 AI pair programming (Github Copilot, TabNine) & MISIM (Machine Inferred code Similarity)

**Pair programming :**

**What is pair programming?**

Pair programming is an Agile software development technique originating from Extreme programming (XP) in which two developers team together on one computer. The two people work together to design, code and test [user stories](https://www.techtarget.com/searchsoftwarequality/definition/user-story). Ideally, the two people would be equally skilled and would each have equal time at the keyboard.

**תרגום :**

מה זה תכנות זוגיות? תכנות זוג היא טכניקת פיתוח תוכנה זריזה שמקורה בתכנות אקסטרים (XP) שבה שני מפתחים עובדים יחד על מחשב אחד. שני האנשים עובדים יחד כדי לעצב, לקוד ולבדוק סיפורי משתמשים. באופן אידיאלי, שני האנשים יהיו מיומנים באותה מידה ולכל אחד יהיה זמן שווה ליד המקלדת

**Part 2 :**

A common implementation of pair programming calls the programmer at the keyboard the driver, while the other is called the navigator. The navigator focuses on the overall direction of the programming. The collaboration between developers can be done in person or remotely.

Pair programming is a collaborative effort that involves a lot of communication. The idea is to have the driver and navigator communicate, discuss approaches and solve issues that might be difficult for a single developer to detect.

יישום נפוץ של תכנות זוגי קורא למתכנת במקלדת כמנהל ההתקן, בעוד שהשני נקרא הנווט. הנווט מתמקד בכיוון הכולל של התכנות. שיתוף הפעולה בין מפתחים יכול להתבצע באופן אישי או מרחוק. תכנות זוגי הוא מאמץ שיתופי הכולל הרבה תקשורת. הרעיון הוא לגרום לנהג ולנווט לתקשר, לדון בגישות ולפתור בעיות שעלולות להיות קשות למפתח יחיד לזיהוי.

### How does pair programming work?

Pair programming requires two developers, one workstation, one keyboard and a mouse. The pairings can be assigned or self-assigned.

Pair programming uses the [four eyes principle](https://www.techtarget.com/whatis/definition/four-eyes-principle), which ensures two sets of eyes review the code that is being produced, even when there is a division of labor. While the driver writes the code, the navigator checks the code being written. The driver focuses on the specifics of coding, while the navigator checks the work, code quality and provides direction.

**Translate**

איך עובד תכנות זוגי? תכנות צמד דורש שני מפתחים, תחנת עבודה אחת, מקלדת אחת ועכבר. ניתן להקצות את הזיווגים או להקצות אותם בעצמם. תכנות זוג משתמש בעקרון ארבע העיניים, המבטיח ששתי קבוצות עיניים בודקות את הקוד שמיוצר, גם כאשר יש חלוקת עבודה. בזמן שהנהג כותב את הקוד, הנווט בודק את הקוד הנכתב. הנהג מתמקד בפרטי הקידוד, בעוד שהנווט בודק את העבודה, איכות הקוד ומספק כיוון.

### Benefits of pair programming

Pair programming includes the following advantages:

* **Fewer coding mistakes.** Another programmer is looking over the driver's code, which can help reduce mistakes and improve the quality of the code.
* **Knowledge is spread among the pairs.** Junior developers can pick up more skills from senior developers. And those unfamiliar with a process can be paired with someone who knows more about the process.
* **Reduced effort to coordinate.** Developers will get used to collaborating and coordinating their efforts.
* **Increased resiliency.** Pair programming helps developers understand each part of a [codebase](https://www.techtarget.com/whatis/definition/codebase-code-base), meaning the environment will not be dependent on a single person for repairs if something breaks.
* Translation is too long to be saved
* היתרונות של תכנות זוגיות תכנות זוגי כולל את היתרונות הבאים: פחות טעויות קידוד. מתכנת אחר בודק את קוד הדרייבר, מה שיכול לעזור להפחית טעויות ולשפר את איכות הקוד. הידע מתפזר בין הזוגות. מפתחים זוטרים יכולים לאסוף יותר מיומנויות ממפתחים בכירים. ומי שלא מכיר תהליך יכול להיות מזווג עם מישהו שיודע יותר על התהליך. מאמץ מופחת לתיאום. מפתחים יתרגלו לשתף פעולה ולתאם את המאמצים שלהם. גמישות מוגברת. תכנות צמד עוזר למפתחים להבין כל חלק של בסיס קוד, כלומר הסביבה לא תהיה תלויה באדם אחד לצורך תיקונים אם משהו נשבר.

### Challenges of pair programming

Pair programming includes the following pitfalls:

* **Efficiency.** Common logic might dictate that pair programming would reduce productivity by 50%, because two developers are working on the same project at one time. According to a blog [post](https://raygun.com/blog/how-good-is-pair-programming-really/) on Raygun, pairs work about 15% slower, which is an improvement but is still less than the productivity of two separate programmers.
* **Equally engaging pairs.** If both developers do not equally engage in the project, then there is less chance that knowledge will be shared, and it is highly probable that one developer will participate less than the other.
* **Social and interactive process.** It is hard for those who work better alone. Pairs that have trouble together might be better suited to work by themselves; forcing them to collaborate may hurt their work ethic.
* **Sustainability.** The pace may not be suited to practicing hours at a time. Likely, developers will need breaks at different times.

**Translate :**Translation is too long to be saved

אתגרים של תכנות זוגי תכנות זוג כולל את המלכודות הבאות: יְעִילוּת. ההיגיון המקובל עשוי להכתיב שתכנות זוגיות יפחית את הפרודוקטיביות ב-50%, מכיוון ששני מפתחים עובדים על אותו פרויקט בו-זמנית. לפי פוסט בבלוג ב-Raygun, צמדים עובדים לאט בכ-15%, שזה שיפור אבל עדיין פחות מהפרודוקטיביות של שני מתכנתים נפרדים. זוגות מרתקים באותה מידה. אם שני המפתחים לא יעסקו באותה מידה בפרויקט, אז יש פחות סיכוי שהידע יתחלק, וסביר מאוד שמפתח אחד ישתתף פחות מהאחר. תהליך חברתי ואינטראקטיבי. זה קשה למי שעובד טוב יותר לבד. זוגות שמתקשים ביחד עשויים להיות מתאימים יותר לעבוד בעצמם; אילוץ אותם לשתף פעולה עלול לפגוע במוסר העבודה שלהם. קיימות. ייתכן שהקצב לא יתאים לתרגול שעות בכל פעם. סביר להניח, מפתחים יצטרכו הפסקות בזמנים שונים.

**Github copilot :**

Trained on billions of lines of public code, GitHub Copilot puts the knowledge you need at your fingertips, saving you time and helping you stay focused-----🡪

מאומן על מיליארדי שורות של קוד ציבורי, GitHub Copilot שם את הידע שאתה צריך בקצות אצבעותיך, חוסך לך זמן ועוזר לך להישאר ממוקד

### Extends your editor

GitHub Copilot is available as an extension for Neovim, JetBrains, and Visual Studio Code. You can use the GitHub Copilot extension on your desktop or in the cloud on [GitHub Codespaces](https://github.com/features/codespaces). And it’s fast enough to use as you type.

מרחיב את העורך שלך GitHub Copilot זמין כהרחבה עבור Neovim, JetBrains ו-Visual Studio Code. אתה יכול להשתמש בתוסף GitHub Copilot בשולחן העבודה שלך או בענן ב- GitHub Codespaces. וזה מהיר מספיק לשימוש תוך כדי הקלדה.

### Speaks all the languages you love

GitHub Copilot works with a broad set of frameworks and languages. The technical preview does especially well for Python, JavaScript, TypeScript, Ruby, Java, and Go, but it understands dozens of languages and can help you find your way around almost anything

דובר את כל השפות שאתה אוהב GitHub Copilot עובד עם מערך רחב של מסגרות ושפות. התצוגה המקדימה הטכנית עושה טוב במיוחד עבור Python, JavaScript, TypeScript, Ruby, Java ו-Go, אבל היא מבינה עשרות שפות ויכולה לעזור לך להתמצא כמעט בכל דבר

### You’re the pilot

With GitHub Copilot, you’re always in charge. You can cycle through alternative suggestions, choose which to accept or reject, and manually edit suggested code. GitHub Copilot adapts to the edits you make, matching your coding style.

**אתה הטייס**

עם GitHub Copilot, אתה תמיד אחראי. אתה יכול לעבור על הצעות חלופיות, לבחור אילו לקבל או לדחות, ולערוך ידנית את הקוד המוצע. GitHub Copilot מסתגל לעריכות שאתה מבצע, בהתאמה לסגנון הקידוד שלך.

## More than autocomplete

GitHub Copilot is powered by Codex, the new AI system created by OpenAI. GitHub Copilot understands significantly more context than most code assistants. So, whether it’s in a docstring, comment, function name, or the code itself, GitHub Copilot uses the context you’ve provided and synthesizes code to match. Together with OpenAI, we’re designing GitHub Copilot to get smarter at producing safe and effective code as developers use it.

**יותר מהשלמה אוטומטית** GitHub Copilot מופעל על ידי Codex, מערכת הבינה המלאכותית החדשה שנוצרה על ידי OpenAI. GitHub Copilot מבין הרבה יותר הקשר מאשר רוב עוזרי הקוד. אז, בין אם זה במחרוזת doc, הערה, שם פונקציה או הקוד עצמו, GitHub Copilot משתמש בהקשר שסיפקת ומסנתז קוד כדי להתאים. יחד עם OpenAI, אנו מעצבים את GitHub Copilot כדי להיות חכמים יותר בהפקת קוד בטוח ויעיל כאשר מפתחים משתמשים בו.

**GitHub Copilot helps you stay focused right in your editor.**

**מה בדיוק עושה ה GitHub copilot : 1-**

Convert comments to code. Write a comment describing the logic you want, and let GitHub Copilot assemble the code for you.

Save translation

המרת הערות לקוד. כתוב הערה המתארת את ההיגיון שאתה רוצה, ואפשר ל-GitHub Copilot להרכיב עבורך את הקוד.

**2-**

Autofill for repetitive code. GitHub Copilot works great for quickly producing boilerplate and repetitive code patterns. Feed it a few examples and let it generate the rest!

Save translation

מילוי אוטומטי עבור קוד שחוזר על עצמו. GitHub Copilot עובד מצוין להפקה מהירה של תבניות קוד ודפוסי קוד חוזרים. הזינו אותו בכמה דוגמאות ותנו לו ליצור את השאר!

**3-** Tests without the toil. Tests are the backbone of any robust software engineering project. Import a unit test package, and let GitHub Copilot suggest tests that match your implementation code.

בדיקות ללא עמל. בדיקות הן עמוד השדרה של כל פרויקט הנדסת תוכנה חזק. ייבא חבילת בדיקת יחידה ואפשר ל-GitHub Copilot להציע בדיקות התואמות את קוד ההטמעה שלך.

**4-**

Show me alternatives. Want to evaluate a few different approaches? GitHub Copilot can show you a list of solutions. Use the code as provided, or edit it to meet your needs.

תראה לי חלופות. רוצה להעריך כמה גישות שונות? GitHub Copilot יכול להראות לך רשימה של פתרונות. השתמש בקוד כפי שסופק, או ערוך אותו כדי לענות על הצרכים שלך.

[(19) Github copilot Preview - AI Pair Programmer - هل يهدد عرش المبرمجين ؟؟ Copilot شرح - YouTube](https://www.youtube.com/watch?v=IH8YkAipcGs)

[(19) #1 كورس تطوير خدمات الويب و API في لغة Restful API With Slim - PHP | الدرس الاول - ما هو | API - YouTube](https://www.youtube.com/watch?v=Ix1iC6nYo00&list=PLdYYj2XLw5BlJr_whdtUxFIuXBX3H2NwY)

Copilot isn’t the first of its kind; **Tabnine** and Kite are two popular tools that cover the same functionality. However, Copilot stands out because it’s powered by Codex, a descendant from GPT-3 that provides a deeper understanding of the context than other assistants. Codex is similar to GPT-3, with an important distinction: It has been trained on huge amounts of coding data publicly available, from GitHub repositories and other sites.

Copilot הוא לא הראשון מסוגו; Tabnine ו- Kite הם שני כלים פופולריים המכסים את אותה פונקציונליות. עם זאת, Copilot בולט מכיוון שהוא מופעל על ידי Codex, צאצא של GPT-3 המספק הבנה עמוקה יותר של ההקשר מאשר עוזרים אחרים. Codex דומה ל-GPT-3, עם הבחנה חשובה: הוא עבר הכשרה על כמויות עצומות של נתוני קידוד הזמינים לציבור, ממאגרי GitHub ואתרים אחרים.

**הסבר על CODEX**

Codex, which OpenAI will integrate into their API this summer, is a programming language model. Its abilities in this area greatly exceed those of GPT-3 — or those of GPT-J. As I argued in a previous article, GPT-like models seem to have a latent power within they release when specializing in a task. GPT-3, the jack of all trades, is amazing at many tasks. Codex is, in contrast, a master at coding.

ש-OpenAI תשלב בקיץ ב-API שלהם, הוא מודל של שפת תכנות. היכולות שלו בתחום הזה עולות בהרבה על אלו של GPT-3 - או של GPT-J. כפי שטענתי במאמר קודם, נראה שלמודלים דמויי GPT יש כוח סמוי בתוכם שהם משחררים כשהם מתמחים במשימה. GPT-3, הג'ק של כל העסקים, מדהים במשימות רבות. קודקס הוא, לעומת זאת, מאסטר בקידוד

**חסרונות שזה לא מושלם :**

But neither Codex nor Copilot is perfect. The tool works fine, given [the testimonies](https://www.cnbc.com/2021/06/29/microsoft-github-copilot-ai-offers-coding-suggestions.html) of those who pre-tested it. But, as with all code written by a human, the Copilot’s code should be “[tested, reviewed, and vetted](https://copilot.github.com/#faqs).” GitHub developers have created safety mechanisms to provide the best experience to the user, but the system “may sometimes produce undesired outputs, including biased, discriminatory, abusive, or offensive outputs.” Copilot code may even not compile, as it doesn’t test the code it outputs. In short, it’s an early version of a new category of tools; it has flaws, but its promises greatly surpass them.

אבל גם Codex וגם Copilot לא מושלמים. הכלי עובד מצוין, בהתחשב בעדויות של מי שבחנו אותו מראש. אבל, כמו בכל קוד שנכתב על ידי אדם, יש "לבדוק, לבדוק ולבדוק את הקוד של Copilot". מפתחי GitHub יצרו מנגנוני בטיחות כדי לספק את החוויה הטובה ביותר למשתמש, אך המערכת "עלולה לפעמים לייצר פלטים לא רצויים, כולל פלטים מוטים, מפלים, פוגעניים או פוגעניים". ייתכן שקוד Copilot אפילו לא יבצע קומפילציה, מכיוון שהוא לא בודק את הקוד שהוא מוציא. בקיצור, זו גרסה מוקדמת של קטגוריה חדשה של כלים; יש לו פגמים, אבל ההבטחות שלו עולות עליהם בהרבה.

**הסבר טוב טוב על GITHUB COPILOT**

## What it does

[On the GitHub page](https://copilot.github.com/), there are examples of Copilot’s amazing performance. It can complete lines of code or write whole functions, transform descriptive comments into code, autofill repetitive code, or create unit tests for your methods. It can take a few hundred lines above the current line as input, but there’s one important limitation: You can’t feed it code from other files.

It works best with Python, JavaScript, TypeScript, Ruby, and Go, but it “understands dozens of languages.” However, because it fails sometimes, it may be more useful when exploring new libraries or writing in an unknown programming language. Otherwise, it could take more time to review Copilot’s code than to write it yourself.

Copilot is based on a language model similar to GPT-3, so it has adapted to the code it’s been trained on. But, as you use it, it starts “understanding” your style and adapts to you. And if you don’t like the first suggestion, you can ask it to provide top-k solutions.

מה שזה עושה בעמוד GitHub, יש דוגמאות לביצועים המדהימים של Copilot. זה יכול להשלים שורות קוד או לכתוב פונקציות שלמות, להפוך הערות תיאוריות לקוד, מילוי אוטומטי של קוד שחוזר על עצמו, או ליצור בדיקות יחידות עבור השיטות שלך. זה יכול לקחת כמה מאות שורות מעל השורה הנוכחית כקלט, אבל יש מגבלה אחת חשובה: אתה לא יכול להזין אותו בקוד מקבצים אחרים

. זה עובד הכי טוב עם Python, JavaScript, TypeScript, Ruby ו-Go, אבל זה "מבין עשרות שפות". עם זאת, מכיוון שהוא נכשל לפעמים, זה עשוי להיות שימושי יותר בעת חקירת ספריות חדשות או כתיבה בשפת תכנות לא ידועה. אחרת, זה יכול לקחת יותר זמן לסקור את הקוד של Copilot מאשר לכתוב אותו בעצמך

. Copilot מבוסס על מודל שפה דומה ל-GPT-3, כך שהוא מותאם לקוד עליו הוכשר. אבל, בזמן שאתה משתמש בו, הוא מתחיל "להבין" את הסגנון שלך ומסתגל אלייך. ואם אתה לא אוהב את ההצעה הראשונה, אתה יכול לבקש ממנה

לספק פתרונות מובילים

.

Important implications of Copilot

The power of language models

GPT-2, GPT-3, the Switch transformer, LaMDA, MUM, Wu Dao 2.0… Pre-trained language models are the cake in AI right now and every major player in the field is fighting to get the biggest portion. The reason is that these models work extremely well. GPT-3 is the most popular, and rightly so. When OpenAI released the GPT-3 API, it allowed the world to take a peek at its power, and it took effect. GPT-3 was so powerful and performed so well, people even dared to call it AGI — which it isn’t.

השלכות חשובות של Copilot כוחם של מודלים לשוניים GPT-2, GPT-3, שנאי ה-Switch, LaMDA, MUM, Wu Dao 2.0... דגמי שפה מאומנים מראש הם העוגה בבינה מלאכותית כרגע וכל שחקן גדול בתחום נלחם כדי להשיג את החלק הגדול ביותר. הסיבה היא שהדגמים האלה עובדים טוב מאוד. GPT-3 הוא הפופולרי ביותר, ובצדק. כאשר OpenAI הוציאה את ה-API של GPT-3, היא אפשרה לעולם להציץ בכוחו, וזה נכנס לתוקף. GPT-3 היה כל כך חזק והופיע כל כך טוב, שאנשים אפילו העזו לקרוא לזה AGI - וזה לא

آثار مهمة مساعد الطيار قوة النماذج اللغوية GPT-2 ، GPT-3 ، محول التبديل ، LaMDA ، MUM ، Wu Dao 2.0 ... نماذج اللغة المدربة مسبقًا هي الكعكة في الذكاء الاصطناعي في الوقت الحالي وكل لاعب رئيسي في هذا المجال يقاتل للحصول على الجزء الأكبر. والسبب هو أن هذه النماذج تعمل بشكل جيد للغاية. GPT-3 هو الأكثر شعبية ، وهو محق في ذلك. عندما أصدرت OpenAI واجهة GPT-3 API ، سمحت للعالم بإلقاء نظرة خاطفة على قوتها ، ودخلت حيز التنفيذ. كان GPT-3 قويًا جدًا وكان أداؤه جيدًا ، حتى أن الناس تجرأوا على تسميته AGI - وهو ليس كذلك..

**משהו חשוב לגבי חסרונות :**

There’s a quote from Joel Spolsky that comes in handy now: “It’s harder to read code than to write it.” An inexperienced programmer may not be aware of issues with Copilot’s code whereas a veteran would prefer to write the code instead of reading what Copilot has generated. There are very few instances in which it's clearly worth it to use Copilot: An experienced programmer who wants to try a new library/language/framework or wants to write unit tests (although those would also need to be reviewed) based on hand-made methods. The other case is an inexperienced programmer that’s starting to learn.

هناك اقتباس من Joel Spolsky أصبح مفيدًا الآن: "قراءة الكود أصعب من كتابته." قد لا يكون المبرمج عديم الخبرة على دراية بالمشكلات المتعلقة برمز Copilot بينما يفضل المخضرم كتابة الكود بدلاً من قراءة ما أنشأه Copilot. هناك عدد قليل جدًا من الحالات التي من الواضح أنه يستحق فيها استخدام Copilot: مبرمج متمرس يريد تجربة مكتبة / لغة / إطار عمل جديد أو يريد كتابة اختبارات وحدة (على الرغم من أن هذه ستحتاج أيضًا إلى المراجعة) بناءً على صنع يدوي طُرق. الحالة الأخرى هي مبرمج عديم الخبرة بدأ التعلم.

יש ציטוט של ג'ואל ספולסקי שבא שימושי עכשיו: "קשה יותר לקרוא קוד מאשר לכתוב אותו." יתכן שמתכנת חסר ניסיון לא מודע לבעיות בקוד של Copilot, בעוד שאדם ותיק יעדיף לכתוב את הקוד במקום לקרוא את מה ש-Copilot יצר. יש מעט מאוד מקרים שבהם ברור ששווה את זה להשתמש ב-Copilot: מתכנת מנוסה שרוצה לנסות ספרייה/שפה/מסגרת חדשה או רוצה לכתוב בדיקות יחידה (אם כי גם אלה יצטרכו לעבור סקירה) על בסיס עבודת יד שיטות. המקרה השני הוא מתכנת חסר ניסיון שמתחיל ללמוד.

**המשך לחסרונות :**

However, there are bigger problems than Copilot writing low-quality code. From GitHub’s FAQ: “There’s a lot of public code in the world with insecure coding patterns, bugs, or references to outdated APIs or idioms. When GitHub Copilot synthesizes code suggestions based on this data, it can also synthesize code that contains these undesirable patterns.” A question worth considering is whether Copilot would reduce or increase the amount of these problems.

עם זאת, יש בעיות גדולות יותר מאשר Copilot לכתוב קוד באיכות נמוכה. מתוך השאלות הנפוצות של GitHub: "יש הרבה קוד ציבורי בעולם עם דפוסי קידוד לא מאובטחים, באגים או הפניות לממשקי API או ניבים מיושנים. כאשר GitHub Copilot מסנתז הצעות קוד על סמך נתונים אלה, הוא יכול גם לסנתז קוד שמכיל את הדפוסים הלא רצויים הללו." שאלה שכדאי לשקול היא האם Copilot יפחית או יגדיל את כמות הבעיות הללו.

**מאמר**

[GitHub Copilot — A New Generation of AI Programmers | by Alberto Romero | Towards Data Science](https://towardsdatascience.com/github-copilot-a-new-generation-of-ai-programmers-327e3c7ef3ae) ----🡪

[Copilot Is Like GPT-3 but for Code—Fun, Fast, and Full of Flaws | WIRED](https://www.wired.com/story/openai-copilot-autocomplete-for-code/)

**Conclusion --- תוצאות :**

GitHub Copilot is predestined to change the day-to-day of many programmers all over the world. Despite the problems I’ve highlighted, the technology is an important milestone and will lead future efforts towards no-code, which Microsoft has been pursuing for some years already.

Save translation

סיכום

GitHub Copilot נועד לשנות את היום-יום של מתכנתים רבים בכל רחבי העולם. למרות הבעיות שהדגשתי, הטכנולוגיה היא אבן דרך חשובה ותוביל מאמצים עתידיים לקראת חוסר קוד, שמיקרוסופט רודפת אחריו כבר כמה שנים.

**חששות :**

Copilot feels like an inflection point, and no one knows how events will unfold from here. Will programmers start losing their jobs a few years from now? Will we manage to find a benefiting pilot-Copilot synergy? Will it provide such an edge that companies will have to adapt to it or die? Will programmers have to trade-off professional survival for privacy?

Many questions have been arising since Tuesday and many more will keep coming. For now, answers will have to wait. GitHub Copilot is up. Let’s keep our eyes open to see where it’s driving us.

يبدو مساعد الطيار وكأنه نقطة انعطاف ، ولا أحد يعرف كيف ستتكشف الأحداث من هنا. هل سيبدأ المبرمجون في فقدان وظائفهم بعد بضع سنوات من الآن؟ هل سننجح في إيجاد تعاون مفيد بين الطيارين؟ هل ستوفر مثل هذه الميزة التي ستضطر الشركات إلى التكيف معها أو تموت؟ هل سيتعين على المبرمجين مقايضة البقاء المهني بالخصوصية؟ ظهرت العديد من الأسئلة منذ يوم الثلاثاء وسيستمر طرح أسئلة أخرى كثيرة. في الوقت الحالي ، يجب أن تنتظر الإجابات. GitHub مساعد الطيار منتهي. دعونا نبقي أعيننا مفتوحة لنرى إلى أين يقودنا.

Translation is too long to be saved

Copilot מרגיש כמו נקודת פיתול, ואף אחד לא יודע איך האירועים יתפתחו מכאן. האם מתכנתים יתחילו לאבד את מקום עבודתם בעוד כמה שנים? האם נצליח למצוא סינרגיה מועילה של טייס-קופיילוט? האם זה יספק יתרון כזה שחברות יצטרכו להסתגל אליו או למות? האם מתכנתים יצטרכו להחליף הישרדות מקצועית עבור פרטיות? שאלות רבות עולות מאז יום שלישי ורבות נוספות ימשיכו להגיע. לעת עתה, התשובות יצטרכו להמתין. GitHub Copilot פועל. בואו נשאיר עיניים פקוחות כדי לראות לאן זה מסיע אותנו.

**MISIM :**

[Intel, MIT and Georgia Tech Deliver Improved Machine-Programming Code Similarity System | Intel Newsroom](https://newsroom.intel.com/news/intel-mit-georgia-tech-machine-programming-code-similarity-system/#gs.1wl5l2)

What’s New: Today, Intel unveiled a new machine

programming (MP) system – in conjunction with Massachusetts Institute of Technology (MIT) and Georgia Institute of Technology (Georgia Tech). The system, machine inferred code similarity (MISIM), is an automated engine designed to learn what a piece of software intends to do by studying the structure of the code and analyzing syntactic differences of other code with similar behavior.

“Intel’s ultimate goal for machine programming is to democratize the creation of software. When fully realized, MP will enable everyone to create software by expressing their intention in whatever fashion that’s best for them, whether that’s code, natural language or something else. That’s an audacious goal, and while there’s much more work to be done, MISIM is a solid step toward it.”

– Justin Gottschlich, principal scientist and director/founder of Machine Programming Research at Intel

מה חדש: היום, אינטל חשפה מערכת חדשה לתכנות מכונות (MP) - בשיתוף עם המכון הטכנולוגי של מסצ'וסטס (MIT) והמכון הטכנולוגי של ג'ורג'יה (Georgia Tech). המערכת, Machine Inferred Code similarity (MISIM), היא מנוע אוטומטי שנועד ללמוד מה תוכנה מתכוונת לעשות על ידי לימוד מבנה הקוד וניתוח הבדלים תחביריים של קוד אחר עם התנהגות דומה. "המטרה הסופית של אינטל לתכנות מכונה היא דמוקרטיזציה של יצירת תוכנה. כשהתממשה במלואה, MP תאפשר לכולם ליצור תוכנה על ידי הבעת כוונתו בכל האופנה המתאימה לו ביותר, בין אם זה קוד, שפה טבעית או משהו אחר. זו מטרה נועזת, ולמרות שיש עוד הרבה עבודה לעשות, MISIM הוא צעד איתן לקראתה". – ג'סטין גוטשליץ', מדען ראשי ומנהל/מייסד מחקר תכנות מכונה באינטל

How It Works: A core differentiation between MISIM and existing code-similarity systems lies in its novel context-aware semantic structure (CASS), which aims to lift out what the code actually does. Unlike other existing approaches, CASS can be configured to a specific context, allowing it to capture information that describes the code at a higher level. CASS can provide more specific insight into what the code does rather than how it does it. Moreover, MISIM can do all of this without using a compiler, which translates human-readable source code into computer-executable machine code. This has many benefits over existing systems, including the ability to execute on incomplete snippets of code that a developer may be currently writing – an important practical characteristic for recommendation systems or automated bug fixing.

Once the code’s structure is integrated into CASS, neural network systems give similarity scores to pieces of code based on the jobs they are designed to carry out. In other words, if two pieces of code look very different in their structure but perform the same function, the neural networks would rate them as largely similar.

By bringing together these principles in a unified system, researchers found that MISIM was able to identify similar pieces of code up to 40x more accurately than prior state-of-the-art systems.

איך זה עובד: הבחנה מרכזית בין MISIM לבין מערכות קיימות של דמיון קוד טמונה במבנה הסמנטי החדש שלה (CASS), שמטרתו להעלות את מה שהקוד עושה בפועל. בניגוד לגישות קיימות אחרות, ניתן להגדיר את CASS להקשר מסוים, מה שמאפשר לו ללכוד מידע המתאר את הקוד ברמה גבוהה יותר. CASS יכול לספק תובנה ספציפית יותר לגבי מה שהקוד עושה במקום איך הוא עושה את זה. יתרה מכך, MISIM יכול לעשות את כל זה מבלי להשתמש במהדר, שמתרגם קוד מקור קריא אנושי לקוד מכונה הניתן להפעלה במחשב. יש לכך יתרונות רבים על פני מערכות קיימות, כולל היכולת לבצע על קטעי קוד לא שלמים שמפתח עשוי לכתוב כעת - מאפיין מעשי חשוב עבור מערכות המלצות או תיקון באגים אוטומטי. ברגע שמבנה הקוד משולב ב-CASS, מערכות רשתות עצביות מעניקות ציוני דמיון לפיסות קוד על סמך העבודות שהן נועדו לבצע. במילים אחרות, אם שתי פיסות קוד נראות שונות מאוד במבנה שלהן אך מבצעות את אותה פונקציה, הרשתות העצבית יעריכו אותן כדומות במידה רבה. על ידי איחוד העקרונות הללו במערכת מאוחדת, חוקרים גילו ש-MISIM הצליח לזהות פיסות קוד דומות עד פי 40 בצורה מדויקת יותר מאשר מערכות קודמות מתקדמות.

**MISIM**

What’s New: Today, Intel unveiled a new machine programming (MP) system – in conjunction with Massachusetts Institute of Technology (MIT) and Georgia Institute of Technology (Georgia Tech). The system, machine inferred code similarity (MISIM), is an automated engine designed to learn what a piece of software intends to do by studying the structure of the code and analyzing syntactic differences of other code with similar behavior.

מה חדש: היום, אינטל חשפה מערכת חדשה לתכנות מכונות (MP) - בשיתוף עם המכון הטכנולוגי של מסצ'וסטס (MIT) והמכון הטכנולוגי של ג'ורג'יה (Georgia Tech). המערכת, Machine Inferred Code similarity (MISIM), היא מנוע אוטומטי שנועד ללמוד מה תוכנה מתכוונת לעשות על ידי לימוד מבנה הקוד וניתוח הבדלים תחביריים של קוד אחר עם התנהגות דומה.

דובריו של מישהו :

“Intel’s ultimate goal for machine programming is to democratize the creation of software. When fully realized, MP will enable everyone to create software by expressing their intention in whatever fashion that’s best for them, whether that’s code, natural language or something else. That’s an audacious goal, and while there’s much more work to be done, MISIM is a solid step toward it.”

– Justin Gottschlich, principal scientist and director/founder of Machine Programming Research at Intel

**אחת הסיבות ליצור מנגנן כזה :**

**Why It Matters**: With the rise of heterogeneous computing, hardware and software systems are becoming increasingly complex. This complexity, paired with a shortage of programmers who can code at an expert level across multiple architectures, spotlights a need for new development approaches. Machine programming, a term coined by Intel Labs and MIT in their “Three Pillars of Machine Programming” paper, aims to improve development productivity through the use of automated tools. A key technology to several of these emerging machine programming tools is code similarity, which has the potential to accurately and efficiently automate some of the software development process to meet this need

Translation is too long to be saved

**למה זה חשוב:** עם עליית המחשוב ההטרוגני, מערכות החומרה והתוכנה הופכות מורכבות יותר ויותר. מורכבות זו, יחד עם מחסור במתכנתים שיכולים לקודד ברמת מומחה על פני ארכיטקטורות מרובות, מדגישה את הצורך בגישות פיתוח חדשות. תכנות מכונה, מונח שנטבע על ידי מעבדות אינטל ו-MIT במאמר "שלושת העמודים של תכנות מכונה", נועד לשפר את פרודוקטיביות הפיתוח באמצעות שימוש בכלים אוטומטיים. טכנולוגיה מרכזית לכמה מכלי תכנות מכונות מתפתחים אלה היא דמיון קוד, שיש לו פוטנציאל לבצע אוטומציה מדויקת ויעילה של חלק מתהליך פיתוח התוכנה כדי לענות על צורך זה

**יתרון של מערכת זו :**

Yet building accurate code similarity systems is a relatively unsolved problem. These systems attempt to determine whether two code snippets show similar characteristics or aim to achieve similar goals — a daunting task when having only source code to learn from. MISIM can accurately determine when two pieces of code perform a similar computation, even when those pieces use different data structures and algorithms. “This is an important step toward the grander vision of machine programming,” Gottschlich said.

עם זאת, בניית מערכות דמיון מדויקות בקוד היא בעיה יחסית לא פתורה. מערכות אלו מנסות לקבוע אם שני קטעי קוד מציגים מאפיינים דומים או שואפים להשיג מטרות דומות - משימה מרתיעה כאשר יש רק קוד מקור ללמוד ממנו. MISIM יכול לקבוע במדויק מתי שני חלקי קוד מבצעים חישוב דומה, גם כאשר חלקים אלה משתמשים במבני נתונים ואלגוריתמים שונים. "זהו צעד חשוב לקראת החזון הגדול יותר של תכנות מכונה", אמר גוטשליץ'.

How It Works: A core differentiation between MISIM and existing code-similarity systems lies in its novel context-aware semantic structure (CASS), which aims to lift out what the code actually does. Unlike other existing approaches, CASS can be configured to a specific context, allowing it to capture information that describes the code at a higher level. CASS can provide more specific insight into what the code does rather than how it does it. Moreover, MISIM can do all of this without using a compiler, which translates human-readable source code into computer-executable machine code. This has many benefits over existing systems, including the ability to execute on incomplete snippets of code that a developer may be currently writing – an important practical characteristic for recommendation systems or automated bug fixing.

Once the code’s structure is integrated into CASS, neural network systems give similarity scores to pieces of code based on the jobs they are designed to carry out. In other words, if two pieces of code look very different in their structure but perform the same function, the neural networks would rate them as largely similar.

By bringing together these principles in a unified system, researchers found that MISIM was able to identify similar pieces of code up to 40x more accurately than prior state-of-the-art systems.

**איך זה עובד:** הבחנה מרכזית בין MISIM לבין מערכות קיימות של דמיון קוד טמונה במבנה הסמנטי החדש שלה (CASS), שמטרתו להעלות את מה שהקוד עושה בפועל. בניגוד לגישות קיימות אחרות, ניתן להגדיר את CASS להקשר מסוים, מה שמאפשר לו ללכוד מידע המתאר את הקוד ברמה גבוהה יותר. CASS יכול לספק תובנה ספציפית יותר לגבי מה שהקוד עושה במקום איך הוא עושה את זה. יתרה מכך, MISIM יכול לעשות את כל זה מבלי להשתמש במהדר, שמתרגם קוד מקור קריא אנושי לקוד מכונה הניתן להפעלה במחשב. יש לכך יתרונות רבים על פני מערכות קיימות, כולל היכולת לבצע על קטעי קוד לא שלמים שמפתח עשוי לכתוב כעת - מאפיין מעשי חשוב עבור מערכות המלצות או תיקון באגים אוטומטי. ברגע שמבנה הקוד משולב ב-CASS, מערכות רשתות עצביות מעניקות ציוני דמיון לפיסות קוד על סמך העבודות שהן נועדו לבצע. במילים אחרות, אם שתי פיסות קוד נראות שונות מאוד במבנה שלהן אך מבצעות את אותה פונקציה, הרשתות העצבית יעריכו אותן כדומות במידה רבה. על ידי איחוד העקרונות הללו במערכת מאוחדת, חוקרים גילו ש-MISIM הצליח לזהות פיסות קוד דומות עד פי 40 בצורה מדויקת יותר מאשר מערכות קודמות מתקדמות.

\\

מאמר חשוב !!!

[MISIM\_An\_End-to-End\_Neural\_Code\_Similarity\_System (1).pdf](file:///C:\Users\user\Downloads\MISIM_An_End-to-End_Neural_Code_Similarity_System%20(1).pdf)

[2006.05265v6.pdf (arxiv.org)](https://arxiv.org/pdf/2006.05265v6.pdf)

[MISIM: A Novel Code Similarity System (upenn.edu)](https://repository.upenn.edu/cgi/viewcontent.cgi?article=1000&context=cps_machine_programming)

Abstract

Code similarity systems are integral to a range of applications from code recom mendation to automated construction of software tests and defect mitigation. In

this paper, we present Machine Inferred Code Similarity (MISIM), a novel end-toend code similarity system that consists of two core components

. First, MISIM

uses a novel context-aware semantic structure, which is designed to aid in lifting semantic meaning from code syntax. Second, MISIM provides a neural-based code similarity scoring algorithm, which can be implemented with various neural

network architectures with learned parameters.

Translation is too long to be saved

תַקצִיר מערכות דימיון קוד הן חלק בלתי נפרד ממגוון יישומים, החל מהמלצת קוד ועד לבנייה אוטומטית של בדיקות תוכנה והפחתת פגמים. ב מאמר זה, אנו מציגים Machine Inferred Code Similarity (MISIM), מערכת דמיון חדשנית מקצה לקצה המורכבת משני מרכיבי ליבה . ראשית, MISIM משתמש במבנה סמנטי חדשני המודע להקשר, שנועד לסייע בהרמה משמעות סמנטית מתחביר קוד. שנית, MISIM מספקת מבוססת עצבים אלגוריתם ניקוד דמיון קוד, שניתן ליישם עם עצבים שונים ארכיטקטורות רשת עם פרמטרים נלמדים.

**2 החידושים במנגנון הזה :**

We principally

focus on two main novelties of MISIM and how they may improve code similarity analysis: (i)

its structural representation of code, called context-aware semantic structure (CASS), and (ii) its

neural-based learned code similarity scoring algorithm. These components can be used individually

or together as we have chosen to do.

אנחנו בעיקרון התמקד בשני חידושים עיקריים של MISIM וכיצד הם עשויים לשפר את ניתוח הדמיון בקוד: (i) הייצוג המבני של הקוד, הנקרא מבנה סמנטי מודע להקשר (CASS), ו-(ii) אלגוריתם ניקוד דמיון נלמד מבוסס עצבי. ניתן להשתמש ברכיבים אלה בנפרד או ביחד כפי שבחרנו לעשות.

MISIM System

Figure 1 shows an overview of MISIM, our end-to-end code similarity system. A core component of MISIM lies is the novel context-aware semantic structure (CASS), which aims to capture semantically salient properties of the input code. CASS is also context-aware, as it can capture information that describes the context of the code (e.g., it is a function call, it is an operation, etc.). Once these CASSes are constructed, they are vectorized and used as input to a neural network, which produces a feature vector. Once a feature vector is generated, a code similarity measurement (e.g., vector dot product [25], cosine similarity [7]) calculates the similarity score between the input program and any other program that has undergone the same CASS transformation process.

**כאן צריך להוסיף תמונה !!!!!**

מערכת MISIM איור 1 מציג סקירה כללית של MISIM, מערכת דמיון הקוד שלנו מקצה לקצה. מרכיב ליבה של MISIM lies הוא המבנה הסמנטי החדש מודע להקשר (CASS), שמטרתו ללכוד באופן סמנטי מאפיינים בולטים של קוד הקלט. CASS גם מודע להקשר, מכיוון שהוא יכול ללכוד מידע זה מתאר את ההקשר של הקוד (למשל, זוהי קריאה לפונקציה, זו פעולה וכו'). פעם אלה CASSs בנויים, הם מוקטורים ומשמשים כקלט לרשת עצבית, המייצרת א וקטור תכונה. לאחר יצירת וקטור תכונה, מדידת דמיון קוד (למשל, נקודה וקטורית תוצר [25], דמיון קוסינוס [7]) מחשב את ציון הדמיון בין תוכנית הקלט לכל תוכנית אחרת שעברה את אותו תהליך טרנספורמציה של CASS.

**הגדרה של CASS**

Definition 1 (Context-aware semantic structure (CASS)) A CASS consists of one or more CASS

trees and an optional global attributes table (GAT). A CASS tree, T, is a collection of nodes,

V = {v1, v2, . . . , v|V |}, and edges, E = {e1, e2, . . . , e|E|}, denoted as T = (V, E). Each edge is

directed from a parent node, vp to a child node, vc, or ek = (vp, vc) where ek ∈ E and vp, vc ∈ V .

The root node, vr, of the tree signifies the beginning of the code snippet and has no parent node, i.e.,

@vp,(vp, vr) ∈ E. A child node is either an internal node or a leaf node. An internal node has at

least one child node while a leaf node has no child nodes. A CASS tree can be empty, in which it

has no nodes. The CASS GAT contains exactly one entry per unique function definition in the code

snippet. A GAT entry includes the input and output cardinality values for the corresponding function.

Translation is too long to be saved

**הגדרה 1** (מבנה סמנטי מודע להקשר (CASS)) CASS מורכב מ-CASS אחד או יותר עצים וטבלת תכונות גלובלית אופציונלית (GAT). עץ CASS, T, הוא אוסף של צמתים, V = {v1, v2, . . . , v|V |}, וקצוות, E = {e1, e2, . . . , e|E|}, מסומן כ-T = (V, E). כל קצה הוא מופנה מצומת אב, vp לצומת ילד, vc, או ek = (vp, vc) כאשר ek ∈ E ו-vp, vc ∈ V . צומת השורש, vr, של העץ מסמל את ההתחלה של קטע הקוד ואין לו צומת אב, כלומר, @vp,(vp, vr) ∈ E. צומת צאצא הוא צומת פנימי או צומת עלה. לצומת פנימי יש ב לפחות צומת צאצא אחד בעוד שלצומת עלה אין צמתים צאצאים. עץ CASS יכול להיות ריק, שבו הוא אין צמתים. ה-CASS GAT מכיל בדיוק ערך אחד לכל הגדרת פונקציה ייחודית בקוד קטע. ערך GAT כולל את ערכי הקרדינליות של הקלט והפלט עבור הפונקציה המתאימה.

Conclusion This paper presented MISIM, a code semantics similarity system. MISIM has two core novelties. First, it uses the context-aware semantics structure (CASS) designed to lift semantic meaning from code syntax. Second, it provides a neural-based code semantics similarity scoring algorithm for learning semantics similarity scoring using CASS. Puri et al. [2021] and our experimental evaluation showed that MISIM outperforms four state-of-the-art code semantics similarity systems and two hand-optimized models. We also provided an anecdotal analysis illustrating that there may not be one universally optimal CASS configuration. An open research question for MISIM is in how to automatically derive the proper configuration of its various components for a given code corpus, specifically the CASS and neural scoring algorithms, which we plan to explore in future work

סיכום מאמר זה הציג את MISIM, מערכת דמיון סמנטית קוד. ל-MISIM שני חידושים מרכזיים. ראשית, הוא משתמש במבנה הסמנטיקה המודע להקשר (CASS) שנועד להרים ממנו משמעות סמנטית תחביר קוד. שנית, הוא מספק עבור אלגוריתם ניקוד דמיון סמנטי מבוסס קוד לימוד ניקוד דמיון סמנטיקה באמצעות CASS. פורי וחב'. [2021] וההערכה הניסיונית שלנו הראה כי MISIM מתעלה על ארבע מערכות דמיון סמנטיקה מתקדמות של קוד ושתיים מודלים מותאמים ביד. סיפקנו גם ניתוח אנקדוטלי הממחיש שאולי אין תצורת CASS אופטימלית אוניברסלית אחת. שאלת מחקר פתוחה עבור MISIM היא כיצד גוזר אוטומטית את התצורה הנכונה של מרכיביו השונים עבור קורפוס קוד נתון, במיוחד את אלגוריתמי ה-CASS והניקוד העצבי, אותם אנו מתכננים לחקור בעבודה עתידית

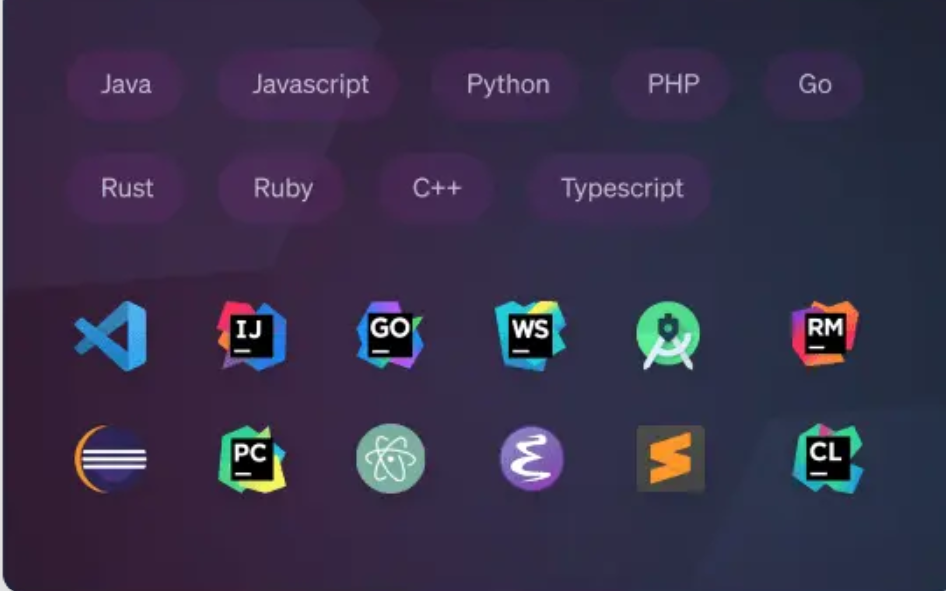
[pdf (openreview.net)](https://openreview.net/pdf?id=AZ4vmLoJft)

TabNine :

**Automatically Writes Parts  
of the Code**

Tabnine predicts your next block of code delivering accurate code completions directly in your editor.

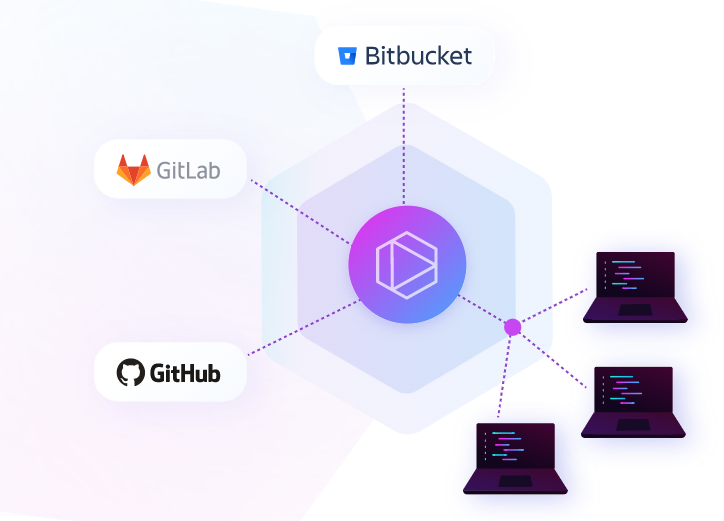
כותב חלקים באופן אוטומטי של הקוד Tabnine חוזה את גוש הקוד הבא שלך ומספק השלמות קוד מדויקות ישירות בעורך שלך

.

**Private AI Model That Trains on Your Team’s Code**

Connect your repositories, and Tabnine's AI assistant will start learning your specific code patterns! Tabnine equips your team with a custom-trained AI model that provides guidance aligned with your code standards and best practices. [Learn more](https://www.tabnine.com/team)

דגם AI פרטי שמתאמן על הקוד של הצוות שלך חבר את המאגרים שלך, ועוזר הבינה המלאכותית של Tabnine יתחיל ללמוד את דפוסי הקוד הספציפיים שלך! Tabnine מציידת את הצוות שלך במודל AI מאומן בהתאמה אישית, המספק הדרכה בהתאם לתקני הקוד ולשיטות העבודה המומלצות שלך

****

Use AI to ship better software faste

השתמש ב-AI כדי לשלוח תוכנה טובה יותר מהר יותר

Code Faster with AI Completions

Tabnine automates repetitive work and reduces the need for expensive and distracting code search. Tabnine can write 30% or even more of the code for you.

Save translation

كود أسرع مع إكمالات الذكاء الاصطناعي يقوم تابنين بأتمتة العمل المتكرر ويقلل من الحاجة إلى البحث عن الكود المكلف والمشتت. يمكن أن يكتب تابنين 30٪ أو حتى أكثر من الكود نيابة عنك.

Save translation

קוד מהיר יותר עם השלמות בינה מלאכותית Tabnine עושה אוטומציה של עבודה חוזרת ומפחיתה את הצורך בחיפוש קוד יקר ומסיח את הדעת. Tabnine יכולה לכתוב עבורך 30% או אפילו יותר מהקוד.

**יתרונות של TABNINE**

Improve Code Quality and Consistency

Tabnine Improves code quality and consistency across your project, suggesting code completions that align with your code patterns. With project-wide consistent code, your projects are easier to read, manage, and maintain.

Save translation

שפר את איכות ועקביות הקוד Tabnine משפרת את איכות הקוד והעקביות בפרויקט שלך, ומציעה השלמות קוד שמתאימות לדפוסי הקוד שלך. עם קוד עקבי בכל הפרויקט, קל יותר לקרוא, לנהל ולתחזק את הפרויקטים שלך.

Reduce Code Review Iterations

Use AI assistance to get the right code the first time. Tabnine provides code guidance that’s correct and consistent with your patterns and style, saving costly and frustrating code review iterations.

צמצם את חזרות סקירת הקוד השתמש בסיוע AI כדי לקבל את הקוד הנכון בפעם הראשונה. Tabnine מספקת הנחיית קוד הנכונה ועקבית עם הדפוסים והסגנון שלך, וחוסכת איטרציות יקרות ומתסכלות של סקירת קוד.

**THE GOAL OF TABNINE**

Our goal at Tabnine is to create and deliver a top-to-bottom AI-assisted development workflow that empowers all code creators, in all languages, from concept through to completion.

המטרה שלנו ב-Tabnine היא ליצור ולספק זרימת עבודה לפיתוח בסיוע בינה מלאכותית מלמעלה עד למטה, המעצימה את כל יוצרי הקוד, בכל השפות, מהרעיון ועד להשלמה.

**למי בדיוק עוזר ?**

Whether you’re part of a team, or a developer working on your own, Tabnine will help you write code faster – all in your favorite IDE.

Save translation

בין אם אתה חלק מצוות, או מפתח שעובד בעצמך, Tabnine יעזור לך לכתוב קוד מהר יותר - הכל ב-IDE המועדף עליך.

# הכלי הזה יעשה השלמה אוטומטית לקוד שלכם

[הכלי הזה יעשה השלמה אוטומטית לקוד שלכם | גיקטיים (geektime.co.il)](https://www.geektime.co.il/tabnine-autocompleter/)

את TabNine תוכלו להטמיע בכל עורך קוד, והוא כבר ידאג להבין בדיוק מה השורה הבאה שאתם רוצים וצריכים לכתוב

כלי חדש בשם TabNine נעזר בבינה מלאכותית ו-Deep Learning על מנת לקצר לכם תהליכים, ולחזות את שורת הקוד הבאה שאתם מעוניינים להקליד. כן, קצת כמו ה-Smart Compose ופיצ׳ר ה-autocomplete האהוב מהג׳ימייל או מהמקלדת שלכם בסמארטפון.

### מתממשק עם עורכי קוד קיימים

המודל מאחורי TabNine, מבוסס על GPT-2 של OpenAI, מודל עיבוד שפה טבעית שאומן על יותר מ-2 מיליון רשומות מפלטפורמת הקוד הפתוח GitHub. היתרון של זה בא לידי ביטוי בכך שמרגע שתוסיפו אותו לעורך הקוד שלכם, תוכלו לצאת לדרך, שכן הוא כבר סיים את מכסת האימונים שלו. יחד עם זאת, כמובן שתוך כדי שימוש הוא ישתפר וילמד אתכם, אבל הרעיון הוא שלא תצטרכו לתת לו תקופת חניכה מסוימת עד שהוא יכנס לתפקיד

לדברי המפתח שעומד מאחורי TabNine, לא מדובר כאן בטכנולוגיה שטרם ראינו, אבל – התוכנה שהם פיתחו טובה יותר, מכיוון שהיא עובדת על בסיס חיזוי. כלומר, בעוד שרוב משלימי הקוד האוטומטיים הקיימים כיום בשוק צריכים לנתח קודם כל את מה שהמשתמש כתב כדי להציע לו את הצעד הבא, TabNine מסתמכת על למידת מכונה, חיזוי, ומנתחת ״רמזים״, על מנת למצוא דפוסים סטטיסטיים, בהתאם לרשומות הרבות שהיא כבר ניתחה. בעזרת ה-AI, המודל מסתכל ומנתח את ״התחביר״ שכתבתם, כדי לזהות למשל מתי אתם מדברים על אובייקט, ומתי על רשימה של דברים  (app.get\_user לעומת app.get\_users), כפי שניתן לראות בתמונה מטה

# What exactly is GitHub Copilot?

GitHub Copilot is an AI tool created by GitHub and OpenAI to help programmers write code using autocompletion. Visual Studio Code, Neovim, and JetBrains users already can use the plugin.

Save translation

מה זה בעצם GitHub Copilot? GitHub Copilot הוא כלי AI שנוצר על ידי GitHub ו-OpenAI כדי לעזור למתכנתים לכתוב קוד באמצעות השלמה אוטומטית. משתמשי Visual Studio Code, Neovim ו-JetBrains כבר יכולים להשתמש בתוסף

GitHub Copilot is powered by the OpenAI Codex model, trained on natural language and billions of public source code lines, including GitHub projects.

The Copilot tool on GitHub can write the code or offer an alternative. The service supports all programming languages but works best with Python, JavaScript, TypeScript, Ruby, Java, and Go.

According to their data, 50% of developers on GitHub continued to use the service after the trial period in July 2021.

מופעל על ידי מודל OpenAI Codex, מאומן על שפה טבעית ומיליארדי שורות קוד מקור ציבורי, כולל פרויקטים של GitHub. כלי Copilot ב- GitHub יכול לכתוב את הקוד או להציע חלופה. השירות תומך בכל שפות התכנות אך עובד בצורה הטובה ביותר עם Python, JavaScript, TypeScript, Ruby, Java ו-Go. לפי הנתונים שלהם, 50% מהמפתחים ב-GitHub המשיכו להשתמש בשירות לאחר תקופת הניסיון ביולי 2021.

## What was good

The most significant value of Github Copilot is saving your time reading the documentation. For example, do you remember the key codes of arrow buttons to handle clicks on them? I don’t know either. Luckily with Copilot, **you don’t have to keep in mind key codes** or search it in google. Instead, just type the comment what do you want.

Translation is too long to be saved

מה היה טוב הערך המשמעותי ביותר של Github Copilot הוא חיסכון בזמן קריאת התיעוד. לדוגמה, האם אתה זוכר את קודי המפתח של לחצני החצים לטיפול בלחיצות עליהם? גם אני לא יודע. למרבה המזל עם Copilot, אתה לא צריך לזכור קודי מפתח או לחפש אותם בגוגל. במקום זאת, פשוט הקלד את התגובה מה אתה רוצה

Awesome, right?

But it works great not only with popular functions. It perfectly recognizes the context of your file and tries to write the code instead of you reusing existing variables and functions.

Let me show you **how to write an API service class** using GitHub Copilot.

