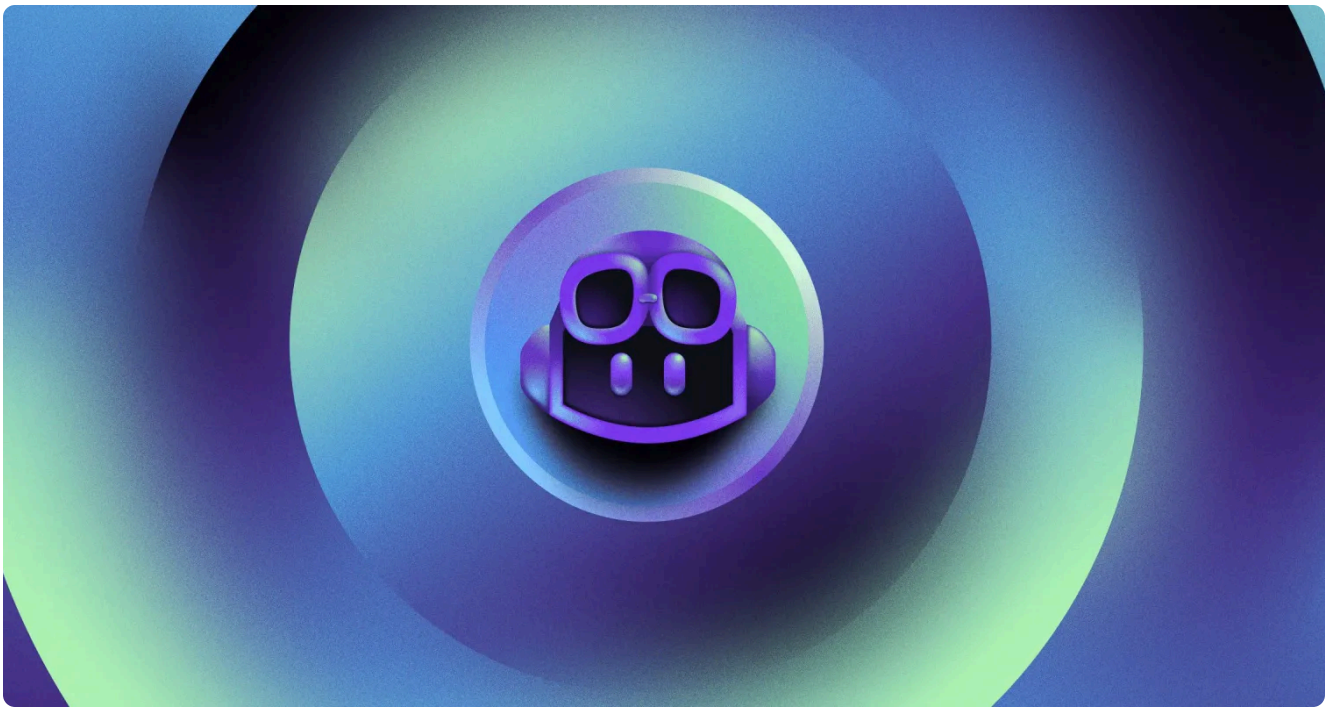
/ Blog

Home / AI & ML / GitHub Copilot

How GitHub Copilot is getting better at understanding your code

With a new Fill-in-the-Middle paradigm, GitHub engineers improved the way GitHub Copilot contextualizes your code. By continuing to develop and test advanced retrieval algorithms, they're working on making our AI tool even more advanced.



Johan Rosenkilde · **@johanrosenkilde**

May 17, 2023

Updated February 7, 2024

10 minutes

Share:

programmer's contextual understanding. That's because good communication is key to pair programming, and inferring context is critical to making good communication happen.

To pull back the curtain, we asked GitHub's researchers and engineers about the work they're doing to help GitHub Copilot improve its contextual understanding. Here's what we discovered.

From OpenAI's Codex model to GitHub Copilot

When OpenAI released GPT-3 in June 2020, GitHub knew developers would benefit from a product that leveraged the model specifically for coding. So, we gave input to OpenAI as it built Codex, a descendant of GPT-3 and the LLM that would power GitHub Copilot. The pair programmer launched as a technical preview in June 2021 and became generally available in June 2022 as the world's first at-scale generative AI coding tool.

To ensure that the model has the best information to make the best predictions with speed, GitHub's machine learning (ML) researchers have done a lot of work called prompt engineering (which we'll explain in more detail below) so that the model provides contextually relevant responses with low latency.

Though GitHub's always experimenting with new models as they come out, Codex was the first really powerful generative AI model that was available, said David Slater, a ML engineer at GitHub. "The hands-on experience we gained from iterating on model and prompt improvements was invaluable."

Inside GitHub: Working with the LLMs behind GitHub Copilot

Get the full, inside story from researchers and engineers at GitHub who built the early versions of GitHub Copilot and have improved it since then.

All that experimentation resulted in a pair programmer that, ultimately, frees up a developer's time to focus on more fulfilling work. The tool is often a huge help even for

/ Blog

> I still find myself impressed and even surprised by what GitHub Copilot can do, even after having worked on it for some time now.
>
> – Alice Li, ML researcher at GitHub

Why context matters

Developers use details from pull requests, a folder in a project, open issues, and more to contextualize their code. When it comes to a generative AI coding tool, we need to teach that tool what information to use to do the same.

Transformer LLMs are good at connecting the dots and big-picture thinking. Generative AI coding tools are made possible by large language models (LLMs). These models are sets of algorithms trained on large amounts of code and human language. Today's state-of-the-art LLMs are transformers, which makes them adept at making connections between text in a user's input and the output that the model has already generated. This is why today's generative AI tools are providing responses that are more contextually relevant than previous AI models.

But they need to be told what information is relevant to your code. Right now, transformers that are fast enough to power GitHub Copilot can process about 6,000 characters at a time. While that's been enough to advance and accelerate tasks like code completion and code change summarization, the limited amount of characters means that not all of a developer's code can be used as context.

So, our challenge is to figure out not only what data to feed the model, but also how to best order and enter it to get the best suggestions for the developer.

How GitHub Copilot understands your code

It all comes down to prompts, which are compilations of IDE code and relevant context that's fed to the model. Prompts are generated by algorithms in the background, at any point in your coding. That's why GitHub Copilot will generate coding suggestions whether you're currently writing or just finished a comment, or in the middle of some gnarly code.

- **Here's how a prompt is created**: a series of algorithms first select relevant code snippets or comments from your current file and other sources (which we'll dive into below). These snippets and comments are then prioritized, filtered, and assembled into the final prompt.

GitHub Copilot's contextual understanding has continuously matured over time. The first version was only able to consider the file you were working on in your IDE to be contextually relevant. But we knew context went beyond that. Now, just a year later, we're experimenting with algorithms that will consider your entire codebase to generate customized suggestions.

Let's look at how we got here:

- **Prompt engineering** is the delicate art of creating a prompt so that the model makes the most useful prediction for the user. The prompt tells LLMs, including GitHub Copilot, what data, and in what order, to process in order to contextualize your code. Most of this work takes place in what's called a **prompt library**, which is where our in-house ML experts work with algorithms to extract and prioritize a variety of sources of information about the developer's context, creating the prompt that'll be processed by the GitHub Copilot model.
- **Neighboring tabs** is what we call the technique that allows GitHub Copilot to process all of the files open in a developer's IDE instead of just the single one the developer is working on. By opening all files relevant to their project, developers automatically invoke GitHub Copilot to comb through all of the data and find matching pieces of code

/ Blog

When developing neighboring tabs, the GitHub Next team and in-house ML researchers did A/B tests to figure out the best parameters for identifying matches between code in your IDE and code in your open tabs. They found that **setting a very low bar for when to include a match actually made for the best coding suggestions**.

By including every little bit of context, neighboring tabs helped to relatively increase user acceptance of GitHub Copilot's suggestions by 5%**.

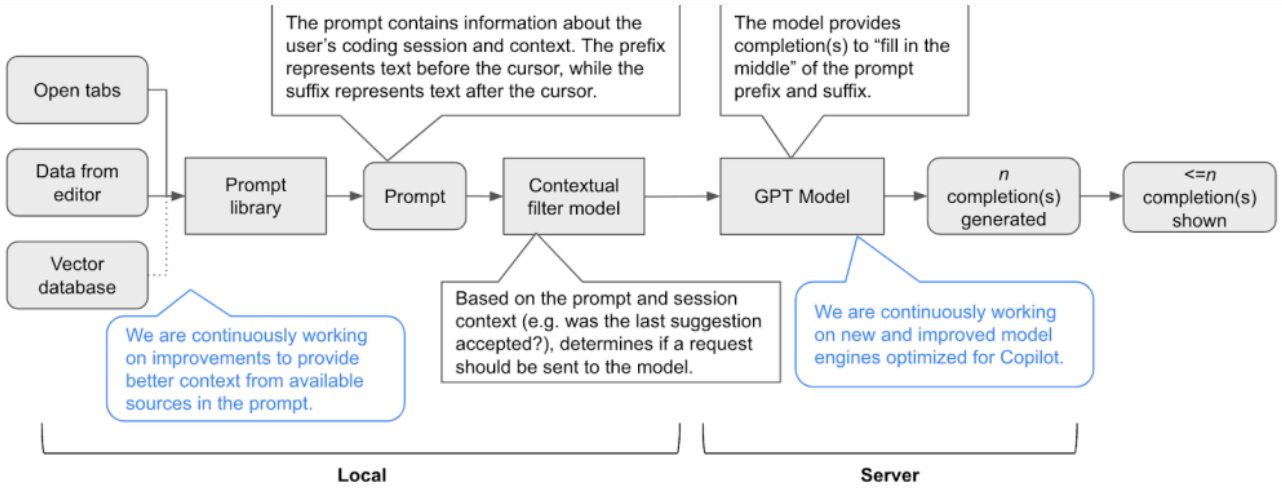
> Even if there was no perfect match—or even a very good one—picking the best match we found and including that as context for the model was better than including nothing at all.
>
> – Albert Ziegler, principal ML engineer at GitHub

- The **Fill-In-the-Middle** (**FIM**) **paradigm** widened the context aperture even more. Prior to FIM, only the code before your cursor would be put into the prompt—ignoring the code after your cursor. (At GitHub, we refer to code before the cursor as the prefix and after the cursor as the suffix.) **With FIM, we can tell the model which part of the prompt is the prefix, and which part is the suffix**.

Even if you're creating something from scratch and have a skeleton of a file, we know that coding isn't linear or sequential. So, while you bounce around your file, FIM helps GitHub Copilot offer better coding suggestions for the part in your file where your cursor is located, or the code that's supposed to come between the prefix and suffix.

Based on A/B testing, **FIM gave a 10% relative boost in performance, meaning developers accepted 10% more of the completions that were shown to them**. And thanks to optimal use of caching, neighboring tabs and FIM work in the background without any added latency.

/ Blog



Simplified system diagram focused on model quality efforts. Made by Alice Li, machine learning researcher at GitHub.

Improving semantic understanding

Today, we're experimenting with **vector databases that could create a customized coding experience for developers working in private repositories or with proprietary code.** Generative AI coding tools use something called embeddings to retrieve information from a vector database.

- **What's a vector database?** It's a database that indexes high-dimensional vectors.
- **What's a high-dimensional vector?** They're mathematical representations of objects, and because these vectors can model objects in a number of dimensions, they can capture complexities of that object. When used properly to represent pieces of code, they may represent both the semantics and even intention of the code—not just the syntax.
- **What's an embedding?** In the context of coding and LLMs, an embedding is the representation of a piece of code as a high-dimensional vector. Because of the "knowledge" the LLM has of both programming and natural language, it's able to capture both the syntax and semantics of the code in the vector.

Here's how they'd all work together:

- Algorithms would create embeddings for all snippets in the repository (potentially billions of them), and keep them stored in the vector database.

> embeddings that are created for your IDE snippets and the embeddings already stored in the vector database. The vector database is what allows algorithms to quickly search for approximate matches (not just exact ones) on the vectors it stores, even if it's storing billions of embedded code snippets.

Developers are familiar with retrieving data with hashcodes, which typically look for exact character by character matches, explained Alireza Goudarzi, senior ML researcher at GitHub. "But embeddings—because they arise from LLMs that were trained on a vast amount of data—develop a sense of semantic closeness between code snippets and natural language prompts."

Read the three sentences below and identify which two are the most semantically similar.

- **Sentence A**: The king moved and captured the pawn.
- **Sentence B**: The king was crowned in Westminster Abbey.
- **Sentence C**: Both white rooks were still in the game.

The answer is sentences A and C because both are about chess. While sentences A and B are syntactically, or structurally similar because both have a king as the subject, they're semantically different because "king" is used in different contexts.

Here's how each of those statements could translate to Python. Note the syntactic similarity between snippets A and B despite their semantic difference, and the semantic similarity between snippets A and C despite their syntactic difference.

Snippet A:

```
if king.location() == pawn.location():
    board.captures_piece(king, pawn)
```

Snippet B:

```
if king.location() == "Westminster Abbey":
    king.crown()
```

/ Blog

```
if len([ r for r in board.pieces("white") if r.type == "rook" ]) == 2:
    return True
```

As mentioned above, we're still experimenting with retrieval algorithms. We're designing the feature with enterprise customers in mind, specifically those who are looking for a customized coding experience with private repositories and would explicitly opt in to use the feature.

Take this with you

Last year, we conducted quantitative research on GitHub Copilot and found that developers code up to 55% faster while using the pair programmer. This means developers feel more productive, complete repetitive tasks more quickly, and can focus more on satisfying work. But our work won't stop there.

The GitHub product and R&D teams, including GitHub Next, have been collaborating with Microsoft Azure AI-Platform to continue bringing improvements to GitHub Copilot's contextual understanding. So much of the work that helps GitHub Copilot contextualize your code happens behind the scenes. While you write and edit your code, GitHub Copilot is responding to your writing and edits in real time by generating prompts–or, in other words, prioritizing and sending relevant information to the model based on your actions in your IDE—to keep giving you the best coding suggestions.

Learn more

- GitHub Copilot X is our envisioned future of AI-powered software development. Discover what's new.
- Learn how the LLMs powering GitHub Copilot are getting better.
- Read our research on how GitHub Copilot is impacting developer productivity.

/ Blog

Tags: AI | AI Insights | generative AI | GitHub Copilot | How GitHub builds GitHub | LLM

Written by

Johan Rosenkilde
@johanrosenkilde

More on AI

Celebrating the GitHub Awards 2024 recipients 🎉

The GitHub Awards celebrates the outstanding contributions and achievements in the developer community by honoring individuals, projects, and organizations for creating an outsized positive impact on the community.

Laura Lindeman & Lee Reilly

Bringing developer choice to Copilot with Anthropic's Claude 3.5 Sonnet, Google's Gemini 1.5 Pro, and OpenAI's o1-preview

At GitHub Universe, we announced Anthropic's Claude 3.5 Sonnet, Google's Gemini 1.5 Pro, and OpenAI's o1-preview and o1-mini are coming to GitHub Copilot—bringing a

/ Blog

Related posts

How developers spend the time they save thanks to AI coding tools

Developers tell us how GitHub Copilot and other AI coding tools are transforming their work and changing how they spend their days.

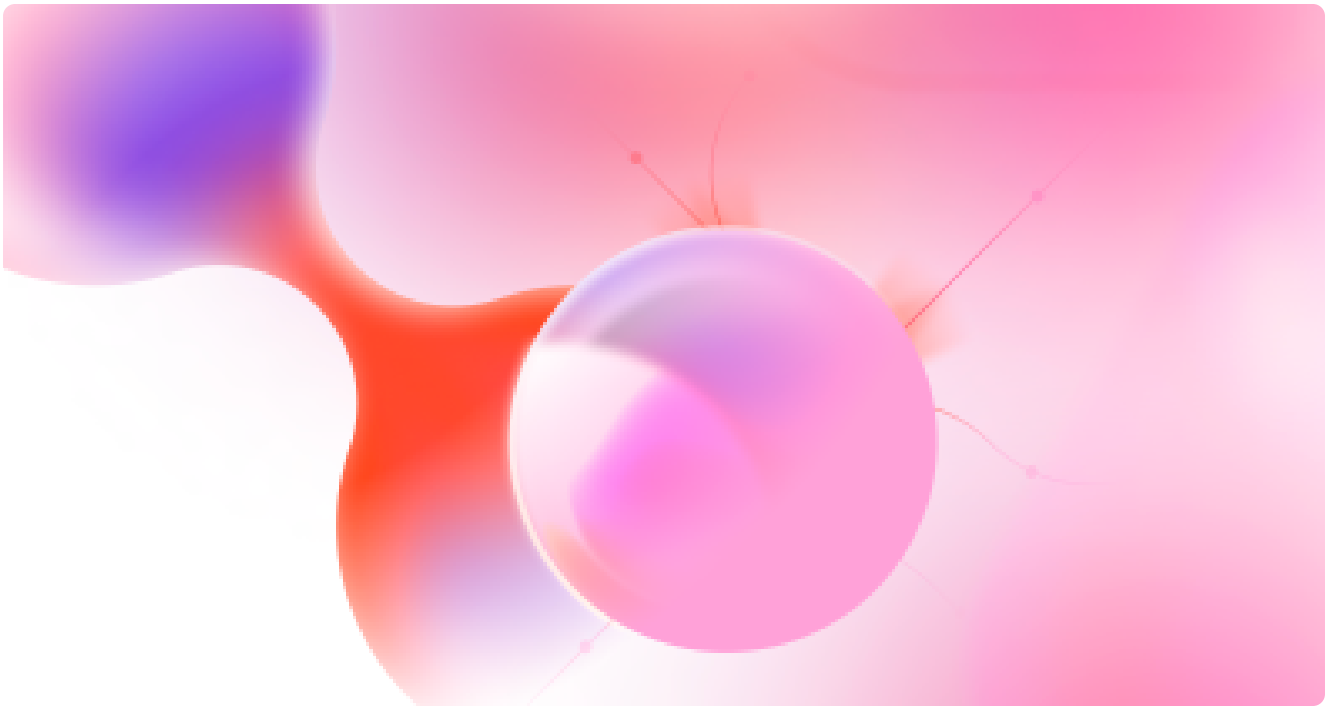
Klint Finley

/ Blog

5 tips and tricks when using GitHub Copilot Workspace

GitHub Next launched the technical preview for GitHub Copilot Workspace in April 2024. Since then, we've been listening to the community, learning, and have some tips to share on how to get the most out of it!

Chris Reddington & Cole Bemis



achieving a groundbreaking historical breakthrough. This is their journey, the technology behind it, and the power of collaboration.

Juan Pablo Flores Cortés

Explore more from GitHub

Docs

Everything you need to master GitHub, all in one place.

Go to Docs

GitHub

Build what's next on GitHub, the place for anyone from anywhere to build anything.

Start building

/ Blog

Customer stories

Meet the companies and engineering teams that build with GitHub.

Learn more

GitHub Universe 2024

Get tickets to the 10th anniversary of our global developer event on AI, DevEx, and security.

Get tickets

We do newsletters, too

Discover tips, technical guides, and best practices in our biweekly newsletter just for devs.

Your email address

Yes please, I'd like GitHub and affiliates to use my information for personalized communications, targeted advertising and campaign effectiveness. See the GitHub Privacy Statement for more details.

/ Blog