

איך Jane Street בונה כלי LLM לפיתוח ב-OCaml: מודלים מותאמים, נתוני אימון ו-AIDE

על חברה שהקוד שלה כתוב בשפה שהמודלים כמעט לא מכירים, ועל מה שעשתה כדי
ש-LLM בכל זאת יעזור למפתחים שלה

John Crepezzi (ב-ASR: John Carpezi) · הרצאה בכנס AI Engineer (על השקפים: Agent Engineering,
ai.engineer), מתוך שיתוף ב-X של Linas Beliūnas · תאריך ההרצאה המקורית לא צוין

משך: כ-17 דקות · X

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה קצרה בכנס AI Engineer. הדובר הוא מהנדס בצוות Jane Street, ולפני כן עבד שנים ב-GitHub ובחברות DevTools אחרות. הוא מתאר את הדרך שבה החברה מנסה להוציא ערך מ-LLM בכלי הפיתוח שלה, למרות שהכול אצלה כתוב ב-OCaml: שפה שהמודלים הציבוריים חלשים בה, וסביבה שמבוססת על build system, מערכת code review ו-monorepo שהחברה בנתה בעצמה.

הטענה המרכזית: כשהסביבה שלך חריגה, אי אפשר פשוט לקנות כלי מהמדף. צריך שלושה רכיבים. הראשון הוא מודל מותאם שאומן על נתונים בצורה המדויקת של המשימה. השני הוא שכבת אינטגרציה אחת (AIDE) שמשרתת את כל העורכים. השלישי הוא תשתית הערכה שבודקת אם הקוד שהמודל כתב באמת מתקמפל ועובר טסטים. הוא גם מראה איפה הגישה הנאיבית נכשלה ומה החליף אותה.

איך לקרוא את המסמך

- **1. תקציר מנהלים:** הטענות המרכזיות עם חותמות זמן, וגם מה לא נאמר בהרצאה.
- **2. למה כלים מהמדף לא עובדים:** OCaml, ה-tooling הפנימי והשאיפה לחבר LLM לכל שלבי הפיתוח.
- **3. מודל מותאם ונתוני אימון:** מה הייתה המטרה, ולמה features ו-commits לא התאימו ו-workspace snapshotting.
- **4. RL ו-evaluation עם CES:** מה נחשב קוד טוב, איך בנוי שירות ההערכה, ומודל ה-code review שענה "אעשה את זה מחר".
- **5. AIDE: sidecar** אחד לשלושה עורכים, החלפת מודלים ו-A/B testing.
- **6. מסקנות ושאלות פתוחות:** מה כדאי לעשות עם זה, ומה נשאר לא ברור.
- **7. מילון מונחים:** המונחים באנגלית שמופיעים במסמך.

הערות מקור המסמך הופק אוטומטית מהתמלול של ההקלטה ומהשקפים שהוצגו בה. ה-ASR עלול לשבש שמות ומונחים: שם הדובר תומלל John Carpezi, אבל ב-commit log שעל השקף מופיע Crepezzi, ושם השירות תומלל "Aid" בעוד שעל השקף כתוב AIDE. הציטוטים מועתקים מילה במילה מהתמלול ונבדקו מול חותמת הזמן שלהם. לא נוסף שום מידע ממקורות חיצוניים. הכיתוב של הפוסט ב-X (כולל הטענה על הכנסות של החברה) לא נאמר בהרצאה ולא שימש כמקור.

1. תקציר מנהלים

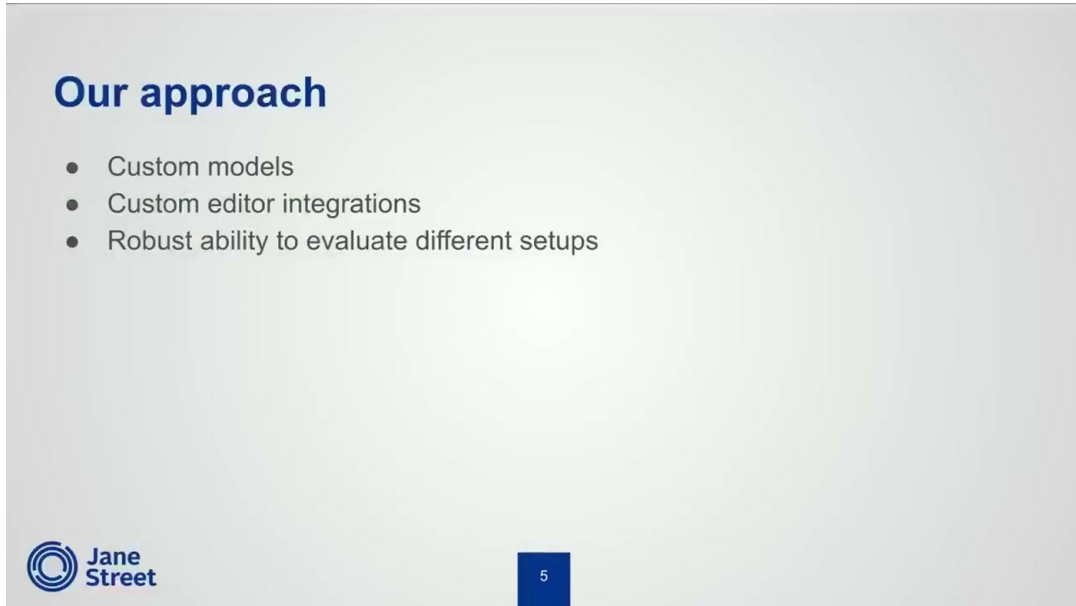
זו לא הרצאה על מסחר ולא על מודלים פיננסיים. היא עוסקת כולה בכלי פיתוח פנימיים. השאלה שהיא עונה עליה: מה עושים כשהמודלים הציבוריים לא מכירים את השפה, את הספריות ואת זרימת העבודה שלך? התשובה של Jane Street היא pipeline שלם: איסוף נתונים בצורה של המשימה, SFT ואחריו RL מול build אמיתי, ושכבת אינטגרציה אחת שמאפשרת להחליף מודלים בלי לגעת בעורכים.

- **המודלים פשוט חלשים ב-OCaml:** הסיבה היא כמות נתוני האימון. לדברי הדובר, ייתכן שיש יותר קוד OCaml בתוך Jane Street מאשר בכל העולם מחוצה לה [1:30].
- **Fine-tuning נאיבי לא עובד:** הניסיון לשחזר את CodeCompose של Meta על ידי "להראות למודל הרבה קוד" נכשל. צריך דוגמאות בצורה המדויקת של השאלה שרוצים לשאול [4:33].
- **המשימה הוגדרה בצורה צרה:** diff, גם על כמה קבצים, שבבנה מ-prompt בתוך העורך, צריך לחול בלי קונפליקטים ויש לו סיכוי טוב לעבור type check. היעד היה עד כ-100 שורות [4:33].
- **features ו-commits לא היו נתוני אימון טובים:** תיאור של feature לא כתוב כמו בקשה בעורך, feature מגיע ל-500 עד 1000 שורות, ול-commits אין תיאור בכלל [6:04].
- **workspace snapshotting פתר את בעיית הנתונים:** מצלמים את מצב תחנת העבודה בערך כל 20 שניות יחד עם סטטוס ה-build. רצף ירוק-אדום-ירוק מסמן שינוי מבודד, ואת התיאור כותב LLM שאחר כך מקצרים אותו [7:34].
- **RL מול build אמיתי:** שירות CES מחזיק build "חם", מחיל עליו diffs מהמודל ומחזיר אדום או ירוק כ-reward. במשך חודשים זה יישר את המודל לכתוב קוד שמתקמפל ועובר טסטים [9:08][10:41].
- **אותה תשתית משמשת גם ל-evaluation:** מספיק לשמור בצד חלק מנתוני ה-RL. מודל code review שאומן חודשים בלי הערכה טובה ענה "I'll do it tomorrow" [10:41].
- **AIDE הוא sidecar אחד לשלושה עורכים:** VS Code, Emacs ו-NeoVim מקבלים שכבה דקה מעל AIDE. עדכון מגיע לכולם ב-restart של השירות, בלי לחכות שמשתמשים יפעילו מחדש את העורך [12:15][13:45].
- **שכבת האינטגרציה מאפשרת להחליף רכיבים ולמדוד:** אפשר להחליף מודלים ואת אסטרטגיית ה-context, להוסיף כלים ייעודיים לתחום, ולהריץ A/B של 50/50 בין מודלים לפי [15:17][13:45] acceptance rate.

מה לא נאמר

- **אין מספרים על תוצאות:** לא נמסרו acceptance rate, שיעור type check או compile, או השוואה בין המודל המותאם למודל ציבורי. הטבלאות מהמאמר של Meta הוצגו, אבל לא נתונים של Jane Street.
- **לא ידוע על איזה מודל בסיס אימנו:** לא צוינו גודל המודל, ספק או עלות האימון. נאמר רק שבהתחלה השתמשו במודל שלא עבר fine-tuning.
- **פרטי ה-RL חסרים:** לא הוסבר איך ה-reward משוקלל בין type check, parse וטסטים, וגם לא כמה דוגמאות נאספו.
- **פרטיות ה-snapshotting לא נדונה:** לא נאמר דבר על הסכמת המפתחים או על אחסון התמונות של תחנות העבודה.
- **אין שום קשר להכנסות של החברה:** הטענה בכיתוב של הפוסט ב-X שהחברה "מרוויחה 30 מיליארד בשנה בעזרת LLM" לא מופיעה בהרצאה. ההרצאה עוסקת רק בכלי פיתוח.

2. למה כלים מהמדף לא עובדים ב-Jane Street



Our approach

- Custom models
- Custom editor integrations
- Robust ability to evaluate different setups

Jane Street

5

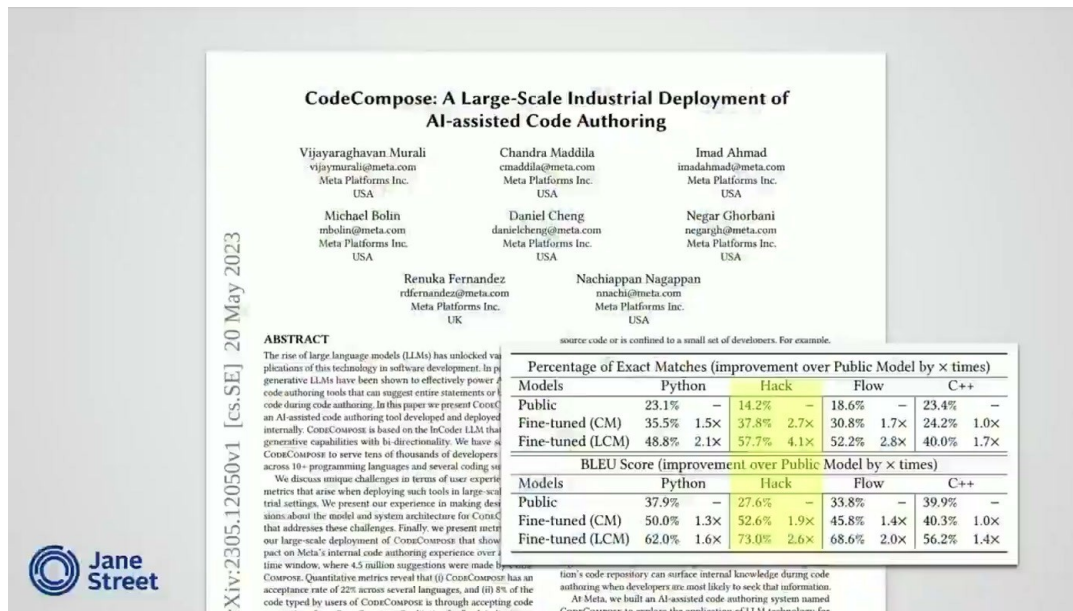
השקף "Our approach" עם שלושה צירים: *custom models*, *custom editor integrations* ויכולת להעריך תצורות שונות

ב-Jane Street כותבים הכול ב-OCaml, גם דברים שבדרך כלל מחייבים שפה אחרת. אפליקציות web נכתבות ב-OCaml ומתורגמות ל-JavaScript בעזרת OCaml plugins, JS of OCaml, Vim נכתבים עם VCaml, ומי שכותב קוד FPGA משתמש ב-HardCaml במקום [1:30][0:00] Verilog.

הדובר מונה שלוש סיבות לכך שכלים מוכנים לא מתאימים. הראשונה היא שהמודלים לא טובים ב-OCaml (השקף: "Models aren't great at OCaml"). השנייה היא שהחברה "made things really hard on ourselves": build system ו-distributed build משלה, מערכת code review בשם Iron, monorepo שמאוחסן ב-Mercurial ולא ב-Git, ו-"67% [1:30] of the firm uses Emacs". השלישית היא השאיפה להשתמש ב-LLM גם ב-merge conflicts, בתיאורי features ובבחירת reviewers, בלי להיתקע בגבולות בין מערכות [3:02].

John Crepezzi [1:30] — So there's a really good chance that the amount of OCaml code that we have inside of Jane Street is just more than like the total combined amount of OCaml code that there exists in the world outside of our walls

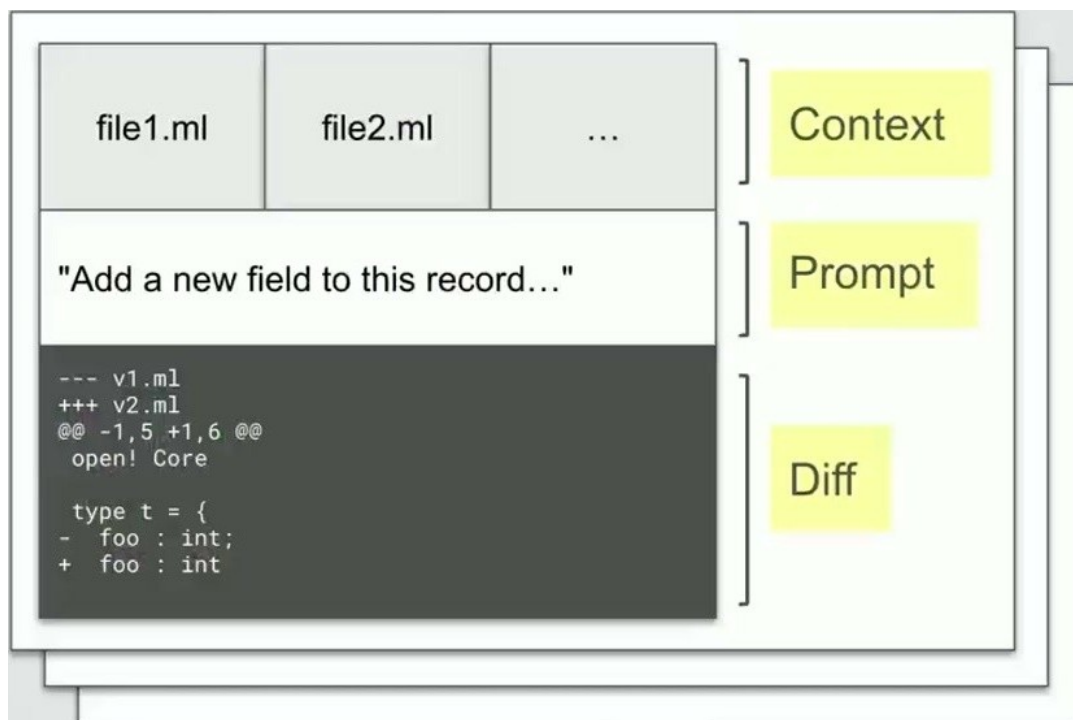
3. מודל מותאם: מטרה, צורת הנתונים ו-workspace snapshotting



המאמר של *CodeCompose: A Large-Scale Industrial Deployment of AI-assisted Code Authoring*, עם טבלאות exact match ו-BLEU שמשוות בין מודל ציבורי למודל fine-tuned ב-Python, Hack, Flow ו-C++

מה שגרם להם לנסות לאמן מודל היה המאמר של Meta על CodeCompose, שבו מודל עבר fine-tuning עבור Hack. כמו OCaml, גם Hack היא שפה שבפועל משתמשת בה חברה אחת (ובמקרה, Hack ממומשת ב-OCaml). הכוונה הראשונית הייתה נאיבית: "We thought we would just take a model off the shelf, we would show it a bunch of our code" [4:33] לקח המפתח: "you have to have the model see a bunch of examples that are in the shape of the type of question that you want to ask the model" [4:33].

לכן קודם הגדירו מטרה: משתמש כותב בעורך מה הוא רוצה, ומקבל diff שאולי נוגע בכמה קבצים, למשל קובץ טסט, קובץ ML וקובץ MLI (שדומה ל-header). ה-diff צריך לחול בלי קונפליקטים, עם סיכוי טוב לעבור type check, והיעד היה עד כ-100 שורות [4:33].



צורת דוגמת האימון: context (קבצים פתוחים כמו file1.ml ו-file2.ml), ("Add a new field to this record...") prompt ו-diff שמממש אותו

כל דוגמה כזו היא שלישייה של context, prompt ו-diff, וצריך הרבה מהן [6:04]. המקום הראשון שבדקו היה features במערכת Iron, שמקבילות ל-pull requests. בלשונית Description יש תיאור שכתב אדם, ובלשונית Diff יש הקוד. אבל תיאור של feature כתוב אחרת לגמרי ממה שמישהו כותב בעורך ("fix that error that's happening" ו-"right now"), ו-feature מגיע בדרך כלל ל-500 עד 1000 שורות [6:04]. ב-Jane Street commits לא עזרו יותר: ב-Jane Street הם משמשים כנקודות שמירה בלבד, אין להם תיאור (ב-log רואים "summary: z"), והם לא שינויים מבודדים [7:34].

Maybe features?

Add User Settings Table To Database

Next Step: [Unlock Release](#)

Compilation
Tip Compiles

Feature is Locked
Release

Owner

Description	Diff	Follow Review	Child Features	History
Adds a new user table to Postgres, to store persistent user-specific data. Columns are:				
1. Username (primary key)				

Prompt

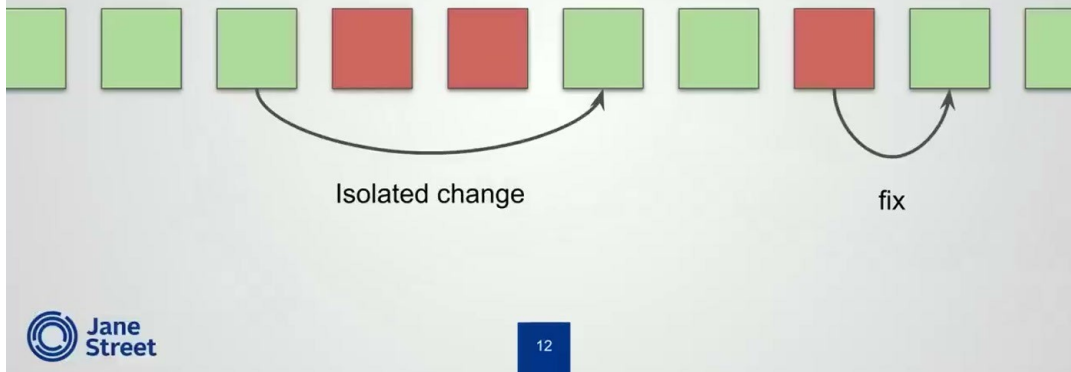
Diff

Jane Street

10

feature במערכת Iron בשם "Add User Settings Table To Database": התיאור מסומן כ-Prompt ולשונית ה-Diff כ-Diff

Workspace snapshotting



workspace snapshotting: רצף snapshots ירוקים ואדומים. ירוק-אדום-ירוק מסומן "Isolated change" ואדום-ירוק מסומן "fix"

הפתרון היה workspace snapshotting. מצלמים את תחנות העבודה של המפתחים לאורך היום, "like every 20 seconds", יחד עם סטטוס ה-[7:34] build. רצף ירוק-אדום-ירוק מסמן בדרך כלל שינוי מבודד. רצף אדום-ירוק מסמן שהמפתח נתקל ב-type error או בשגיאת קומפילציה ותיקן אותה. כשלוקחים את השגיאה במצב האדום ואת ה-diff עד הירוק, מקבלים נתונים שמלמדים את המודל להתאושש מטעויות.

את התיאורים כותב LLM: קודם תיאור מפורט ככל האפשר, ואחר כך מסננים ומקצרים עד שהוא נראה כמו משהו שאדם היה כותב [7:34]. השקף שמדגים את זה משתמש בסיפור שלושת החזירים הקטנים, שמתקצר בהדרגה עד "Brick wins".

Three pig brothers set out to build their own homes, with the first pig quickly constructing a house of straw, the second pig building a house of sticks, and the third pig diligently creating a sturdy house of bricks. When a hungry wolf came along, he easily blew down the straw house and then the stick house, forcing both pigs to flee to their brother's brick house for safety. The wolf tried repeatedly to blow down the brick house but failed, then attempted to enter through the chimney, only to fall into a pot of boiling water the clever third pig had prepared, sending the wolf running away and never returning, while the three pigs lived safely together in the brick house thereafter.

Three pigs built houses of straw, sticks, and bricks respectively. A wolf blew down the first two houses, forcing those pigs to flee to their brother's brick house. When the wolf tried entering through the chimney, he fell into a pot of boiling water and ran away, leaving the pigs to live safely ever after.

Three pigs built homes of straw, sticks, and bricks. A wolf destroyed the first two houses, but failed to blow down the brick one and fled after falling into a boiling pot, leaving the pigs safe.

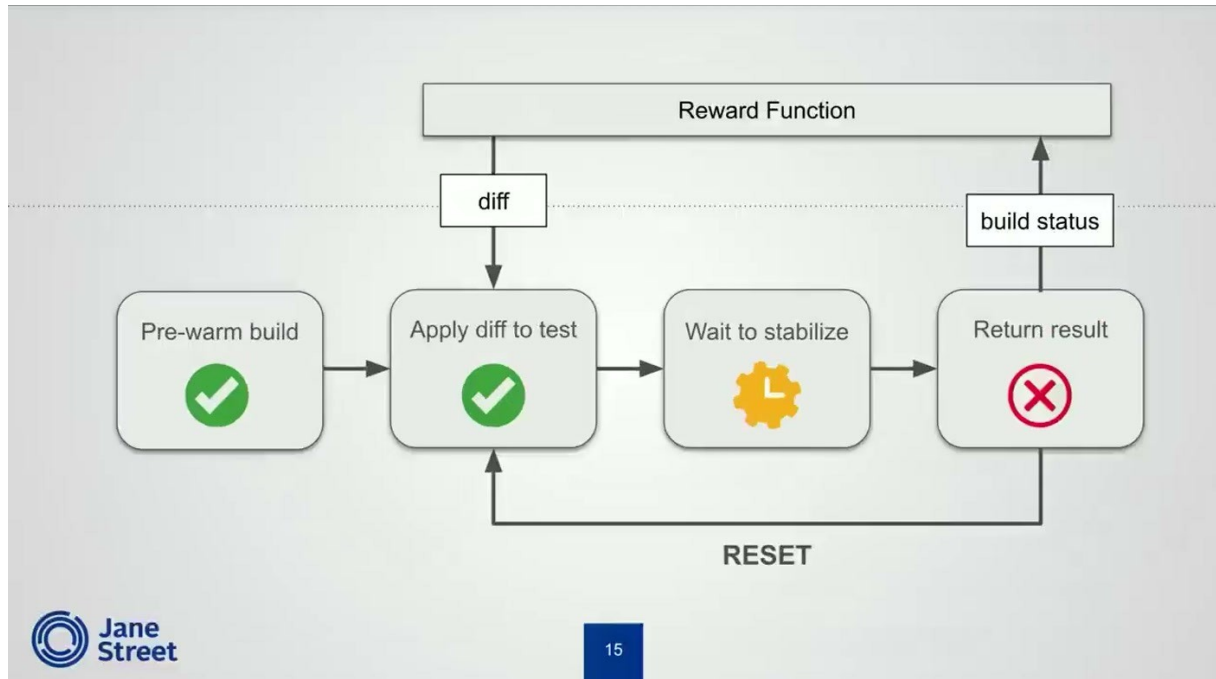
Three pigs made houses of straw, sticks, and bricks. A wolf blew down two but failed at the third, falling into a pot and fleeing.

Wolf failed to blow down third pig's brick house.

Brick wins.

המחשה של קיצור תיאור בעזרת LLM: סיפור שלושת החזירים מתקצר בשלבים, מפסקה מלאה עד "Brick wins".

4. RL-evaluation: מה זה קוד טוב, ושירות CES



לולאת ה-Reward Function של build status: Pre-warm build, Apply diff to test, Wait to stabilize, Return result. CES חוזר ל-reward ואחר כך יש RESET

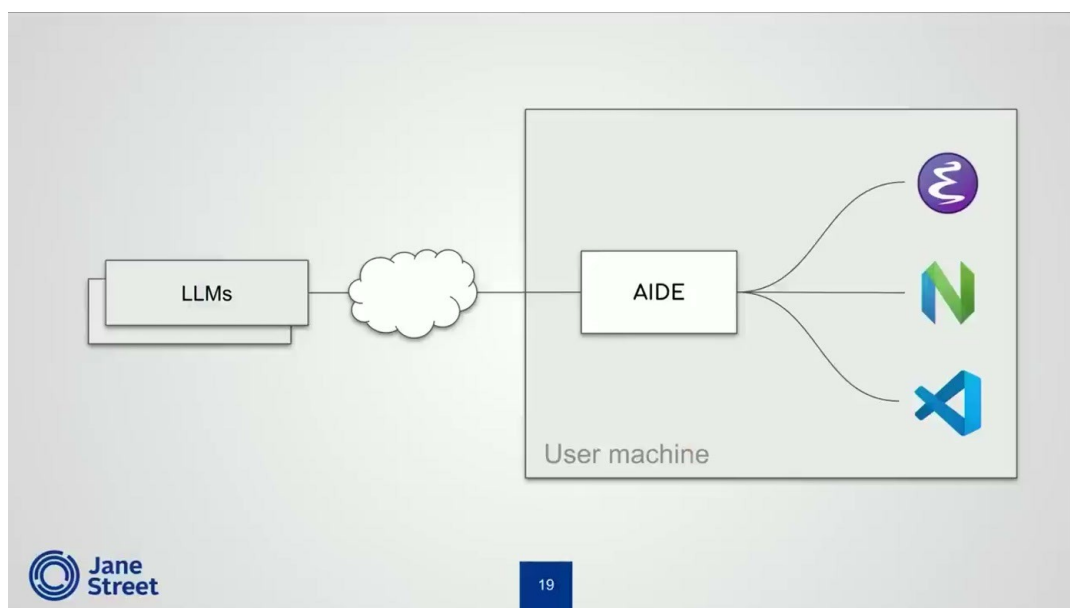
נתוני ה-SFT הם רק חצי מהעבודה. החצי השני הוא RL, ושם "models get a lot of the power" [9:08]. ההגדרה של קוד טוב מדורגת: קוד שעובר את ה-parser של OCaml, אחר כך קוד שעובר type check כשמחילים אותו על revision בסיס, ובשלב הגבוה ביותר קוד שמתקמפל ועובר טסטים [9:08].

בשביל זה בנו את CES (Code Evaluation Service). זה מעין build service עם שינוי שמאיץ אותו: build "חם" שעומד על revision ירוק, ו-workers שכל היום מקבלים diffs מהמודל, מחילים אותם ומדווחים אם ה-build נשאר ירוק או הפך לאדום [9:08][10:41]. אחרי חודשים של שימוש, המודל התיישר לכתוב קוד שמתקמפל ועובר טסטים. אותו מנגנון משמש גם ל-evaluation: שומרים בצד חלק מנתוני ה-RL, נותנים למודל בעיה ובודקים אם הקוד שלו עובד [10:41].

— John Crepezzi, על מודל code review שאומן חודשים על דוגמאות של בני אדם [10:41] It's fun for a bit and it came back with something along the lines of, hmm, I'll do it tomorrow

המסקנה שלו מהסיפור: הערכות משמעותיות הן "kind of a cornerstone" כדי שמודלים לא יירדו מהפסים ושלא יתבזבזו זמן וכסף [10:41].

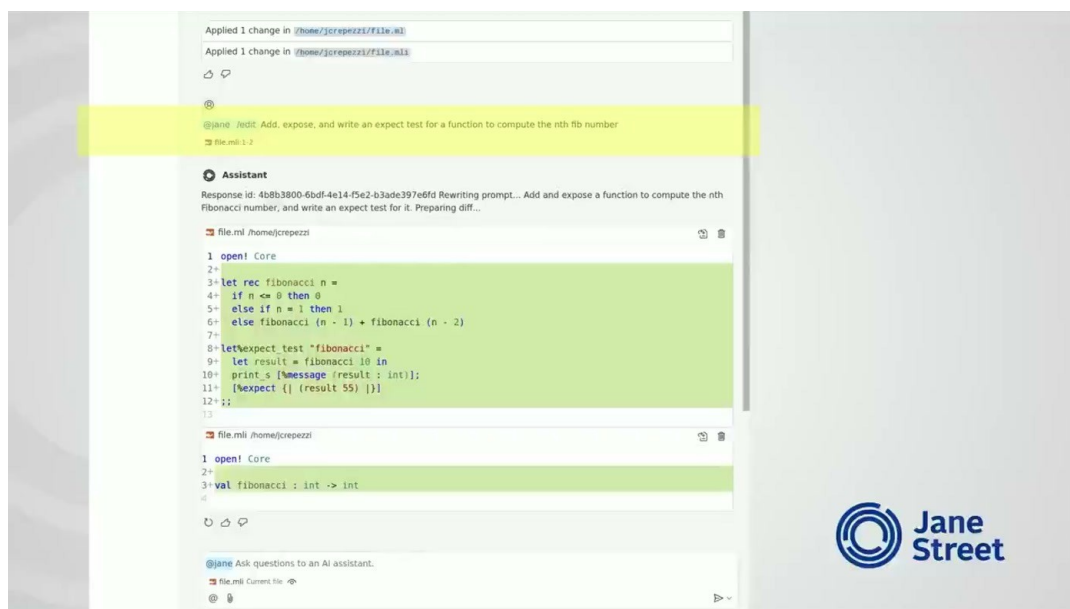
5. אינטגרציה לעורכים: AIDE



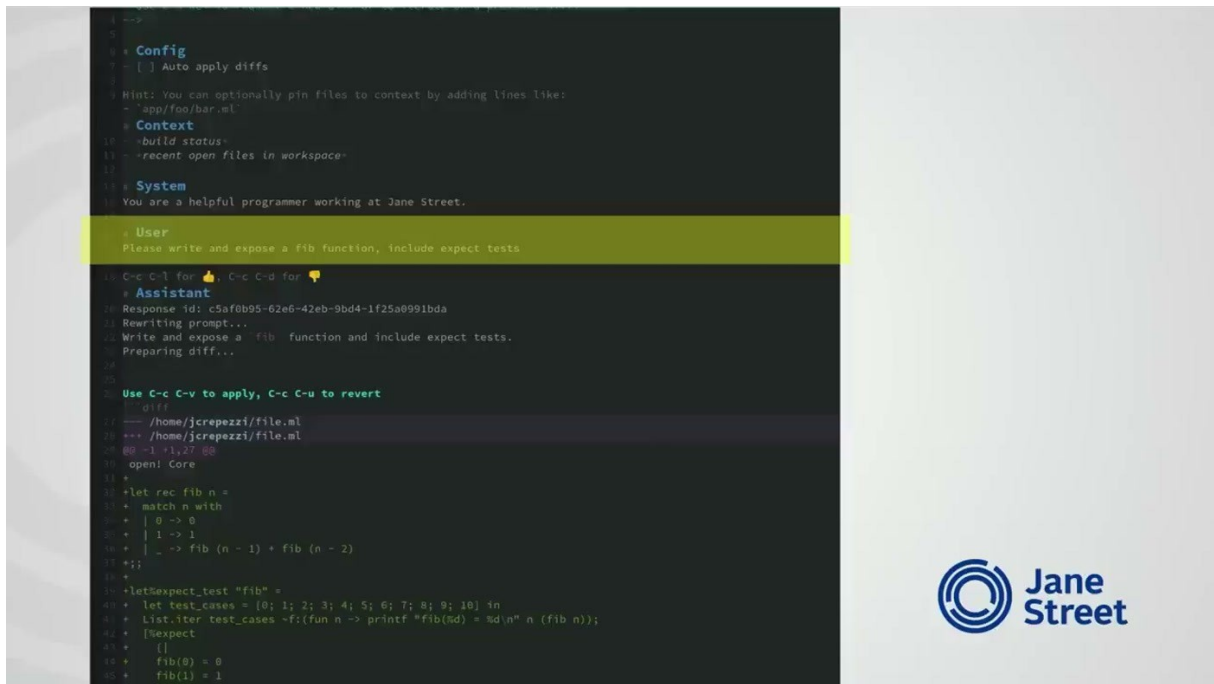
ארכיטקטורת LLMs: AIDE בענן מחברים ל-AIDE שרץ על מכונת המשתמש, וממנו יוצאות שכבות דקות ל-Emacs, NeoVim ו-VS Code

שלושה עקרונות הנחו את האינטגרציה [12:15]. הראשון: לא לבנות את אותו דבר שלוש פעמים, כלומר אסטרטגיות context ו-prompting נכתבות פעם אחת לשלושת העורכים. השני: גמישות, כי כבר אז היה ברור שיגיע מודל fine-tuned ושיהיה צורך להחליף מודלים או prompting. השלישי: איסוף מדדים כמו latency והאם ה-diffs באמת חלים.

התוצאה היא AIDE (AI development environment). AIDE בונה את ה-prompt ואת ה-context ורואה את סטטוס ה-build, ומעליו יושבת שכבה דקה לכל עורך. הוא רץ כ-sidecar על המכונה של המפתח, ולכן לא צריך לשנות את העורכים ולקוות שאנשים יפעילו אותם מחדש: "So we restart aid and then everyone gets the most recent copy." [13:45]



AIDE בתוך VS Code: סרגל צד בסגנון Copilot שמחזיר diff על כמה קבצים (דוגמה של פונקציית fibonacci ב-OCaml עם טסט)



AIDE ב-Emacs: buffer של Markdown עם בלוקים של Config, Context, System, User, ו-diffs שמתווספים בסוף ה-buffer בעזרת keybinds

ב-Emacs המימוש שונה בכוונה. המפתחים שם רגילים לעבוד עם text buffers, ולכן החוויה בנויה כ-buffer של Markdown: אפשר לזוז בו ולהעתיק ממנו כרגיל, ו-keybinds מוסיפים תוכן בסופו [13:45]. מה שהארכיטקטורה מאפשרת (לפי השקף "AIDE Pluggability"): החלפת מודלים, שינויים ב-context building וב-prompting, תמיכה בעורכים חדשים ("something we're actually just doing right now"), וכלים ייעודיים שצוותים בחברה מוסיפים ומופיעים אוטומטית בכל העורכים. היא גם מאפשרת A/B testing, למשל 50% מהחברה על מודל אחד ו-50% על מודל אחר, ובחירה לפי [15:17][13:45] acceptance rate.

— **John Crepezzi [15:17]** Every time something changes in large language models, we're able to change it in one place downstream of the editors and then have it available everywhere

6. מסקנות ושאלות פתוחות

- **להגדיר משימה צרה לפני שמאמנים:** תוצר מדיד (diff שחל ועובר type check, עד כ-100 שורות) קובע אילו נתונים צריך לאסוף. "להראות למודל את הקוד שלנו" זו לא משימה.
- **לאסוף נתונים בצורת השימוש בפועל:** אם הבקשות בעורך קצרות ומעורפלות, תיאורים של PR לא ילמדו את המודל להבין אותן. כדאי לחפש אותות טבעיים, כמו מעברי build מאדום לירוק, ולא ארטיפקטים של תהליך.
- **לבנות קודם את סביבת ההערכה:** build חם שמחיל diffs ומחזיר אדום או ירוק משמש גם כ-reward ל-RL וגם כ-eval. בלי הערכה כזו אפשר לגלות אחרי חודשים שהמודל עונה "מחר".
- **להפריד את הלוגיקה מהעורך:** sidecar אחד עם שכבות דקות חוסך מימוש כפול, מאפשר לעדכן במקום אחד ומאפשר A/B בין מודלים.
- **להתאים את ה-UX לתרבות של כל עורך:** סרגל צד ב-VS Code ו-buffers של Markdown ב-Emacs. ה-backend משותף, הממשק לא.

שאלה פתוחה	למה זה חשוב
כמה המודל המותאם טוב יותר ממודל ציבורי ב-OCaml?	לא הוצג אף מספר של Jane Street, רק הטבלאות מהמאמר של Meta.
מה היה מודל הבסיס ומה עלה האימון?	בלי זה אי אפשר להעריך אם הגישה משתלמת לארגון קטן יותר.
איך משוקלל ה-reward בין type check, parse וטסטים?	זו ההחלטה שקובעת מה המודל לומד בשלב ה-RL.
כמה דוגמאות הניב ה-snapshotting, ואיך מסננים רעש?	צילום כל 20 שניות מייצר הרבה מעברים, ולא כל ירוק-אדום-ירוק הוא שינוי בעל משמעות.

שאלה פתוחה	למה זה חשוב
איך מטופלים פרטיות והסכמה של מפתחים בצילום תחנות העבודה?	הנושא לא עלה בהרצאה בכלל.
מה הממצאים מה-A/B testing ומ-acceptance rate?	המנגנון תואר, אבל לא התוצאות.
איך זה מתרחב ל-RAG, multi-agent ו-reasoning models?	הוזכרו רק כעבודה שנמשכת [15:17], בלי פרטים.

7. מילון מונחים

מונח	משמעות
OCaml	שפה פונקציונלית עם טיפוסים סטטיים. ב-Jane Street כותבים בה כמעט הכול
Fine-tuning	המשך אימון של מודל קיים על נתונים ייעודיים
Supervised training (SFT)	אימון על דוגמאות של קלט ופלט רצוי, כאן context, prompt ו-diff
Reinforcement learning (RL)	אימון לפי תגמול. כאן התגמול הוא האם ה-build והטסטים עברו
Reward function	הפונקציה שקובעת את התגמול על פלט של המודל
Diff	שינוי בקוד, אולי בכמה קבצים, שמוחל על revision בסיס
Type check	בדיקת טיפוסים של הקומפיילר. ב-OCaml זה סף ראשון לקוד תקין
MLI	קובץ ממשק של OCaml, דומה ל-header
Monorepo / Mercurial	מאגר קוד אחד לכל החברה, שמנוהל ב-Mercurial ולא ב-Git
Iron / Feature request	מערכת ה-code review הפנימית. feature מקביל ל-pull request
Workspace snapshotting	צילום תקופתי של תחנת העבודה וסטטוס ה-build כדי לחלץ ממנו דוגמאות אימון
CES (Code Evaluation Service)	שירות שמחיל diffs על build חם ומחזיר אם הוא ירוק או אדום, ל-RL ול-eval
Evaluation (eval)	מדידה של ביצועי המודל על נתונים שהוחזקו בצד
AIDE	AI development environment: שירות מקומי שבונה context ו-prompt ומשרת את כל העורכים
Sidecar	תהליך שרץ ליד האפליקציה על אותה מכונה ומספק לה שירות
A/B test / Acceptance rate	חלוקת משתמשים בין שתי גרסאות ומדידה של שיעור ההצעות שהתקבלו
RAG	שליפת מידע רלוונטי והזנתו ל-prompt
Multi-agent workflows	תהליכים שבהם כמה agents עובדים יחד על משימה