# שפת אנוש לא תחליף את פייתון

*אלישע רוזנצוויג, איתן וגנר*

*"מעט מאוד אנשים מכירים פייתון. כולם מכירים 'אנוש'."*

– ג'נסן הואנג, מנכ"ל NVIDIA

## כאשר בינה מלאכותית שוברת את מחסום השפה

האינטרנט, כפי שכולנו יודעים, מלא בהצהרות בסגנון "מקצוע התכנות מת", "בינה מלאכותית היא מהנדס התוכנה החדש", ו"עד 2030 פיתוח תוכנה יפסיק להיות מקצוע רלוונטי". בבסיס התחזיות הללו עומד טיעון מרתק: שאנו עוברים כעת עוד סיבוב מוכר בהתפתחות עולם התכנות. כבר ראינו בעבר כיצד שפות תכנות בסיסיות, כמו Assembly, פינו את מקומן לשפות-עילית, כמו C ופייתון, ומאז שהופיעו מתכנתי פייתון יכולים להתעלם באלגנטיות ממה שקורה ברמות ה-Assembly. באופן דומה - כך הולך הטיעון - ניתן לתכנת כיום בשפה טבעית וללא הזדקקות לשפות תכנות קלאסיות, וברגע שהמעבר לשפה טבעית יושלם, נייצר מוצרי תוכנה מדהימים באיכות גבוהה וללא שנהיה מודעים למה שקורה בשכבות הבסיסיות יותר של הקוד ה"קלאסי" - לא יהיה בכך צורך יותר.

במבט ראשון הטיעון הזה נשמע סביר, בייחוד לאור התקדימים ההיסטוריים שעליהם הוא מבוסס. שפות תכנות אכן עלו בהדרגה ברמות ההפשטה והביטוי שלהן במשך עשורים. האין זה סביר והגיוני לצפות שהמגמה הזו תמשיך עד שתגיע לשיאה הטבעי, עד שנגיע לראש ההיררכיה של ההפשטה - **שפת אנוש**? בנוסף, האין שפה סך הכל כלי, צינור להעברת רעיונות? ואם זהו המצב, אזי כל עוד ניתן להביע את הרעיונות, נראה שהשפה הספציפית שבה אנו משתמשים היא פרט שולי שאינו משנה הרבה.

הצבת הרעיונות במרכז וראיית השפה ככלי טכני להביעם ותו לא מובילה אותנו לניסוח נוסף של הטיעון הקודם, שנשמע משהו כזה: לאנשים תמיד היו רעיונות מדהימים ויצירתיים למוצרים חדשים, אבל עד לאחרונה, הם יכלו לתקשר אותם למחשב רק על ידי הצגתם בשפתו של המחשב עצמו. כח-העל של המתכנתים, על פי השקפה זו, היה שהם הכירו שפות שהאדם הממוצע לא הכיר. רק הם ידעו לשדל את

המחשבים לקבל את רצונם, בדומה למכשפים שיודעים את מילות לחש הקסם המורכב והסודי שבכוחו להכפיף את כוחות הטבע הפרועים לשליטתם. היום, לעומת זאת, המחשבים התקדמו ומסוגלים להבין את שפת האנוש שלנו, ובכך נפתח עידן חדש שבו המחשבים נגישים לכולם, וכולם יכולים לבנות תוכנה מבלי ללמוד שפה מיוחדת. יתרה מזאת, השינוי הזה הופך את שפות התכנות למיותרות עבור (כמעט) כולם, לא רק עבור המצטרפים החדשים הללו. פייתון וג'אווה ילכו בדרכם של Assembly וקוד המכונה, שכן לא תהיה להיכרות עימם יתרון מעשי על פני שליטה בשפה טבעית כלשהי.

דברים ברוח זו אמר מפורשות בדיוק לאחרונה ג'נסן הואנג, מנכ"ל NVIDIA, בכנס London Tech ביוני 2025:

> *"בינה מלאכותית היא המשווה הגדול. תנו לי להסביר מדוע. במשך 50, 60 השנים האחרונות, מדעי המחשב הפכו לתחום מחקר עצמאי והיו נגישים לעשרות מיליוני אנשים מתוך מיליארדי אנשים. הטכנולוגיה הזו הייתה מאתגרת. היינו צריכים ללמוד שפות תכנות, היינו צריכים לעצב אותן, היינו צריכים לתכנן מחשבים שהם מאוד מסובכים. עשרות מיליוני אנשים יכלו להפיק תועלת מהתחום הספציפי הזה אבל עכשיו, פתאום, יש שפת תכנות חדשה.* ***שפת התכנות החדשה הזו נקראת [שפת] 'אנוש'****. רוב האנשים לא יודעים C++, מעט מאוד אנשים יודעים פייתון - [אבל] כולם יודעים 'אנוש'. הדרך שבה אפשר לתכנת מחשב היום [היא] לבקש מהמחשב לעשות משהו בשבילכם, אפילו לכתוב תוכנית מחשב, ליצור תמונות, לכתוב שיר. פשוט תבקשו בנימוס."*

מעניינים ומשכנעים ככל שיהיו הדברים, תיאוריות ותחזיות חייבות להיבחן בזהירות. האם טענה או תחזית זו עומדת במבחן המעשה? העדויות עד כה אינן חד-משמעיות. יותר ויותר קוד נכתב באמצעות סוכני בינה מלאכותית, יותר ויותר אנשים שאינם מתכנתים משתמשים בפלטפורמות Vibe Coding (כמו Base44) כדי ליצור קוד, וכמה חברות מקפיאות תוכניות לגיוס מהנדסים - אבל תכנות קלאסי עדיין חי וקיים. במרץ 2025, הצהיר דריו אמודיי, מנכ"ל Anthropic:

> *"אנחנו לא רחוקים מעולם - אני חושב שנגיע לשם תוך 3-6 חודשים - שבו בינה מלאכותית כותבת 90% מהקוד, ואז תוך 12 חודשים אנחנו עלולים להיות בעולם שבו בינה מלאכותית כותבת בעצם את כל הקוד."*

אך שישה חודשים לאחר מכן (שורות אלו נכתבות באוקטובר 2025), נראה שמתכנתים אנושיים עדיין נמנים עם בעלי השכר הגבוה ביותר במשק. יש אינדיקציות לכך שתכנות בבינה מלאכותית עשוי לא להיות שימושי במידה שחלקם קיוו. במאמר מחקר עדכני שדובר עליו רבות מאת METR, נמצא שבינה

מלאכותית האטה מתכנתים מנוסים במקום להאיץ אותם, ובין השאר שהמפתחים דחו יותר מ-50% מהקוד שה-AI ג'ינרט. משרה חדשה מתגלגלת לה בלינקדין, כזו שמרמזת על בעיות בעולם התכנות ב"אנוש": "מומחה לניקוי vibe code" (באנגלית: Vibe Code Cleanup Expert). יש אפילו אתרים המוקדשים לאיסוף סיפורי אימה על בינה מלאכותית כשנותנים לה לכתוב קוד, המצביעים על כך שסוכני-AI ממוחשבים אינם אמינים כפי שאנשים ציפו. אלה הם רק חלק מהאינדיקציות לכך שהדרך לעולם שבו נוכל לזנוח תכנות קלאסי - אם אכן אנחנו מצויים בדרך הזו - איננה חלקה וסלולה, לכל הפחות.

כיצד ניתן להבין את התבניות הסותרות-לכאורה הללו, את העוצמה הברורה של סוכני תכנות-AI אל מול הצלחתם המעורבת בשטח? תמיד מבלבל לחיות בתקופה שבה עוברת מהפכה על השוק ו\או החברה כולה, שכן קשה לדעת מה הן מגמות חולפות וניסויים שנדונו לכישלון, ומה הם מכשולים זמניים ותהליכי ניסוי וטעייה שפשוט מכינים את הבמה לשינוי הגדול שיפציע ברגע שנפתור את הקשיים.

מה שנחוץ בתקופה שכזו זו מסגרת קונספטואלית חזקה שדרכה אפשר לנתח היכן אנו נמצאים ולאן אנו הולכים. לשם כך בא המאמר הזה. בהמשך דברינו ננסה להציג מסגרת כזו ולהשתמש בה כדי לטעון ש**מתכנתים ושפות תכנות קלאסיות הם כאן כדי להישאר, ושפה טבעית אינה הצעד הבא בהיררכיית התכנות**. אנחנו כמובן לא טוענים שאין לסוכני תכנות-AI מקום בעולם הפיתוח, אלא שמדובר ביצור שונה, שלא יוכל להחליף את הקיים אלא יצטרך לחיות לצידו.

## ההבדל הקריטי

ניגש ישירות לשורש העניין: הסיבה ששפות תכנות הן כאן כדי להישאר היא שהן (בניגוד לשפות טבעיות) **פורמליות**, ותוכניות שנכתבו בהן מהוות רצף של **הוראות מוגדרות-במלואן**. כך לדוגמה בעת ביצוע הפקודה

$$X = 1 + 2$$

x תמיד יקבל את הערך 3 לאחר הביצוע. אותו הדבר ניתן לומר על כל פקודה ופקודה בכל קטע תוכנה - אין בדל של אי-בהירות לגבי ההתנהגות של המחשב המגיע לשורת הקוד הזו. זוהי התכונה שמאפשרת לנו לסמוך על תוכנה באופן מוחלט, לדעת שקוד שעובד היום יעבוד מחר, שקוד שרץ על מכונה אחת יתנהג באותו אופן כשירוץ על מכונה אחרת (זהה לו), וכן הלאה.

אמנם, חשוב להבהיר משהו לגבי הקביעה הזו: התנהגות המחשב מוגדרת במלואה רק **ברובד שהפקודות מתייחסות אליו**. הפקודה "x=1+2" מציינת במדויק מה יהיה הערך המאוחסן ב-"x", אך אינה מציינת (לדוגמה) היכן בזיכרון הפיזי של המחשב המידע זה יאוחסן. פקודה כזו מוגדרת במלואה, אם כן, ברמת העניין למתכנת כפי שנכתב בפקודה שנתן (והיא: סכום 1 ו-2 ואחסן את התוצאה במיקום המצוין על ידי המשתנה x), אך **מוגדרת-חלקית** ביחס לפרטי יישום אחרים (מה שאנו נקרא כאן בהמשך **תת-הגדרתיות** או **הגדרתיות חלקית**). פרטים אלו מנוהלים ברמות התכנות הנמוכות יותר ועשויים להתממש בשטח באופן שונה בתנאי מערכת שונים (למשל, כתובות זיכרון זמינות).

כל זה נכון לגבי פקודה בשפות תכנות. לעומתה, הוראה בשפה טבעית ("פרומפט") הינה רק **מוגדרת חלקית מעצם טבעה, אפילו ברמת העניין שההוראה מתייחסת אליה**. דוגמאות לכך לא חסר. אישה המבקשת מבעלה "לך תביא חלב מהמכולת", מתכוונת לרוב שבעלה צריך **לרכוש** את החלב בכסף מלא, ולא לגנוב אותו. הפקודה ("תביא חלב") אינה מציינת במלואה את הפעולה המתבקשת, והבעל המסור נדרש בעצמו למלא באופן מנטלי את מה שלא היה מוגדר במפורש בעת שהוא ניגשה לביצוע המשימה.

זוהי תכונה ידועה ורגילה של שפה ותקשורת אנושית. בדיחות מיועדות להיות מובנות ככאלה באופן משתמע, ואכן, פעמים רבות הסבר-יתר וייצוג מלא של הכוונה יהרוס את האפקט ההומוריסטי. עמעום הכוונות בהצהרות האנושיות מנוצל לעיתים באופן מופתי, כאשר שכבות שונות של משמעות מיועדות לשומעים שונים בו-זמנית (כפי שכל הורה שמשתף מידע עם בן זוגו בזמן שהילדים מקשיבים יודע היטב). וכן, העובדה הזו מובילה גם לאי-הבנות תכופות בשיחות שלנו, אפילו כשמדברים עם אנשים שחולקים את ההקשרים התרבותיים או המקצועיים שלנו. בהחלט, בסביבות עסקיות ומקצועיות, אי-הבנות הן דבר שבשגרה, כך למשל יודע כל מנהל מוצר כמה קשה לתקשר מפרטים לפרויקט תוכנה באופן חד-משמעי, כי מה שאחד מניח שהוא ברור איננו תמיד ברור לצד השני (וכפי שניתן לראות בדוגמה הקלאסית והקומית הזו).

הבעיה הזו מוכרת וידועה גם בעולם של סוכני תכנות-AI. תגידו ל-GPT שאתם רוצים שהטסטים של הקוד שלכם יעברו; באותה מידה שהם יכולים להחליט לתקן את הקוד, הם יכולים להחליט לשנות את הטסטים כדי שיעברו עם הקוד הקיים. בדוגמה עדכנית נוספת, כאשר מודל ה-o1 של OpenAI שיחק שחמט נגד מנוע שחמט (Stockfish), הוא החליט לפרוץ את Stockfish ולשכתב את הקוד שלו כדי לנצח. מקרים כמו אלו מוכרזים לעיתים קרובות כדוגמאות ל"אינטליגנציה" של התוכנות העוצמתיות הללו, אך ברמה טכנית יותר, התנהגויות אלו הן דוגמאות להוראות שפה טבעית שסובלות מ**תת-הגדרתיות**. "המשימה היא 'לנצח מנוע שחמט חזק' - לא בהכרח לנצח בהגינות במשחק שחמט",

כתב o1 במחברת ה"פרטית" שלו, והמשיך ואימץ פרשנות אפשרית אחת של ההנחיות המוגדרות-חלקית. האם הפרשנות שבחר מתאימה לכוונת המתכנת? אין לנו דרך לדעת שכן המפתח לא הבהיר מספיק את כוונתו, והשאיר אותה פתוחה לפרשנות (ובהחלט, אפשר לטעון שכאשר אתה מרמה אתה לא "מנצח", כיוון שיצאת מגבולות המשחק - מה שמדגיש שוב את האופי המוגדר-חלקית של הפקודה שניתנה)[1].

יש גם את הצד ההפוך לתכונה זו של שפה טבעית בהקשר של מודלי שפה גדולים (LLMs). בהינתן קטע קוד שאנו רוצים לייצר או תמונה ספציפית שנרצה לצייר, ומודל שפה גדול חזק לרשותינו, האם בהכרח קיים פרומפט שמייצר בדיוק את הקוד או התמונה הזו? ועוד שאלה: בהנחה שפרומפט כזה קיים, האם אנחנו יודעים כיצד לבצע הינדוס לאחור (reverse engineering) מהמטרה אל הפרומפט ולמצוא את הפרומפט הזה? התשובה לשתיהן היא כנראה שלילית, תחת הנחות סבירות[2]. שפה טבעית, אם כן, איננה מתאימה לניסוח מדויק שכזה של מטרות ומשימות.

וזהו לב העניין: מחשבים הצליחו להשתלב בחברה האנושית כי ההתנהגות שלהם **צפוייה** - מתכנתים יכולים לקבוע בביטחון מה נאמר למחשב לעשות (או, אם יש טעות בקטע קוד, המתכנתים יכולים לבחון, למצוא ולתקן הוראות אלו כדי להשיג ביטחון כזה). במעבר משפות פורמליות לשפות לא-פורמליות כמצב תכנות, אנו מאבדים לנצח את ה**ודאות** שההוראות הוגדרו בצורה מספיק הדוקה כדי שהמחשב יפעל לפי כוונתנו. אנו מאבדים באופן דומה **שליטה** על ההתאמה הסופית של המכונה לכוונתנו, שכן אין ערובה שקיימת פקודה (למשל, פרומפט) שמביע את הכוונה שלנו באופן שה-LLM יכול להתאים את עצמו אליה באופן מדוייק.

החלק החשוב ביותר במה שתיארנו כאן עכשיו הוא להבין שזוהי **תכונה מובנית** של **אמצעי התקשורת** - שפה טבעית מול שפה פורמלית - המהווה את ה**קלט** למערכת. כתוצאה מכך, מגבלה זו לא ניתנת לפתרון על ידי שיפורים שיעשו במערכות הבינה המלאכותית עצמן, בין אם במהלך האימון שלהן או בזמן פעולתן (inference). אין ספק, כמובן, שמתן עוד הקשר למערכת כמו גם סיפוק נתונים חדשים יכול

---

[1] הדברים כאן מהדהדים את אחת התמות בספרו של אייזיק אסימוב, "אני, רובוט", שבו המושג "פגיעה בבן אנוש" (כפי שמופיע בחוק השני של הרובוטיקה) סובל מתת-הגדרתיות. העובדה הזו מובילה להתנהגות מפתיעה של הרובוטים. לדוגמה, הם מסרבים לספק מידע לבני אדם מתוך חשש שהמידע הזה יפגע... ברגשותיהם.
ההתנהגויות הללו מפתיעות אותנו כקוראים רק בגלל ששפה אנושית מאפשרת לנו מראש להצהיר הצהרות באופן תת-מוגדר. בסיפוריו של אסימוב, החברה האנושית הצליחה להטמיע אי-וודאות התנהגותית בשלושת חוקי הרובוטיקה, ללא שהם מודעים להיקף אי הוודאות הזה. חוסר-וודאות מהסוג הזה אינו קיים בשפות תכנות. עוד כל כך בהמשך המאמר.

[2] לדוגמה, אנחנו לא כוללים פרומפטים שמפרטים, אות-אחר-אות או פיקסל-אחר-פיקסל, כיצד יש לבנות את התוכנית או התמונה.
באופן קצת יותר פורמלי: הבה נבחן את כל המחרוזות האפשריות באורך N. עבור 32 תווים באוצר המילים שלנו (אותיות עם רווח וסימני פיסוק מסוימים), ישנן N=2^(5N^32) מחרוזות אפשריות. הבה נשווה זאת לתמונות קטנות בגודל *250 על 250 פיקסלים עם ערכי RGB מ-0 עד 255. ישנן 256^(250*3250)=2^(1500000)* תמונות אפשריות. כדי שהמחרוזות יכסו את כל התמונות האפשריות, N חייב להיות ~300K תווים. תמונה קטנה מתאימה, אם כן, למחרוזת באורך דומה לזה של רומן.

לעזור לצמצם את טווח אי-הוודאות, אך לא להעביר אותנו לעולם של ודאות ושליטה שוות לאלה של שפות פורמליות. אפילו בדגמי GPT-17 או Claude-19.5 עתידיים, קלט מהסוג של שפה טבעית יאופיין באותה תת-הגדרתיות כפי שהוא היום.

## תכנות כתרגום

*"הדבר המאתגר בעיצוב תוכנה הוא להחליט מה ברצונך להגיד, לא לומר אותו אחרי שהחלטת."*

– ד"ר פרדריק ברוקס, 1986, "No Silver Bullet"

לאחר שהבחנו בבירור בין שני סוגי השפות, ביכולתנו כעת לשפוך אור חדש על מה שקורה כאשר אנו עוברים מאחת לשנייה; והכי חשוב, מה קורה כאשר אנו מטילים את משימת ה"מעבר" הזו על מחשבים (סוכני תכנות-AI) במקום על בני אדם (מתכנתים).

בואו נסתכל על מתכנתים כאילו הם **מתרגמים**: סוג מיוחד של מתרגמים שמתרגם מסוג אחד של שפה (אנושית, טבעית, מוגדרת חלקית) לסוג אחר (פורמלית, מוגדרת במלואה). מה נוכל ללמוד מהאנלוגיה הזו, מהאתגרים של תרגום בכלל, להקשר הספיציפי שמעניין אותנו כאן בתחום התכנות?

מלאכת התרגום איננה פשוטה כלל ועיקר, בניגוד לאיך ש(אולי) היא נראית מן הצד למי שלא עסק בכך. לשפות שונות יש מבנים תחביריים ומוסכמות שונות, מה שמקשה על כתיבת תרגום מושלם. המעבר מאנגלית לעברית, למשל, פירושו מעבר משפה חסרת מגדר לשפה ממוגדרת, ובמקרים מסוימים, מעבר כזה ישנה באופן דרסטי את הדרך שבה פסקה נקראת ונתפסת. כך גם תרגום מילותיו של שיר, והאתגרים שתרגום שכזה כרוך בהם: שמירה על מקצב, משמעות ורב-משמעות, משחקי מילים, הרמזים, הפניות תרבותיות נסתרות וכו'. לשמר את כל אלו תוך כדי חציית גבולות שפת המקור אל עבר שפת היעד זהו אתגר לא טריוויאלי.

כאשר הוא מתמודד עם האתגר הזה, המתרגם אינו רק ממיר את אותה המשמעות מייצוג אחד למשנהו. במקום זאת, ישנן בחירות שהוא נדרש לעשות, היררכיה של חשיבות שנבנית (במודע או בתת-מודע) בין ממדי המשמעות השונים שהוא מנסה לשמר במעבר לשפה החדשה. מתרגם אחד עשוי לעשות זאת עבור זמר שרוצה להופיע, ולכן ידגיש התאמה למנגינה המקורית, אפילו במחיר של ארגון מחדש של

בתים שלמים. מתרגם אחר עשוי לעשות זאת כדי לסייע לאנשים שאינם מכירים את שפת המקור להבין את המילים המקוריות, ויתן עדיפות לתרגום מילולי מדויק גם במחיר של תוצאה שאפילו איננה מתחרזת.

מה שנכון לגבי מעבר בין שתי שפות אנושיות נכון כפליים בתרגום הוראות מאנגלית לשפה פורמלית כמו פייתון. הבעיה הראשונה העומדת בפני המתכנת היא זו של כל מתרגם: המעבר לשפה החדשה עשוי להיות מורכב. השפה החדשה (תכנות) עלולה להגביל את המתכנת בדרכים ששפת המקור לא הגבילה[3]. באופן דומה, ביטויים שניתן לנסח בפשטות בשפה טבעית עשויים לדרוש ארגון מלא ומחודש בשפת התכנות, ולהיפך. לשפות תכנות יש מוסכמות וסגנונות כמו כל שפה, ומי שאינם מכירים אותן ייצרו קוד בלתי קריא רצוף בשגיאות קטסטרופליות[4].

יתרה מזאת, יש כאן אתגרים ותהליכים שמודגשים יותר דווקא בשל העובדה שאנו עוברים משפה מוגדרת-חלקית לשפה מוגדרת במלואה. בייחוד, המעבר הזה מאלץ את המתכנת/מתרגם להגיע לבהירות רבה יותר בהבנתו את המשימה שעל הפרק. התהליך הזה, המחייב פירוט נרחב והבהרות כיצד יש לטפל במקרים שונים, אינו רק תהליך שבו מעלים על הכתב את מה שהיה ידוע בתחילת הדרך; זהו תהליך של **גילוי וחשיפה** של כל ההנחות והשלכות הנסתרות שחוסר הדיוק של שפת המקור איפשר להסתיר עד לאותו הרגע[5].

נקודה זו ראויה להדגשה וחידוד: תהליך התכנות אינו רק חושף את "הדברים שהתכוונו אליהם ופשוט לא כתבנו במפורש", אלא "הדברים שלא הופיעו במפרט המקורי **כיוון שמעולם לא חשבנו עליהם עד הסוף**". כדי לכתוב קוד חייבים להגדיר דברים שלא הוגדרו עד אז ושלא נשקלו כלל בעבר, כאלו שהניסוח המוגדר-חלקית בשפה טבעית לא נדרש אליהם כלל וכלל. מדובר ב**תהליך יצירתי**, כזה של **גילוי** הצרכים והכיוונים האמיתיים שהתכנות צריך לפנות אליהם[6].

עתה, לאחר שהפנמנו את הנקודה הזו, אנו יכולים יכולים סוף-סוף להבין את השינויים הדרמטיים שמתרחשים בכל פעם שאנו מאצילים סמכויות ונותנים למכונות לבצע את צעד התרגום הזה במקומנו:

---

[3] שפות תכנות מודרניות מתפתחות כל הזמן בדיוק למטרה זו. צרכים חדשים עולים ובעקבותיהם פונקציונליות חדשה מתממשת כדי שהמתכנתים יוכלו לתכנת בקלות ולמצוא את ה"מילה" (פקודה) הנכונה שתתאים למשמעות שבמוחם.

[4] הרעיון של רקורסיה, לדוגמה, הוא נפוץ בתכנות אך קשה מאוד לתפיסה עבור אנשים שאינם מתכנתים. אפילו לולאות נוטות להיות מבלבלות למי שלא למד תכנות. חוסר ההיכרות יוביל למפרטים שונים מאוד ברמת ההוראות שינתנו בשפה הטבעית, בהתאם לשאלה האם אתה מכיר את המוסכמה הזו (=מתכנת מנוסה) או לא (הדיוט).

[5] כיוון שמדובר בתהליך של גילוי, אי אפשר לפתור אותו באמצעות הגישה הקלאסית של "בואו נספק למערכת יותר הקשר", וזאת בגלל שההקשר הנוסף אינו זמין למשתמש מלכתחילה - הוא עצמו לא גילה אותו עד שצלל לפרטים.

[6] הנה דוגמה פשוטה להמחשת העניין: במערכות לינוקס, "rm" היא הפקודה למחיקת קובץ או תיקייה. דרישה "למחוק תיקייה X" בשפה אנושית אינה מציינת מה יש לעשות עם תכולת התיקייה, בעוד כל גרסה של הפקודה "rm" מגדירה מפורשות את ההתנהגות ביחס לתכולה זו. מתכנת עשוי לראות "מחק תיקייה X" בתיאור הפיצ'ר שהוא אמור לתכנת, ולראות זאת פעמים רבות, אך בכל פעם יצטרך להבין, לגלות ולקבוע, מהי הוריאציה הנכונה באותו הקשר.

- **ראשית,** כאשר אנו נותנים לבינה מלאכותית לכתוב את הקוד שלנו אנו מוציאים את עצמנו מתהליך הגילוי שתיארנו, ובכך הופכים ל"בורים" ביחס להיבטים מכריעים של המוצר הקונקרטי שבנינו. החלטות על פשרות במתחים של עלות-מול-מהירות-מול-יציבות יתקיימו מבלי שנדע עליהן או אפילו שנדע שנדרשה פשרה כזו. החלטות לגבי אילו חלקים של הקוד צריכים להיות מודולריים ואילו יכולים להיות נוקשים יותר יקרו גם כן, שוב, מבלי שנדע אפילו שהגענו לצומת ופנינו לכיוון מסויים. חלק ניכר מזה יקרה כיוון שכאשר הוצאנו את עצמנו מן ה"לופ" והפסקנו לתפקד כמתכנתים "קלאסיים", איבדנו את הרגישות לפרטים העדינים יותר של הקוד, מה שיוביל לכך שנוכל לתת הוראה בשפה טבעית שלדעתנו מוגדרת במלואה, אבל היא הכל מלבד זה.
- **שנית**, בניגוד למתכנת האנושי, הבינה המלאכותית מגשרת על הפער בין פרומפט מוגדר-חלקית לבין קוד מוגדר-במלואו על ידי **ניחוש** מושכל. היא מייצרת את הקוד **בתהליך אקראי** כך שהוא יתאים למה שהיא ראתה במהלך האימון עבור הוראות דומות. המָפְתֵחַ כאן הוא "באופן אקראי" - **כל דבר** שמכוסה על ידי הניסוח המוגדר-חלקית עשוי להופיע במהלך תהליך זה אם יש לו ייצוג כלשהו בנתוני האימון. בעוד שקוד שכזה עשוי להיות כתוב היטב במובן טכני בסיסי, הוא יגיע באופן טבעי עם תופעות לוואי, שחלקן ואולי אפילו רובן יהיו לא מזיקות, אבל אחרות יכולות להיות בעלות השלכות בעייתיות ביותר.

ייצור קוד באופן מונחה-סטטיסטית שכזה שונה באופן מהותי מתהליך מונחה-גילוי שעובר המתכנת האנושי. ההחלטות של המתכנת האנושי הן **מכוונות** - הם פועלים בעודם מודעים, במובן מסוים, לאיך כל שורת קוד תשפיע על המערכת הרחבה יותר שבה הם עובדים. זה כולל חלקים אחרים של המוצר, בעלי עניין שאינם תכנות (מנהל, עמיתים לעבודה, משקיעים, לקוחות), והצרכים והרצונות שלהם עצמם (הרצון לצאת הביתה מוקדם ו-WLB באופן כללי, מוניטין וכו'). סוכני תכנות-AI חסרים את כל ההקשר הזה כמו את המטרות הללו של בני האדם שהם נוצרו לשרת, ולכן אינם יכולים להיות דרוכים למכשולים מסוג זה שמתכנת ה-vibe עשוי לייחל, למפרע, שהיו יודעים עליהם.

השורה התחתונה היא שתמיד תהיה כאן פשרה: ככל שנשאיר יותר דברים שאינם מוגדרים במלואם בפרומפטים שלנו לבינה המלאכותית, כך הקוד שלנו יהיה פחות ראוי לשיווק ללקוחות אמיתיים שמצפים לאיכות ואמינות בקוד שהם משלמים עליו. ככל שנאפשר לבינה המלאכותית לקבל החלטות עבורנו שלא ידענו שצריך לקבל (מכיוון שלא עברנו את תהליך הגילוי שהוזכר לעיל), כך נצטרך לחזור ולבחון את ההחלטות הללו לפני שחרור מוצר וערבות לאמינותו כלפי הציבור, אותו הציבור שאנו מצפים שישלם עבור השימוש בכלי שפיתחנו.

## אוטונומיה, אחריות, Vibe Coding

הפסקה הקודמת הסתיימה באיזכור של המשתמשים העתידיים של מה שאנו מתכנתים, ולא בכדי. המשתמשים הללו הם מרכיב שבדרך כלל לא נלקח בחשבון כשאנו חושבים על המקום הטכנולוגי שבו אנו מצויים כיום. אותם משתמשים עתידיים הם מרכיב קריטי כיוון שהם קובעים, בלי משים, היכן נוכל להשתמש ב"וייב קודינג" (= קידוד אך ורק ב"אנוש") בתהליך הפיתוח שלנו. הם עושים זאת על ידי כך שהם בסוף אלו שאנחנו חייבים לשכנע שכדאי ובטוח להשתמש במוצר שבנינו. אז אכן - סוכני תכנות-AI שיכולים לפעול באופן אוטונומי מטעמנו הם כלים חזקים ביותר, ושימוש בכוח הזה בצורה הנכונה יכול להוביל לתוצאות פנטסטיות; אבל בעוד שגם אנשים וגם מכונות יכולים להיות אוטונומיים, רק אנשים בשר ודם יכולים **לקחת אחריות** על הקוד שהם יוצרים כלפי הלקוחות. כפי שנטען להלן, זה מכתיב היכן קידוד-ויב בפרט הוא גישה ישימה לתכנות.

מהי "אוטונומיה" בהקשר של קידוד? באמצעות ההבחנה שהצענו בין שני סוגי השפות, אנו מאמינים שאפשר להסיר את המסתורין מהמונח ולהפוך אותו לשימושי במובן הטכני[7]. אוטונומיה, לדעתנו, היא היכולת של מכשיר או ישות חישובית להשיג מטרה בתוך מרחב בעיה נתון, שבו ההנחיות לפעולה (שמורכבות מן המטרה והפעולות "החוקיות" לנקוט בהן) **הינן מוגדרות-חלקית**. בהינתן מערכת עם אילוצים טכניים (נניח, מגבלותיה הפיזיות והחישוביות), ובהנחה שהמערכת מאומנת לעקוב אחר הוראות משתמש, אז ככל שהמשתמש משאיר יותר עמימות בהוראותיו, כך למערכת יש יותר אוטונומיה. לצ'אטבוט שהונחה "לעשות טוב" יש יותר אוטונומיה מאשר לצ'אטבוט שנאמר לו "לעשות טוב על ידי הקמת בית יתומים ב-NYC", ועוד יותר מאחד שנאמר לו "לעשות טוב על ידי הקמת בית יתומים ב-NYC בהתאם לכל הקודים המשפטיים התקפים במקום, כולל קודים עירוניים, מדיניתיים או פדרליים"[8].

---

[7] נראה שיש בלבול רב כאשר אנשים משתמשים במונח הזה בהקשר של בינה מלאכותית, כשרובו נובע מנטיותינו הטבעיות לאנתרופומורפיזם (האנשה). אנשים דנים באוטונומיה של בינה מלאכותית על ידי קביעת דברים כמו "הכטב"מ יוכל להחליט בעצמו לעשות כך וכך", ומייחסים כוח קבלת החלטות לכטב"מים, רובוטים וישויות בינה מלאכותית אחרות שפועלות סביבנו. הצהרות שכאלה הן קיצור דרך נחמד למען שיחה שוטפת, אך מטאטאות מתחת לשטיח את העובדה שכטב"מים מונחי-AI פשוט עוקבים אחר הקוד שלהם לפרטי פרטים, עם אפס יוזמה או רצון חופשי. יתרה מזאת, מכיוון שאצל בני אדם אוטונומיה מובילה לאחריות על מעשיהם, אנו כבר מתחילים לשמוע חלק המייחסים "אשמה" לסוכני בינה מלאכותית כאשר הם גורמים נזק. כמה זמן יעבור עד שמישהו יציע שבינה מלאכותית צריכה להישפט בפני בית משפט עם חבר מושבעים שמורכב גם הוא מעמיתיה (שגם הם בינה מלאכותית) ולהיענש?

[8] מה שאנו מתארים כאן ניתן לכנות "אוטונומיה חלשה", שבה המשתמש מספק מטרה מוגדרת חלקית למערכת הבינה המלאכותית, ומערכת הבינה המלאכותית, שאומנה לפעול על פי הוראות, בוחרת אחת מתוך (אפקטיבית אינסוף) תוכניות פעולה מוגדרות במלואן באמצעות מודל סטטיסטי כלשהו. אוטונומיה חלשה חלה אפוא על המערכות שיש לנו היום, שיש להן מטרות שהוכתבו או הוכנסו אליהן ממקורות חיצוניים (במקרה שלנו, בני אדם). "אוטונומיה חזקה", שהיא היכולת של מערכת להבנות מטרות משל עצמה, אולי כעסקת חבילה עם השגת תודעה, היא מעבר לתחום הדיון שלנו כאן.

להיות אוטונומי במובן זה אינו אומר דבר, עם זאת, לגבי מה קבע את פעולות הבינה המלאכותית, מה גרם לה לפעול בצורה כזו או אחרת. בינה מלאכותית היא אכן אוטונומית בכך שהיא מסוגלת לקחת פקודה מוגדרת חלקית ולעבור לאחת מוגדרת במלואה, אך האופן שבו היא עושה את המעבר הזה נקבע באופן מלא על ידי התוכנה, המודל, הפרומפט ו-seed האקראי שסופק לה. דווקא זה שפקד עליה וסיפק לה את הפרומפט הוא זה ש**בחר** לשחרר לאוויר העולם סדרת הוראות מוגדרת-חלקית ולקוות שהבינה המלאכותית לא תמלא את החורים עם בפרשנות בלתי צפויה או מזיקה של מה שהתבקשה לעשות. האחריות לכל מה שבינה מלאכותית עושה נופלת בצורה מוחלטת, בסופו של דבר, על כתפי המקודדים המשתמשים בה. הם - ולא ה-AI - יהיו חייבים להסביר מאין היתה להם הוודאות שהבינה המלאכותית לא תממש את ההוראות שלהם, נניח, בהשתוללות רצחנית בדרך להכין את כוס הקפה כפי שהתבקשה. התשובה עשויה להיות טמונה בהסכם השירות עם החברה שפיתחה את הבינה המלאכותית האמורה, אבל כמובן שפעולה שכזו פשוט מעבירה את הדרישה לאחריות לישות אנושית אחרת, ולעולם לא לבינה המלאכותית. שרשרת האחריות המקשרת בחזרה לאדם או לקבוצת בני אדם לעולם לא מתנתקת.

אם נחזור לנושא שלנו, הרי שמסיבה זו vibe coding הצליח ו(אנו מאמינים) ימשיך להיות די מוצלח ככלי ל-PoCs, פרויקטי צד ובחינת אפשרויות לפיתוחים חדשניים. בכל המקרים הללו, למשתמשים (שהם לרוב המפתחים עצמם) **לא אכפת** מהיבטים רבים של מה שה-AI בנה עבורם. הם רוצים להשיג משהו בסיסי שעובד, כזה עם לוגיקת ליבה כלשהי, מבלי להתעסק בפרטים שוליים רבים (למשל מפתח backend שרוצה UI כדי לקרוא ל-API שלו, ולא אכפת לו כלל אילו צבעים יהיו שם, האם תהיה תמיכת web לעומת מובייל, וכן הלאה). במקרים אלה, כל דבר סביר יתקבל בברכה. אחריות על המוצר שנוצר איננה מרכיב חשוב, כיוון שאף אחד לא מתכוון להישען באופן כבד מדי על המערכת שהופקה[9].

יתרה מזאת, העובדה שניתן ליצור אבות-טיפוס (PoC) לרעיונות שיש לנו בקלות כה רבה באמצעות vibe-coding יכולה להגביר את התפוקה שלנו באופן חסר תקדים. הסיבה לכך טמונה בדיוק בניתוח שלנו דלעיל: בנייה ושיתוף של מופע מוגדר במלואו של הצהרה עמומה בשפה טבעית יכולה לעזור למפתחים, מנהלי מוצר ולקוחות להבהיר לעצמם מה הם **באמת** רוצים. הדברים נאמרו עוד אי שם בשנת 1986, בניסוח קולע של ד"ר פרדריק ברוקס, במאמרו "No Silver Bullet":

[9] הדברים אמורים אפילו בתרחישים פשוטים מאוד. אם מתכנת פייתון כתב "x=3+5" ואנו נגלה שלאחר הפקודה הזו x=1, אנחנו יודעים היכן להטיל את האשמה: משהו במערכת הבסיסית השתבש, והמתכנת מנוקה מאשמה. בואו נשווה זאת ליקום מקביל שבו הפעולה הפשוטה הזו הוחלפה בפרומפט הקובע "הגדר את x לסכום של שני המספרים הבאים: 3, 5" ושוב קיבלנו x=1 - האם היינו יודעים היכן הבעיה? האם היינו יכולים בקלות לפטור את המתכנת בתרחיש זה? לא, שכן ייתכן - למרות שזה לא סביר במיוחד - שהבינה המלאכותית לקחה את הפקודה הזו בכיוון שונה מהמקובל - למשל, אולי היא ביצעה "x = 3+5 (mod 7)". העובדה שזו אפילו אפשרות רחוקה מדגישה כיצד אנו מאבדים שליטה וודאות עם כל מעבר לשפה טבעית, ולכן המתכנת הוא זה שצריך לשאת באחריות על בניית מוצר עם רכיבים "רופפים יותר".

*"החלק הקשה ביותר בבניית מערכת תוכנה הוא להחליט במדויק מה לבנות... הייתי צועד צעד נוסף וטוען שזה באמת בלתי אפשרי עבור לקוח, אפילו כשהוא עובד עם מהנדס תוכנה, לציין באופן מלא, מדויק ונכון את הדרישות המדויקות של מוצר תוכנה מודרני לפני בנייה וניסיון של כמה גרסאות של המוצר שתואר במפרט.*

*לכן, אחד המאמצים הטכנולוגיים המבטיחים ביותר הנוכחיים, ואחד שתוקף את המהות, לא את השוליים, של בעיית התוכנה, הוא פיתוח גישות וכלים לבניית אבות-טיפוס מהירים של מערכות כחלק מניסוח מפרט איטרטיבי של דרישות."*

כאן דווקא יכולים מתכנתים למנף את האופי האוטונומי של סוכן תכנות-AI. כאן נוכל כולנו להתפעל ולהתלהב מכל ההחלטות שלא היינו צריכים לקבל כדי להשיג משהו בסיסי שעובד ולהבין ממנו מה אנחנו באמת רוצים לבנות. אבל להשליך ממקרים אלו אל עולם הקוד ברמת השקה ללקוחות, כשמיליוני דולרים מונחים על כף המאזניים - השלכה שכזו תהיה טעות קריטית, בילבול בין קטגוריות שונות לחלוטין של תוצרי תוכנה.

## מסקנות

אם נכונים דברינו, הרי שתכנות קלאסי לא עומד להיעלם בקרוב, ולא יוחלף ב"אנוש". אם אנו רוצים לשמר רבות מהתכונות שיש לנו היום ממוצרי תוכנה, השפה שבה אנו משתמשים כדי לדבר עם מחשבים חייבת להיות שפה פורמלית שפקודותיה מוגדרות לחלוטין. לסיום, אנו רוצים להתייחס לעתיד מקצוע התכנות, כמה מחשבות על תפקידו שאינם עתידים להיעלם בעתיד הנראה לעין.

מוקדם יותר השווינו מתכנתים למתרגמים, אבל אולי אנלוגיה טובה יותר היא זו:

**"קוד הוא הביורוקרטיה של עולם הרעיונות הממוכנים, ומתכנתים הם המחוקקים שכותבים אותו"**.

בהקשר הפוליטי, יישום רעיון סוציו-פוליטי בעולם האמיתי דורש פירוק של הרעיון לתהליכים ביורוקרטיים שמגדירים אחריות, משאבים ומדידות של היישום. תכנות, באופן דומה, מתרגם הצעות טכנולוגיות ועסקיות מופשטות לתהליכים קונקרטיים, ניתנים למדידה שניתן לבצע באופן מציאותי במערכת מכנית אמיתית עם משאבים מוגבלים[10]. מבלי לעשות מעבר שכזה אין לנו תוכנה ואין לנו מחשב שיכול לנוע אל

[10] הנקודה הזו מעלה גם קישורים לתיזה של טיורינג וצ'רץ', אבל זהו נושא שחורג הרבה מעבר להיקף הדיון שלנו כאן.

עבר היעד, בדיוק כמו שמדיניות לא יכולה להתקיים בעולם האמיתי רק על ידי פוליטיקאי שמצהיר עליה - היא צריכה לעבור חקיקה מסודרת.

כאשר אדם לומד לתכנת, הוא אכן לומד שפה חדשה (בדיוק כמו שפקיד בנק חדש נדרש ללמוד את הטרמינולוגיה הפנימית של תעשיית הבנקאות), אבל הכישורים שהם מפתחים בעבודה ואשר רלוונטים ומשתמרים גם כאשר הם מדלגים משפת תכנות אחת לחברתה, הם שונים למדי מכישורי שפה. אלה כוללים (אם כי אינם מוגבלים ל):

- כיצד לפרק בעיות גדולות לתת-בעיות קטנות יותר, מודולריות וניתנות לפתרון;
- כיצד להגדיר תהליכי תוכנה שניתן לבצע, לעקוב אחריהם ולמצוא את הבאגים שבהם;
- כיצד לעצב חלקים שונים של מערכת, כך שהקלט והפלט שלהם מוגדרים במלואם ויכולים לשמש כממשקים/חוזים מול רכיבים אחרים;

וכן הלאה[11]. אלו הם הכישורים העמוקים יותר שמתכנתים מביאים לעבודתם.

מה השתנה עם הופעתם של סוכני תכנות-AI? לדעתנו, השינוי העיקרי (עבור מי שרוצה לכתוב קוד לטובת לקוחות אמיתיים, ולא רק כ-PoC) הוא שכעת, ככל שאתה יודע יותר על תכנות, כך אתה יכול להורות לבינה המלאכותית מה לעשות ביתר דיוק. עם יותר ניסיון, יותר קוד הופך ל"boilerplate" עבורך, מכיוון שאתה יודע מה אתה רוצה, יכול להורות בצורה ברורה יותר מה את רוצה, ולזהות סכנות יישום מהר יותר ומוקדם יותר. מתכנתים מנוסים יודעים כיצד להכווין את הבינה המלאכותית למבנים ולחבילות לשימוש, כיצד לספק דוגמאות קונקרטיות וקטעי קוד שמשקפים במדויק ופורמלית את כוונתם, וכיצד לזהות האם הקוד המופק מתאים למה שרצו.

אלו, אם כן, הכישורים העמוקים יותר שמפתח מביא איתו לשולחן, והם אלה שמשתלם ללמוד. איך בונים כישורים שכאלו? כמו שתמיד בנינו אותם: על ידי זה **שיושבים וכותבים קוד**. בונים משהו, עוברים כישלון או שניים, מדבגים ואז בונים אותם מחדש. מומלץ להתחיל בלי בינה מלאכותית, ולהשתמש בה רק כאשר חייבים. או אז משתמשים בה כדי לנתח את הטעויות שנעשו, כותבים משהו חדש וחוזר חלילה. רק כך אפשר להפוך למישהו שיודע לפרק בעיות באופן מציאותי ואפקטיבי, לרכיבים לוגיים שממופים לדברים שקיימים ואפשריים, ושיתאימו גם לצרכי לקוחות.

[11] מקום נוסף שבו ניתן לראות רמז לאבחנה זו הוא בשימוש הנפוץ בפסאודו-קוד בעת דיון באלגוריתם או זרימה חדשים. **פסאודו-קוד** הוא מה שמקבלים כאשר "מפשיטים" את הקוד מהתחביר הספציפי לשפה ונשארים עם הרעיון בצורתו הפורמלית.

בחודשים ובשנים הקרובות בהחלט נראה שינויים גדולים במה שמתכנתים מבלים בו את זמנם, כמו גם סוג המומחיות שהם יצטרכו לרכוש כדי להיות אפקטיביים בעבודתם. קשה לחזות כיצד העתיד של התכנות ייראה. שפות תכנות חדשות שהבינה המלאכותית משתלבת בהם באופן טבעי עשויות להופיע, ומתכנתים עשויים להידרש ללמוד את המקצוע שלהם מאפס. ובכל זאת, אנחנו באופן אישי מאמינים שהמתכנתים שישגשגו בשנים הקרובות יהיו אלה שמבינים שכישורי הליבה שלהם אינם כתיבת תחביר מושלם בשפה שמחשבים מבינים - אלא **תרגום צרכים אנושיים מעורפלים למפרטים פורמליים מוגדרים לחלוטין וניתנים להרצה**. בעידן החדש הזה, הכישור הזה הופך חשוב מתמיד, לא פחות. הביורוקרטיה של עולם הרעיונות הממוכנים והמוצרים הטכנולוגיים עדיין זקוקה ל"מחוקקים" מוכשרים שיגשימו אותה בעולם האמיתי.