

80% Boris Cherny: מחקנו מה-prompt של Claude Code

שיחה עם יוצר Claude Code ב-Startup School 2026: מה Opus 5 משנה, למה מוחקים את ה-harness בכל דור מודל, ואיך מריצים אלפי agents

Boris Cherny, Diana Hu · Y Combinator · 27.07.2026

משך: כ-36 דקות · YouTube

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו שיחה על הבמה ב-Startup School 2026 של Y Combinator. היוצר של Claude Code, Anthropic, משוחח עם Diana Hu מ-YC. השיחה הוקלטה יום אחרי השקת Opus 5. אין בה שקפים, רק שאלות ותשובות. השיחה עוברת על היכולות החדשות של המודל, על ההחלטה למחוק יותר מ-80% מה-system prompt של Claude Code, על הרעיון של product overhang, ועל הדרך שבה הצוות מריץ היום אלפי agents בבת אחת.

הטענה המרכזית של Cherny פשוטה: בכל דור מודל, חלק גדול מה-harness (ה-prompts, הכלים וההוראות) מפסיק לעזור ומתחיל להפריע. לכן צריך למחוק באופן קבוע, להחזיר הוראות רק אחרי שרואים את המודל נכשל שוב ושוב באותו מקום, ולתת לו משימות קשות יותר יחד עם דרך לבדוק את העבודה של עצמו. לדבריו, העבודה עם מודלים הפכה למדע אמפירי ולא תיאורטי.



Diana Hu (משמאל) ו-Boris Cherny על הבמה ב-Startup School 2026. בשיחה לא הוצגו שקפים.

איך לקרוא את המסמך

- **תקציר מנהלים:** הטענות העיקריות עם חותמות זמן, ומה לא נאמר בשיחה.
- **פרק 2, Opus 5:** ריצות של ימים ושבועות, ושלוש שכבות ההגנה מפני prompt injection.
- **פרק 3, מחיקת ה-prompt:** ablation, מצב simple, ואיך בונים prompt מחדש.
- **פרק 4, product overhang:** הסיפור של Claude Code עצמו, השכתוב של Bun מ-Zig ל-Rust, וציוור עם OpenCV.
- **פרק 5, אימות במקום prompt engineering:** משימת שכתוב ה-desktop ל-Swift שרצה כבר שבועיים.
- **פרק 6, אלפי agents:** dynamic workflows, loops ו-routines שמתחזקים את הקוד של Claude Code עצמו.
- **פרק 7, קידוד כמעט פתור:** איפה המודל עדיין נכשל, ומה כדאי לסטודנטים ל-CS ללמוד.
- **פרקים 8 ו-9:** פעולות מומלצות, שאלות פתוחות ומילון מונחים.

הערת מקור המסמך נוצר אוטומטית מהתמלול של ההקלטה ומהפריימים שלה. בשיחה אין שקפים. ה-ASR שיבש שמות רבים: Claude Code הופיע בתמלול כ-Cloud Code, Plot Code או quad code, ו-Sonnet 3.5 הופיע כ-SANA 3.5. כמה שמות לא ודאיים ומסומנים ככאלה בטקסט: מודל בשם Fable, המוצר quad tag (כנראה Claude ב-Slack), חוקר הפרשנות Chrysola (כנראה Chris Olah), Jared מצוות Bun (כנראה Jarred Sumner) ושם משתנה הסביבה של מצב simple. הציטוטים מועתקים מילה במילה מהתמלול ונבדקו מול חותמת הזמן שלהם, ולכן הם משמרים גם את שיבושי ה-ASR. לא נוסף מידע ממקורות חיצוניים.

1. תקציר מנהלים

Cherny מציג את Opus 5 כקפיצה בשלושה תחומים: ריצה ארוכה ועצמאית, עמידות ל-prompt injection, וחשיבה שמיתרת חלק גדול מההוראות שנכתבו עבור מודלים קודמים. מזה הוא גוזר שיטת עבודה למי שבונה מוצרי AI: למחוק, למדוד, להחזיר רק את מה שבאמת חסר, ולהשקיע ב-verification וב-orchestration במקום בהוראות מפורטות.

- **ה-system prompt קוצץ ביותר מ-80%:** לדבריו, רוב ההוראות תיקנו התנהגויות שהמודל היה אמור לדעת לבד, ו-Opus 5 פשוט עושה אותן נכון [3:00]. במצב simple, בלי שום prompt, המודל אפילו מעט חכם יותר [4:33].
- **Ablation בכל דור מודל:** מוחקים את כל ה-prompt ומחזירים אותו שורה אחר שורה כדי למדוד את התרומה של כל שורה. כך עושים גם עם כלים וקוד [4:33], [6:09].
- **ההמלצה למשתמשים:** פעם בחצי שנה למחוק את CLAUDE.md, את ה-skills ואת ה-hooks ולבדוק מה המודל עושה בלעדיהם [6:09].
- **Prompt injection:** לטענתו, השילוב של מודל מיושר, classifier מבוסס interpretability שרץ על כל התעבורה ו-classifier של auto mode הגיע למצב שבו הם כבר לא מצליחים להדגים [3:00], [1:30] prompt injection.
- **Product overhang:** למודלים של היום יש יכולות שאף מוצר עוד לא מנצל. Claude Code עצמו נולד מזיהוי כזה בתקופת Sonnet 3.5 [10:47], [12:18].
- **השכתוב של Bun מ-Zig ל-Rust:** אחד, dynamic workflow ו-11 ימי ריצה עם steering. לדבריו זו עבודה של יותר משנה למהנדסים, והתוצאה כבר ב-production בתוך [16:53] Claude Code.
- **Verification היא המיומנות החשובה:** משימה שנראית קשה מדי, ודרך שבה המודל יכול לבדוק את עצמו. משימת השכתוב ל-Swift רצה כבר יותר משבועיים [19:57], [21:28].
- **Routines מתחזקים את הקוד:** 20 עד 30 routines יומיים על ה-iOS, Android, CLI, desktop, כולל ניקוי dead code ואיחוד של abstractions כפולות [27:36], [29:06].
- **קידוד פתור, אבל לא לכולם:** לא עבור קוד מערכות עמוק, מערכות מבזרות ואימות UI ברמת הפיקסל [30:38].

מה לא נאמר

- **אין מספרים על prompt injection:** נאמר שהם לא מצליחים להדגים מתקפה, אבל לא הוצגו eval, שיעור הצלחה או בדיקה של red team חיצוני.
- **80% ממה?** לא צוין אם מדובר בטוקנים, בשורות או בתוכן, ולא הוצגו מדדי איכות לפני המחיקה ואחריה.
- **אין נתוני עלות:** לא נאמר כמה עלו ריצה של 11 ימים, ריצה של שבועיים או אלפי agents ביום.
- **מספר ה-agents לא ידוע:** Cherny אמר שהוא לא יודע כמה agents רצו במשימת ה-Swift, וההערכה שלו היא אלפים עד עשרות אלפים [24:42].
- **איכות השכתוב של Bun:** המשפט "over 100,000" נקטע ללא יחידה. לא נאמר כמה steering נדרש ולא נמסרו נתונים על באגים אחרי המעבר ל-Rust.
- **אין סיפורי כישלון:** לא תוארה אף ריצה ארוכה שנכשלה, וגם לא מה קורה כשה-verification עצמה שגויה.

2. מה Opus 5 עושה אחרת: ריצות ארוכות ו-prompt injection

השיחה נפתחת בנתון שמביאה המראיית: Opus 5 הגיע ל-30% ב-ARC-AGI, לעומת ספרות בודדות או עשרות נמוכות קודם לכן. Cherny מדגיש שבאימון לא כל מה שמנסים ללמד את המודל נקלט, ושלפעמים מופיעות יכולות שאף אחד לא לימד. היכולת הראשונה שהוא מציין היא ריצה ארוכה. בשילוב עם auto mode המודל יכול לעבוד "for days, weeks, months at a time", ולדבריו בלי scaffolding כמו [1:30] goal.

היכולת השנייה היא עמידות ל-prompt injection. לפי Cherny זה תהליך שנמשך מאז Opus 4.7 ו-Opus 4.8, ו-Opus 5 קובע בו רף חדש. ההגנה בנויה משלוש שכבות. הראשונה היא מודל מיושר, תוצאה של שלוש שנות מחקר alignment. השנייה היא classifier ל-prompt injection שרץ על כל התעבורה ומבוסס על מחקר mechanistic interpretability (השם בתמלול הוא Chrysola, כנראה Chris Olah): מזהים נירונים שנדלקים כשמתרחש injection, גם אם המודל עצמו לא מדווח על כך. השלישית היא ה-classifier של auto mode. לדבריו, להגנה הזו יש השלכות ישירות על התכנון של harness, של agents ושל מוצרים, בגלל ה-lethal trifecta.

Boris Cherny [3:00] And with these three layers, we just cannot demonstrate prompt injection — anymore.

3. למה נמחקו 80% מה-system prompt ואיך בונים אותו מחדש

לפי Cherny, Claude Code משתנה כל הזמן. עם כל מודל חדש מוחקים ומשנים חלקים מה-system prompt, מסט הכלים ומה-prompts של הכלים, כי "every model is very different". ב-Opus 5 רוב ה-prompt תיקן התנהגויות שהמודל כבר יודע לבד, ולכן נמחקו יותר מ-80% ממנו [3:00].

אפשר להמשיך את הניסוי לבד: הדגל --system-prompt מחליף את ה-prompt, ויש גם מצב simple לא מתועד שמופעל דרך משתנה סביבה (השם בתמלול הוא Cloud Code simple equals one, והשם המדויק לא ודאי). המצב הזה מוחק את כל ה-prompts, כולל אלה של הכלים. הצוות משתמש בו כ-ablation. הממצא שלהם: "the model is actually a little bit more intelligent without these prompts" [4:33]. Claude Code כמוצר, חלק מה-prompts עדיין נדרשים כדי שהמודל יתנהג כמו שאדם מצפה. לדבריו, רוב הקוד שנשאר היום ב-harness עוסק ב-safety, ב-permissions וב-static analysis, ולצדו קוד UI [6:09].

Boris Cherny [6:09] every six months, delete your Cloud MD. Delete your skills. Delete your — hooks. See what the model does, and it might surprise you

השיטה לבנייה מחדש: קודם מוחקים, אחר כך משתמשים. לא מנחשים איזו הוראה תידרש, כי לרוב הניחוש שגוי. מריצים את המוצר, רואים איפה המודל נכשל ואיפה הוא מצליח, ומחזירים הוראה רק אחרי שרואים אותו נכשל שוב ושוב באותו מקום. הנימוק: "the model is going to read this instruction every single time you use it" [7:42]. Cherny משווה את זה להנדסה הקלאסית, שבה re-architecture הוא פרויקט של חודשים ושנים, ומתאר את המודל כ-"almost like a living creature": לכל דור יש אישיות קצת שונה, וצריך להכיר אותו מחדש.

גם ל-evals יש תאריך תפוגה. הם שורדים יותר מה-harness, אבל לדבריו רק דור עד שלושה דורות של מודלים. אחרי שהמודל רווה את ה-eval, זורקים אותו ובונים חדש מהנקודות שבהן המודל עדיין נתקע [9:13].

4. Product overhang ו-unhobbling: לתת ל-Claude משימות קשות יותר

Cherny מגדיר שני צדדים של אותה תופעה. Product overhang הוא מצב שבו למודל של היום יש יכולת שאף מוצר עוד לא מנצל. Hobbling הוא מצב שבו המוצר מפריע למודל [10:47]. Claude Code עצמו נולד מזיהוי של overhang בתקופת Sonnet 3.5, לפני שנה וחצי עד שנתיים. כלי הקוד של אז הציעו autocomplete של שורה אחת או כמה שורות, וצ'אט עם הרשאות קריאה בלבד. ההימור היה שהמודל מסוגל לכתוב קובץ שלם בבת אחת, ולכן נבנה "the simplest possible harness" [12:18]. לטענתו, היום יש הרבה overhang שסטארטאפים לא מנצלים.

ההמלצה המעשית: לתת משימה קצת יותר ממה שנראה אפשרי, ולא להכתיב שלבים. במקום רשימה של אחת, שתיים, שלוש, ארבע, מתארים את המשימה, את ה-guardrails ואת תנאי היציאה, ונותנים למודל לעבוד [15:20].

פרמטר	מה נאמר על השכתוב של Bun
מקור ויעד	Bun, ה-runtime של JavaScript ש-Claude Code רץ עליו, נכתב ב-Zig ושוכתב ל-Rust
הבעיה	ב-Zig מנהלים זיכרון ידנית, ולכן היו memory leaks. Claude עשה fuzzing ומצא אותם אחד-אחד
מי	Jared מצוות Bun (השם לפי התמלול). הוא ניסה את המשימה על כל דור מודל, וזה התחיל לעבוד מ-Fable (שם לא ודאי)
איך	test suite קיים של Bun ושל Node.js כמנגנון אימות, אחד prompt ו-dynamic workflow
משך	11 ימים, עם steering. לא נעשה ב-one shot
הערכה ידנית	לדבריו, "Definitely over a year" של מהנדסים

פרמטר	מה נאמר על השכתוב של Bun
סטטוס	ב-production. זה ה-runtime ש-Claude Code משתמש בו היום

— And it ran for 11 days. And it rewrote the entire code base [16:53] Boris Cherny.

הדוגמה השנייה היא משחק חופשי. בתוך Anthropic התגלה שאם נותנים ל-Opus 5 את OpenCV, הוא יודע לצייר פורטרטים, חיות ונופים, למרות שאף אחד לא אימן אותו לצייר. Cherny מעריך שיש עשרות עד מאות פערי elicitation כאלה שעוד לא התגלו [18:25]. העצה שלו: לזרוק את המודל החדש על כל בעיה עסקית שמודל קודם לא הצליח לפתור.

5. Prompt engineering משתנה: אימות, ומשימה שרצה שבועיים

לדברי Cherny, המיקוד עבר מ-prompt engineer ל-context engineer, ומשם הלאה. המיומנות היום היא לתת ל-Claude משימה שנראית קשה מדי, ולאפשר לו לאמת את העבודה תוך כדי. לדבריו, ב-verification רוב האנשים טועים [19:57].

הדוגמה: אפליקציית ה-desktop של Claude כתובה ב-Electron, וכניסוי Cherny רצה לראות איך היא תרגיש כאפליקציה native. הוא פתח session של quad tag (כנראה Claude שרץ ב-Slack, השם לא ודאי), חיבר runner של macOS ב-GitHub כדי שהמודל יוכל להרים מכונה וירטואלית של Mac, ונתן גישה ל-repo ריק של גרסת Swift. ההוראה הייתה לשכתב את האפליקציה, להריץ את גרסת ה-Electron במכונה הווירטואלית, לצלם מסך ולהשוות פיקסל אחר פיקסל לגרסת ה-Swift, ואז: "Don't stop until you're done". בזמן השיחה המשימה רצה כבר 14 עד 15 ימים. Claude גם החליט ביוזמתו לפתוח ערוץ Slack ולפרסם בו צילומי מסך של ההתקדמות כל כמה דקות [21:28].

— But really all you need is give the model the task, give it a way to verify the output of its work so it doesn't get stuck and it'll just go Boris Cherny [21:28].

לשאלה איך להגיע לרמה של משתמש מוביל, התשובה של Cherny היא "don't listen to the LinkedIn influencers", כי "one weird trick" לא קיים [23:01]. השיטה שלו: לתת משימה קשה ואת הכלים לבדוק אותה, לראות איפה המודל נתקע, ולתקן באמצעות prompt, skill או MCP שמביא את ה-context החסר. לדבריו, ה-failure mode הנפוץ אצל מהנדסים ותיקים הוא over-specification, כלומר ניסיון לגרום למודל לעשות את העבודה בדיוק כמו שהם היו עושים אותה.

6. אלפי routines, dynamic workflows, loops ו-routines

הדרך הקלה להריץ אלפי agents היא dynamic workflows. מספיק לכתוב "use a workflow" [24:42]. מתחת למכסה המנוע, Bun משמש sandbox שבתוכו עולה מכונה וירטואלית, ו-Claude מתזמר בה agents בשלבים: fan-out לעבודה הראשונית, שלב של agents שמאמתים או מסכמים, ואחריו fan-out נוסף. Cherny, שהרקע שלו בתכנות פונקציונלי, מתאר את התכנון כ-"essentially an algebra for agents": יש צורה להריץ agents ברצף וצורה להריץ אותם במקביל. לטענתו זה ציר חדש של test-time compute, נוסף על גודל הרשת, נתוני האימון וה-FLOPs באימון [26:05].

הדרך השנייה היא loops ו-Loop routines. routine הוא cron מקומי, ו-routine הוא אותו דבר בענן, כך שאפשר לסגור את המחשב. ההבדל מ-workflow: כאן מדובר במשימה חוזרת שלא חולקת context בין הרצות, אבל יכולה לחלוק memory [27:36]. ב-Claude, Anthropic מתחזק כך את הקוד של עצמו.

Routine	מה היא עושה (לפי השיחה)
ניקוי dead code	prompt של משפט אחד שרץ כל יום. Claude בחר לבד ב-static analysis ו-dynamic analysis ופותח PR יומי
סגירת ניסויים	ניסוי שכבר נפתח ל-100% מוסר מהקוד ונשלח
כתיבת טסטים	מוסיפה כיסוי לאזורים בלי טסטים
מחיקת טסטים	מסירה טסטים חסרי ערך שכתבו מודלים קודמים או אנשים

מה היא עושה (לפי השיחה)	Routine
מאתרת abstractions כמעט כפולות בכל ה-codebases ומאחדת אותן	abstraction police

Boris Cherny [29:06] And so now we have every day maybe 20 or 30 of these routines. It's —
running across all of our code bases

ה-routines רצים על ה-CLI, על אפליקציות iOS ו-Android ועל ה-desktop. לדבריו מדובר במאות agents ביום, לפעמים באלפים, שעושים את העבודה של עשרות עד מאות מהנדסים. Cherny מודה ש-"it's not totally there yet", אבל הכיוון הוא אוטומציה מלאה של התחזוקה, כך שהמהנדסים יתפנו לפיתוח מוצר ולשיחות עם משתמשים.

7. קידוד (כמעט) פתור, ומה סטודנט ל-CS עדיין צריך ללמוד

Cherny מסייג את הטענה ש-coding is solved: היא נכונה לסוג הקוד שהוא כותב. Claude עדיין מתקשה ב-codebases של מערכות עמוקות, במערכות מבוזרות ובאימות UI ברמת הפיקסל (בתמלול "off by Pixar", כנראה pixel). Opus 5 שיפר מאוד את הראייה ואת ה-computer use, אבל הוא עדיין לא מושלם [30:38]. בסקר קצר בקהל, הרבה ידיים עלו גם לשאלה מי כותב 100% מהקוד עם agents.

התכונה שמבדילה את המשתמשים הטובים לדבריו היא אמפיריות: לשכוח את מה שעבד במודלים קודמים, וגם את התיאוריה מהקורסים, ולנסות שוב דברים שבעבר לא עבדו. לסטודנטים הוא מספר שלמד לתכנת ב-BASIC על מחשבון TI-83 בחטיבת הביניים, כדי לרמות במבחנים במתמטיקה, ובהמשך עבר ל-assembly כשהחומר הגיע לחדו"א. המסקנה שלו: ללמוד לא רק מדעי המחשב אלא גם איך ליישם אותם, כלומר design sense, business sense, data science ושיחה עם משתמשים [33:42]. השיחה נסגרת בהכרזה שכל המשתתפים מקבלים מנוי Max 20x.

Boris Cherny [30:38] So it's just like very much become, it's not a theoretical science, it's become —
an empirical science

8. מה לעשות עם זה

- להריץ **ablation על ה-harness שלכם**: להסיר את כל ה-system prompt (או את CLAUDE.md, ה-skills וה-hooks), להריץ את המשימות הרגילות, ולהחזיר הוראה רק אחרי שרואים את אותו כישלון חוזר.
- לתת לכל שורה ב-prompt להצדיק את עצמה**: המודל קורא כל הוראה בכל הרצה, אז כל שורה צריכה לתקן כישלון שנצפה בפועל ולא כישלון שמישהו ניחש.
- לבנות verification לפני שכותבים prompt: test suite**, השוואת צילומי מסך או כל oracle אחר שמאפשר למודל לדעת מתי סיים. זה מה שאפשר את ריצות ה-11 ימים ואת ריצת השבועיים.
- לנסח משימות ברמה גבוהה**: מטרה, guardrails ותנאי יציאה, בלי רשימת צעדים.
- להתייחס ל-evals כמתכלים**: להחליף eval שהמודל רווה, ולבנות חדש מהמקומות שבהם המודל נתקע בפועל.
- לנסות routine תחזוקה אחת**: למשל ניקוי dead code או איחוד abstractions כפולות, כ-prompt של משפט אחד שרץ יומית ופותח PR.
- לחפש overhang במוצר שלכם**: לזרוק כל מודל חדש על בעיה שהמודל הקודם נכשל בה.

שאלה פתוחה	למה זה חשוב
איך נמדד שאי אפשר להדגים prompt injection?	בלי eval או בדיקה חיצונית, הטענה הזו לא מספיקה כדי לבטל הגנות ב-harness שלכם
כמה עולה ריצה של 11 ימים או של שבועיים?	בלי מספרי עלות קשה לדעת אם הגישה מתאימה גם למי שלא עובד ב-Anthropic
מה איכות השכתוב של Bun ל-Rust?	לא נמסרו נתוני באגים, ביצועים או היקף ה-steering
האם מצב simple ו-dynamic workflows זמינים לכל המשתמשים?	Cherny הגדיר את simple כלא מתועד, ושם משתנה הסביבה לא ודאי בתמלול
מה קורה כשה-verification שגויה?	כל השיטה נשענת על oracle, והשיחה לא עסקה במקרה שבו הוא עצמו טועה

שאלה פתוחה	למה זה חשוב
איפה עובר הגבול של 'coding is solved'?	נאמר רק באופן כללי: מערכות עמוקות, מערכות מבוזרות ואימות UI מדויק

9. מילון מונחים

מונח	פירוש
system prompt	ההוראות הקבועות שהמודל מקבל בכל הרצה
harness	הקוד, הכלים וה-prompts שעוטפים את המודל והופכים אותו למוצר
ablation	הסרה של רכיב ובדיקה של ההשפעה, למשל החזרת prompt שורה אחר שורה
eval	מערך בדיקות שמודד את ביצועי המודל במשימה
saturate	מצב שבו המודל מגיע לתקרה של eval והוא כבר לא מבחין בין גרסאות
prompt injection	הוראה זדונית שמוסתרת בתוכן שהמודל קורא, למשל דף אינטרנט
lethal trifecta	שילוב מסוכן של גישה לנתונים פרטיים, חשיפה לתוכן לא אמין ויכולת לשלוח מידע החוצה
auto mode	מצב שבו Claude Code פועל בלי לבקש אישור על כל פעולה, בפיקוח classifier
mechanistic interpretability	מחקר שמפענח מה קורה בתוך הרשת ברמת נירונים
product overhang	יכולת שקיימת במודל אבל אף מוצר עוד לא מנצל
hobbling / unhobbling	מצב שבו המוצר מגביל את המודל, והסרה של ההגבלה
elicitation	הדרך לגרום למודל להפגין יכולת שכבר קיימת בו
scaffolding	מבנה עזר חיצוני שמכוון את המודל, כמו goal/ או loop/
guardrails / exit criteria	הגבולות של המשימה והתנאים שלפיהם היא נחשבת גמורה
verification	מנגנון שמאפשר למודל לבדוק בעצמו שהעבודה נכונה
dynamic workflow	פיצ'ר ב-Claude Code לתזמור של עשרות עד אלפי agents בשלבים, ברצף ובמקביל
test-time compute	כמות החישוב (הטוקנים) שמושקעת בזמן הריצה ולא בזמן האימון
loop / routine	משימה מתוזמנת שחוזרת על עצמה, מקומית (loop) או בענן (routine)
fuzzing	הזנה של קלטים אקראיים כדי לחשוף באגים, למשל דליפות זיכרון
CLAUDE.md / skills / hooks / MCP	קובץ הוראות לפרויקט, יכולות ארזות, פעולות שרצות באירועים, ופרוטוקול לחיבור מקורות context