

Claude עם Agentic engineering Android CLI-ו Code

איך בונים לסוכן סביבה שבה הוא מצליח לבד: `context`, `agents.md`, `skills`, `hooks`
ולולאות אימות. בהדגמה: אפליקציית Android אמיתית

Ben Trengrove, Steven Jenkins · Android Developers (Android Dev Summit 2026) · 29.10.2026

משך: כ-30 דקות · YouTube

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זה סיכום של הרצאה מ-Android Dev Summit 2026. מציגים בה Ben Trengrove מצוות Developer Relations של Android ו-Product Manager, Steven Jenkins. ההרצאה משלבת עקרונות עבודה עם דמואים חיים על Sideline, אפליקציית טניס שהשניים בנו. האפליקציה מריצה מודלי TF Lite על המכשיר ומחזירה בזמן אמת מיקום של הכדור, של השחקן ושל המגרש. הם עבדו עם Claude Code, אבל מדגישים שהתוכן לא תלוי ב-agent מסוים: "it's more of a mindset".

הטענה המרכזית: התפקיד של המפתח עובר מכתובת קוד לאופטימיזציה של הסביבה שבה ה-agent עובד. כש-agent נכשל, לא מסיקים ש-AI עוד לא יודע לעשות את זה. שואלים איזה מידע או כלי חסר לו, ומטפלים בזה כמו בבאג. ההרצאה בנויה מארבעה צירים: Context, Environment, Tools, ו-Loops. בכל אחד מהם יש מספרים קונקרטיים: חלון context של כ-150 אלף טוקנים, agents.md של פחות מ-250 שורות, וזינוק משיעור הצלחה של 22% ל-90% כשה-agent מקבל test suite כושל מראש.

איך לקרוא את המסמך

- 1. **תקציר מנהלים:** הטענות המרכזיות עם המספרים, ומה לא נאמר.
- 2. **סביבה:** מה ה-agent צריך מעבר לקוד, ו-Android CLI בדמו של תיקון ANR.
- 3. **ניהול context:** החלון של כ-150 אלף טוקנים, workflow מבוסס skills וכללי agents.md.
- 4. **כלים ומנופים:** hooks, skills, CLI מול MCP, ותיקון ביצועים בעזרת profiler.
- 5. **לולאות אימות:** Android Bench 2.0, לולאות מדורגות, דמו XR ו-schedule.
- 6. **מה עושים מחר:** צעדים מעשיים וטבלת שאלות פתוחות.
- 7. **מילון מונחים:** המונחים באנגלית שמופיעים במסמך.

הערות מקור המסמך הופק אוטומטית מהתמליל ומהשקפים של ההקלטה. שמות ומונחים עלולים להשתבש בתמלול האוטומטי (ASR), למשל "Cloud Code" במקום Claude Code ו-"AndrewBench" במקום Android Bench. הציטוטים מועתקים מילה במילה ובבדקו מול חותמת הזמן שלהם. לא נוסף מידע ממקורות חיצוניים.

1. תקציר מנהלים

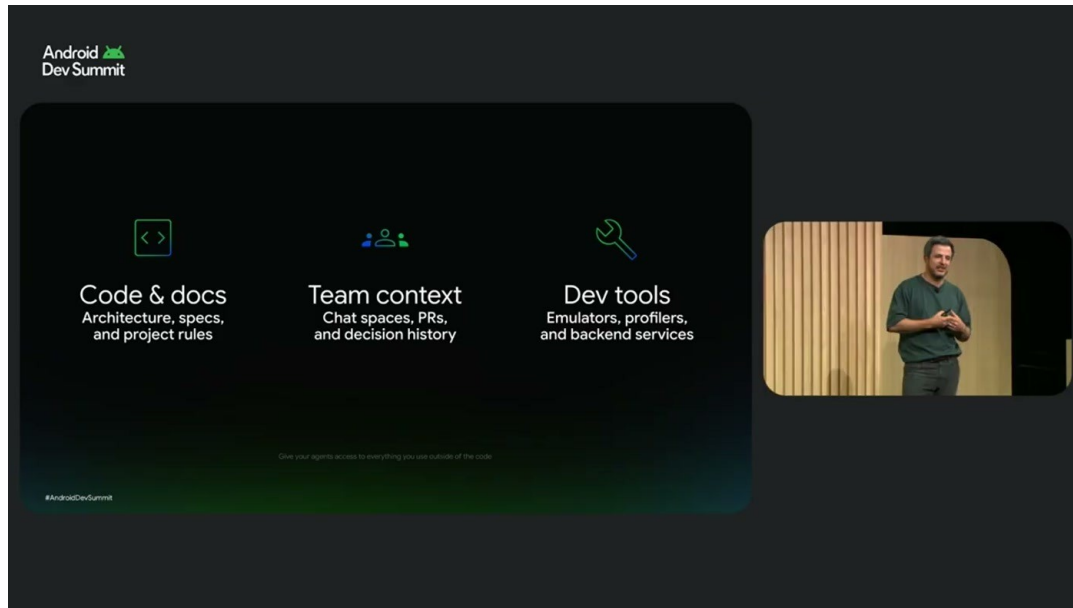
המסר בשורה אחת: אם רוצים להריץ צוותים של agents במקביל, צריך להפסיק לתקן אותם ידנית בכל צעד. עושים את זה בשלוש דרכים: נותנים להם סביבה מלאה (מכשירים, profilers, Crashlytics), שומרים על context רזה, ומגדירים מראש מה נחשב "גמור" כדי שילופו עד שזה מתקיים.

- **סביבה קודמת למודל:** Android CLI חיבר את ה-agent ל-Crashlytics ול-device streaming. כך התגלה ANR על Pixel 11 Pro Fold שלא שוחזר באמולטור, ותוקן בשתי שורות.
- **Context הוא תקציב קבוע:** החלונות נעצרו בסביבות מיליון טוקנים, אבל ה-sweet spot הוא שיחות של כ-150 אלף טוקנים בלי context rot.
- **workflow שמפרק עבודה ליחידות שנכנסות בחלון:** ה-skills של Matt Pocock (שאלות, spec, פירוק ל-issues). כל issue אמור להיכנס בכ-150K, ובסופו מנקים את ה-context.
- **agents.md קצר ונכתב בידי הצוות:** פחות מ-250 שורות. לפי מחקר מפברואר, agent שעורך לעצמו את הקובץ מוריד את הביצועים ב-2-3% ומעלה את העלות ב-20%.
- **Hooks לדטרמיניזם, skills ל-context לפי הצורך:** stop hook שמריץ lint או בדיקות במקום הוראה בטקסט. תיאור של skill מוגבל ל-1,024 תווים, ו-100 skills מוסיפים כ-100 אלף תווים עוד לפני שהתחלתם.
- **CLI לכלים מקומיים, MCP לשירותים חיצוניים:** CLI מהיר, ניתן להרכבה והרצה שלו כמעט חנימית, והמודלים אומנו עליו באגרסיביות (TerminalBench). MCP מתאים ל-GitHub, Figma ו-Slack.
- **ה-agent תיקן ביצועים לבד:** מ-22-FPS ל-30-FPS. הוא מצא ב-system traces שתהליכוני ה-AI רצים על הליבות האיטיות, ושלף בעצמו את התיעוד של Performance Hint API.
- **הגדרת "גמור" שווה פי ארבעה:** ב-Android Bench 2.0 מודל אחד עלה משיעור הצלחה של 22% ל-90% כשקיבל test suite כושל מראש, בלי שקיבל את הפתרון.
- **לולאות מדורגות ותיקון עצמי:** קודם JVM tests, אחר כך Compose previews ובסוף האמולטור. עם schedule/ ב-Claude Code התהליך רץ במחזוריות, מתקן באגים מ-Crashlytics ופותח PRs.

מה לא נאמר

- **מחקר agents.md:** לא צוין מקור. נאמר רק "a study came out in February this year".
- **מספר ה-90%:** שם המודל בתמליל הוא "Fable 5.1", ולא ברור מה המודל באמת (ייתכן שיבוש של ASR). גם לא ברור אם 90% מתייחס למשימת nav2→nav3 או לכל הבנצ'מרק. הדובר אומר "on the whole benchmark".
- **עלות וזמן:** לא נמסרו נתוני עלות, צריכת טוקנים או זמן ריצה לדמואים, לרבות הלולאה המתוזמנת עם schedule/.
- **מידה:** לא הוצגו מדדים לאיכות ה-PRs האוטונומיים או לשיעור ה-PRs שנדחו.
- **ה-sweet spot של 150K:** הוצג כמספר משוער שמשתנה בין מודלים, בלי נתונים שתומכים בו.

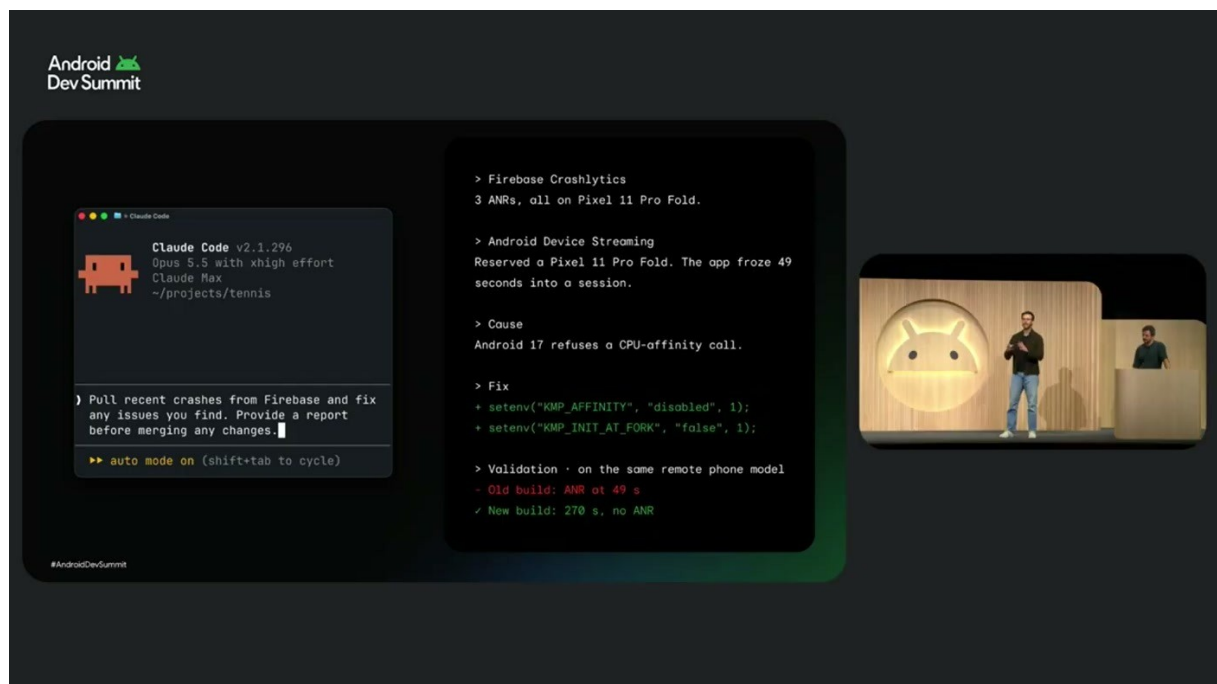
2. סביבה: מה ה-agent צריך כדי להצליח לבד



שלוש שכבות של סביבת *agent*: *Code & docs* (ארכיטקטורה, specs, כללי פרויקט), *Team context* (ערוצי צ'אט, PRs, היסטוריית החלטות), ו-*Dev tools* (אמולטורים, profilers ושירותי backend)

לפי Ben, גישה לקוד היא רק ההתחלה. יש צוותים שמחברים את ה-agent לצ'אט הצוותי כדי שיבין איך התקבלו ההחלטות שהובילו לפיצ'ר, וזה אותו ידע "תת-מודע" שמפתח משתמש בו. גם הערות ב-PRs, מכשיר או אמולטור, profiler ו-backend הם חלק מהסביבה. כאן נכנס Android CLI. זה לא מוצר AI אלא ממשק שורת פקודה לכלי Android: SDK manager, הפעלת אמולטורים, Crashlytics. הוא רץ ב-shell ולכן עובד עם כל agent.

בדמו הראשון Steven חיבר את Firebase Crashlytics לסביבת הפיתוח וגילה ANR על Pixel 11 Pro Fold. הבעיה לא שוחזרה באמולטור, ולא היה לו מכשיר כזה. ה-agent התחבר בעצמו למכשיר מרוחק דרך Android device streaming. הסיבה הייתה קריאת affinity שקשורה ל-Android 17, והתיקון היה שתי שורות. Steven מיזג, העלה build ל-Play, ובגרסה 1.0 הבעיה נעלמה. מנגד, צצו שתי בעיות חדשות ב-Crashlytics, והוא חוזר אליהן בסוף.



סיכום הדמו ב-Claude Code: שליפת crashes מ-Crashlytics, הזמנת Pixel 11 Pro Fold מרוחק ב-device streaming, אבחון קריאת CPU-affinity ב-Android 17, התיקון ואימות על אותו מכשיר מרוחק

Ben Trengrove we need to step back and think, what piece of information was missing and kind of —
.treat it how we treat bugs today

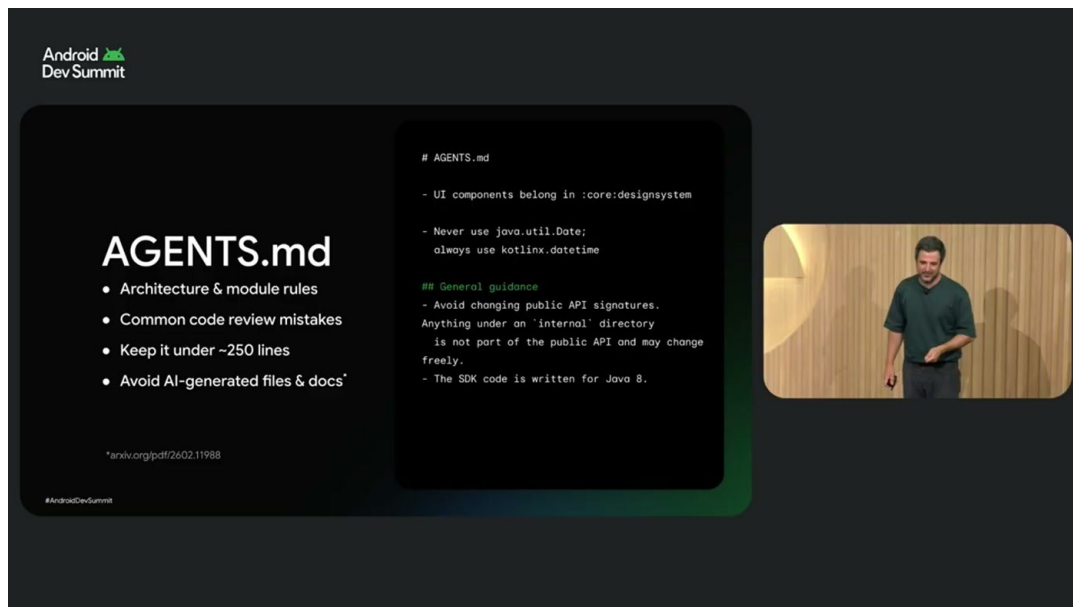
3. ניהול context ו-agents.md



"What's in context?": מה נשלח עם כל prompt. system instructions, מידע על ה-workspace וה-SDK, skills ו-MCP tools מותקנים, וכל היסטוריית ההודעות וקריאות הכלים

לפני כשנה וחצי עד שנתיים היה מרוץ להגדיל את חלונות ה-context. הצוות אפילו ניסה להכניס את כל prompt ל-developer.android.com, וזה לא עבד. לפי Android Bench 1.0, החלונות נעצרו בסביבות מיליון טוקנים, אבל יכולות המודלים ממשיכות לעלות. חלק מה-harnesses לא חושפים את כל המיליון, כי התוצאות נעשות גרועות יותר. ה-sweet spot, שמשתנה ממודל למודל, הוא כ-150 אלף טוקנים.

הצרכן הגדול של context הוא היסטוריית ההודעות וקריאות הכלים, שנשלחת מחדש עם כל הודעה. לוגים של Gradle בולטים במיוחד. הטיפ: להריץ Gradle ב-quiet mode. Android Studio ו-Claude Code מקצצים את הלוג בעצמם, אבל agents מינימליסטיים כמו mini SWE agent שולחים אותו במלואו, וזה נעשה יקר מהר. כדי להישאר בחלון Ben ממליץ על ה-workflow של Matt Pocock. ה-skill בשם Grill Me שואל אתכם שאלות עד שהעמימות נעלמת. אחריו skill שכותב spec, ו-skill נוסף שמפרק אותו ל-issues בגודל של כ-150K כל אחד. כל issue עובר לולאת build-verify, ובסוף מנקים את ה-context.



כללי AGENTS.md: כללי ארכיטקטורה ומודולים, טעויות נפוצות ב-code review, פחות מ-250 שורות, ולא לתת ל-AI לייצר את הקובץ. לצד זה דוגמה ('UI components belong in :core:designsystem', 'Never use java.util.Date')

agents.md נטען עם כל prompt, ולכן צריך להכניס אליו רק מה שרלוונטי כמעט תמיד. מה שעובד: כללי ארכיטקטורה ומודולים שה-agent לא יכול לנחש, במיוחד באפליקציה closed source, וטעויות שחוזרות ב-code review ("we prefer to use Kotlin X date time") שיטות עבודה מומלצות משתנות עם כל מודל. בימי Opus 4.6 נהגו לכתוב הוראות באותיות גדולות ("NEVER"), ועם Anthropic Opus 5 הודיעה שזה כבר לא עובד טוב.

שני כללים נשארו יציבים: לשמור על הקובץ קצר, מתחת ל-250 שורות, ולא לתת ל-agent לערוך אותו. לפי Ben, מחקר מפברואר מצא ירידה של 2-3% בביצועים ועלייה של 20% בעלות. החלופה שהוא מציע: ה-agent כותב רטרוספקטיבה בסוף כל PR, והצוות קורא אותן יחד ומחליט מה משנים. Steven מוסיף שארכיטקטורה מוגדרת היטב חוסכת טוקנים, כי ה-agent לא צריך לחפור בקוד כדי להסיק איפה דברים יושבים.

4. כלים ומנופים: skills, hooks, CLI מול MCP

Hooks הם אירועים שה-agent חושף: תחילת session, סיום session, קריאת קובץ וכתובת קובץ. במקום לכתוב ב-agents.md "תמיד תריץ linter לפני שאתה מסיים", מגדירים stop hook שמריץ סקריפט דטרמיניסטי. אותו מנגנון חוסם קריאה של קובץ secrets או כתיבה ל-agents.md. בלשונו של Ben: "It's just a way to add determinism".

Skills טוענים context רק כשהוא רלוונטי. כל skill בנוי מ-name, מ-description (עד 1,024 תווים, ותמיד נמצא ב-context) ומ-body שנטען רק כשצריך. ה-body יכול להיות ענק ולכלול סקריפטים.

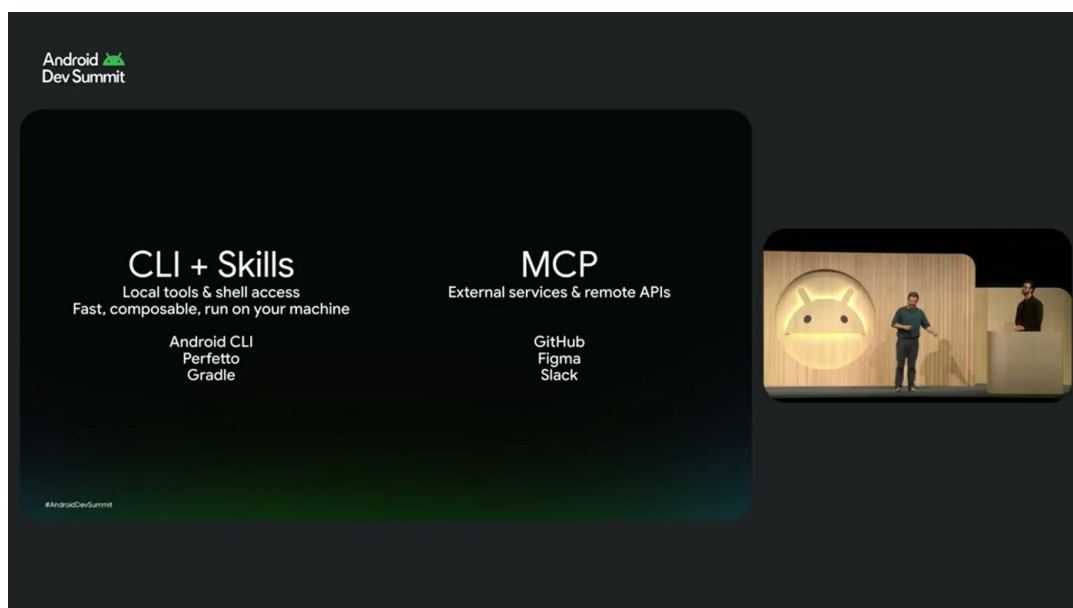


Be selective: לא להתקין מאות skills בלי לבדוק. skills שלא נבדקו הם סיכון אבטחה, ו-100 תיאורים מנפחים את ה-context. מתחילים מה-base skill של android-cli ומבקרים עם skill-doctor/ Stats. לצדו רצף פקודות android skills

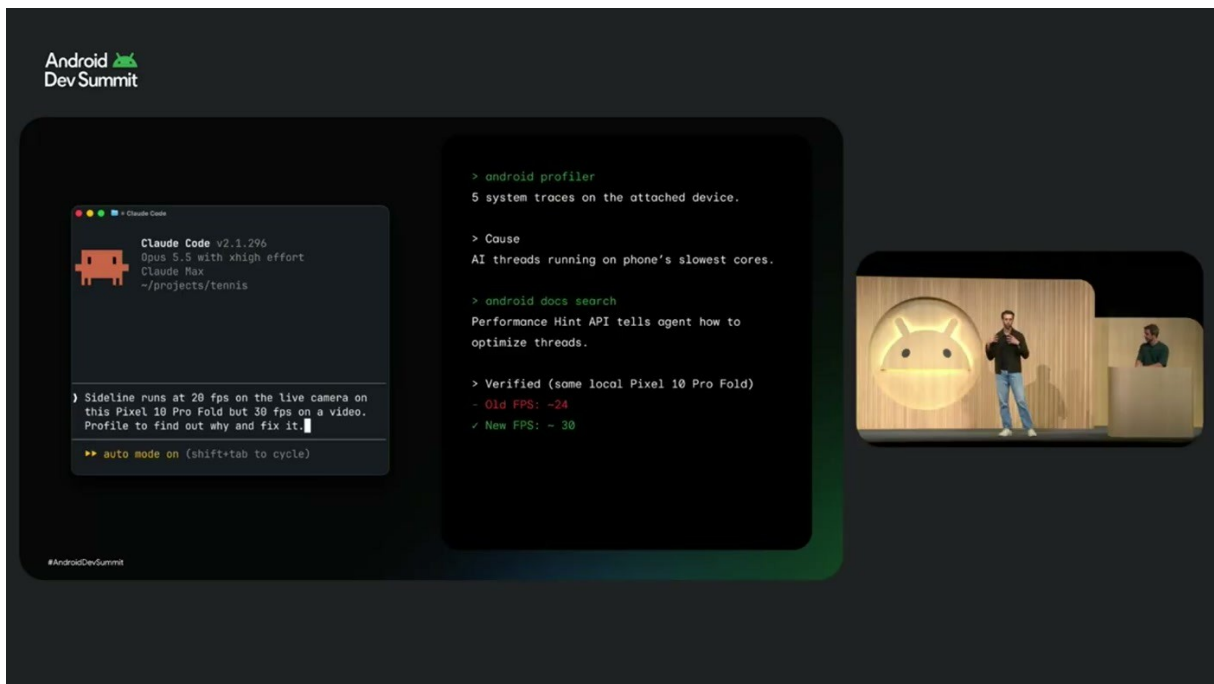
Ben Trengrove One, that is a massive security risk. You have no idea what's in those skills. It — could be telling your agent to do anything

ההמלצה: להתקין את ה-base skill של Android CLI ולתת לו לחפש ולהתקין skills שמתאימים לפריקט, או להוסיף ידנית את האוניברסליים (Compose, edge-to-edge). ספריות חדשות כמו Navigation 3, שהמודלים כמעט לא אומנו עליהן, הן מקרה מובהק ל-skill ייעודי. ב-Claude Code יש skill doctor שמראה אילו skills אף פעם לא מופעלים.

למה Android CLI הוא CLI ולא שרת MCP? כי "fast, they're composable, and because they run locally in your shell session, they're effectively free to run". בנוסף, יצרני המודלים משפרים אותם באגרסיביות מול TerminalBench. ל-MCP יש מקום בשירותים חיצוניים שכבר חושפים שרת, כמו GitHub, Figma, Slack, ו-Gradle. נשארים ב-shell.



החלוקה: CLI + Skills: לגישה מקומית ל-shell (Android CLI, Perfetto, Gradle), ו-MCP לשירותים חיצוניים ול-APIs מרוחקים (GitHub, Slack, Figma)



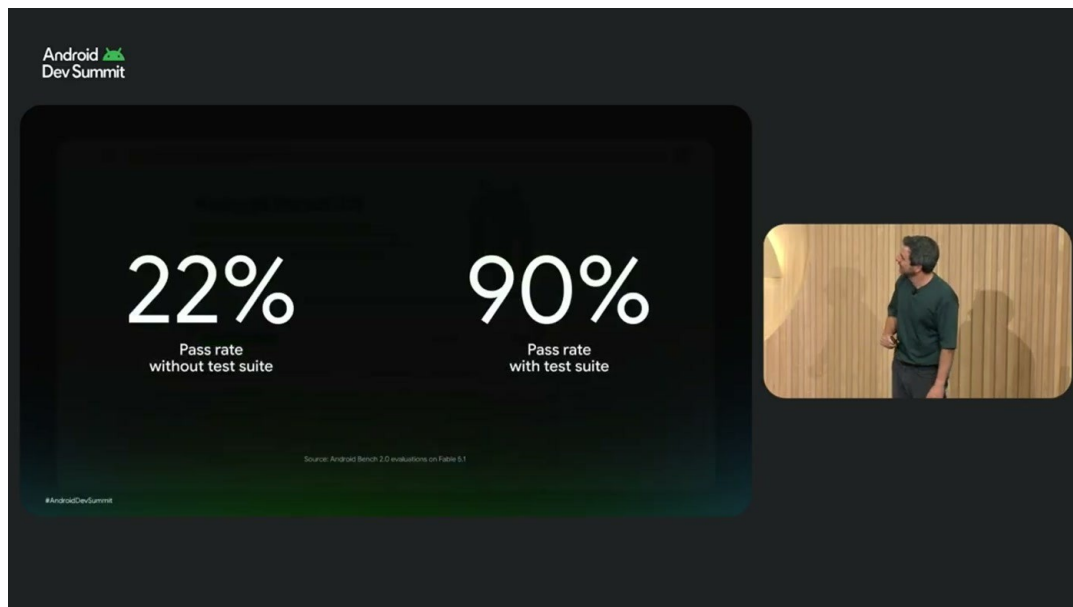
דמו ביצועים ב-Claude Code: Sideline רץ ב-20-22 FPS. ה-agent הקליט system traces עם android profiler, מצא תהליכוני AI על הליבות האיטיות, שלף את תיעוד Performance Hint API ואימת את התיקון על המכשיר

Sideline צריכה לרוץ ב-FPS 30 בזמן אמת, ועל Pixel 10 Pro Fold היא הגיעה ל-20-22 בלבד. בתמליל השם השתבש ל-"Pixel 10 profile", ושם המכשיר כאן נלקח מהשקף. ה-agent קיבל את הפקודה android profiler ואת הדרישה ל-FPS 30. הוא הקליט system traces וגילה שתהליכוני ה-AI רצים על הליבות האיטיות. לאחר מכן, בלי שהתבקש, הוא השתמש ב-Android Docs Search כדי לשלוף את ההנחיות הרשמיות ל-Performance Hint API, ושינה את התזמון בהתאם. לדברי Steven, "the fix followed the guidance". הערך לדבריו: ה-agent מוצא את הכלי הנכון גם כשהמפתח לא מכיר אותו. Ben מודה שעקומת הלמידה של Perfetto תלולה מאוד.

5. לולאות אימות: להגדיר מראש מה זה "גמור"

התסריט המוכר: מבקשים מה-agent להעביר את כל הפרויקט ל-Nav3. הוא חוזר ומודיע "migration complete", ואז מסובבים את הטלפון והאפליקציה קורסת. מעברי ה-slide חסרים וקובץ ה-Gradle לא נוקה. המסקנה של Ben:

Ben Trengrove We need to stop being our agent's QA tester and define upfront what we consider — done to be

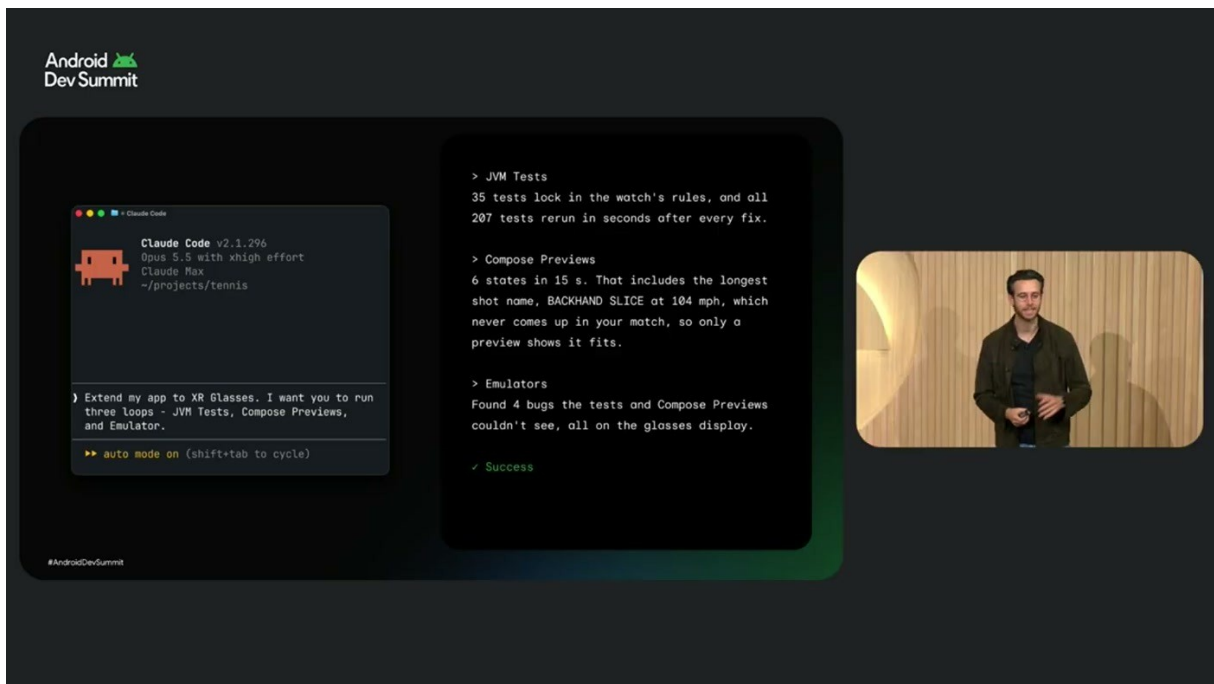


Android Bench 2.0: שיעור הצלחה של 22% בלי test suite לעומת 90% עם test suite כושל שניתן מראש

Android Bench 2.0 כולל משימות גדולות ומציאותיות, ושיעורי ההצלחה בו עומדים כיום על כ-40%. אחת המשימות היא מעבר מ-nav2 ל-nav3. מודל שבתמליל נקרא "Fable 5.1" (השם לא ודאי) הגיע ל-22% עם prompt טבעי, ולדברי הדובר ל-90% בכל הבנצ'מרק כשקיבל test suite כושל מראש. הוא לא קיבל את התשובה, רק הגדרה של מה נחשב גמור. "these models are really capable now of implementing pretty much any Android tasks we can think of. They just don't always know what done means".

לא חייבים test suite מלא. אפשר להוסיף ל-prompt: לרנדר Compose preview ולוודא שהוא תואם ל-spec ב-Figma, ולהראות את התוצאה. בקשה ל"show me the results" עובדת טוב יותר מלקבל "I am done". בלוגיקה עסקית אפשר לבקש להריץ unit tests, ובתיקון קריסה לבקש אימות על המכשיר האמיתי דרך Remote Test Lab. ב-Claude Code יש skill מובנה לבניית לולאות אימות, ו-Bring Your Own Agent (Android Studio) חושף כלים כמו lint ו-Journeys.

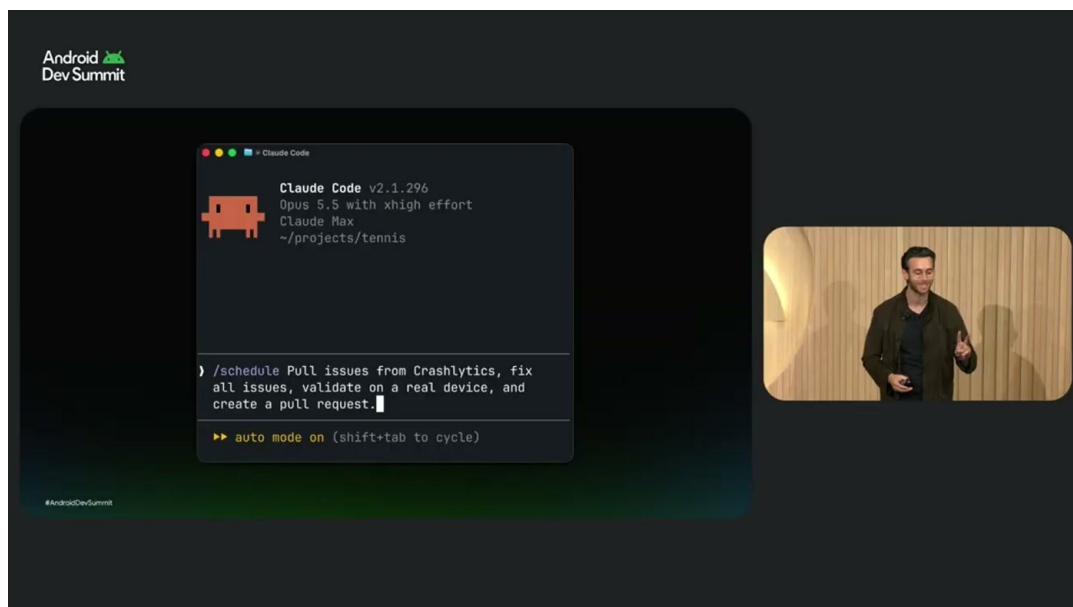
- **רמה 1, מהירה וזולה:** unit tests ו-Compose previews.
- **רמה 2:** connected tests ו-Android Journeys, שהם prompts שמתארים את הפיצ'ר.
- **רמה 3, איטית וחזקה:** full test suite ב-Remote Test Lab, שעלול לקחת שעות.
- **למה לדרג:** כשהרמות מוגדרות במפורש, המודל לופ בתוך כל רמה במקום להריץ הכול בכל סבב, כי "models are really confident".



דמו XR: prompt ב-Claude Code להרחבת Sideline למשקפי XR עם שלוש לולאות (JVM tests, Compose previews, אמולטור), וסיכום מה כל לולאה תפסה

Steven הרחיב את Sideline למשקפי XR שעוד לא יצאו לשוק, על אמולטור XR, בשלוש לולאות שכל אחת בונה על קודמתה. ה-JVM tests הגדירו שש מצבי Compose preview בלי להפעיל אמולטור, ולכן רצו מהר. ה-compose previews תפסו מחרוזות ארוכות עוד לפני האמולטור. האמולטור מצא באגים שהשניים הראשונים לא יכלו לתפוס. בתוצאה, המשקפיים מציגים את סוג החבטה ואת מהירות הכדור בזמן אמת. העיקרון: "we want to run the fastest but" "least powerful loops first, followed by the slowest but most powerful".

בסוף Steven חוזר לשתי הקריסות החדשות מ-Crashlytics. מכיוון שהגישה כבר הוכיחה את עצמה, הוא משתמש ב-schedule ב-Claude Code כדי להריץ אותה במחזוריות שנבחרת: שליפת issues, תיקון, אימות על מכשיר אמיתי ופתיחת PR שמהנדסים בודקים וממזגים. הוא קורא לזה "self-healing application".



פקודת /schedule ב-Claude Code: שליפת issues מ-Crashlytics, תיקון, אימות על מכשיר אמיתי ופתיחת pull request

6. מה עושים מחר

- לטפל בכישלון של agent כמו בבאג: לשאול איזה מידע או כלי היה חסר לו, ולהוסיף אותו לסביבה.

- **לחבר את ה-agent לכלים אמיתיים:** Android CLI, Crashlytics, device streaming ו-profiler. לא להסתפק בגישה לקוד.
- **לשמור על 150K:** לפרק עבודה ל-issues שנכנסים בחלון, לנקות context בין issues ולהריץ Gradle ב-quiet mode.
- **לקצץ את agents.md:** פחות מ-250 שורות, רק ארכיטקטורה וטעויות חוזרות, ובלי עריכה עצמית של ה-agent. שינויים נכנסים דרך רטרוספקטיבה צוותית.
- **להעביר הוראות "תמיד" ל-stop hook:** lint ולבדיקות, ו-hook שחוסם גישה ל-secrets.
- **לבקר skills:** להתחיל מה-base skill של Android CLI ולהסיר skills שלא מופעלים (skill doctor).
- **להגדיר "גמור" מראש:** test suite כושל, או לפחות בקשה ל"show me the results". לדרג לולאות מהמהירה לאיטית.
- **לכתוב evals לפרויקט:** כך אפשר למדוד אם עצה חדשה או מודל חדש באמת עוזרים. העצות ברשת סותרות זו את זו.

שאלה פתוחה	למה זה חשוב
מה המקור של המחקר מפברואר על עריכה עצמית של agents.md?	המספרים (2-3% - ביצועים, +20% עלות) מצדיקים שינוי תהליך, אבל לא ניתן לבדוק אותם
האם 90% מתייחס ל-nav2→nav3 או לכל Android Bench 2.0, ומהו המודל?	ההבדל משנה את המסקנה על משקל הבדיקות בכלל המשימות
מה העלות של לולאה מתוזמנת עם /schedule?	צריכת טוקנים ומכשירים ב-Remote Test Lab לא דווחה
כמה מה-PRs האוטומטיים מוזגו בלי שינוי?	"self-healing" נשען על איכות התיקונים, ולא הוצגו נתונים
האם 150K נשאר ה-sweet spot גם במודלים הבאים?	הדובר עצמו אומר שהמספר משתנה ממודל למודל

7. מילון מונחים

מונח	משמעות
Android CLI	ממשק שורת פקודה חדש לכלי Android (SDK, אמולטורים, profiler, Crashlytics, device streaming). עובד עם כל agent
ANR	Application Not Responding: מצב שבו האפליקציה לא מגיבה
Context rot	ירידה באיכות התשובות כשה-context ארוך מדי
agents.md	קובץ הנחיות קבועות ל-agent שנטען עם כל prompt
Hook	סקריפט דטרמיניסטי שרץ באירוע של ה-agent (session start/stop, קריאה או כתיבה של קובץ)
Skill	חבילת ידע עם name, description (עד 1,024 תווים) ו-body שנטען רק כשצריך
MCP	פרוטוקול לחיבור agent לשירותים חיצוניים כמו GitHub, Slack ו-Figma
Verification loop	לולאה שבה ה-agent ממשיך לעבוד עד שקריטריוני "גמור" שהוגדרו מראש מתקיימים
Android Bench	הבנצ'מרק של Google למדידת יכולת של מודלים לכתוב קוד Android
Device streaming	גישה מרחוק למכשיר פיזי לצורך שחזור ואימות
Journeys	בדיקות ב-Android Studio שנכתבות כ-prompts על הפיצ'ר
Performance Hint API	API של Android לרמזי תזמון וביצועים למערכת
Harness	סביבת ההרצה של ה-agent, כמו Claude Code, Codex או Antigravity
Evals	הערכות מדידות לפרויקט, שבדקות אם שינוי עוזר