

Generative AI Tools for Developers: A Practical Guide

Copyright © 2023 SitePoint Pty. Ltd.

- **Author:** Timi Omoyeni
- **Cover Design:** Volodymyr Boyarinov
- **Technical editor:** Christian Heilmann
- **Series editor:** Oliver Lindberg

Notice of Rights

All rights reserved. No part of this book may be reproduced, stored in a retrieval system or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical articles or reviews.

Notice of Liability

The author and publisher have made every effort to ensure the accuracy of the information herein. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors and SitePoint Pty. Ltd., nor its dealers or distributors will be held liable for any damages to be caused either directly or indirectly by the instructions contained in this book, or by the software or hardware products described herein.

Trademark Notice

Rather than indicating every occurrence of a trademarked name as such, this book uses the names only in an editorial fashion and to the benefit of the trademark owner with no intention of infringement of the trademark.



Published by SitePoint Pty. Ltd.

71 Balmain Street, Cremorne

VIC Australia 3121

Web: www.sitepoint.com

Email: books@sitepoint.com

ISBN: 978-1-925836-59-2

About the Author

Timi is a frontend engineer who specializes in building web applications using Vue.js. He is also a technical writer who enjoys simplifying the process of learning for his readers. He writes on topics ranging from Vue.js to accessibility on the web, for platforms like Smashing Magazine and Vue Mastery.

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit sitepoint.com to access our blogs, books, newsletters, articles, and community forums. You'll find a stack of information on JavaScript, PHP, Ruby, mobile development, design, and more.

Table of Contents

Preface	v
----------------------	----------

Who Should Read This Book?	v
----------------------------------	---

Chapter 1: An Introduction to AI Chatbots:

ChatGPT vs. Bing Chat vs. Bard.....	7
--	----------

Introducing Chatbots	8
----------------------------	---

Large Language Models	12
-----------------------------	----

The State of Chatbots Today	15
-----------------------------------	----

ChatGPT	17
---------------	----

Bing Chat.....	21
----------------	----

Bard	24
------------	----

Wrapping Up	27
-------------------	----

Chapter 2: How Generative AI Can

Augment the Work of Developers.....	28
--	-----------

Learning in Software Development	29
--	----

AI in Software Development.....	31
---------------------------------	----

Pair Programming.....	33
The Pros and Cons of Using AI in Programming.....	43
Wrapping Up.....	45

Chapter 3: A Practical Guide to AI-

powered Pair Programming 47

Integrating AI-powered Tools.....	48
Wrapping Up.....	79

Preface

Who Should Read This Book?

This book is for developers who are interested in using AI tools professionally. It doesn't assume any existing familiarity with AI tools.

Code Samples

Code in this book is displayed using a fixed-width font, like so:

```
<h1>A Perfect Summer's Day</h1>
<p>It was a lovely day for a walk in the park.
The birds were singing and the kids were all back at school.</p>
```

Where existing code is required for context, rather than repeat all of it, `:` will be displayed:

```
function animate() {
  :
  new_variable = "Hello";
}
```

Some lines of code should be entered on one line, but we've had to wrap them because of page constraints. An `↵` indicates a line break that exists for formatting purposes only, and should be ignored:

```
URL.open("https://www.sitepoint.com/responsive-web-
↵design-real-user-testing/?responsive1");
```

Tips, Notes, and Warnings



Hey, You!

Tips provide helpful little pointers.



Ahem, Excuse Me ...

Notes are useful asides that are related—but not critical—to the topic at hand. Think of them as extra tidbits of information.



Make Sure You Always ...

... pay attention to these important points.



Watch Out!

Warnings highlight any gotchas that are likely to trip you up along the way.

An Introduction to AI Chatbots: ChatGPT vs. Bing Chat vs. Bard

Chapter

1

Artificial intelligence (AI) has been in the spotlight for a while now, and it's dividing opinions across the world. While some are enthusiastic about the advances in AI and welcome it, others have raised concerns surrounding job security, ethics, and privacy of users' data—all of which are valid concerns. Artificial intelligence is by no means a new topic or area of tech, but developers have spent years trying to improve on the technology and introduce new tools that are powered by AI.

In this series, we'll look at some of the applications of AI we're seeing today, and how AI can be a very useful tool in our day-to-day work as engineers. Innovations in AI have introduced some obvious benefits, such as efficiency and increased productivity. A nice example is [GitHub Copilot](#), an AI-powered, peer-coding partner that helps engineers improve their standard practices, write better tests, and be more productive.

The introduction of ChatGPT—a language model developed by [OpenAI](#)—has led to a recent buzz around AI. Several other AI-powered chatbots have also been introduced by rival companies. Given the current proliferation of AI-powered chatbots, it's a good time to take a look at chatbots and how they work.

In this part, we'll be looking at what chatbots are, their history, and the state of chatbots today. We'll also compare the newly introduced chatbots like ChatGPT, Bing Chat, and Bard.

Introducing Chatbots

A **chatbot** is a software application that aims to mimic human conversation through text or voice interactions, typically online. Chatbots first came into existence in 1966 when an MIT professor named Joseph Weizenbaum created [ELIZA](#), an early [natural language processing](#) computer program created to explore communication between humans and machines.

The image below ([from Wikipedia](#)) shows ELIZA in action.


```

Welcome to

EEEEEE LL      IIII ZZZZZZ AAAAA
EE      LL      II     ZZ  AA  AA
EEEEEE LL      II     ZZ  AAAAAA
EE      LL      II     ZZ  AA  AA
EEEEEE LLLLLL IIII ZZZZZZ AA  AA

Eliza is a mock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in 1966.
This implementation by Norbert Landsteiner 2005.

ELIZA: Is something troubling you ?
YOU:  Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:  They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:  Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:  He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:  It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:

```

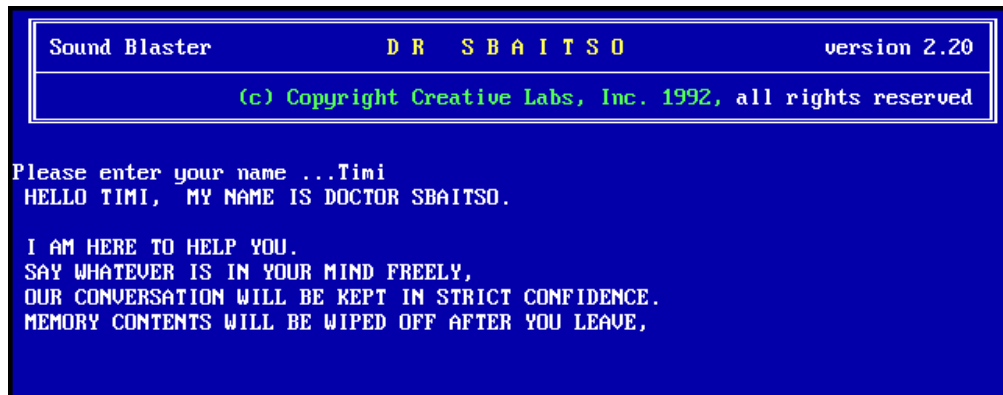
1-1. ELIZA in action

In 1994, computer scientist Michael Mauldin decided to call this kind of program a “chatbot”, after inventing Verbot, a chatterbot program and artificial intelligence software development kit for Windows and the Web.

The Evolution of Chatbots

Chatbots continued to evolve after ELIZA, finding different purposes ranging from entertainment (with Jabberwacky) to healthcare (with PARRY). The chatbots created during this period were intended to mimic human interaction under different circumstances. But in 1992, Creative Labs built Dr Sbaitso, a chatbot with speech synthesis. This was the first time machine learning was integrated into a chatbot, though it only recognized limited or pre-programmed responses and commands.

The image below shows the Dr Sbaitso interface.



1-2. Dr Sbaitso

Another chatbot called ALICE (Artificial Linguistic Internet Computer Entity) was developed in 1995—a program engaging in human conversation using heuristic pattern matching to conduct conversations.

All the chatbots released during this period are termed **“Rule-based chatbots”**, because they all operated on a set of predefined rules and patterns created by human developers or conversational designers to generate responses. This means these chatbots had limited flexibility, due to their reliance on predetermined rules. They lacked the ability to learn from a user’s message and generate a new response to it. Examples of such rules include:

- If a user asks about product pricing, respond with information about pricing plans.
- If a user mentions a technical issue, provide troubleshooting steps.
- If a user expresses gratitude, respond with a thankyou message.

In 2001, ActiveBuddy, Inc. publicly launched a new chatbot that was called SmarterChild. It was an intelligent bot distributed across global instant messaging networks (AIM, MSN, and Yahoo Messenger) that was capable of providing information ranging from news, weather, sports, stock information, and so on, and that allowed users to play games and also access the START Natural Language Question Answering System by MIT’s Boris Katz. It was revolutionary, as it demonstrated the power of conversational computing, and in many ways it can be said to have been a precursor of Siri.

The next set of remarkable developments in chatbots came in the 2010s, partly due to the growth of the Web and the availability of raw data. During this period, great progress was made in natural language processing (NLP), as representation learning and deep neural network-style machine learning methods became widespread in NLP.

Some of the achievements of this period include:

- **Deep learning and neural networks.** Significant developments were made in recurrent neural networks (RNNs) that made them capable of capturing complex linguistic patterns, contextual relationships, and semantic understanding, contributing to significant improvements in chatbot performance.
- **Sentiment analysis and emotion understanding.** Sentiment analysis and emotion understanding were added to NLP techniques in the 2010s. Chatbots also incorporated these capabilities, allowing them to recognize user sentiments and emotions while responding appropriately to them. This development enhanced the chatbot's ability to provide empathetic and personalized interactions.
- **Named entity recognition and entity linking.** The process of named entity recognition (NER) and entity linking also got better when Alan Ritter used a hierarchy based on common Freebase entity types in ground-breaking experiments on NER over social media text.
- **Contextual understanding and dialogue management.** Language models became more proficient at understanding and maintaining contexts within a conversation, and consequently chatbots got better at handling conversations while providing more coherent responses. The flow and quality of interactions also improved as a result of reinforcement-learning techniques.
- **Voice-activated virtual assistants.** There was massive development in areas like NLP, AI, and voice recognition technologies from the 1990s to the 2010s. The combination of these led to the development of smart, voice-activated

virtual assistants with better audio than Dr Sbaitso, which was the first voice-operated chatbot. A notable example of assistants developed in this era was Apple's Siri, which was released in 2011, and which played a pivotal role in popularizing voice-based interactions with chatbots.

- **Integration of messaging platforms and APIs.** As a result of the progress being made in the field of AI, there's been a rise in the adoption of chatbots by messaging platforms such as Facebook, Slack, WhatsApp, and so on. These platforms have also made it possible for users to develop and integrate into them their personalized chatbots with different capabilities, by providing them with APIs and developer tools—all of which have ultimately led to the adoption of chatbots across various industries.

All of these advancements made it possible to build chatbots that were capable of having better conversations. They had a better understanding of topics, and they offered an experience that was better than the scripted feel of their predecessors.

Large Language Models

In the early days of the Internet, search engines weren't as accurate as they are now. Ask.com (originally known as Ask Jeeves) was the first search engine that allowed users to get answers to questions in everyday, natural language. Natural language search uses NLP, a process which uses a vast amount of data to run statistical and machine learning models to infer meaning in complex grammatical sentences. This has made it possible for computers to understand and interact with human language, and it has paved the way for various applications. NLP has facilitated a remarkable evolution, with the emergence of large language models.

A **large language model** (LLM) is a computerized language model that can perform a variety of natural language processing tasks, including generating and classifying text, answering questions in a human-like fashion, and translating text from one language to another. It's trained on a massive trove of articles, Wikipedia entries, books, internet-based resources and other input, so it can learn how to generate responses based on data from these sources.

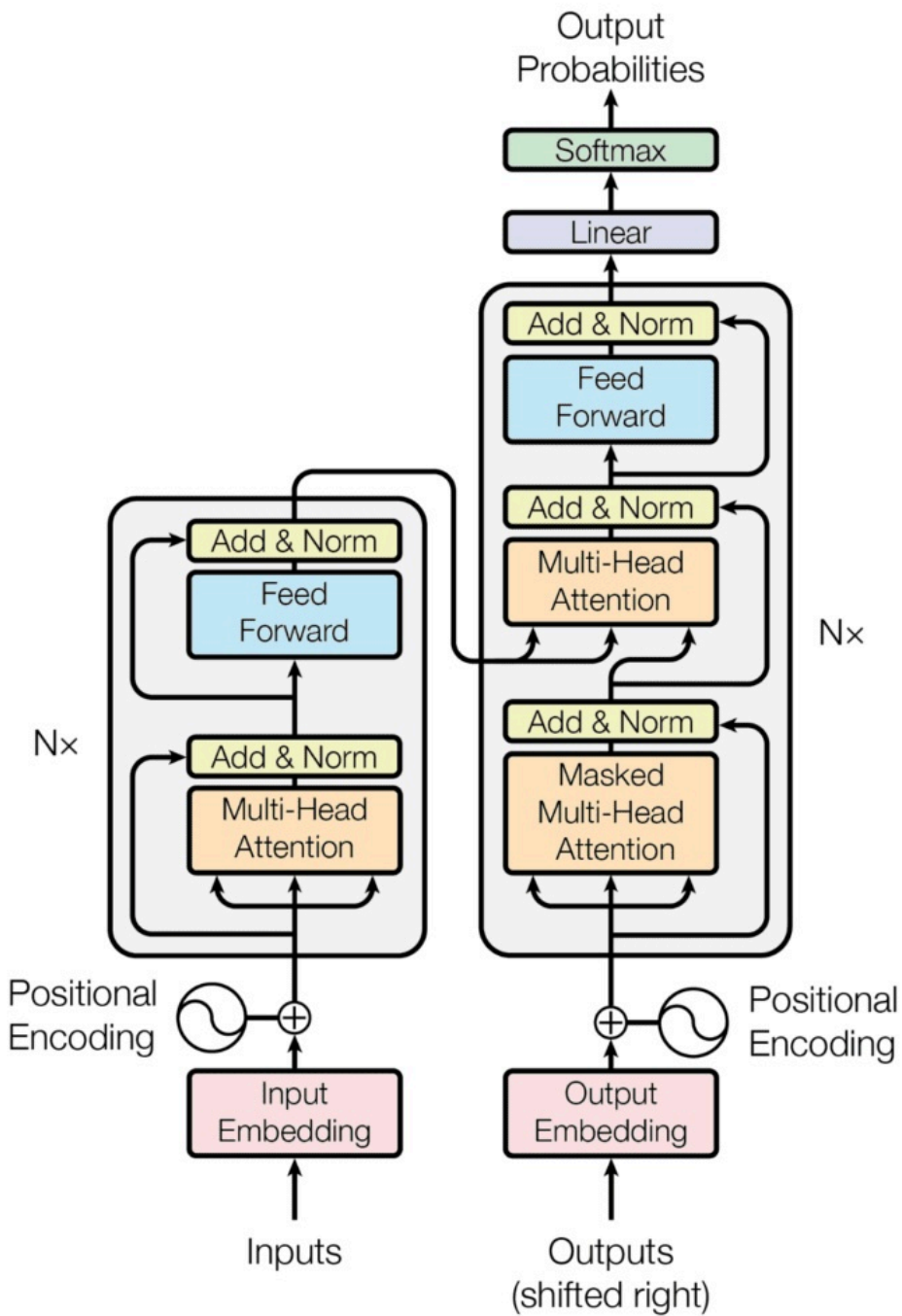
The underlying architecture of most LLMs is one of two types:

- Bidirectional Encoder Representations from Transformers (BERT)
- Generative pre-trained transformers (GPTs)

These LLMs are all based on the transformer model architecture. Transformers are a type of neural network architecture that has revolutionized the field of natural language processing and enabled the development of powerful large language models.

It uses self-attention mechanisms to calculate a weighted sum for an input sequence and dynamically determine which tokens in the sequence are most relevant to each other.

The image below depicts how the transformer model architecture works.



How LLMs Work

In order to understand how LLMs work, we must first look at how they're trained. Using large amounts of text from books, articles, and various parts of the Internet, they learn the patterns and connections between words. This is the first step, known as **pre-training**. It utilizes distributed computing frameworks and specialized hardware such as graphics processing units (GPUs) or tensor processing units (TPUs), which allow for efficient parallel processing. After this is done, the pre-trained model still needs to know how to perform specific tasks effectively, and this is where fine-tuning comes in.

Fine-tuning is the second step in training LLMs. It involves training the model on specific tasks or data sets to make it more specialized and useful for particular applications. For example, the LLM can be fine-tuned on tasks like text completion, translation, sentiment analysis, or question-answering.

The State of Chatbots Today

Today, we have chatbots that are more powerful than ever before. They can perform more complex tasks and are also better at handling conversations. This is because there have been significant advancements in AI, NLP, machine learning, and an increase in computing power and internet speed.

Chatbots have continued to take advantage of these advancements. Some of the notable aspects of these advancements include:

- **Advanced AI models.** The introduction of advanced AI models has revolutionized the capabilities of chatbots in recent years. Models such as OpenAI's GPT series have immensely helped to push the boundaries of natural language processing and machine learning. These models are trained on extensive datasets and can generate contextually relevant responses, making conversations with chatbots more engaging and human-like.
- **Multichannel and multimodal capabilities.** Chatbots are no longer limited to a

single platform or interface, as they can seamlessly operate across channels like websites, messaging apps/platforms, and mobile apps. Although they're often behind a paywall, chatbots have also expanded beyond text-based interactions and now support multimodal inputs, including images and voice, providing users with the freedom to engage through different mediums.

- **Continuous learning and adaptability.** Continuously learning and improving from user interactions, chatbots employ reinforcement learning and feedback mechanisms to adapt their responses over time, refining their performance and better meeting user needs.
- **Industry applications.** Extensive applications across industries have been found for chatbots. For instance, Airbnb makes use of chatbots in assisting users to answer FAQs, resolving booking issues, and finding accommodation, while Duolingo utilizes a chatbot in simulating conversations in foreign language learning and feedback. They're also used in other industries such as financial institutions, healthcare, and ecommerce. This usually requires providing these bots with domain-specific knowledge in order for them to do a great job in their respective use cases.
- **Integration with backend systems.** Due to this immense growth, we now have chatbots that are being integrated with backend systems and databases. This allows them to access and provide up-to-date information, which further enhances their ability to provide accurate and up-to-date responses to user queries.

As a result of all these developments, we now have far more intelligent chatbots that are capable of handling several tasks on different scales, ranging from booking a reservation at your favorite restaurant, or performing extensive research on various topics with references, to solving technical issues in software development. Some of the most popular chatbots that we have today include Google's [Bard](#), Microsoft's [Bing Chat](#), and OpenAI's [ChatGPT](#), all of which are powered by large language models.

ChatGPT

ChatGPT (Chat Generative Pre-trained Transformer) is an AI program built on a model that's based on a transformer architecture. It was trained on vast amounts of text data and code, and it can generate natural language responses, write code, translate languages, and so on. It was developed by OpenAI and was launched on November 30, 2022. Within a few days of the launch, it had over a million users. In January 2023, it had about 100 million monthly active users. This rapid user growth also led to the reemergence of AI as a trending topic among people and companies alike.

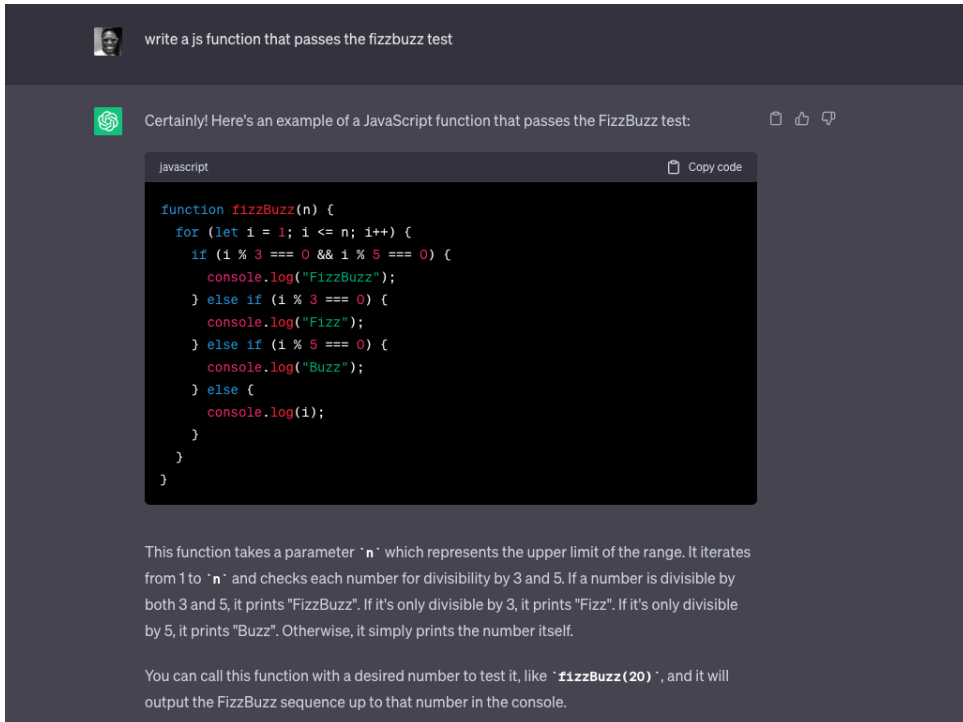
ChatGPT is built on GPT-3.5 and GPT-4, both of which are products of OpenAI's proprietary series of foundational GPT models.

Features of ChatGPT

ChatGPT has a lot of amazing features, some of which we'll look at below:

- **Natural language understanding.** Based on the type of data ChatGPT was trained on, it comes with the ability to generate human-like text, making the conversations feel more natural and engaging. It can understand and generate responses across a wide range of topics, handling both open-ended discussions and specific queries.
- **Knowledge of code.** ChatGPT was trained on a vast amount of text data and code, including programming languages, code snippets, documentation, and discussions on platforms like GitHub and Stack Overflow. This makes it possible for it to understand and write code in several languages. For example, it can solve the FizzBuzz test, a popular interview question that involves looping through numbers in a certain range to determine if they're either multiples of 3, 5, or both 3 and 5 (15, 30). This is powered by OpenAI Codex, a system that translates natural language into code.

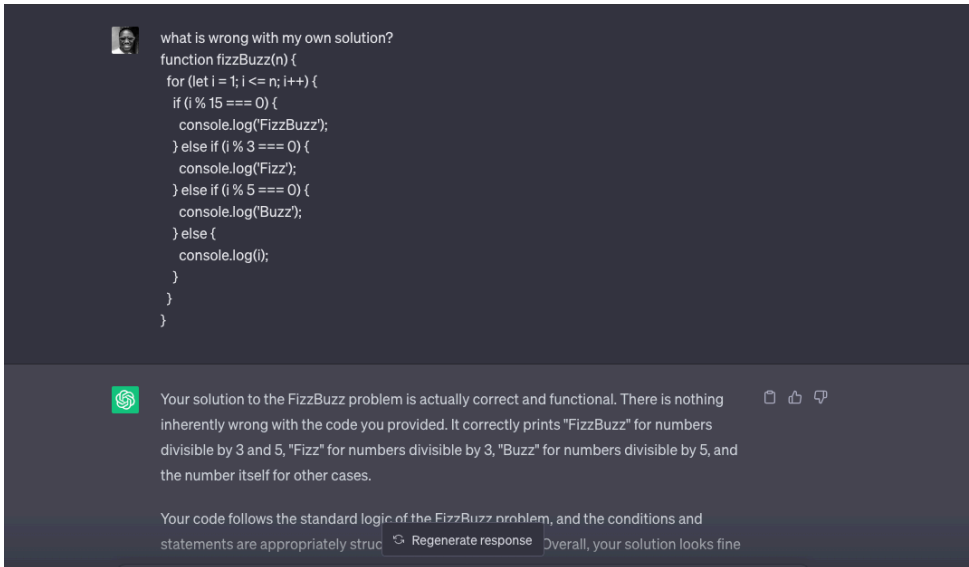
The image below shows a ChatGPT FizzBuzz response.



1-4. ChatGPT fizzBuzz response

- **Contextual responses.** One of the best features of ChatGPT is the ability for it to provide responses based on the flow of the current conversation being held. This means it's capable of storing users' messages in memory to analyze and provide future responses concerning them.

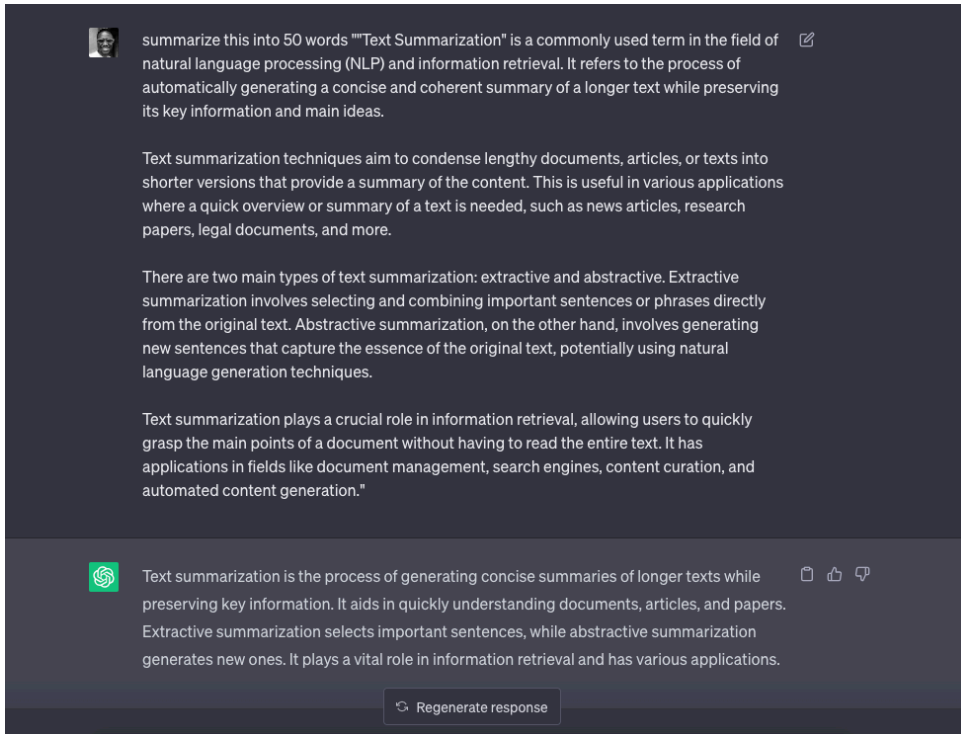
The image below shows a ChatGPT contextual response.



1-5. ChatGPT contextual response

- **Text summarization.** Another popular feature of ChatGPT is the ability to take lengthy text, documents, or articles and provide a summary, while also preserving key information and ideas.

The image below shows ChatGPT summarizing an article.



1-6. ChatGPT summarizes an article

While the knowledge of ChatGPT's free version is limited to information gathered up until 2021, users who want to use it to fetch up-to-date information from the Web—along with other cool benefits like plugins—can do so on ChatGPT Plus, a premium version that costs \$20 per month. Some of the benefits of ChatGPT Plus include:

- **Greater availability.** One of the benefits of being a ChatGPT Plus user is the unlimited access to the service that comes with it, even during periods of high demand.
- **Access to new features.** As ChatGPT gains new features, paying users get access to these improvements before anyone else.
- **Access to GPT-4.** The free version of ChatGPT is currently powered by the GPT-3.5, which has proven to be a very powerful model. However, OpenAI also

has GPT-4, a more capable language model that's only available to users on the paid version of the app.

Limitations of ChatGPT

As amazing as ChatGPT is, there's still room for improvement:

- **Input sensitivity.** ChatGPT currently exhibits a sensitivity to how user input is phrased, thereby leading to different responses for the same request. For instance, the model may claim not to know the answer to a question when presented with one phrasing, but provide the correct answer when the same question is slightly rephrased.
- **Lack of updated data.** The free version of ChatGPT doesn't have access to the Internet and, as such, it's only capable of providing information regarding certain topics up until September 2021. However, users can get access to up-to-date information by upgrading to ChatGPT Plus.
- **Verbosity.** The model is often excessively verbose and overuses certain phrases. These concerns can be attributed to biases in the training data, where trainers favor longer responses for perceived comprehensiveness.
- **Inability to ask clarifying questions.** The model has a limitation where it's not able to seek clarification from users when their request is ambiguous or unclear. Instead, it usually suggests or guesses what the user intended.

Bing Chat

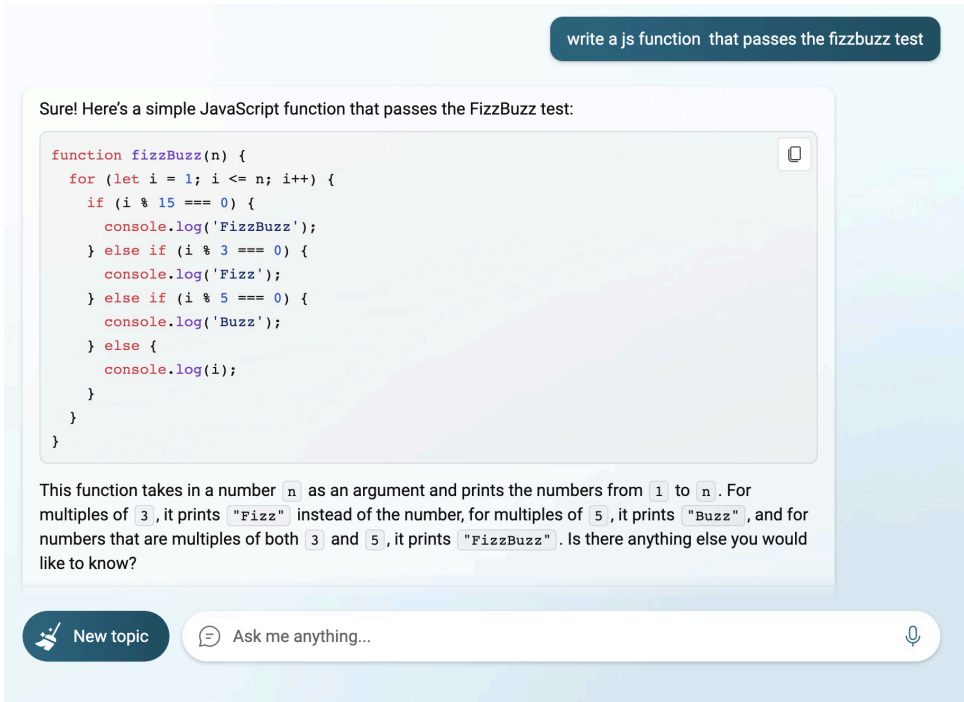
Bing Chat is an AI-powered chatbot built on the next-generation OpenAI large language model, GPT-4, which is faster, more accurate, and more capable than GPT-3 and GPT-3.5. It comes with the version of Bing that was announced on February 7, 2023. It's currently only available to Microsoft Edge users, as it comes integrated with Bing Search.

Bing Chat Features

Some of Bing Chat's features are listed below:

- **Multilingual support.** Users of Bing Chat aren't limited to a particular language, as it comes with support for several languages both via voice input and text.
- **Up-to-date information.** Bing Chat is integrated with Bing, a search engine owned by Microsoft, so it has access to the Internet. This makes it possible to provide its users with updated information in real time.
- **Code generation.** Another exciting feature of Bing Chat is the ability to both read and write code. This is because it's powered by a language model that's been trained on a large dataset that includes examples of code in various programming languages.

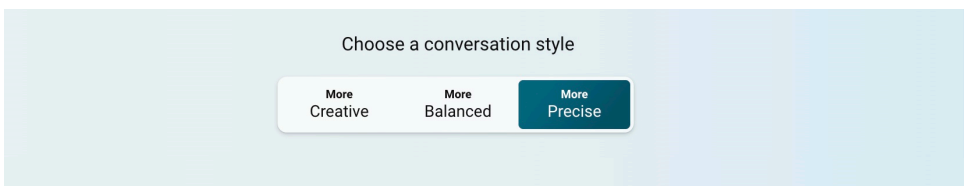
The image below shows Bing Chat attempting the FizzBuzz test.



1-7. Bing Chat attempting the FizzBuzz test

Conversation style. As part of the flexibility of Bing Chat, users can choose a conversation style depending on what they're trying to achieve. For instance, "More Creative" will work best when trying to draft a cover letter, while "More Precise" will be a better option when conducting research for work or an assignment.

The image below shows Bing Chat conversation options.



1-8. Bing Chat conversation options

Limitations of Bing Chat

As with ChatGPT, there's still room for Bing Chat to improve:

- **Limited number of questions.** Bing Chat users currently have a cap on the number of questions they can ask per day. This limit was set in place after multiple users reported that it had either been providing inaccurate responses or rude answers. It's currently limited to 30 chats per session and 200 total chats per day. Once this limit has been exceeded, users get a prompt to try again in 24 hours.
- **Only available to Microsoft Edge users.** Currently, Bing Chat lives inside the Microsoft Edge browser, as it's integrated with the search engine and can only be accessed from the browser. This makes it impossible for non-Microsoft Edge users to take advantage of its many features.

Bard

Bard is an AI-powered chatbot developed by Google AI. It was originally powered by the Language Model for Dialogue Applications (LaMDA) family of large language models. Later, it was updated to be powered by the PaLM 2 LLM (Pathways Language Model). PaLM is a state-of-the-art language, Transformer-based model with improved multilingual, reasoning, and coding capabilities, trained using a mixture of objectives similar to UL2 (Unified Language Learner), a breakthrough language pre-training paradigm that boosts the effectiveness of language models in every given setting and data set.

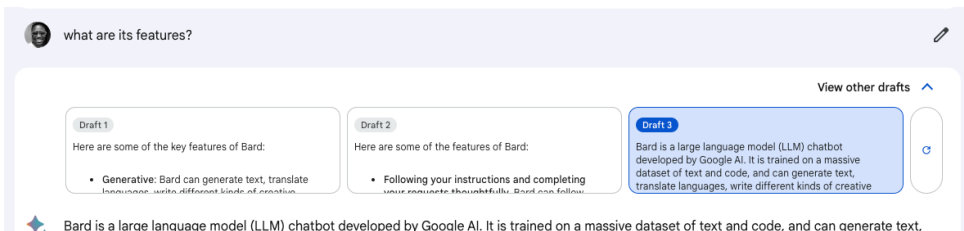
Bard was first released on March 21, 2023, in a limited capacity depending on users' location, before it was finally opened to the general public on May 10, 2023.

Bard Features

Bard has a lot of interesting features and capabilities, some of which are listed below:

- **Multiple response options.** Bard offers the ability for users to switch between a variety of available responses without having to rephrase their message. This presents users with a variety of answers to the same question or request so they can have a different, better, and clearer understanding of their request.

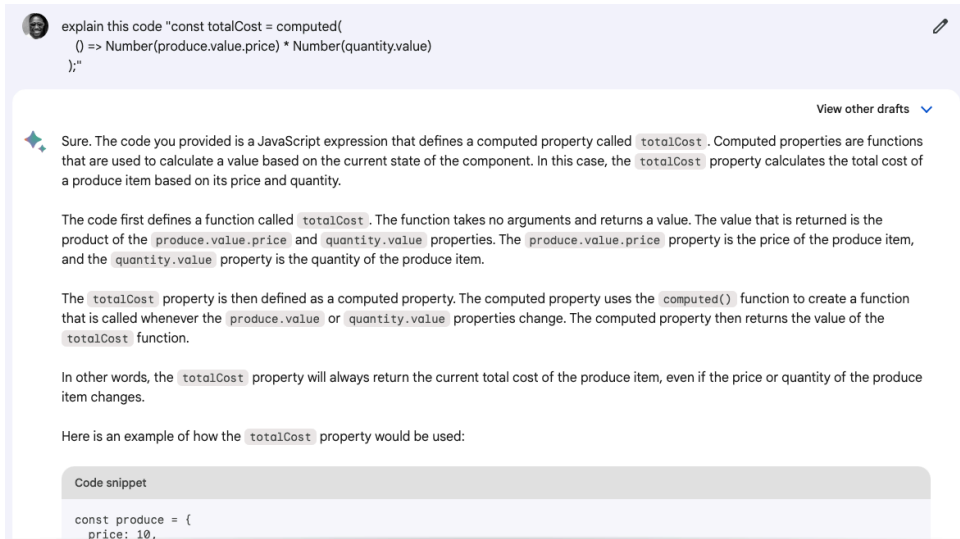
The image below shows Bard offering multiple answers.



1-9. Bard multiple answers

- **Voice input.** Bard offers a voice feature where its users can interact using their voice. This feature comes in handy when trying to dictate a lengthy sentence, and it's also good for accessibility.
- **Code explanation.** Bard can also serve as a peer coding partner, as it's capable of understanding and explaining code snippets. While the code in question may not always be accurate, it can serve as a good starting point for understanding the block of code.

The image below shows Bard explaining a code snippet.



1-10. Bard explaining a code snippet

Limitations of Bard

Since its release, Bard has proven to be quite useful as both a research assistant and a personal assistant due to the number of features it comes with, but it's still under development and has some limitations. A few of those limitations are listed below:

- **Bias.** Bard was trained on a data set of text and code that may be biased, so it can be biased in its responses.
- **Inconsistencies.** Bard has the potential to deliver inconsistent and inaccurate responses, which can lead to user confusion. It's important for users to be mindful of these inconsistencies and to exercise critical evaluation when considering the reliability of information obtained from Bard.
- **Security.** Bard possesses the capability to generate content that may be harmful or offensive, which can be based on the user's original input. To ensure responsible usage, it's important to exercise caution and be aware of the limitations associated with Bard.

Wrapping Up

Chatbots have come a long way since ELIZA, the first chatbot. In the last few years, there's been a lot of development and growth in AI and machine learning, with more advanced language models and NLP techniques. Despite these developments, chatbots are still far from perfect, as there are concerns around issues such as hallucinations. But one thing is clear: there will be continuous developments and improvements in this space.

In the next tutorial, we'll explore how we can enhance our learning and skill acquisition using the latest advancements in AI and chatbot technologies.

How Generative AI Can Augment the Work of Developers

Chapter

2

There's no easy path to becoming a programmer. Learning to be a software engineer takes a lot of time and effort, and because of the rapid rate of change in the software industry, the learning process must remain an on-going effort. Thankfully, there are both collaborative industry practices that help with the learning process, and also new and better tools for helping developers get up to speed and *stay up to speed* in this evolving industry.

There's a lot of pressure on software engineers to work efficiently and to stay up to date with advances in technology, programming languages and best practices. In this tutorial, we're going to cover how we can harness the power of AI to get started in programming, work efficiently, and continue to grow as software engineers. In particular, we'll focus on tools that are revolutionizing pair programming, which is a fundamental practice for learning web development skills and staying up to date. We'll also look at how these tools can be used to increase efficiency and improve the quality of our code.

Learning in Software Development

Wikipedia defines learning as the process of acquiring new understanding, knowledge, behaviors, skills, values, attitudes, and preferences. Learning is a lifelong journey that starts the moment we're born, including learning to crawl, walk, talk, and all the other skills we need to acquire to function as full-fledged participants in the world. Nowadays, we're even seeing the rise of machines that can learn, which is key to the power of recent developments in AI.

The ability and willingness to learn are key to being successful as a professional developer—whether it's because we find ourselves in situations that require us to pick up a new skill, or because we make a thoughtful and deliberate decision to learn something new (known as **intentional learning**). Either way, we need to be able to seek out information, resources, or opportunities for learning, driven by a clear intention to achieve certain outcomes.

Even if we're planning to specialize in a narrow field of engineering, software development requires a wide range of expertise and knowledge. This often involves learning a new technology to be able to understand and execute the task at hand. Continuous learning is vital in our lives as engineers in the dynamic world

of technology. As professionals, we constantly encounter evolving frameworks, programming languages, and best practices. Embracing lifelong learning becomes imperative to stay current and progress in our careers, irrespective of experience levels. Employers expect engineers to be well versed in the rapidly changing landscape of technological advancements and possess the necessary software engineering skills. Consequently, investing in continuous learning is a strategic approach that ensures engineers remain competent and relevant in the industry.

To meet these expectations, many companies recognize the significance of upskilling their workforce and allocate budgets for professional development. By providing engineers with opportunities to expand their knowledge and expertise, companies foster a culture of growth and innovation. Engineers are encouraged to pursue specialized training, attend workshops, or engage in courses to enhance their abilities. This not only benefits individual engineers but also enhances the overall productivity and competitiveness of the organization.

Here are a few reasons why continuous learning is important in software development:

- **Personal growth.** As engineers continue to adapt to the changes in the tools, frameworks, and languages they use in their day-to-day work, it helps increase their knowledge base in said technology. It also adds to their skill set, especially if learning results in acquiring a new tool (such as new programming language). It also provides an edge in the job market, as employers and interviewers often look out for engineers who are up to date with changes in their field.
- **Enhancing problem-solving skills.** During the process of learning, we're often faced with the frustration that arises from unfamiliar situations. But with perseverance, we eventually find new and innovative ways to solve a problem. Although this is a result of the growth mindset, constantly doing this helps us develop various ways in which we can solve problems.
- **Keeping up with industry standards.** As technologies continue to evolve, some of the industry standards associated with them also change. With

continuous learning, developers can remain compliant with these standards, which is crucial for producing high-quality, secure, and maintainable code.

- **Embracing new challenges.** Continuous learning also allows developers to develop an open mind towards taking on new opportunities, which can come in various forms, ranging from a new project, a new role, or a new framework.

AI in Software Development

Generative AI (also known as GenAI) is a type of artificial intelligence involved in generating a range of media using generative models. These models take in massive amounts of training data and use it to learn patterns and structures, which they then use to generate new data with similar characteristics.

Generative AI systems include large language model chatbots such as ChatGPT, Bing Chat, and Bard—which we covered in the first tutorial—and other tools such as text-to-image art systems like Stable Diffusion, Midjourney, and DALL-E.

With the rapid rise of AI, all sorts of AI-powered tools are starting to appear in areas such as photography, video editing, image creation, writing, and of course coding. Companies are integrating AI into the workflow of their users—such as Adobe, with its Generation AI for designers. AI's ability to analyze, process, and understand vast amounts of data has paved the way for a new generation of tools that can significantly enhance the process of software development and of learning to code.

Contrary to the belief that AI will take the jobs of engineers, it's merely a tool that can enhance human capabilities. AI can be integrated into a developer's workflow to handle certain tedious tasks like testing and debugging, enabling developers to spend less time on mundane tasks and more time doing meaningful and innovative work. The new AI-based tools are a boon not only because they help developers work more efficiently, but also because they introduce new opportunities for learning more quickly, efficiently and reliably.

With the introduction of language models like GPT-3.5 and GPT-4, chatbots like ChatGPT and Bing Chat can understand, write, and explain code. While these

chatbots are mostly found in the browser, AI-powered tools created for the sole purpose of performing tasks related to software development are also starting to appear.

GitHub was one of the first companies to create an AI-powered tool capable of reading, understanding, and generating code, called GitHub Copilot, which is currently powered by OpenAI Codex.

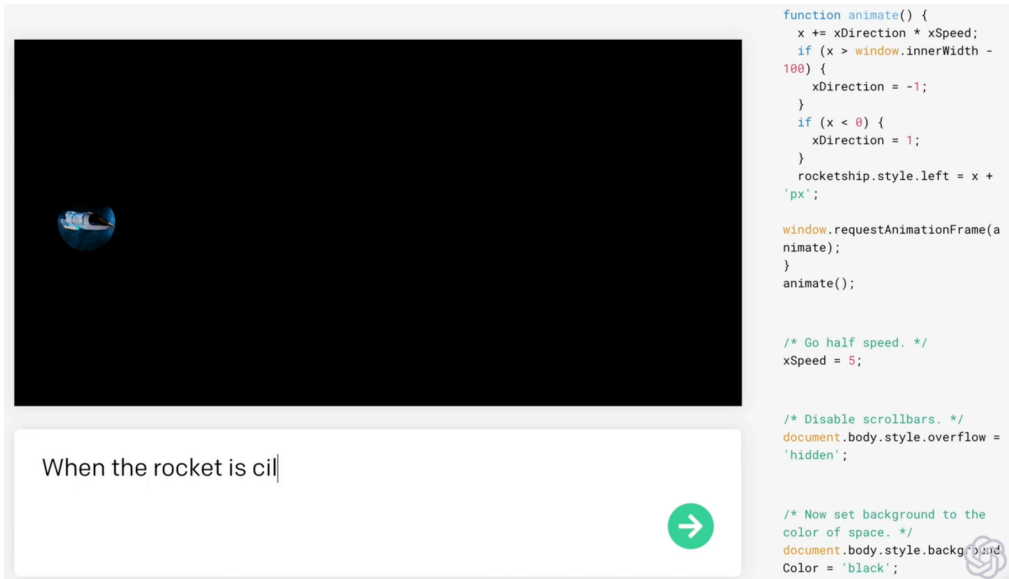
OpenAI Codex

OpenAI Codex is a language model developed by OpenAI that's capable of interpreting simple commands in natural language and executing on a user's behalf. This makes it possible to build a natural language interface for existing applications. It's a descendant of GPT-3 and is proficient in over a dozen programming languages including JavaScript, Go, Ruby, Swift, TypeScript, and Python.

Although it was built on GPT-3 (a model trained using data from publicly available sources), Codex was also trained on 54 million public repositories hosted on GitHub—159 gigabytes of code—making it better at understanding and processing code.

As a result of its understanding of natural language, Codex can take commands in English such as “Create a div with a box shadow and rounded edges” and produce working code. It may not work every time, but it produces a coded solution that's close to the intended result.

The image below shows OpenAI Codex in action.



2-1. OpenAI Codex in action

Additionally, OpenAI Codex is a versatile programming model, capable of handling various programming tasks beyond writing code from scratch, including explaining and refactoring code.

However, despite all the capabilities and promises of the model, in March 2022 OpenAI closed API access to Codex, instead offering access through its [researcher access program](#).

Pair Programming

Pair programming (also sometimes called “peer programming”) is a process in software development in which two programmers work together at one workstation to improving quality, share knowledge, and increase efficiency.

In pair programming, one of the programmers takes on the role of the **driver**—the person who actively writes the code and implements the logic—while the other team member takes up the role of the **observer** or **navigator**, reviewing the code being written, checking for errors, suggesting improvements, and thinking strategically about the design. These roles aren’t fixed, as they can be switched

at any time depending on the situation at hand.

Pair programming has several benefits, such as:

- **Improved code quality.** The popular saying “two heads are better than one” is one of the benefits of pair programming. It allows both engineers to make the best choice on logic and approach to issues. Ultimately, this leads to higher-quality code that’s more reliable and easier to maintain.
- **Efficiency.** By improving the quality of code in the codebase, developers get to be more efficient by spending time accomplishing new goals and adding new features to the codebase, thereby increasing productivity in general.
- **Fewer bugs.** During pair programming, it’s easier to spot bugs and fix them. This not only improves code quality in production but also leads to fewer bugs that will need to be fixed in the future. This allows developers to focus on adding new features and improving the software.
- **Good for upskilling.** Pairing is a great way to expose junior team members to advanced coding techniques. It’s also an opportunity to witness firsthand how more experienced teammates approach complex problems and implement efficient solutions.
- **Enhanced code understanding.** Both the driver and observer gain a deeper understanding of the codebase through pair programming. This knowledge transfer ensures that team members are familiar with all parts of the project.

The options for pair programming have increased significantly with developments in artificial intelligence. In the next section, we’ll look at some examples of AI-powered tools that are proving to be game changers in the field of software engineering and learning how to code.

AI-powered Pair Programming Tools

As a result of all the work that’s gone into AI and language models, a lot of AI-powered pair programming tools are starting to appear—tools that are capable of

understanding both natural language and code. There are now tools that can process natural language and produce code in return, translate code from one language to another, and also explain blocks of code in English. These tools offer a significant boost in productivity for developers and options for learning new skills.

Some of these tools are powered by OpenAI Codex, GPT-4, and other language models. They include:

- GitHub Copilot
- Replit's Ghostwriter
- Sourcegraph's Cody
- Amazon's CodeWhisperer

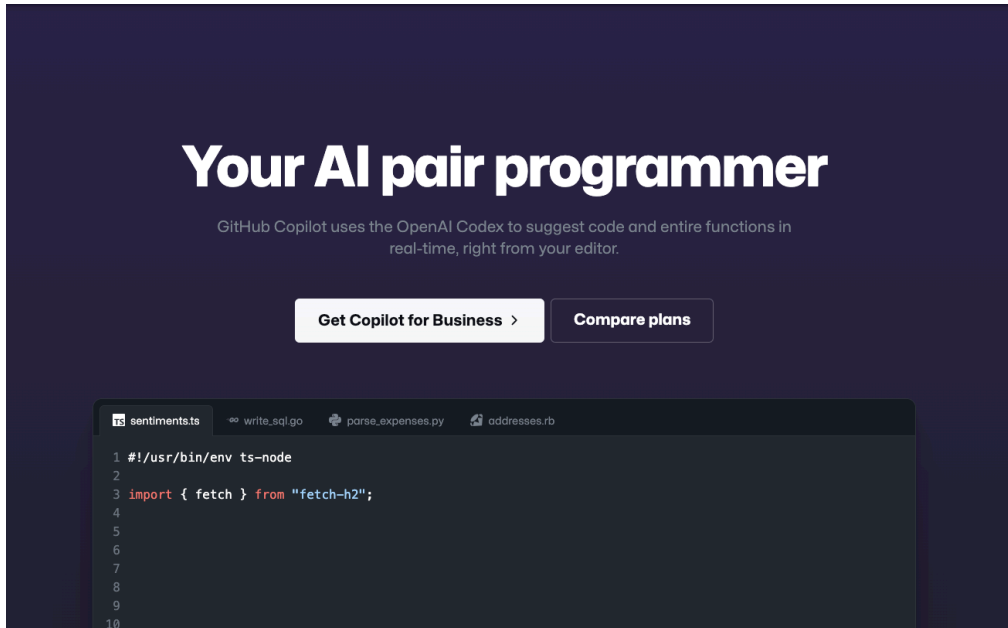
Let's look at each of these tools in turn below.

GitHub Copilot

GitHub Copilot is an AI-powered tool that acts as a pair programmer, allowing developers to provide commands using natural language while offering autocomplete-style suggestions in several programming languages directly in their code editor.

By analyzing the context within the file being edited, and considering related files in the codebase, Copilot takes nuances into account and provides highly relevant and valuable suggestions.

Copilot was first announced by GitHub on June 29, 2021, for technical preview as an extension in the Visual Studio Code text editor, and it's currently powered by OpenAI Codex. It's available to everyone as an extension for Neovim, Visual Studio, Visual Studio Code, and the JetBrains suite of IDEs.



2-2. GitHub Copilot

Currently, Copilot is capable of suggesting multi-line functions, speeding up test generation, and filtering out common, vulnerable coding patterns.

GitHub currently offers two types of subscriptions to Copilot customers:

- Copilot for individuals: \$10/month or \$100/year
- Copilot for business: \$19/user per month

Along with Copilot, GitHub also offers a companion VS Code extension called Copilot Labs. It's a separate extension that depends on GitHub Copilot. It offers users access to experimental features that help make Copilot better. Some of its features include:

- **Explaining code.** This feature allows developers to highlight a block of code and ask Copilot to explain it using natural language. This is a very useful feature to have, especially when dealing with a codebase that has several people working on it. With the ability for Copilot to explain code, time spent on onboarding new team members or understanding a codebase is reduced. It

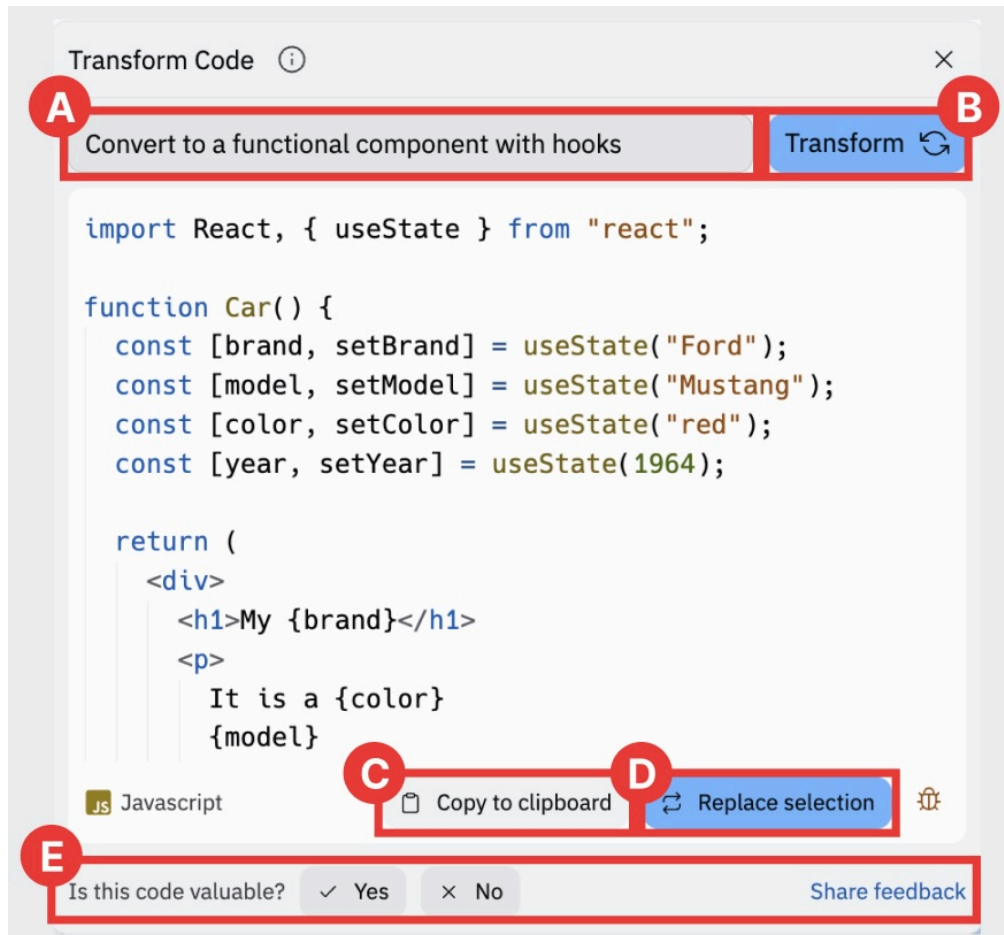
also provides an excellent resource for increasing the confidence and experience of those learning to code.

- **Writing tests.** Writing tests is another important part of software development that can sometimes be tedious and overlooked. With TestPilot, Copilot can suggest tests based on existing code and documentation.
- **Code translation.** Language translation is another amazing Copilot power. It involves converting a block of code from one language (such as JavaScript) to another (such as Ruby). Language translation can come in handy when dealing with unfamiliar languages. Translating the code into a familiar language makes it easier to understand how it works, and it provides an exciting new avenue for gaining experience with new languages.
- **Task automation.** Repetitive tasks like writing documentation, code refactoring, adding comments, and so on, are often tedious and error-prone when done manually. Such tasks can be handled much more reliably and efficiently with Copilot.

Ghostwriter

Ghostwriter is a pair programmer powered by machine learning that completes code in real time. It includes tools for generating, transforming, and explaining code, and an in-editor search utility that helps developers find and import open-source code without leaving the editor. Created by Replit, it was first announced for beta testing on September 8, 2022, and then opened to the public on October 31, 2022.

The image below shows Ghostwriter's transform feature.



2-3. Ghostwriter at work

Ghostwriter was built using [Salesforce's CodeGen](#) open-source large language model (LLM). It has been optimized using several techniques to achieve super-low latency and a median response time of less than 400ms.

It's capable of the following:

- **Code completion.** Ghostwriter can generate code and provide autocomplete suggestions for developers as they type.
- **Code explanation.** Ghostwriter provides step-by-step explanations of code in

the editor using the [Explain Code](#) feature.

- **Refactoring blocks of code.** Ghostwriter can refactor blocks of code and translate them into different languages using the [Transform Code](#) feature.
- **Generating code.** Ghostwriter can also [generate code](#) when given a command using a natural language prompt (English). The length of the code generated can vary from a few lines of code to several related functions at once.

Combining all of its features allows users to automate repetitive tasks and refactor their codebase faster and much more easily. It's also a great resource for those learning to be software engineers, with its ability to explain and generate code with prompts.

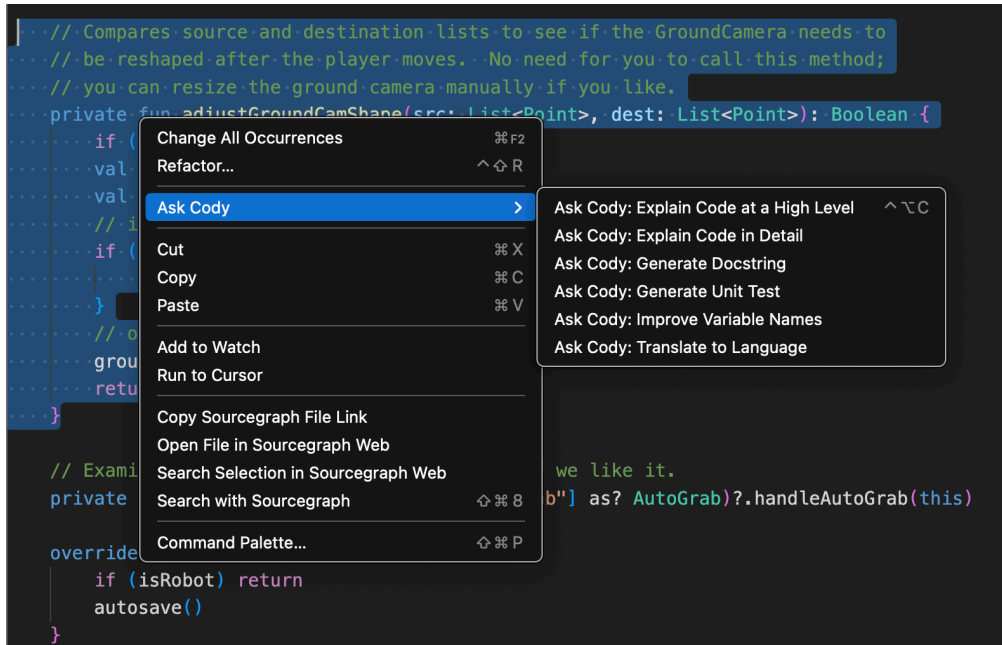
Replit's Ghostwriter can be accessed either through the [Pro plan](#) or through [Cycles](#) (for 1,000 cycles per month).

Cody

[Cody](#) is an AI coding assistant created by Sourcegraph. It writes code and answers questions by reading an entire codebase and examining a [code graph](#). It's capable of writing unit tests, generating doc comments, summarizing code, and generating code autocomplete suggestions. It can do this by looking around a codebase for context to accurately predict what a developer is trying to implement before providing autocomplete suggestions. It uses a combination of Sourcegraph's code graph and LLMs to eliminate toil and keep human developers in flow.

It also comes with a chat interface that has complete knowledge of a user's codebase and allows for questions about it and other things like the coding environment. It was first [announced](#) on March 23, 2023.

The image below shows Cody on VS Code.



2-4. Cody Interface on VS Code

Cody is currently open to both new and existing Sourcegraph users. Those looking to try out Cody can easily [create an account](#) and get started with Cody for free. This can be done either on the [Sourcegraph site](#) or via a [VS Code extension](#).

Based on its knowledge of a codebase and added context depending on current and recently opened files, Cody is capable of doing the following:

- **Test generation.** Cody can generate unit tests using its knowledge of a codebase.
- **Natural language processing.** Cody is capable of understanding natural language and, as a result, can write and refactor code using natural language instructions.
- **Autocomplete suggestions.** Cody takes into account a developer's current file and workflow and uses it to offer autocomplete suggestions in real time during coding.

- **Chatbot.** Cody also offers a chatbot interface that has knowledge of a codebase. This chatbot can write code and answer questions regarding a developer's codebase using this knowledge.

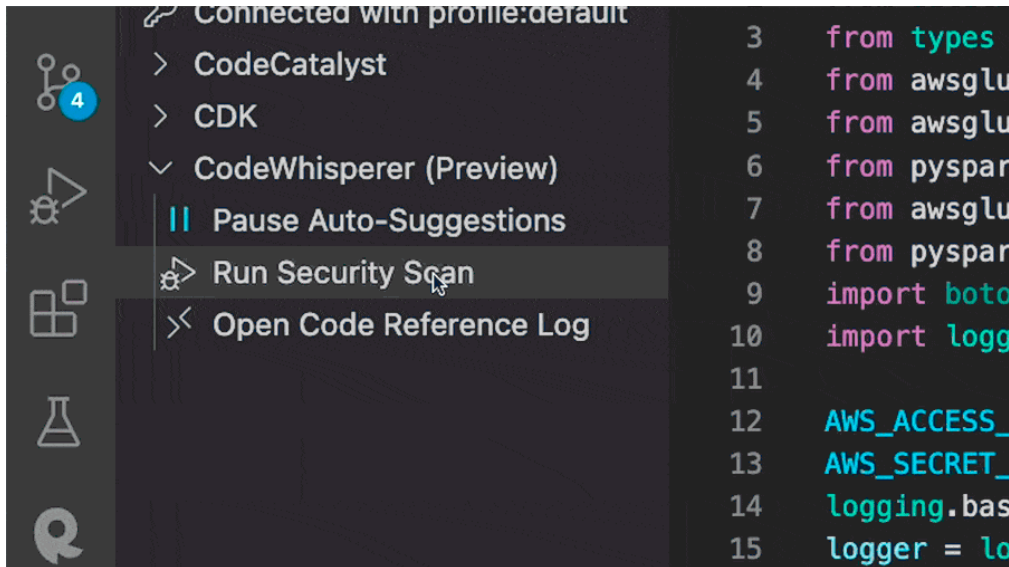
CodeWhisperer

Amazon's CodeWhisperer is a general-purpose service powered by machine learning that helps improve developer productivity by providing code recommendations based on a developer's natural comments and prior code. In other words, it works by taking commands left using comments (depending on the comment style of the user's programming language) in English (natural language) and translating it to working code suggestions.

CodeWhisperer is trained on billions of lines of code drawn from open-source repositories, internal Amazon repositories, API documentation, and forums. It was developed by Amazon and was first announced for preview on June 23, 2022, and was subsequently opened to the public on April 13, 2023.

CodeWhisperer uses multiple contextual clues to drive recommendations, including the cursor location in the source code, code that precedes the cursor, comments, and code in other files in the same projects. CodeWhisperer offers support for up to 15 languages, with Java, Python, JavaScript, TypeScript, and C# having the most support.

The image below shows the CodeWhisperer vulnerability scanning feature in action.



2-5. CodeWhisperer vulnerability scanning feature

CodeWhisperer is available in several IDEs, such as Visual Studio Code and JetBrains. However, language support differs based on a user's IDE. For instance, Python is the only language supported when using CodeWhisperer on Amazon Sagemaker Studio and JupyterLab.

CodeWhisperer currently has an individual tier plan that's free to use for all developers. Some of its features include:

- **Code generation.** CodeWhisperer can generate code and suggestions using code comments. This aims to increase a developer's productivity by 27%.
- **Code referencing.** CodeWhisperer can also scan generated code for repository URLs and licenses when code similar to its training data is generated, to lower the risk of using that code and also boost the developer's confidence in the code.
- **Vulnerability check.** CodeWhisperer can scan through code for hard-to-detect vulnerabilities, such as those in the top ten listed in the Open Web Application Security Project, and it also offers suggestions on how to fix such vulnerabilities.

Remote Pair Programming

Traditionally, pair programming happens with two people physically present in the same place working on one computer. The AI-powered pair programming tools we've looked at above free us from having to partner with another human, instead collaborating with a machine.

But what if you prefer to collaborate with real humans, even if they can't be in the same room? There are some excellent options for remote pair programming that don't require driver and navigator to be in the same location. Some of the impetus towards remote pair programming was provided by Covid-19, which forced many developers to work from home.

There has now been widespread adoption of remote pair programming. Several tools allow for remote pair programming from anywhere, as long as there's a stable internet connection. Examples include:

- [Replit](#)
- [Visual Studio Live Share](#)
- [CodeSandbox Live](#)
- [CodePen Pro](#)

While these tools aren't necessarily powered by AI, we thought they were worth mentioning here while discussing new advances in pair programming and opportunities for learning and developing one's software engineering skills.

The Pros and Cons of Using AI in Programming

We've covered some of the new AI tools that are now available to enhance the learning and day-to-day work of programmers and make their life a little easier. But there are other things about AI-based programming that we should consider, at least in passing, such as their pros and cons.

The Pros of AI in Programming

While there are mixed views about the use of artificial intelligence in

programming, AI has the potential to revolutionize the industry in various ways. Its benefits include:

- **Task automation.** With the introduction of AI programming tools that can generate code and offer autocomplete suggestions, it's possible to automate tedious tasks, allowing developers to focus on more complex, higher-value tasks.
- **Improved efficiency.** AI has the potential to enhance processes and workflows, thereby creating time to focus on more pressing tasks and business logic. This has the potential to lead to improved efficiency and productivity.
- **Debugging.** AI tools that are capable of working with codebase context—such as Cody—can make it easier to debug blocks of code, search out errors in a codebase, and suggest possible solutions.
- **Learning.** Learning to program is no easy task, and AI tools offer powerful new ways for both beginners and experienced programmers to learn new skills without having to rely on the availability of other humans to help or random searches of the Web to find information.

The Cons of Using AI in Programming

Despite the great tools available and the features they offer, it's also important to consider some of the challenges and downsides of using AI in programming. Some of them include:

- **Security vulnerabilities and malicious use.** AI-powered tools might introduce new security vulnerabilities and be susceptible to malicious use. Hackers could exploit AI-generated code to infiltrate systems or deploy attacks, making it crucial to thoroughly test and secure AI-driven software.
- **Legal use of AI-generated code.** AI-generated code may be subject to intellectual property laws, just like human-created code. It's essential to determine the ownership of the code and any associated copyrights, patents,

or licenses before deciding to use such code.

- **Unintended biases and discrimination.** A lot of the models powering these tools have been trained on publicly available data/code that's biased. This can result in the generation of code that exhibits discriminatory bias. Inexperienced programmers or those just starting out may be led down unhelpful paths without even realizing it.

Ethical Considerations around Using AI to Learn Programming

AI offers seemingly limitless opportunities for developers to learn programming and develop their skills. But there are also risks that come with doing so, such as:

- **Data privacy and security.** AI-powered systems always rely on data to work, and this requires them to collect, store, and process data to provide personalized learning experiences. Users' data must be properly protected and only used for learning and not for other purposes such as commercialization, surveillance, or profiling.
- **Bias and fairness.** AI models are trained on publicly available data, and as such they often exhibit or amplify human bias and prejudice present in the data on which they're trained. In the context of learning, this may lead to unequal educational opportunities or even reinforce stereotypes. It's important to regularly evaluate and mitigate bias in AI models to ensure fair and unbiased learning experiences for all learners.
- **Autonomy and human oversight.** While AI can be a valuable learning companion, it shouldn't replace human teachers or mentors entirely. Human oversight is necessary to provide emotional support, and moral guidance, and to address complex questions that AI may struggle to answer. Maintaining a balance between AI assistance and human interaction is essential for fostering a holistic learning and working environment.

Wrapping Up

Since generative AI is still in its early days, we should expect it to be applied in

more and more innovative ways in future. The potential for new and even better tools for helping in the learning of software engineering, personal development, and learning in general are seemingly limitless.

In this tutorial, we've looked at some of the new, AI-based tools that can enhance the work of developers and make the process of learning to code and working as a programmer easier. We've also considered some of the ethical concerns that arise from the applications of AI in programming and learning to program.

In the next tutorial, we'll look in detail at the benefits of AI-powered tools, considering several areas of programming and how the tools we've covered can be used to tackle them. This will involve practical examples of how to use these tools to boost productivity.

A Practical Guide to AI- powered Pair Programming

Chapter

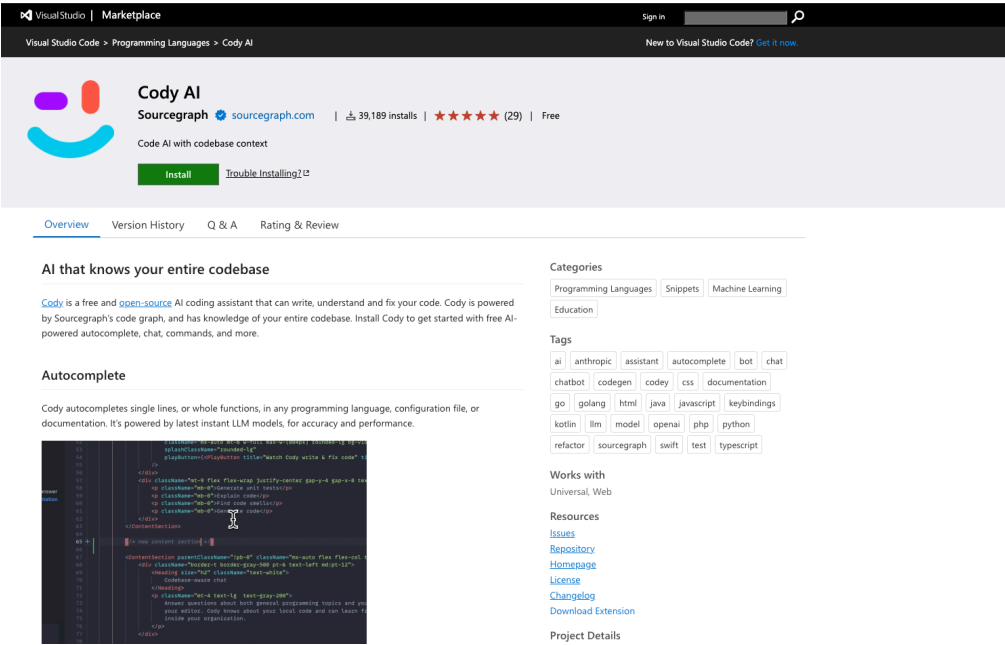
3

So far, we’ve looked at how AI tools can augment the work of software engineers. In this final tutorial, we’ll explore some practical examples of how we can integrate these tools into our day-to-day workflow.

Integrating AI-powered Tools

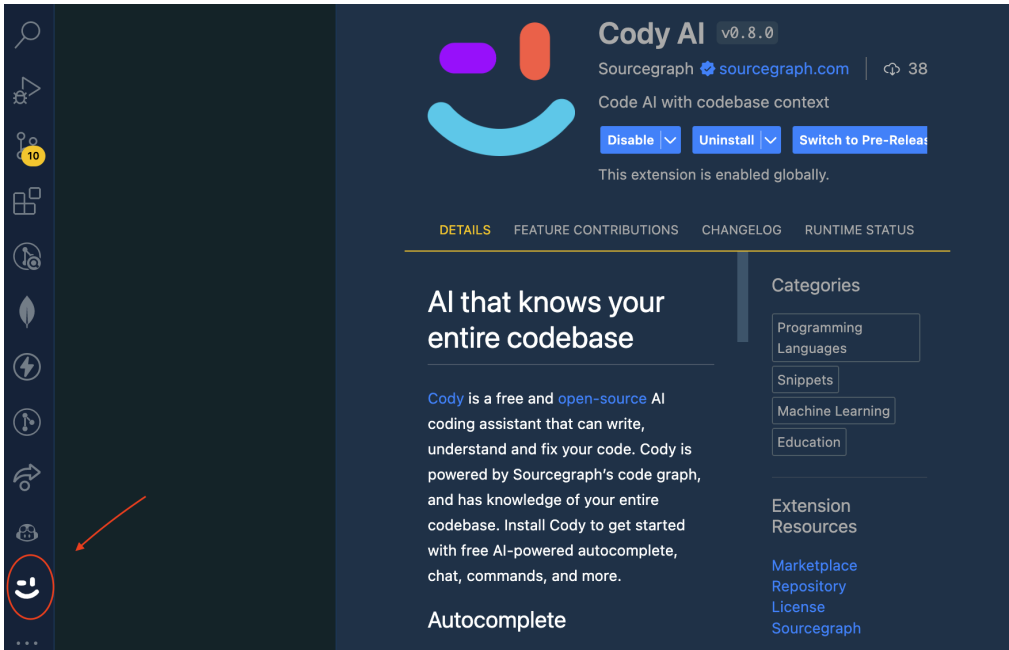
While some of the tools we’ve covered require an active paid subscription, other tools like Codeium and Cody are free. Once we have an account (or an active subscription) with these tools, the next step is to install the relevant extensions for our IDE/text editor. Most tools provide extensions for popular editors like VS Code or Sublime Text.

Using Cody as an example, if we’re using VS Code, the first step is to install the Cody extension for that editor. The image below shows the Cody VS Code extension home page.



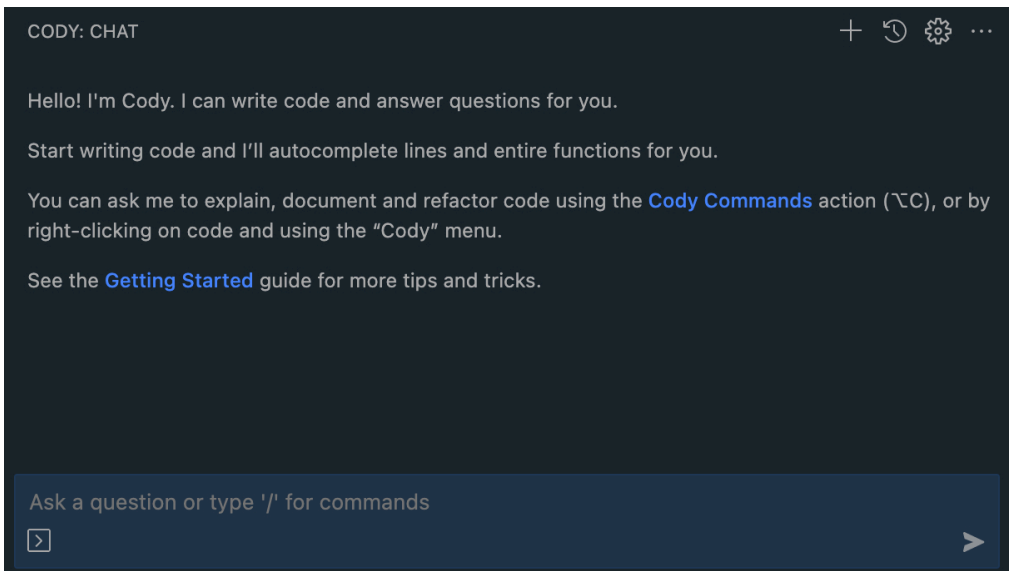
3-1. Cody VS Code extension home page

Once installation is complete, there should be a Cody icon at the side of VS Code.



3-2. Cody VS Code extension

We can click on that icon to open the Cody chat interface.



3-3. Cody Welcome message

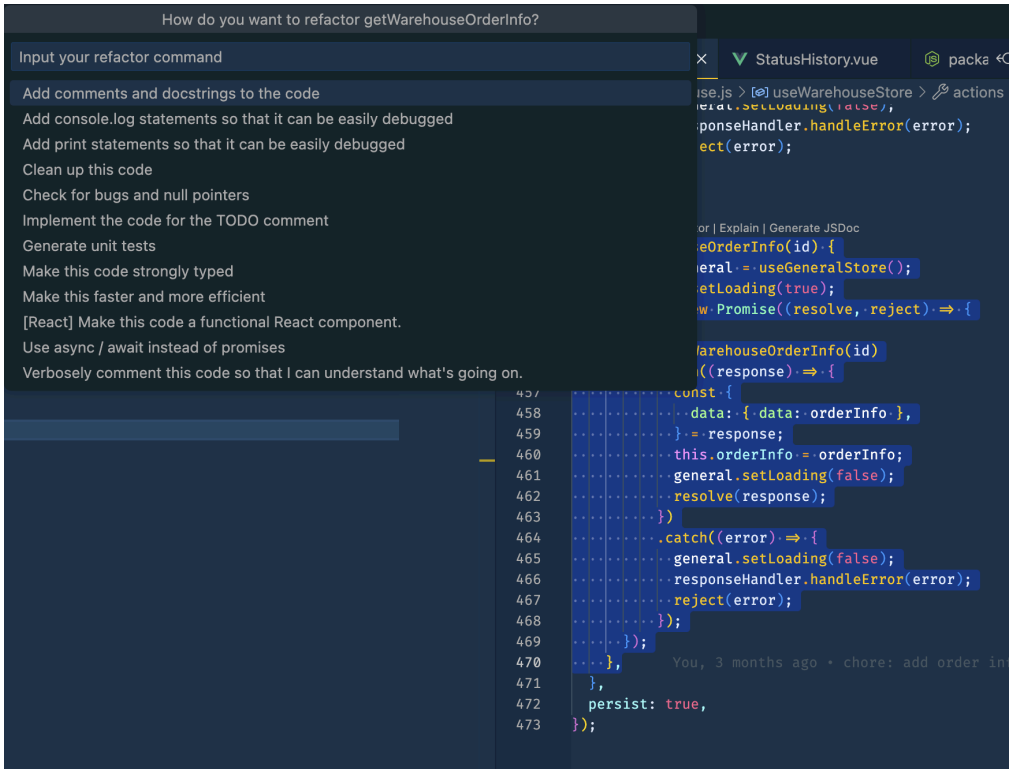
We can then start making use of Cody's features.

Now that we've successfully set up our first AI-powered tool, let's look at some of the ways we can integrate it into our workflow to achieve the following:

- task automation
- debugging
- using it as a digital tutor
- language translation
- documentation

Task Automation

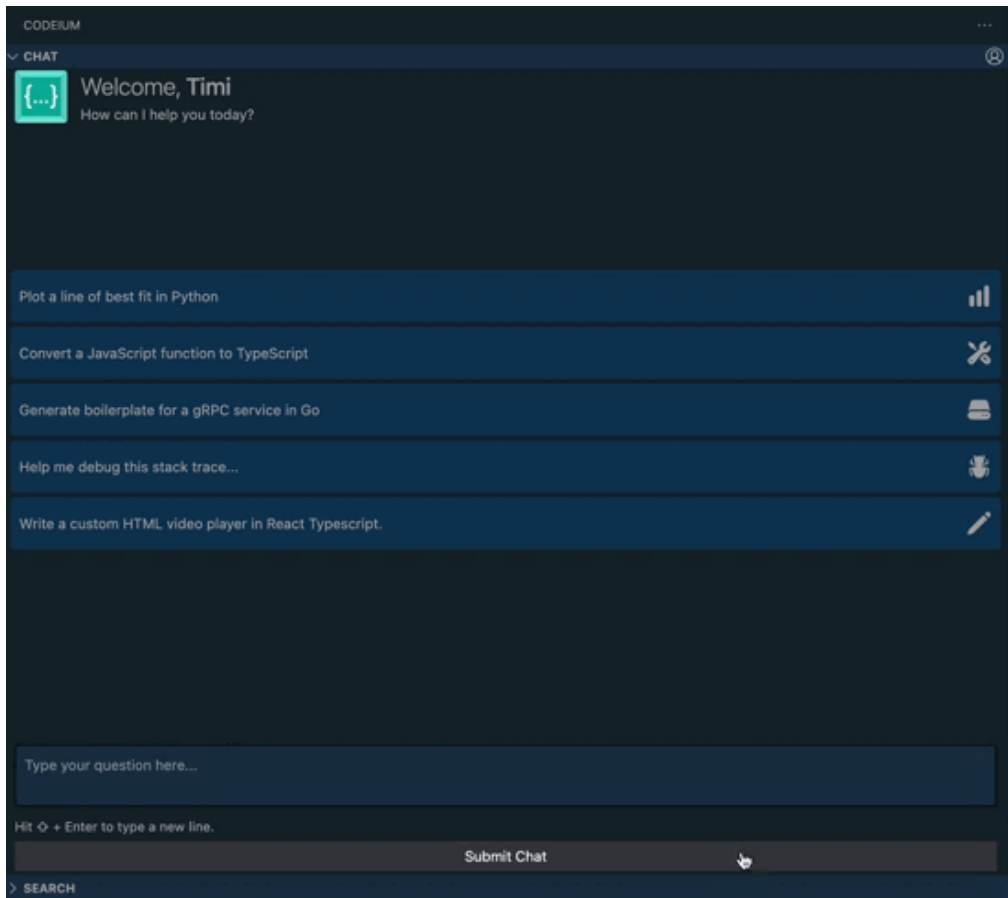
One of the ways engineers can make use of AI-powered tools is with task automation. Many mundane and repetitive tasks can be automated with the help of AI coding assistants.



3-4. AI automation tasks list

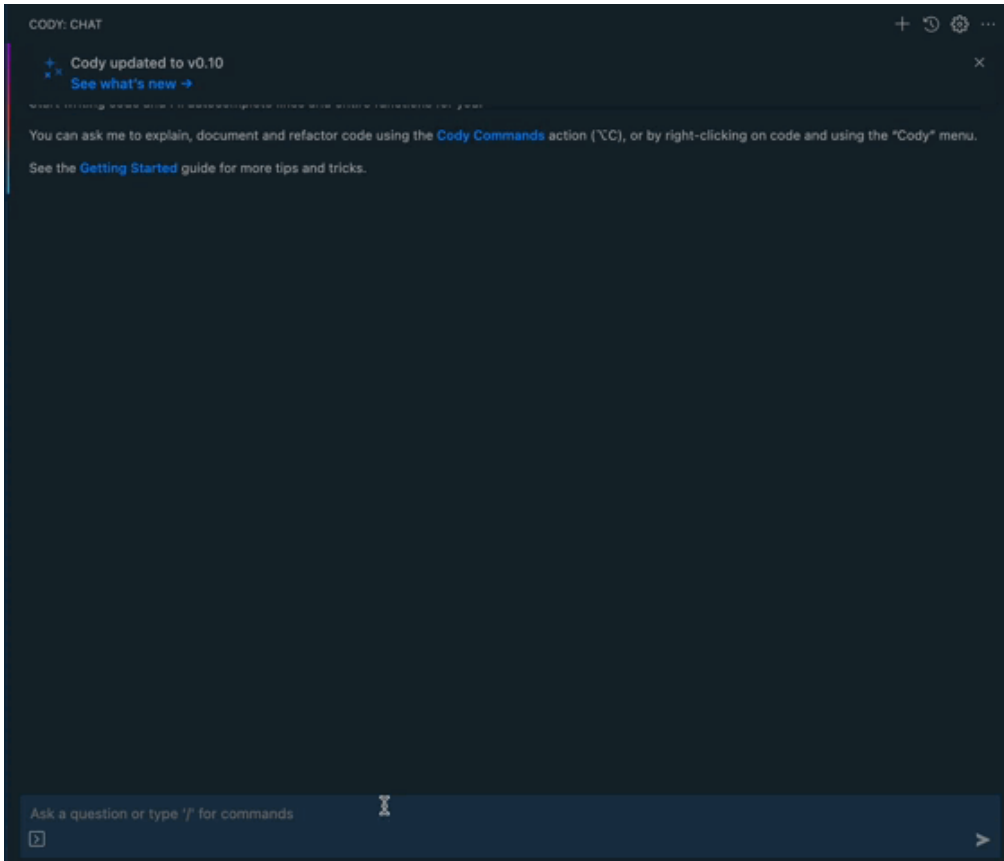
For example, we can ask our AI assistant to **generate boilerplate code**—for common tasks like creating a basic REST API with routes, controllers, models, and so on. This saves time initially setting up a project skeleton.

Using Codeium, here's an example of how we can generate boilerplate code.



3-5. Codeium generating boilerplate code

We can also use Cody to generate boilerplate code.



3-6. Cody generating boilerplate code

Here, we can see that, using an AI coding assistant like Codeium or Cody, we're able to quickly create a simple Express server that's enough to get started with the basic structure and routes defined. To do this, we have to click on the Codeium icon, which should be among the list of extensions on the left side of our editor (or right, depending on where we've placed the icons). Doing this opens up the Codeium chat interface that allows us to communicate with the Codeium server. This is a lot quicker than manually writing all the boilerplate code from scratch.

Here's what Codeium gives us:

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(port, () => {
  console.log(`Server listening on port ${port}`);
});
```

Using Cody, here's what we get:

```
const express = require('express');

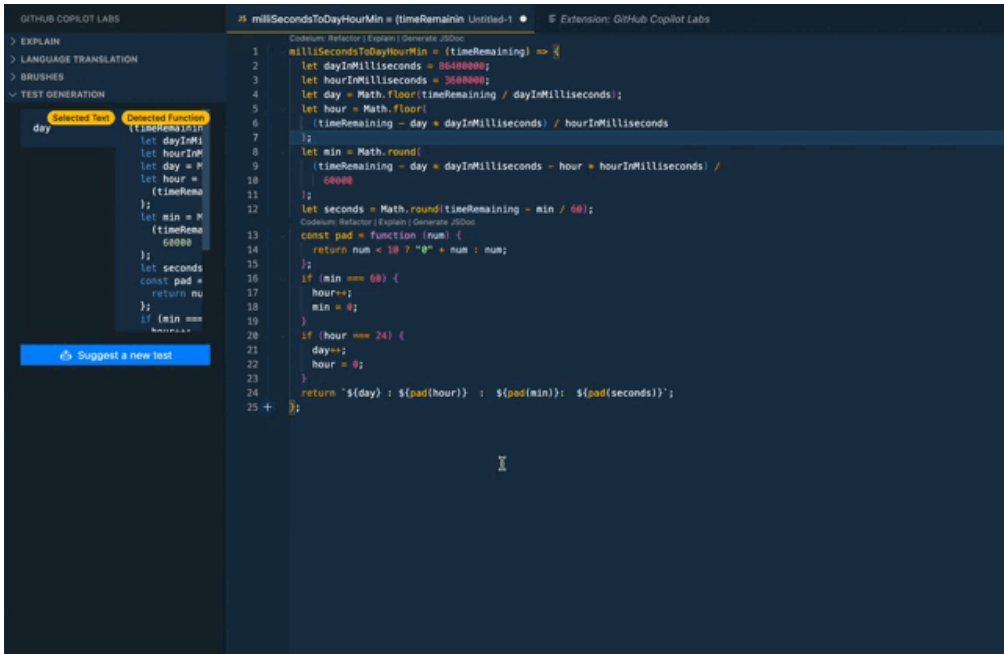
const app = express();

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

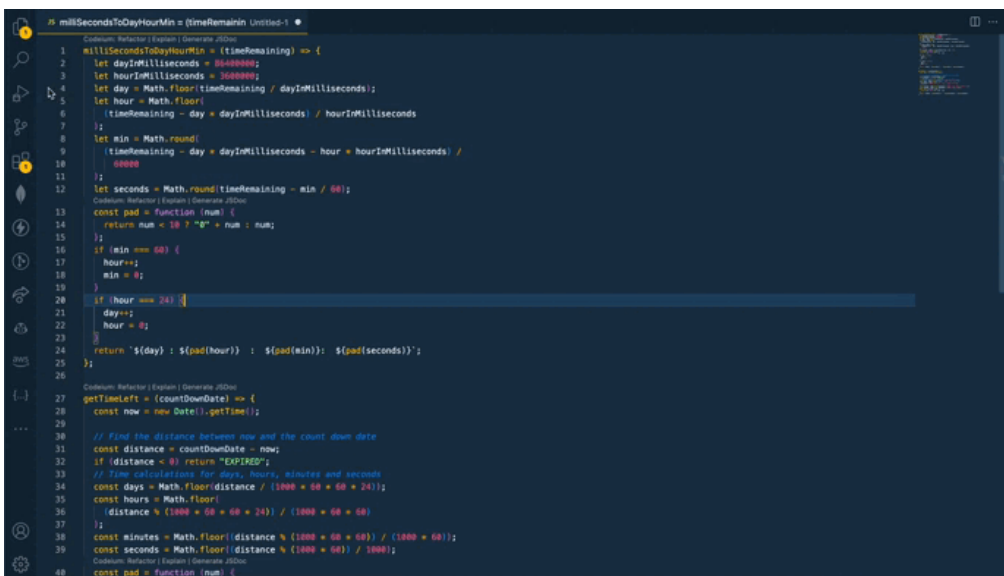
Now let's ask our AI assistant to **write some tests**. AI tools can also help us generate test cases as a starting point for testing our code. This ensures code is tested from the beginning, which is important for quality and avoiding bugs. This can help engineers who are still learning how to write tests and also help introduce test-driven development practices into a codebase without tests.

Here's an example of how we can do this using GitHub Copilot Labs.



3-7. Copilot writing tests

We can also generate tests using Codeium.



3-8. Codeium unit test

In this example, we see the Copilot Labs test feature and Codeium in action. For it to work, we need to provide a function for it to test against:

```
function milliSecondsToDayHourMin = (timeRemaining) => {
  let dayInMilliseconds = 86400000;
  let hourInMilliseconds = 3600000;
  let day = Math.floor(timeRemaining / dayInMilliseconds);
  let hour = Math.floor(
    (timeRemaining - day * dayInMilliseconds) / hourInMilliseconds
  );
  let min = Math.round(
    (timeRemaining - day * dayInMilliseconds - hour * hourInMilliseconds) /
    60000
  );
  let seconds = Math.round(timeRemaining - min / 60);
  const pad = function (num) {
    return num < 10 ? "0" + num : num;
  };
  if (min === 60) {
    hour++;
    min = 0;
  }
  if (hour === 24) {
    day++;
    hour = 0;
  }
  return `${day} : ${pad(hour)} : ${pad(min)} : ${pad(seconds)}`;
};
```

After doing this, it's able to generate test cases to cover different scenarios, such as valid, invalid, and edge cases. The test case generated can act as a good starting point for developers to implement proper testing.

Here's what Copilot gives us:

```
describe('test Untitled_1', function() {
  it('test Untitled-1.milliSecondsToDayHourMin', function(done) {
    assert.deepEqual(Untitled_1.milliSecondsToDayHourMin(0), '0 : 00 : 00: 00');
    assert.deepEqual(Untitled_1.milliSecondsToDayHourMin(60000), '0 : 00 : 01:
    ↳ 01');
    assert.deepEqual(Untitled_1.milliSecondsToDayHourMin(3600000), '0 : 01 : 00:
```



```

    ↪ 60');
    assert.deepEqual(Untitled_1.millisecondsToDayHourMin(86400000), '1 : 00 : 00:
    ↪ 60');
    assert.deepEqual(Untitled_1.millisecondsToDayHourMin(172800000), '2 : 00 :
    ↪00: 60');
    assert.deepEqual(Untitled_1.millisecondsToDayHourMin(259200000), '3 : 00 :
    ↪00: 60');
    assert.deepEqual(Untitled_1.millisecondsToDayHourMin(604800000), '7 : 00 :
    ↪00: 60');
    assert.deepEqual(Untitled_1.millisecondsToDayHourMin(604800000), '7 : 00 :
    ↪00: 60');
    assert.deepEqual(Untitled_1.millisecondsToDayHourMin(86400000 + 3600000 +
    ↪60000), '1 : 01 : 01: 01');
    assert.deepEqual(Untitled_1.millisecondsToDayHourMin(86400000 + 3600000 +
    ↪60000 + 1000), '1 : 01 : 01: 01');
    assert.deepEqual(Untitled_1.millisecondsToDayHourMin(86400000 + 3600000 +
    ↪60000 + 1000), '1 : 01 : 01: 01');
    assert.deepEqual(Untitled_1.millisecondsToDayHourMin(86400000 + 3600000 +
    ↪60000 + 1000), '1 : 01 : 01: 01');
    assert.deepEqual(Untitled_1.millisecondsToDayHourMin(86400000 + 3600000 +
    ↪60000 + 1000), '1 : 01 : 01: 01');
    done();
  })
})

```

And here's what Codeium gives us:

```

test("millisecondsToDayHourMin - should return 0 : 00 : 00 : 00 for 0 milliseconds",
  ↪ () => {
    expect(millisecondsToDayHourMin(0)).toBe("0 : 00 : 00 : 00");
  });

test("millisecondsToDayHourMin - should return 1 : 00 : 00 : 00 for 86400000
  ↪milliseconds", () => {
    expect(millisecondsToDayHourMin(86400000)).toBe("1 : 00 : 00 : 00");
  });

test("millisecondsToDayHourMin - should return 1 : 01 : 00 : 00 for 90000000
  ↪milliseconds", () => {
    expect(millisecondsToDayHourMin(90000000)).toBe("1 : 01 : 00 : 00");
  });

```

```
test("millisecondsToDayHourMin - should return 1 : 01 : 01 : 00 for 90060000  
↳milliseconds", () => {  
  expect(millisecondsToDayHourMin(90060000)).toBe("1 : 01 : 01 : 00");  
});  
  
test("millisecondsToDayHourMin - should return 1 : 01 : 01 : 01 for 90061000  
↳milliseconds", () => {  
  expect(millisecondsToDayHourMin(90061000)).toBe("1 : 01 : 01 : 01");  
});
```

Debugging

AI-powered tools are also useful for debugging. Debugging is one of the most time-consuming tasks any engineer deals with on a daily basis, regardless of their experience level. Integrating AI tools into the debugging workflow reduces the time spent trying to understand what's going wrong.

Here's an example of how we can debug using Codeium:

```
function cleanupFounders() {  
  const founders = this.orgInfo.founders;  
  founders.map((document) => {  
    delete document.organisation;  
    delete document.createdAt;  
    delete document.updatedAt;  
    delete document.__v;  
  });  
  this.orgInfo.founders = founders;  
}
```

CHAT



Welcome, Timi

How can I help you today?

```
why is this code not working? cleanupFounders() {  
  const founders = this.orgInfo.founders;  
  foundeers.map((document) => {  
    delete document.organisation;  
    delete document.createdAt;  
    delete document.updatedAt;  
    delete document.__v;  
  });  
  this.orgInfo.founders = founders;  
},
```

Wed, Aug 9, 2023, 1:21 pm

TO

The code snippet you provided seems to have a typo. In the `cleanupFounders()` function, you are calling the `map()` method on the `foundeers` array, which should be `founders` instead.

Here's the corrected code:

Copy code Insert code

```
cleanupFounders() {  
  const founders = this.orgInfo.founders;  
  founders.map((document) => {  
    delete document.organisation;  
    delete document.createdAt;  
    delete document.updatedAt;  
    delete document.__v;  
  });  
  this.orgInfo.founders = founders;  
}
```

With this correction, the `cleanupFounders()` function should work as intended, removing the specified properties (`organisation`, `createdAt`, `updatedAt`, `__v`) from each object in the `founders` array.



Wed, Aug 9, 2023, 1:21 pm

Clear conversation

Type your question here...

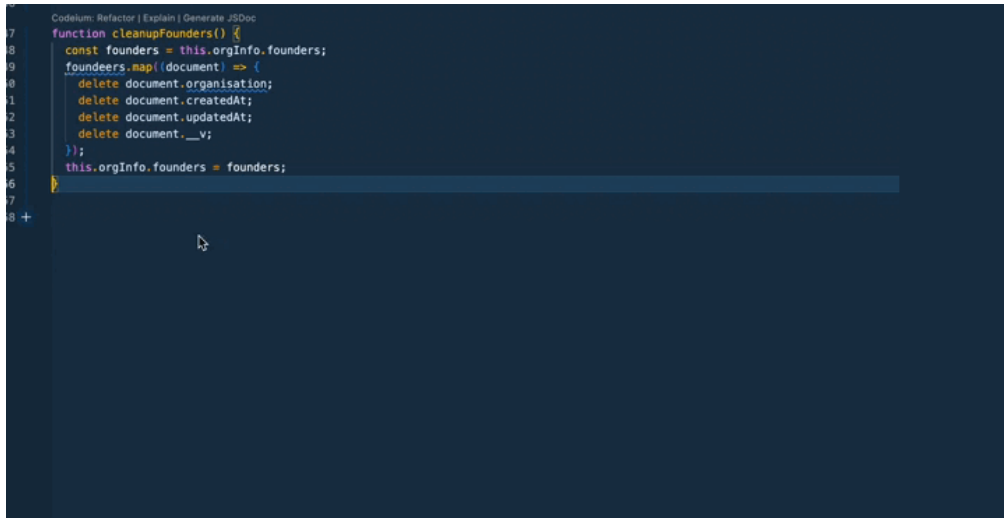
Hit + Enter to type a new line.

Submit Chat

> SEARCH

3-9. Codeium Debugging

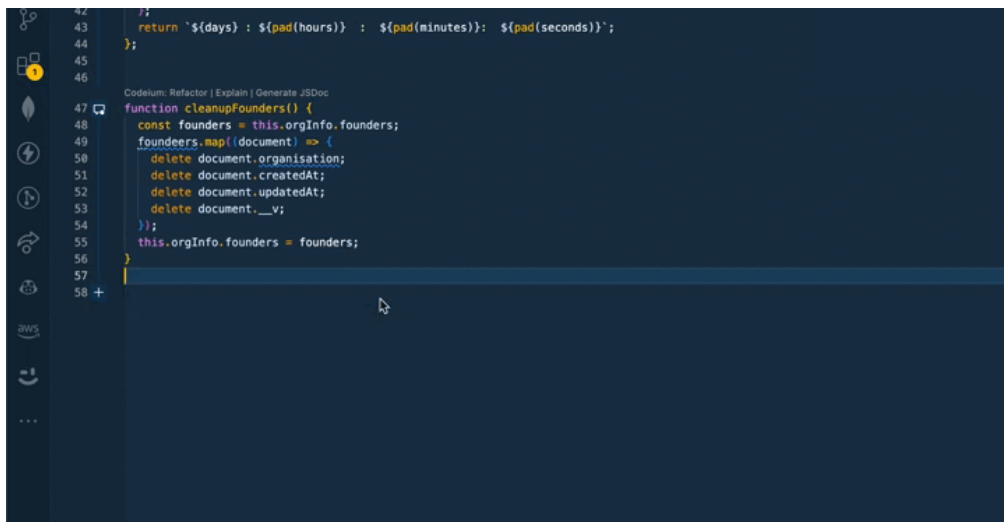
We can also use Cody to debug code.



```
7 function cleanupFounders() {
8   const founders = this.orgInfo.founders;
9   foundeers.map((document) => {
10     delete document.organisation;
11     delete document.createdAt;
12     delete document.updatedAt;
13     delete document.__v;
14   });
15   this.orgInfo.founders = founders;
16 }
17
18 +
```

3-10. Cody Debugging

And here's another example of how we can do this using GitHub Copilot Labs.



```
42 //
43 return `${days} : ${pad(hours)} : ${pad(minutes)}: ${pad(seconds)}`;
44 };
45
46
47 Codium: Refactor | Explain | Generate JSDoc
48 function cleanupFounders() {
49   const founders = this.orgInfo.founders;
50   foundeers.map((document) => {
51     delete document.organisation;
52     delete document.createdAt;
53     delete document.updatedAt;
54     delete document.__v;
55   });
56   this.orgInfo.founders = founders;
57 }
58 +
```

3-11. Copilot Debugging

We can see above that Codeium points out an issue in the code by first understanding the `founders` variable declaration and finding the typo in the

code. It then produces a working version of the code (without the typo) while also explaining the true intention of the function.

Here's the Codeium corrected code:

```
function cleanupFounders() {
  const founders = this.orgInfo.founders;
  founders.map((document) => {
    delete document.organisation;
    delete document.createdAt;
    delete document.updatedAt;
    delete document.__v;
  });
  this.orgInfo.founders = founders;
}
```

Using Cody, we're also able to debug code by first providing a function for it to debug. It then produces a working version of the code while also explaining the true intention of the function. It also goes a step further by suggesting extra ideas to improve the code.

When using the debug feature in Copilot Labs, we're required to provide a function for it to debug. When we do this, we can see that it automatically fixes our code and gives us a working version of the code:

```
function cleanupFounders() {
  const founders = this.orgInfo.founders;
  this.orgInfo.founders = founders.map((document) => {
    delete document.organisation;
    delete document.createdAt;
    delete document.updatedAt;
    delete document.__v;
    return document;
  });
}
```

With these tools, debugging is easier, faster, and more efficient. However, while these tools can help us out with all these tasks, we still need to go over the solutions in order to verify the quality of the code. This way, we get to spend time

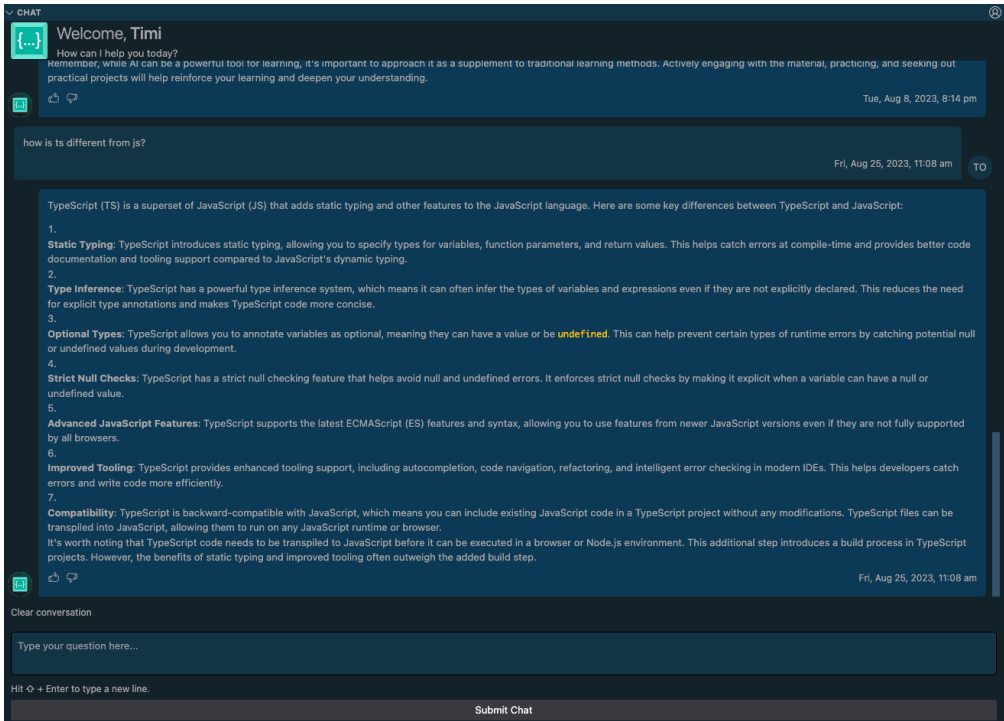
learning rather than debugging, searching the Internet, and getting frustrated over bugs in our code.

Using AI as a Digital Tutor

Thanks to advancements in artificial intelligence, chatbots have evolved into remarkably powerful tools. Trained on vast amounts of data from sources such as blogs, Wikipedia, public GitHub repositories, and [Stack Overflow](#), they have knowledge of a wide array of subjects available on the Internet (depending on the chatbot platform). This proficiency makes chatbots invaluable for tasks beyond mere text generation or summarization. They're adept at assisting developers in seeking knowledge and support.

For instance, imagine a JavaScript engineer who's exploring TypeScript and encountering a question about the differences between the two. In such cases, turning to a chatbot for these explanations often proves faster and more efficient than a traditional Google search.

The image below shows Codeium explaining the differences between TypeScript and JavaScript.



3-12. Codeium explaining the differences between TS and JS

We can see that Codeium has put together a detailed list of the differences between TypeScript and JavaScript, and that it has done so in under a minute. Here's how it explains the difference:

TypeScript (TS) is a superset of JavaScript (JS) that adds static typing and other features to the JavaScript language. Here are some key differences between TypeScript and JavaScript:

- 1 Static Typing: TypeScript introduces static typing, allowing you to specify types for variables, function parameters, and return values. This helps catch errors at compile-time and provides better code documentation and

tooling support compared to JavaScript's dynamic typing.

2 **Type Interface:** TypeScript has a powerful type inference system, which means it can often infer the types of variables and expressions even if they are not explicitly declared. This reduces the need for explicit type annotations and makes TypeScript code more concise.

3 **Optional Types:** TypeScript allows you to annotate variables as optional, meaning they can have a value or be undefined. This can help prevent certain types of runtime errors by catching potential null or undefined values during development.

4 **Strict Null Checks:** TypeScript has a strict null checking feature that helps avoid null and undefined errors. It enforces strict null checks by making it explicit when a variable can have a null or undefined value.

5 **Advanced JavaScript Features:** TypeScript supports the latest ECMAScript (ES) features and syntax, allowing you to use features from newer JavaScript versions even if they are not fully supported by all browsers.

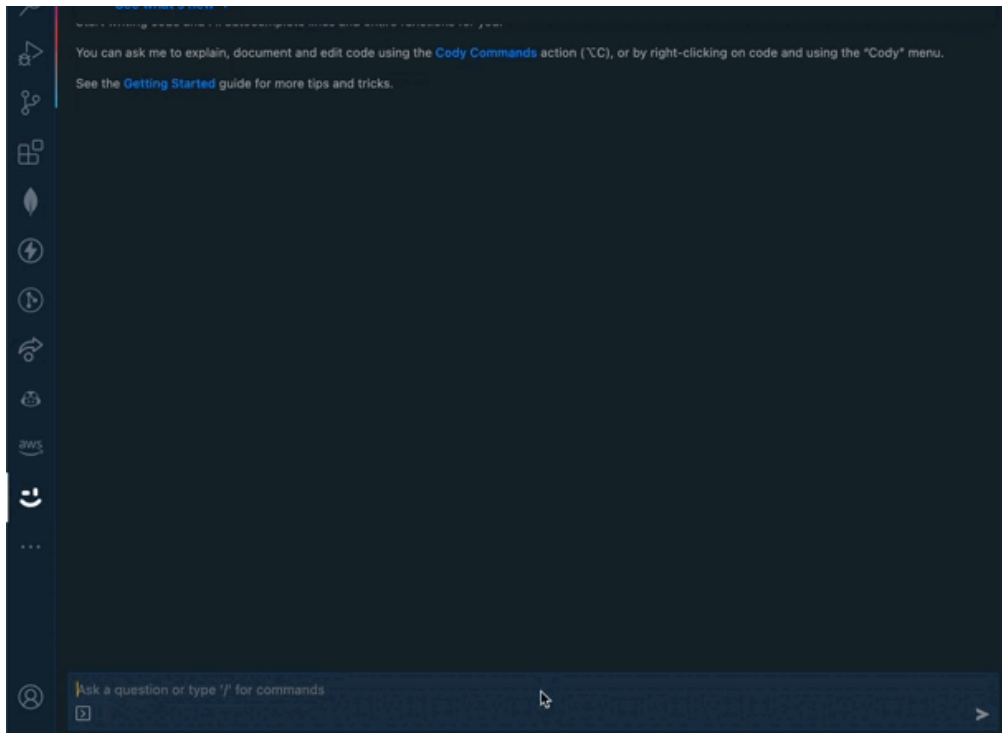
6 **Improved Tooling:** TypeScript provides enhanced tooling support, including autocompletion, code navigation, refactoring, and intelligent error checking IDEs. This helps

developers catch errors and write code more efficiently.

- 7 Compatibility: TypeScript is backward-compatible with JavaScript, which means you can include existing JavaScript code in a TypeScript project without any modifications. TypeScript files can be transpiled into JavaScript, allowing them to run on any JavaScript runtime or browser.

It's worth noting that TypeScript code needs to be transpiled to JavaScript before it can be executed in a browser or Node.js environment. This additional step introduces a build process in TypeScript projects. However, the benefits of static typing and improved tooling often outweigh the added build step.

We can also use the chat feature of Cody to get explanations for technical terms and concepts. The image below shows Cody explaining the benefits of TDD.



3-13. Cody explaining the benefits of TDD

Here are some of the key benefits of Test-Driven Development (TDD):

Improved code quality and design — TDD encourages writing small, modular code that is easy to test. This tends to result in better overall code quality and design.

Increased test coverage - With TDD, you are writing the tests upfront, which helps ensure thorough test coverage.

Faster feedback - By running tests frequently as you code, you get rapid feedback on whether changes broke existing functionality. This allows catching bugs early.

Confidence in refactoring - Existing tests give you confidence to refactor code without breaking existing behavior.

Documentation - Well-written unit tests effectively document

how a piece of code should work.

Reduced debugging time - With good test coverage, you tend to spend less time debugging since tests help identify issues quickly.

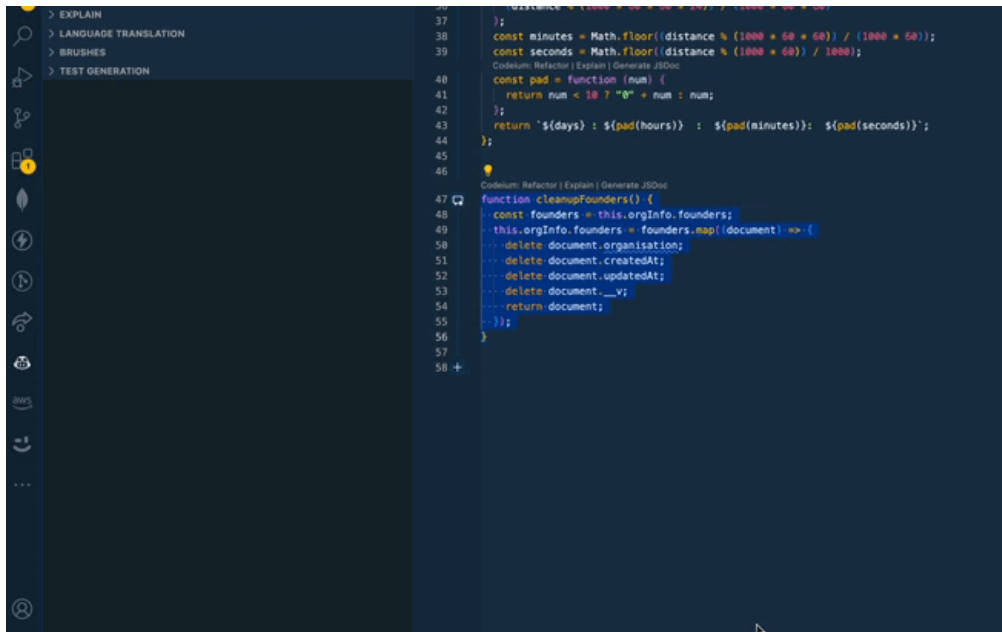
In summary, TDD promotes writing clean, well-tested code through rapid feedback cycles of writing tests, writing code, and refactoring. This results in robust, maintainable code with confidence that changes don't break existing functionality.

In this example, we can see Cody explaining the benefits of test-driven development. It also suggests a few follow-up questions at the bottom of its answers in case the user is interested in knowing more about the topic.

This helps the engineer quickly understand the key differences without having to spend time searching through various blogs and documentation pages. These tools are able to provide contextual and tailored responses very quickly, thanks to their training on vast amounts of online data related to programming topics.

In addition to being able to explain technical terms and concepts, AI-powered tools are also capable of explaining blocks of code by breaking them down into smaller bits before explaining what each line and variable in the code does.

The image below shows Copilot explaining a block of code.



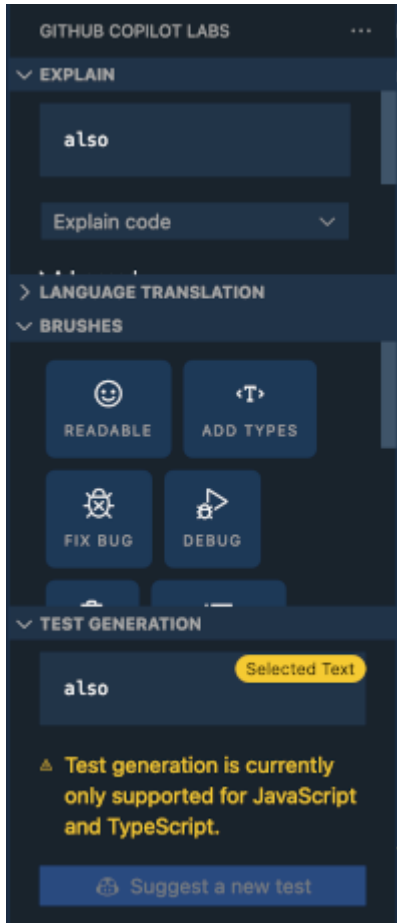
3-14. Copilot explaining code

Here, we can see that Copilot makes an attempt to explain what this code does by breaking it down into smaller bits and explaining what each line and variable in the code does. This can be very useful for engineers who are still learning a new language or framework. In this particular example, we can see that Copilot doesn't really help us understand this function, and that's partly because it's a block of code taken from an existing codebase.

There are other AI-powered tools that have a chat interface built into them. These tools are able to help us with tasks like code translation, code generation, and code completion. Having the chat interface built into these tools makes it easier to use them, as we can easily interact with them and get the results we need without having to leave our IDE. This also makes it easier to integrate these tools into our workflow.

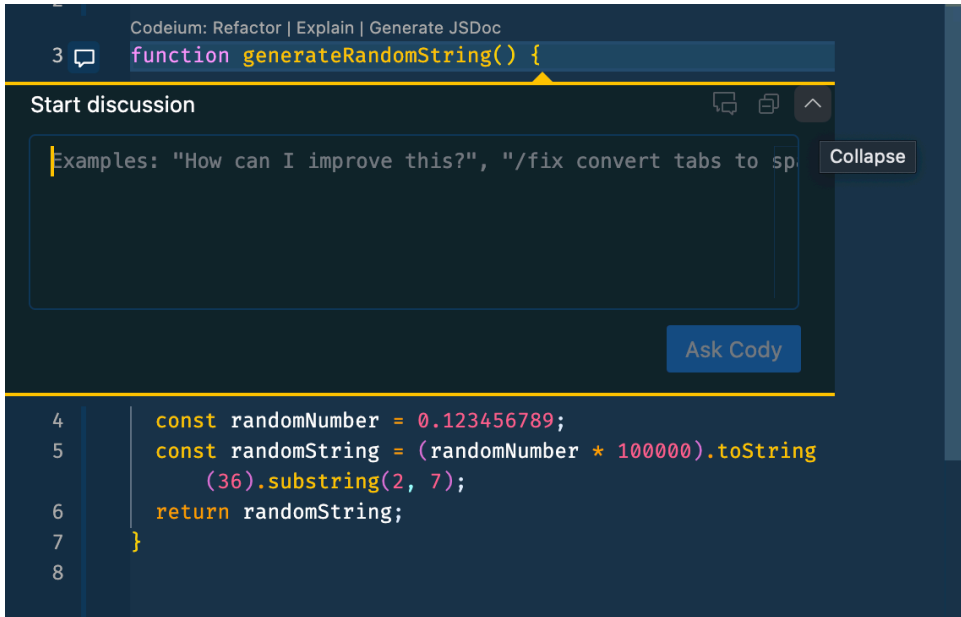
Examples of these tools include two we met in the previous tutorial:

- [GitHub Copilot Labs](#) (which requires active GitHub Copilot subscription). The image below shows the Copilot Labs interface in VS Code.



3-15. GitHub Copilot Labs

- Cody, by Sourcegraph. The image below shows the Cody chat interface.



3-16. Cody by Sourcegraph

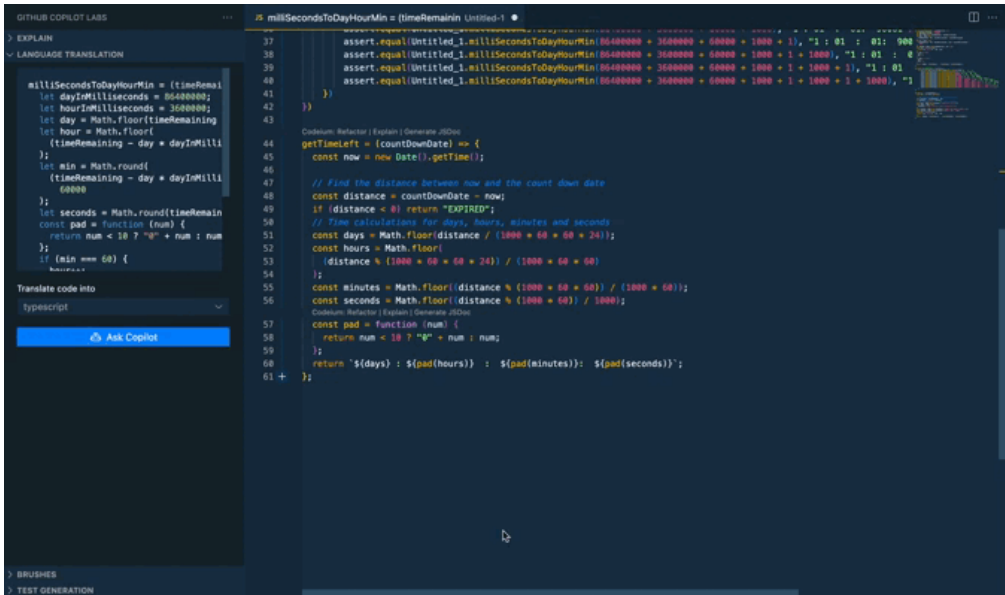
Language Translation

Another unique way AI-powered tools can be useful to engineers is with language translation. The need for language translation arises in software development for various reasons, such as:

- working with a codebase written in another programming language (such as a JavaScript developer working on a legacy Python project)
- integrating code or APIs from libraries, frameworks, or services written in other languages
- a beginner learning a new programming language who finds examples and documentation written in other languages

With AI-powered tools, we can easily translate code snippets, documentation, or even entire files between programming languages. This helps us learn new languages faster by being able to understand code examples and documentation written in languages we're not familiar with.

The image below shows Copilot translating JavaScript code to Go.



3-17. Copilot translation example

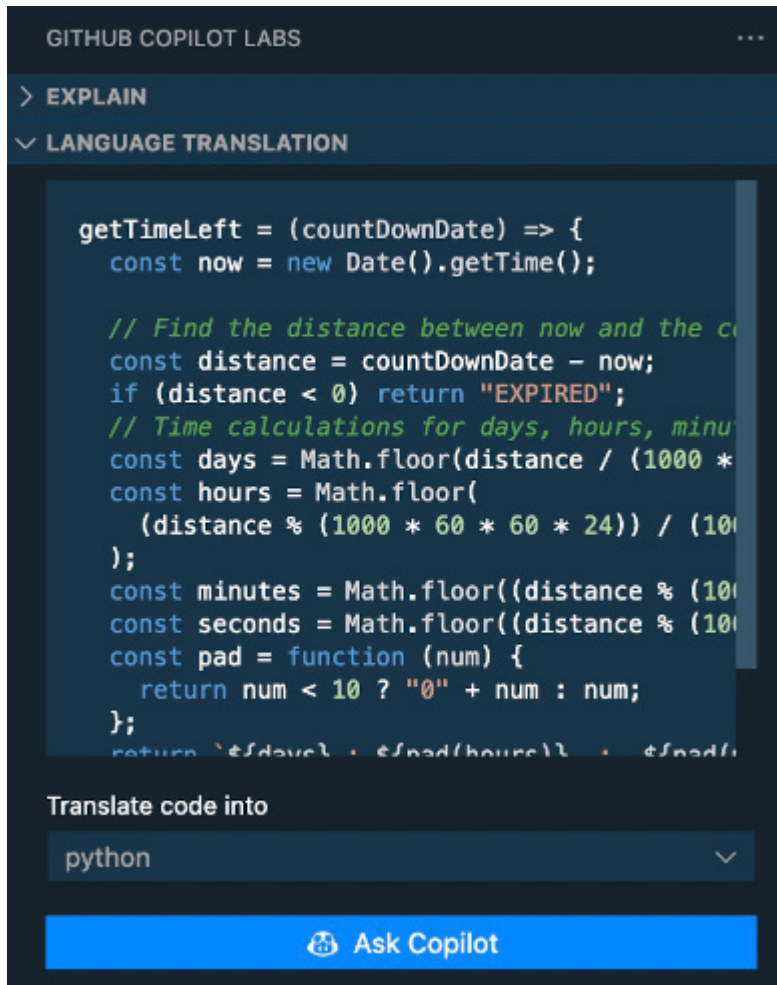
In this example, we have a JavaScript function called `getTimeLeft` that we want to translate to Go:

```
getTimeLeft = (countDownDate) => {
  const now = new Date().getTime();

  // Find the distance between now and the count down date
  const distance = countDownDate - now;
  if (distance < 0) return "EXPIRED";
  // Time calculations for days, hours, minutes and seconds
  const days = Math.floor(distance / (1000 * 60 * 60 * 24));
  const hours = Math.floor(
    (distance % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60)
  );
  const minutes = Math.floor((distance % (1000 * 60 * 60)) / (1000 * 60));
  const seconds = Math.floor((distance % (1000 * 60)) / 1000);
  const pad = function (num) {
    return num < 10 ? "0" + num : num;
  };
};
```

```
return `${days} : ${pad(hours)} : ${pad(minutes)}: ${pad(seconds)}`;  
};
```

Using Copilot, we need to highlight the block of code we want to translate. After that, we need to click on the language translation section in the Copilot extension.



3-18. Copilot language translation feature

After doing this, we should see the selected code in the window, along with a variety of options to translate between languages like JavaScript, Python, Java,

C#, PHP, and more, which may vary depending on the tool. By selecting **Go**, this is what we get:

```
func getTimeLeft(countDownDate time.Time) string {
    now := time.Now()
    // Find the distance between now and the count down date
    distance := countDownDate.Sub(now)
    if distance < 0 {
        return "EXPIRED"
    }
    // Time calculations for days, hours, minutes and seconds
    days := distance / (time.Duration(1000) * time.Duration(60) * time.Duration(60)
↳* time.Duration(24))
    hours := (distance % (time.Duration(1000) * time.Duration(60) * time.Duration(60)
↳* time.Duration(24))) / (time.Duration(1000) * time.Duration(60) *
↳time.Duration(60))
    minutes := (distance % (time.Duration(1000) * time.Duration(60)
↳* time.Duration(60))) / (time.Duration(1000) * time.Duration(60))
    seconds := (distance % (time.Duration(1000) * time.Duration(60)))
↳/ time.Duration(1000)
    pad := func(num int) string {
        if num < 10 {
            return "0" + strconv.Itoa(num)
        }
        return strconv.Itoa(num)
    }
    return strconv.Itoa(days) + " : " + pad(int(hours)) + " : " + pad(int(minutes))
↳+ " : " + pad(int(seconds))
}
```

We can also make use of Codeium to translate the same code to Go, Python, and Java.

```

20 if (hour == 24) {
21     days++;
22     hour = 0;
23 }
24 return `${day} : ${pad(hour)} : ${pad(min)} : ${pad(seconds)}`;
25 }
26
27 // Codium: Refactor | Explain | Generate | JSDoc
28 getTimeLeft = (countDownDate) => {
29     const now = new Date().getTime();
30     // Find the distance between now and the count down date
31     const distance = countDownDate - now;
32     if (distance < 0) return "EXPIRED";
33     // Time calculations for days, hours, minutes and seconds
34     const days = Math.floor(distance / (1000 * 60 * 60 * 24));
35     const hours = Math.floor(
36         (distance % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60)
37     );
38     const minutes = Math.floor((distance % (1000 * 60 * 60)) / (1000 * 60));
39     const seconds = Math.floor((distance % (1000 * 60)) / 1000);
40     // Codium: Refactor | Explain | Generate | JSDoc
41     const pad = function (num) {
42         return num < 10 ? "0" + num : num;
43     };
44     return `${days} : ${pad(hours)} : ${pad(minutes)} : ${pad(seconds)}`;
45 }
46
47 // Codium: Refactor | Explain | Generate | JSDoc
48 function cleanupFounders() {
49     const founders = this.orgInfo.founders;
50     this.orgInfo.founders = founders.map((document) => {
51         delete document.organisation;
52         delete document.createdAt;
53         delete document.updatedAt;
54         delete document.__v;
55         return document;
56     });
57 }

```

3-19. Codeium translating code

Here's an example of the Go output:

```

package main

import (
    "fmt"
    "time"
)

func getTimeLeft(countDownDate int64) string {
    now := time.Now().UnixNano() / int64(time.Millisecond)

    // Find the distance between now and the count down date
    distance := countDownDate - now
    if distance < 0 {
        return "EXPIRED"
    }

    // Time calculations for days, hours, minutes and seconds
    days := distance / (1000 * 60 * 60 * 24)
    hours := (distance % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60)
    minutes := (distance % (1000 * 60 * 60)) / (1000 * 60)
    seconds := (distance % (1000 * 60)) / 1000

```

```

    pad := func(num int64) string {
        if num < 10 {
            return "0" + fmt.Sprintf("%d", num)
        }
        return fmt.Sprintf("%d", num)
    }

    return fmt.Sprintf("%d : %s : %s : %s", days, pad(hours), pad(minutes),
        ↪pad(seconds))
}

```

It's important to note that these suggestions won't always be perfect and may require some manual tweaking to get the desired result. However, they can serve as a good starting point for engineers who are still learning a new language or framework.

Documentation

Another important aspect of programming is documentation. Well-documented code and projects are easier to understand, maintain, and expand. Good documentation also makes it easy to transfer the thought process of the writer and contributors to whoever picks up such a project.

Despite all of the benefits of documentation, not all projects and codebases are well documented. This can be due to various reasons such as lack of time, lack of documentation skills, or just forgetting about it. With AI-powered tools like Codeium and GitHub Copilot, developers can generate documentation for their code and projects with just a few clicks. This can be useful for engineers who are working on a codebase that has little or no documentation. This way, they can get a good starting point for writing documentation for their codebase.

To try this out, let's take a look at this code snippet:

```

import { gsap } from "gsap/dist/gsap";

export default {

```

```

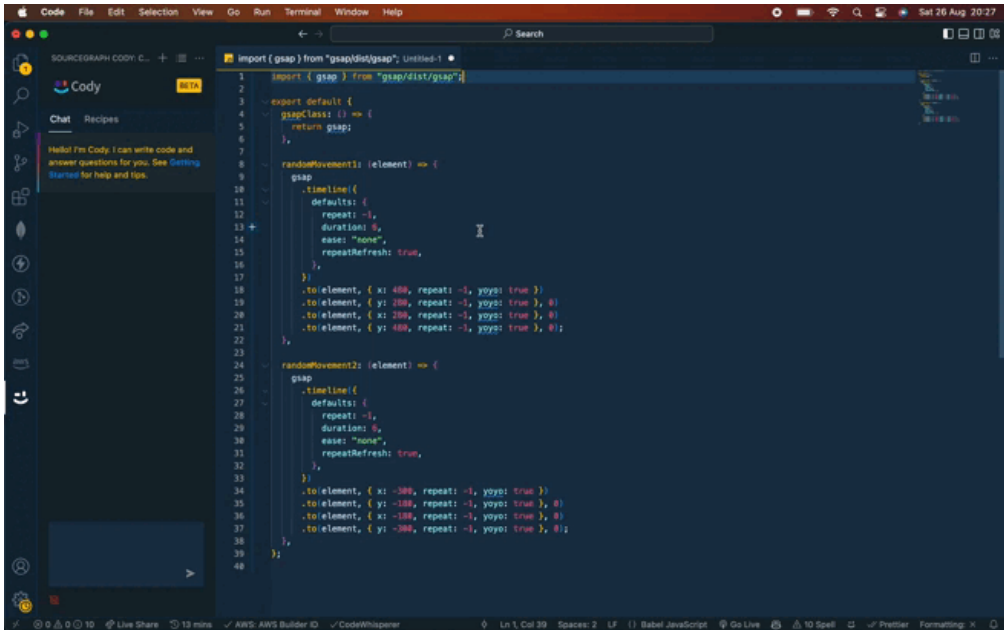
gsapClass: () => {
  return gsap;
},

randomMovement1: (element) => {
  gsap
    .timeline({
      defaults: {
        repeat: -1,
        duration: 6,
        ease: "none",
        repeatRefresh: true,
      },
    })
    .to(element, { x: 480, repeat: -1, yoyo: true })
    .to(element, { y: 280, repeat: -1, yoyo: true }, 0)
    .to(element, { x: 280, repeat: -1, yoyo: true }, 0)
    .to(element, { y: 480, repeat: -1, yoyo: true }, 0);
},

randomMovement2: (element) => {
  gsap
    .timeline({
      defaults: {
        repeat: -1,
        duration: 6,
        ease: "none",
        repeatRefresh: true,
      },
    })
    .to(element, { x: -300, repeat: -1, yoyo: true })
    .to(element, { y: -180, repeat: -1, yoyo: true }, 0)
    .to(element, { x: -180, repeat: -1, yoyo: true }, 0)
    .to(element, { y: -300, repeat: -1, yoyo: true }, 0);
},
};

```

The image below shows Cody writing documentation.



3-20. AI writing documentation

We're able to ask Cody to generate documentation for our GSAP animation code. It's able to understand the code and provide a basic explanation of how the code is being used to animate elements on the page. This serves as a good starting point for writing comprehensive documentation.

Here's the response we get from Cody, writing documentation for our `gsap.js` file:

This file exports an object with methods to create GSAP animations.

It imports gsap from the GSAP library to use its timeline and animation methods.

Methods gsapClass

■ Returns the imported gsap instance.

randomMovement1

- Creates a GSAP timeline with infinite repeating and yoyo.
- Animates the provided element by translating it horizontally and vertically in a random pattern.

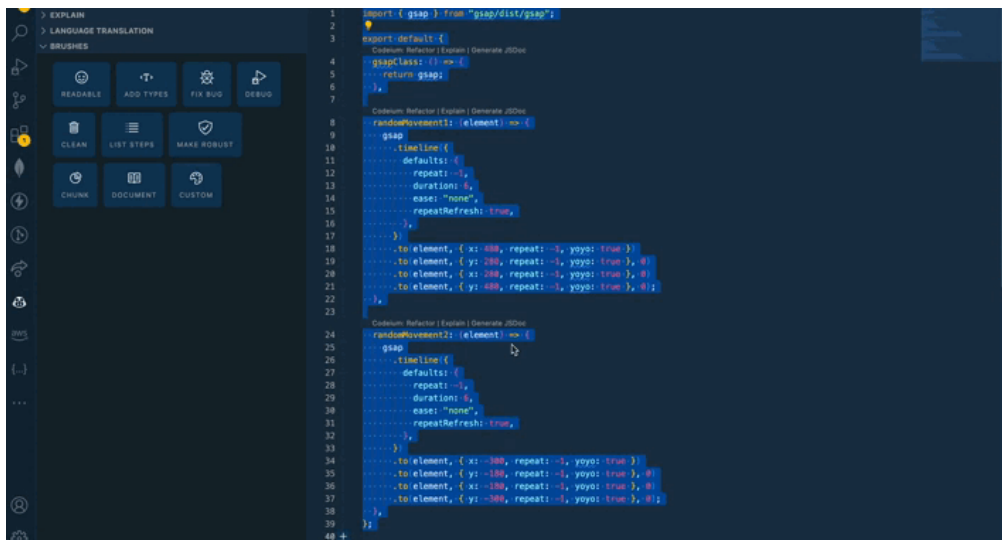
randomMovement2

- Creates a GSAP timeline with infinite repeating and yoyo.
- Animates the provided element by translating it horizontally and vertically in a different random pattern from *randomMovement1*.

The methods *randomMovement1* and *randomMovement2* can be used to animate elements in different random patterns by passing the element as a parameter.

The *gsapClass* method returns the *gsap* instance which can be used to create other custom GSAP animations.

Using Github Copilot, we can also generate documentation for our code. Here's an example of how we can do this.



The screenshot shows a VS Code editor with a dark theme. On the left, there's a sidebar with various icons and a 'BRUSHES' section containing buttons like 'READABLE', 'ADD TYPES', 'FIX BUG', 'DEBUG', 'CLEAN', 'LIST STEPS', 'MAKE ROBUST', 'CHUNK', 'DOCUMENT', and 'CUSTOM'. The main editor area displays two JavaScript functions, `randomMovement1` and `randomMovement2`, with their corresponding JSDoc comments generated by Github Copilot. The code is as follows:

```

1  import { gsap } from "gsap/dist/gsap";
2
3  export default {
4    gsapClass: () => {
5      return gsap;
6    },
7
8    randomMovement1: (element) => {
9      gsap
10       .timeline({
11         defaults: {
12           repeat: -1,
13           duration: 6,
14           ease: "none",
15           repeatRefresh: true,
16         },
17         to: element, { x: 400, repeat: -1, yoyo: true },
18         to: element, { y: 200, repeat: -1, yoyo: true }, 0,
19         to: element, { x: 200, repeat: -1, yoyo: true }, 0,
20         to: element, { y: 400, repeat: -1, yoyo: true }, 0,
21       },
22     ),
23
24    randomMovement2: (element) => {
25      gsap
26       .timeline({
27         defaults: {
28           repeat: -1,
29           duration: 6,
30           ease: "none",
31           repeatRefresh: true,
32         },
33         to: element, { x: 300, repeat: -1, yoyo: true },
34         to: element, { y: 100, repeat: -1, yoyo: true }, 0,
35         to: element, { x: 100, repeat: -1, yoyo: true }, 0,
36         to: element, { y: 300, repeat: -1, yoyo: true }, 0,
37       },
38     ),
39   },
40 }
  
```

3-21. Copilot generating documentation

Copilot is able to understand the code and provide a basic explanation of what the code is doing. This serves as a good starting point for writing comprehensive documentation.

Wrapping Up

With AI-powered tools offering features that help automate routine tasks, debug blocks of code, point out issues, suggest possible solutions, help navigate the codebase, suggest code snippets along with function names and comments, prepare documentation, and translate code into different languages, developers can save a lot of time and thereby focus on the more important (and fun) aspects of their work, such as learning and building. All of these benefits offered by AI tools lead to improved efficiency.

Given the rate at which AI-powered tools are being developed, and the rate at which they're being adopted into developer workflows, it's safe to say they're here to stay. There are other new and exciting AI-based products currently on the horizon, such as [Copilot for Pull Requests](#), which aims to help us with better PR descriptions, reviews and merges.

In these tutorials, we've explored some of the benefits of integrating AI tools into our development workflow through pair programming. We've also looked at practical examples of how tools like GitHub Copilot and Cody can be used in a pair programming setting. Hopefully you now have a good understanding of how AI-powered pair programming works and how to get started with it.

As we've already stressed, AI isn't here to take our jobs. Rather, it's here to augment our toolkit and make us better engineers. AI-powered pair programming is the future, and we should embrace it to become better software engineers. Try experimenting with these tools and learn more about how they can enhance your pair programming experience.