיקרון רביעי: org-level harness



קרחון שרובו מתחת למים. השקף טוען שהממשק ב-Slack הוא רק החלק הגלוי של מערכת עמוקה בהרבה

הדובר מדבר על מוצר שהתמלול קורא לו "Cloud Tag". נראה שהכוונה ל-Claude עם תיוג ב-Slack, אבל השם המדויק לא ודאי. לדבריו, רוב התגובות היו "ah, Slack bot", והוא מודה שבנה בעצמו הרבה Slack bots גרועים. הייחוד לא נמצא ב-Slack. הייחוד הוא שזה org-level harness. עד היום סוכנים היו single player: Claude Code

על המחשב שלך, עם הקונטקסט המקומי שכיוונת לעצמך. כאן יש harness אחד שכל הארגון משתמש בו. יש לו זהות credentials משלו שלא קשורים למשתמש מסוים, ויש לו גישה לקונטקסט של הארגון.

מה יוצא מזה לפי הדובר: אפשר לבדוק עבודה של אחרים לפני שמריצים ניסוי, למנוע ממצאים כפולים, לעשות מחקר פנימי, ולתת לכל עובד harness מפותח מהיום הראשון. בלי זה, עובדים חדשים צריכים שבועות ואפילו חודשים כדי להגדיר את כל ה-connectors. עוד שני כיוונים שהוא צופה: סוכנים אסינכרוניים יהיו פרואקטיביים, למשל "alert me with things I might need to know about", ו-harness אחד יקבל הכוונה מהרבה אנשים במקביל, כלומר multiplayer [17:08].

— הדובר, Anthropic [17:08] So org-level harnesses are a real leveler of the playing field.

7. מה לעשות עם זה

- להוציא את ה-state מה-container: אם יש לכם סוכן שרץ שעות, שמרו את ה-session כ-event log חיצוני שאפשר רק להוסיף אליו, והפכו את ה-harness ל-stateless כדי שנפילה לא תמחק את העבודה.
- להוציא secrets מה-sandbox: ה-credentials צריכים לשבת ב-vault נפרד ולא בסביבה שבה המודל מריץ כלים.
- לבדוק אם אפשר לוותר על compaction הרסני: נסו לתת למודל לשלוח מה-log המלא במקום לסכם ולזרוק.
- לבנות verifier בקונטקסט נפרד: הגדירו מצב סיום מדיד עם goal ב-Claude Code או outcomes ב-Managed Agents, ותנו לקונטקסט עצמאי ומכוון לביקורת להחליט מתי יוצאים מהלולאה.
- להוסיף שלב offline לזיכרון: אם הסוכן כותב לתיקיית זיכרון, תזמנו תהליך שעובר על הזיכרון ועל ה-traces הקודמים ומתקן טעויות, לפני שזיכרון שגוי יחזור בכל הריצות.
- לחשוב על harness ארגוני: harness משותף עם זהות משלו וקונטקסט ארגוני מקצר את זמן הקליטה של עובדים חדשים ומאפשר למנוע עבודה כפולה.

שאלה פתוחה	למה זה חשוב
מה מפעיל את ה-dreaming ואיך מוודאים שהוא לא מכניס טעויות חדשות?	התיקון הוצג רק בתרחיש אחד ב-Pokemon עם חמש ריצות
מה העלות וה-latency של harness stateless מול session log חיצוני?	לא נמסרו מספרים, וזה קובע אם הארכיטקטורה כלכלית
מה יצא בפועל ב-Parameter Golf (loss) לכל מודל?	הוצג רק שהדפוס עובד, בלי ערכים להשוואה
איך verifier נמנע מסלחנות כלפי פלט של LLM?	השקף מהבלוג מודה שה-evaluator נוטה להיות generous
איך מנהלים הרשאות ופרטיות ב-org-level harness?	יש לו גישה לקונטקסט של כל הארגון וזהות שלא קשורה למשתמש
מה עושים כשהרבה אנשים מכוונים harness אחד ומבקשים דברים סותרים?	ה-UX של multiplayer הוזכר ולא פורט

8. מילון מונחים

מונח	פירוש
Task horizon	כמה זמן עבודה אוטונומית מודל מסוגל לבצע, נמדד בזמן שאדם היה צריך למשימה
Harness	הקוד שעוטף את המודל ומנהל את הלולאה, הכלים והקונטקסט. בהרצאה הוא נקרא brain
Sandbox / hands	container מבודד שבו רצים הכלים. ל-harness אחד יכולים להיות הרבה כאלה
Session / append-only event log	יומן אירועים שאפשר רק להוסיף אליו. משמש כ-state של הסוכן
Vault	מאגר נפרד ל-credentials, מחוץ ל-sandbox
Compaction	דחיסת הקונטקסט לסיכום. בדרך כלל זורקת את כל מה שלא נכנס לסיכום
Recursive language models	גישה שבה הקונטקסט הוא אובייקט חיצוני שהמודל שולף ממנו
Verifier / Judge	סוכן או קונטקסט נפרד שבודק את העבודה מול rubric

פירוש	מונח
הכלים להגדרת מצב סיום מדיד. Claude Code, Goal, Managed Agents, Outcomes	Goal / Outcomes
לולאת build ו-verify שממשיכה עד שה-verifier מאשר	Loop
זיכרון שהסוכן כותב במהלך המשימה	In-band memory
תהליך offline שעובר על הזיכרון ועל ה-traces ומתקן אותם	Dreaming
harness משותף לכל הארגון, עם זהות וקונטקסט ארגוניים	Org-level harness
harness אחד שהרבה אנשים מכוונים במקביל	Multiplayer
סוכן שפונה למשתמש ביוזמתו במקום רק להגיב	Proactive agent
שלוש שכבות ה-API של Anthropic: prompt ותשובה, harness של Claude Code, ו-harness עם תשתית מנוהלת	Messages API / Agent SDK / Managed Agents