

מתי לבנות agent harness משלכם

Harrison Chase מ-LangChain מסביר מה ה-harness עושה, איך מתאימים אותו עם middleware, ולמה evals ו-observability הם הבסיס ל-data flywheel

13 · Sequoia Capital · Harrison Chase באוגוסט 2026

משך: כ-24 דקות · YouTube

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה של כ-24 דקות מכנס Own Your Intelligence של Sequoia Capital. מרצה בה - Harrison Chase, co-founder ו-CEO של LangChain. בערך 17 דקות הן מצגת ודמו של LangSmith Engine, והשאר שאלות מהקהל. המנחה שהציגה אותו לא מזוהה בשמה בתמליל.

הטענה המרכזית: agent מורכב משלושה חלקים, model, harness ו-context, ומי שרוצה לשלוט ב-intelligence שלו צריך לשלוט בשלושתם. ההרצאה מתמקדת ב-harness. היא מסבירה מה התפקיד שלו, איך מתאימים את הלולאה הבסיסית בעזרת middleware, ומתי כדאי לבנות harness משלכם במקום להשתמש ב-Claude Code או ב-Codex. מכאן היא עוברת ל-evals ול-observability, שהם הבסיס ל-data flywheel שמשפר את ה-agent לאורך זמן.

איך לקרוא את המסמך

- 1. **תקציר מנהלים:** הטענות המרכזיות עם חותמות זמן, ומה לא נאמר.
- 2. **שלושת החלקים של model, context ו-agent:** harness, ומה ה-harness בעצם עושה.
- 3. **התאמת הלולאה:** DeepAgents, middleware ו-cognitive architectures.
- 4. **לבנות או לקחת מוכן:** in-distribution מול out-of-distribution, ודוגמת עריכת הקבצים.
- 5. **Harbor ו-Evals:** איך בנויה משימת eval, ומה מודדים מלבד accuracy.
- 6. **Observability, flywheel ו-LangSmith Engine:** מ-traces לשיפורים, כולל הדמו.
- 7. **שאלות ותשובות:** Engine על Engine, codexification, והשאלה אם ה-harnesses יתכנסו.
- 8-9. **המלצות ומילון מונחים:** מה לעשות עכשיו, שאלות פתוחות, ופירוש המונחים.

הערות מקור המסמך הופק אוטומטית מהתמליל ומהשקפים של ההקלטה. התמליל נוצר ב-ASR, ולכן שמות ומונחים עלולים להשתבש. למשל LangChain הופיע בתמליל כ-Claude Code, LinkChain כ-Cloud Code ו-Harbor כ-Harvard. במקומות כאלה תיקנו את השם לפי השקפים או סימנו אותו כלא ודאי. הציטוטים מועתקים מילה במילה ונבדקו מול חותמת הזמן שלהם. לא נוסף שום מידע ממקורות חיצוניים.

1. תקציר מנהלים

Chase מציג גישה מעשית: כמעט כל agent היום הוא אותה לולאה פשוטה של LLM שקורא ל-tools. ההבדלים בין agents נובעים ממה שמוסיפים מסביב ללולאה, וממידת ההתאמה בין המשימה למה שהמודל אומן עליו. כדי לדעת אם שינוי עזר צריך evals, וכדי להבין למה ה-agent נכשל צריך observability. שני אלה ביחד מאפשרים להפוך traces מ-production לשיפורים.

- **agent = harness + model + context**: לשליטה ב-intelligence צריך לשלוט בשלושתם. לגבי model, הוא מדגיש את היכולת להחליף מודלים כדי להימנע מ-lock-in, בדומה לגישת cloud agnostic של פעם [0:00, 1:31].
- **התפקיד של ה-harness**: להביא למודל את ה-context הנכון בזמן הנכון. הוא מתזמר context קבוע ודינמי, ומחזיר ללולאה את מה שמערכות חיצוניות פולטות [1:31].
- **כל ה-harnesses המתקדמים מריצים אותה לולאה**: LLM שקורא ל-tools. ההתאמה נעשית דרך middleware: summarization, context, before_agent, before_model, wrap_model_call, wrap_tool_call, sandbox [3:03, 4:34].
- **מתחילים מ-harness כללי**: זו הדרך המהירה ביותר להגיע לערך. אחר כך, ככל שמצמצמים את ה-use case, מוסיפים gates ובדיקות [6:07].
- **לבנות או לקחת מוכן תלוי במרחק מה-distribution**: ככל שהמשימה רחוקה יותר ממה שהמודלים אומנו עליו, צריך יותר התאמה. גם ב-harness מותאם כדאי להשאיר רכיבים שהמודל אומן עליהם, כמו עריכת קבצים, קרובים למימוש המקורי של אותו מודל [6:07, 7:40].
- **כל agent קריטי ידרוש benchmark**: Harbor, eval runner בקוד פתוח מהיוצרים של TerminalBench2, הופך לסטנדרט. מודדים בו accuracy, latency ועלות [9:11, 12:13].
- **רוב הכשלים נובעים מה-context, לא מהמודל**: לכן observability נשמעת בסיסית אבל לדבריו היא מוערכת בחסר [12:13].
- **ה-data flywheel**: מריצים agent, אוספים traces, אוצרים אותם ומריצים ניסויים. feedback מגיע מ-UX חכם או מ-online evaluators זולים, למשל SLMs שעברו [13:46, 15:17].
- **LangSmith Engine: agent** שרץ ברקע על ה-traces, פותח issues ומציע תיקונים ל-prompt, ל-context ולקוד של ה-harness [16:49, 18:22].

מה לא נאמר

- **אין מספרים על Engine**: לא הוצגו נתונים על הדיוק שלו, על כמה מההצעות שלו מתקבלות או על החיסכון בזמן.
- **החיסכון עם Harvey לא כומת**: נאמר שהעלות של LLM-as-a-judge ירדה "significantly" בעזרת SLMs, בלי מספר ובלי השוואת איכות.
- **אין קריטריון ל-out-of-distribution**: הכלל ברור, אבל לא הוצג אף מדד שעוזר להחליט מתי משימה רחוקה מספיק כדי להצדיק harness מותאם.
- **תוצאות ה-benchmarks לא נותחו**: שקף FrontierBench הוצג כדוגמה לממשק, בלי מסקנות לגבי harness או מודל ספציפיים.
- **אין סיפורי כישלון**: לא סופר מה נשבר ב-DeepAgents או ב-Engine בדרך, מלבד הלקח מ-Codex.

2. שלושת החלקים של agent ומה ה-harness עושה

To own your intelligence, you have to own all three parts... 🌸

Model

- Open weights
- Ability to switch (offensive and defensive)

Context

- Episodic (chat history), semantic (RAG), procedural (skills)

Harness...

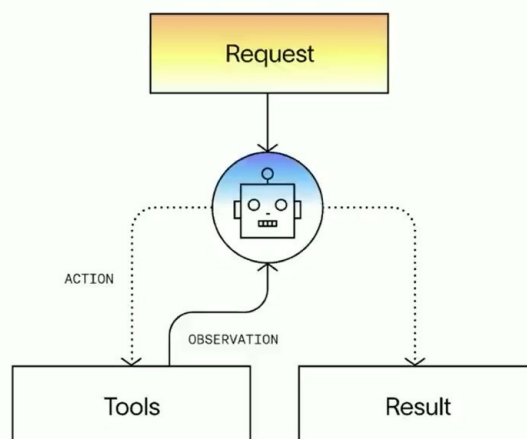
"To own your intelligence, you have to own all three parts" בצד ה-Model, open weights ויכולת החלפה (התקפית והגנתית). בצד ה-Context, episodic (היסטורית שיחה), semantic (RAG) ו-procedural (skills).

על המודל עצמו Chase כמעט לא מדבר. הוא מפנה להרצאה קודמת בכנס על open weight models של מרצה מ-Fireworks (השם בתמליל, "Lynn", לא ודאי). מה שחשוב לו הוא היכולת להחליף מודל, גם כדי להימנע מ-lock-in וגם כדי לעבור למודל הטוב ביותר ברגע שהוא יוצא. את ה-context הוא מחלק ל-memory, לידע סמנטי ולשיחות קודמות, וכולם משמשים להתאמה אישית ולהכוונה של ה-agent.

את התפקיד של ה-harness הוא מגדיר כך: "The main job of the harness is to bring context to the model" [1:31] "at the right point in time." שקף נוסף מפרט מה agent צריך לעשות: לפעול בסביבה שבה הוא יכול לבצע פעולות, להתחבר לנתונים, לנהל context שגדל לאורך ריצות ארוכות, להריץ משימות במקביל, לשלב human-in-the-loop ולהשתפר עם הזמן.

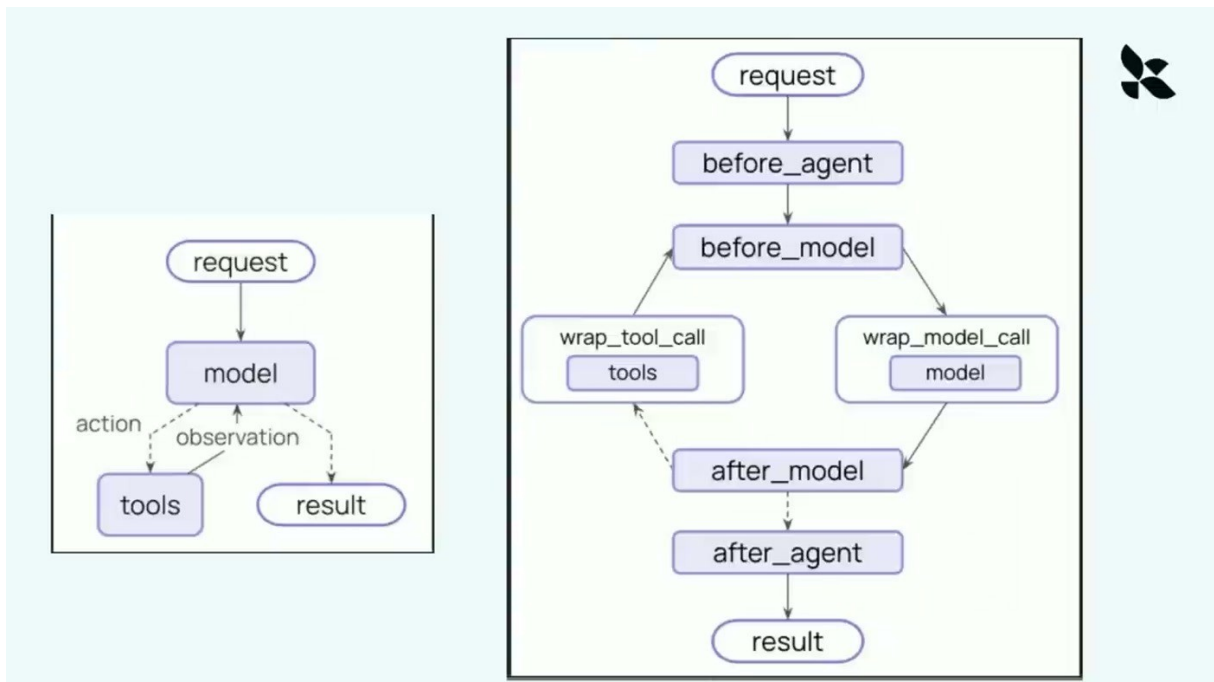
What is an agent?

A model calling tools in a loop until it completes a task and returns a result.



"What is an agent?" מודל שקורא ל-tools בזמנה עד שהמשימה מושלמת ומוחזרת תוצאה. Request נכנס, ה-agent מבצע action מול Tools, מקבל observation, ובסוף מחזיר Result.

3. התאמת הלולאה: DeepAgents, cognitive-middleware, architectures



משמאל הלולאה הבסיסית (request, model, tools, result). מימין נקודות ה-, before_model, before_agent, middleware: wrap_tool_call, wrap_model_call, after_model, after_agent

LangChain הוא harness מינימלי במכוון. DeepAgents, שלדברי Chase הוא "model agnostic and more", Claude Code [3:03], בנוי מעליו ומוסיף file systems, skills, sub-agents, הכול בעזרת אותם hooks. לדבריו גם מגנוני ה-hooks וה-plugins ב-coding agents עושים בעצם אותו דבר: לוקחים את הלולאה הבסיסית ומוסיפים נקודות התאמה.

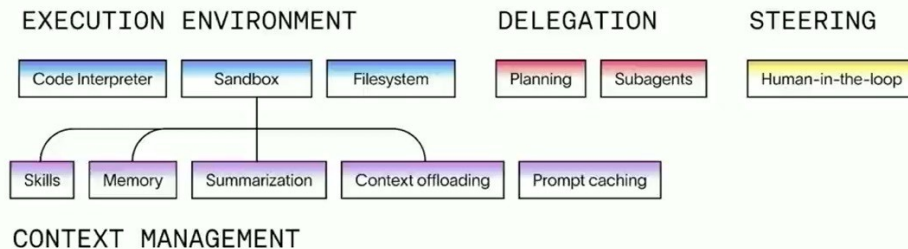
שתי דוגמאות קונקרטיות: summarization נכנס לפני קריאה למודל. בודקים אם ה-context ארוך מדי, ואם כן מסכמים אותו. context offloading עוטר את קריאת ה-tool ומוריד מה-context תוצאות גדולות [4:34]. אלה בדיוק הנקודות שבהן אפשר להכניס לוגיקה ייעודית לתחום שלכם.

Case study: deepagents on top of langchain



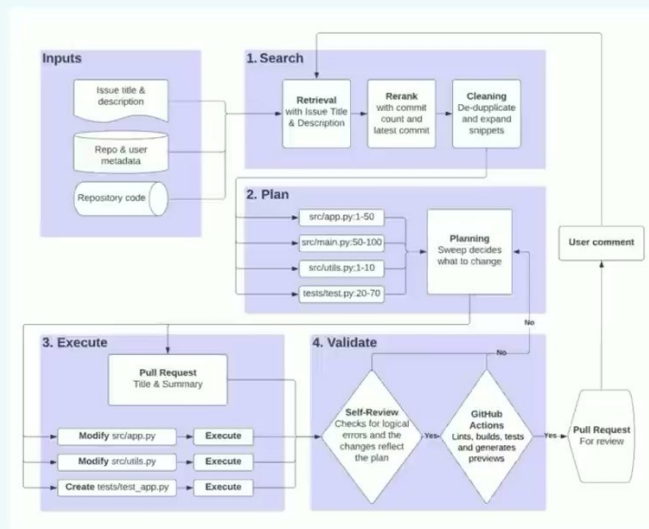
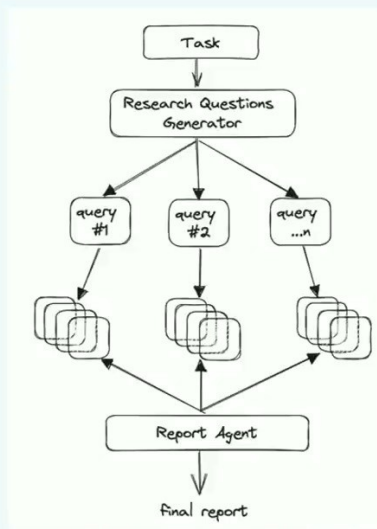
Base deepagents harness

ANATOMY OF A HARNESS



langchain. Execution environment (Code interpreter, Sandbox, Filesystem), האנטומיה של deepagents על גבי, Case study Context management (Skills, Memory, Summarization, -ו, Delegation (Planning, Subagents), Steering (Human-in-the-loop) .Context offloading, Prompt caching)

Custom cognitive architectures



Custom cognitive architectures: משמאל זרימת deep research (יצירת שאלות מחקר, fan-out לביצוע מקבילי, ואז Report Agent). מימין code review bot עם שלבי Search, Plan, Execute ו-Validate.

הדרך השנייה להתאים harness היא cognitive architecture מפורשת. כך בנו agents ב-2023–2024, כשהמודלים לא היו מספיק טובים כדי לרוץ לבד בלולאה. לדבריו חלק גדול מזה עבר היום לתוך ה-harness, כשינויים בלולאה הבסיסית, אבל עדיין משתמשים בזה בזרימות ספציפיות מאוד. ההמלצה שלו: "One thing that we recommend to people is to start with a general harness." [6:07] שרוצים להצטיין בו, מוסיפים gates ובדיקות.

4. לבנות או לקחת מוכן: in-distribution מול out-of-distribution

ה-harnesses המוכנים קשורים בדרך כלל למודלים מסוימים: Claude Code ו-Claude Agent SDK עובדים עם המודלים של Anthropic, ו-Codex עם המודלים של OpenAI. Chase מגדיר את השאלה הזו כשאלה פתוחה בתעשייה, ונותן כלל אצבע.

— Harrison Chase [6:07] the more in distribution you are of what the models are trained on, then the better the off-the-shelf harness will be. As soon as you start to move further and further out of distribution, then you'll probably want to tune your harness in some way

הדוגמה שלו היא legal AI, בהתייחסות למרצה קודם בשם Gabe (השם והחברה לא ודאיים, וההקשר מרמז על Harvey), שבנה harness משלו. המשימה הכוללת, משפט, רחוקה מה-distribution של המודלים. אבל עריכת קבצים היא בדיוק מה שכל המודלים עברו עליו RL, וכל אחד בדרך אחרת: מודלים של OpenAI ושל Claude עורכים קבצים בצורה שונה ב-harness שלהם, ולכן כל מודל הכי טוב בשיטה שלו.

המסקנה: harness מותאם למשימה הכוללת, אבל עם edit file tool שתואם למודל שבשימוש. ב-DeepAgents זה ממומש בעזרת model profiles, שמחליפים את מימוש עריכת הקבצים לפי המודל. בלשונו: "keeping smaller in" [7:40] "distribution parts as close to the model layer as possible".

5. Harbor ו-Evals



"Create your private evals, because evals define what "good" looks like inside the organization"

"retain ownership of your organization's memory, **traces, feedbacks, decisions**, and institutional context"

"you create your own continuous learning loop (i.e. hill climbing machine) that will allow your AI investments to compound the value of your firm"

שלושה ציטוטים מהמאמר של Satya Nadella, "The Reverse Information Paradox": evals מגדירים מה "טוב" בארגון, לשמור בעלות על memory, traces, feedbacks ו-decisions, וליצור continuous learning loop שגורם להשקעות ה-AI לצבור ערך.

Chase טוען ש-observability ו-evals חיוניים לכל אחד משלושת החלקים, לא רק ל-harness. לדבריו כל חברה שבונה mission critical agent תבנה לו benchmark, גם כדי לתפוס רגרסיות וגם כדי לעשות hill climbing, בין אם על ידי שינוי ה-harness ובין אם על ידי שינוי המודל [9:11].

הכלי שהוא רואה כסטנדרט המתהווה הוא Harbor, eval runner בקוד פתוח מהיוצרים של TerminalBench2. כדוגמה הוא מציג את benchmark FrontierBench, נוסף ל-coding, שבו משווים harnesses, מודלים ו-reasoning efforts על אותה משימה [10:43].

Datasets > frontier-bench > frontier-bench

frontier-bench/frontier-bench

The frontier is wide and dynamic. Frontier-Bench measures general agent capabilities on a diverse set of difficult tasks.

Published 7/22/2026 by Kevin Xiang Li

+ New Job [rev. 3 \(v0.1.0, latest\)](#)

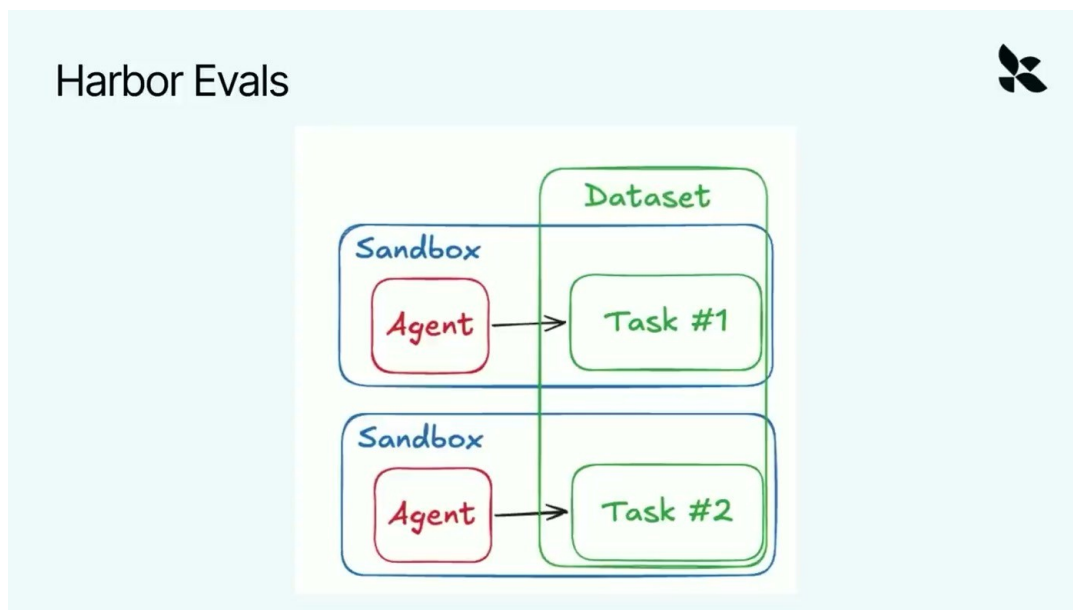
```
harbor run -d frontier-bench/frontier-bench
```

README Tasks **Leaderboard**

FRONTIER-BENCH V0.1

#	Agent	Model	Reasoning Effort	Accuracy	Release Date	Agent Org	Model Org	Tokens
1	mini-SWE-agent	Opus 5	max	43.5% ± 1.7%	Jul 24, 2026	Princeton	Anthropic	
2	Codex	GPT-5.6 Sol	max	32.7% ± 1.6%	Jul 9, 2026	OpenAI	OpenAI	
3	Claude Code	Fable 5	max	31.7% ± 1.7%	Jun 9, 2026	Anthropic	Anthropic	
4	Claude Code	Opus 4.8	max	21.1% ± 1.6%	May 28, 2026	Anthropic	Anthropic	
5	Codex	GPT-5.6 Terra	max	20.8% ± 1.4%	Jul 9, 2026	OpenAI	OpenAI	
6	Cursor CLI	Grok 4.5	xhigh	15.7% ± 1.5%	Jul 8, 2026	Cursor	xAI	
7	Claude Code	Sonnet 5	max	14.6% ± 1.5%	Jun 30, 2026	Anthropic	Anthropic	

דף ה-leaderboard של frontier-bench: עמודות Accuracy, Reasoning Effort, Model, Agent, תאריך וחברה, עם שורות ל-Claude Code ול-Codex על מודלים שונים.



Dataset של Harbor Evals: משימות, שכל אחת רצה ב-Sandbox נפרד מול (Task #1, Task #2) Agent.

משימת Harbor בנויה מארבעה רכיבים [12:13, 10:43]:

environment: הסביבה שבה ה-agent רץ, מוגדרת ב-Dockerfile ומורצת ב-sandbox. זה חשוב ל-agents ארוכים ו-stateful שצריכים לעבוד מול סביבה.

solution: פתרון golden לבדיקת שפיות. Chase מציין שזה "not that interesting".

tests: ה-verifier וה"עיקר" לדבריו. סקריפט שיכול להריץ קוד, LLM as a judge, unit tests או אפילו agent as a judge.

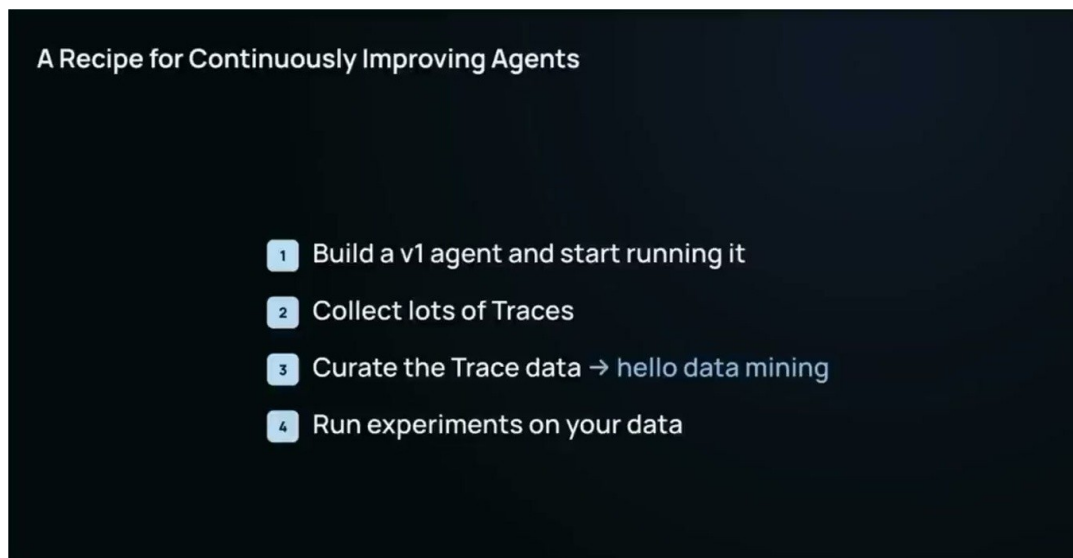
instruction.md: ה-prompt שה-agent מקבל.

את התוצאות מציגים ב-LangSmith, שלו יש אינטגרציה עם Harbor. מעבר ל-feedback score, שבדוגמה הוא reward function אחת, עוקבים גם אחרי latency ו-"you probably don't just care about accuracy. You also probably care about latency and cost." [12:13]

6. Observability, ה-data flywheel ו-LangSmith Engine

Chase מסביר כאן למה observability חשובה. כש-agent נכשל, זה תמיד בגלל שקריאה ל-LLM השתבשה, ויש לכך שתי סיבות אפשריות: המודל לא מספיק טוב, או שה-context שהוא קיבל לא מספיק טוב. "And so I actually think it's the second one that more often than not causes issues." [12:13] לכן צריך לראות מה נכנס ל-context window, איך ה-context הצטבר ואילו tools רצו.

ב-LangSmith יש שתי תצוגות. הראשונה היא תצוגת trajectory ידידותית בסגנון Claude Code: רשימת ההודעות, כשקריאות tool מקופלות ("seven tool calls"). השנייה היא trace מלא, שבו אפשר להיכנס לכל צעד ולראות בדיוק מה נכנס למודל. לדבריו ה-trajectory לבדה "not enough to fully debug it" [13:46]



"A Recipe for Continuously Improving Agents": בונים agent v1 ומתחילים להריץ, אוספים הרבה traces, אוצרים את ה-trace data ("hello data mining"), ומריצים ניסויים על הנתונים.

המתכון הוצג במקור, לדבריו, בכנס AI Engineer (השם בתמליל משובש). את ה-feedback, שהוא המרכיב הקריטי, אפשר לקבל משני מקורות. הראשון הוא ה-UX: "They may not click thumbs up or thumbs down explicitly. No one really is doing that." [15:17] ולכן צריך לתכנן את הממשק כך שיפיק feedback עקיף. השני הוא online evaluators סינתטיים. הרצת Opus על כל trace יקרה מדי, ולכן LangChain עשו fine-tuning ל-SLMs לתפקיד הזה. אפשר גם להשתמש במודל מוכן עם custom prompting, או בקוד פשוט לבדיקות פשוטות.

את הנתונים שנאספו אפשר להפנות לכל אחד משלושת החלקים: harness engineering ל-fine-tuning harness, ל-memory ל-LangSmith Engine context, שהוצג בדמו, מנסה להפוך את זה לאוטומטי. זה coding agent שרץ ברקע עם גישה ל-LangSmith CLI (סינון traces לפי feedback וכדומה), עם prompt גדול ו-sub-agents. הוא מחפש דפוסים, פותח issues עם תיאור וקישורים ל-traces, ומציע תיקונים: שינוי prompt, עדכון הוראות והוספת קוד ל-Chase. harness מסכם את זה כ-"run agent, get traces, see patterns, fix." [18:22]

7. שאלות ותשובות: Engine על Engine, ולאן הולכים harnesses

לשאלה אם הריצו את Engine על עצמו: כן. Engine מחובר ל-Slack ושולח דוחות על עצמו. בנוסף נבנה Issue benchmark, Bench בפורמט Harbor, שעליו משווים מודלים ו-harnesses. כשהריצו עליו את DeepAgents, Claude Code ו-Codex, ראו ש-Codex כותב לעצמו באגרסיביות הרבה סקריפטים קטנים שרצים על ה-traces, ושה משפר את הביצועים שלו. "So we did a sprint to do what we call the kind of like codexification of"

[19:53] "engine". זו לדבריו עוד תועלת של benchmark: אפשר ללמוד מ-harnesses מתחרים ולהכניס את הלקח ל-harness שלכם.

לשאלה אם ה-harnesses יתכנסו לאחד או יתפצלו, התשובה הכנה שלו: "I don't know is the honest answer." [19:53]. ה-harnesses הכלליים כבר מספיק טובים כדי להתחיל, ולכן הוא ממליץ לפתוח עם DeepAgents, Codex או Claude Code. זה ספקטרום: בצד אחד harness מוכן, באמצע hooks ו-middleware, ובקצה השני cognitive architecture מלאה. סיבה נוספת לבחור בקצה הזה היא predictability ושליטה. לקוחות מ-financial services שראו את DeepAgents הגיבו "[21:25] 'whoa, whoa, whoa, that's way too scary an agent for us.' לגבי המעבדות, הוא מעריך שהן מתכנסות לכיוון coding, אבל יש ביניהן הבדלים קטנים ולא מוסברים, כמו עריכת קבצים. אם מעבדה אחת תתמקד, למשל, ב-bio, גם ה-harnesses יתחילו להתפצל." [22:56] "It's a fast-moving space."

8. מה לעשות עכשיו

- **להתחיל עם harness מוכן:** Claude Code, DeepAgents או Codex. לעבור להתאמה רק כשה-use case מצמצם וברור שהמשימה רחוקה מה-distribution של המודל.
- **להתאים דרך middleware לפני שבונים מאפס:** summarization לפני המודל, context offloading סביב tools, ו-gates ייעודיים לתחום.
- **להשאיר רכיבים in-distribution קרובים למודל:** edit file tool בפורמט שהמודל אומן עליו, ובחירת המימוש לפי מודל (בדומה ל-model profiles).
- **לבנות benchmark פרטי בפורמט Harbor:** environment ב-Docker, tests כ-verifier, ו-instruction.md. למדוד accuracy, latency ועלות, ולהריץ עליו גם harnesses מתחרים כדי ללמוד מהם.
- **להשקיע ב-observability ברמת ה-context:** לראות מה בדיוק נכנס ל-context window בכל קריאה, לא רק את ה-trajectory.
- **לתכנן את ה-UX כך שיפיק feedback:** ולהוסיף online evaluators זולים (prompt, SLM, או קוד) במקום להריץ מודל frontier על כל trace.

שאלה פתוחה	מה נאמר בהרצאה
איך מודדים כמה משימה רחוקה מה-distribution?	רק כלל אצבע איכותי, בלי מדד או סף.
האם ה-harnesses יתכנסו לאחד או יתפצלו?	"I don't know". ההימור שלו: התכנסות סביב coding, ופיצול אם מעבדות יתמחו בתחומים שונים.
כמה אמינות ההצעות של LangSmith Engine?	הוצג דמו בלבד, בלי שיעורי קבלה או דיוק.
מה התמורה בפועל של SLM כ-online evaluator?	נאמר שהעלות ירדה משמעותית בניסוי עם Harvey, בלי מספרים או השוואת איכות.
למה OpenAI ו-Antropic עורכים קבצים בצורה שונה, ואיזו שיטה עדיפה?	גם Chase לא יודע להסביר. הוא ציין שמישהו שדיבר איתו חשב ששיטה אחת עדיפה בבירור.

9. מילון מונחים

מונח	פירוש
Harness	הקוד שמתזמר את הלולאה סביב המודל ומביא לו את ה-context הנכון בזמן הנכון
Middleware / Hooks	נקודות התאמה בלולאה (before_model, wrap_tool_call וכדומה) שמאפשרות להוסיף לוגיקה בלי לשנות את הלולאה
Summarization	סיכום ה-context כשהוא ארוך מדי, לפני הקריאה למודל
Context offloading	הוצאה של תוצאות tool גדולות מה-context, למשל לקובץ
Sub-agents	agents משניים שה-agent הראשי מאציל להם משימות
Cognitive architecture	זרימה מפורשת ומוגדרת מראש של שלבים, במקום לולאה חופשית
In-distribution / Out-of-distribution	משימה שדומה או רחוקה ממה שהמודל אומן עליו
Model profiles	מנגנון ב-DeepAgents שבוחר מימוש tool (למשל עריכת קבצים) לפי המודל
Benchmark / Evals	אוסף משימות ומודדים להערכת agent, לתפיסת רגרסיות ולשיפור
Hill climbing	שיפור איטרטיבי מול benchmark קבוע

פירוש	מונח
eval runner בקוד פתוח מהיוצרים של TerminalBench2	Harbor
הרכיב שנותן ציון לריצה, בקוד או בעזרת מודל	Verifier / LLM as a judge
רשימת ההודעות וקריאות ה-tool של ריצה	Trajectory
תיעוד מלא של ריצה, כולל מה בדיוק נכנס לכל קריאה למודל	Trace
בודקים אוטומטיים שרצים על traces מ-production	Online evaluators
Small language model, מודל קטן וזול, כאן לתפקיד evaluator	SLM
מחזור של traces, אוצרות, ניסויים ושיפור, שחוזר על עצמו	Data flywheel
שיפור ה-harness עצמו על סמך נתונים	Harness engineering