

# **How to Use Your Development Data to Make LLMs Code Like You and Your Team**

Tyler Dunn, Co-founder & CEO of Continue



# Continue is on a mission to make building software feel like making music

Continue is a modular, open-source Copilot alternative

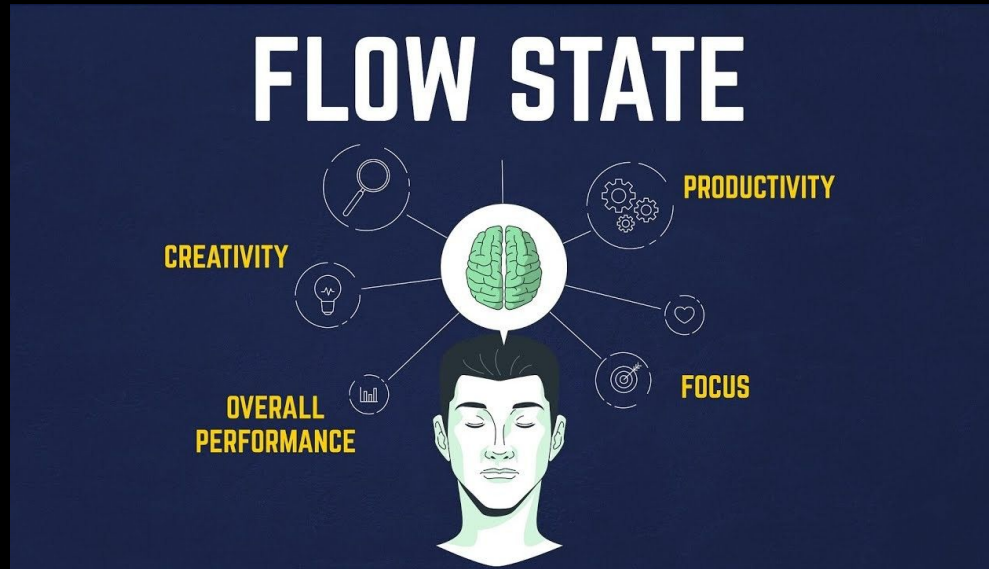
It's built as a reusable set of components that enable developers to create their own copilot

```
43 def from_name(cls, name: str):
44     """Construct from name"""
45     if name in transformer_configs:
46         return cls(**transformer_configs[name])
47     # Fuzzy search
48     config = findconfig_for_config_in(transformer_configs if config in str(name).upper() or config in str(name).lower() else None)
49     assert len(config) == 1, name
50     return cls(**transformer_configs[config])
51
52
53 transformer_configs = {
54     "gpt4o": {
55         "vocab_size": 50257, "vocab_size_embeddings": 50257, "vocab_size_embeddings_embeddings": 50257, "vocab_size_embeddings_embeddings_embeddings": 50257,
56         "num_layers": 32, "num_heads": 128, "num_heads_kv": 128, "num_heads_kv_embeddings": 128, "num_heads_kv_embeddings_embeddings": 128,
57         "num_heads_kv_embeddings_embeddings_embeddings": 128, "num_heads_kv_embeddings_embeddings_embeddings_embeddings": 128,
58         "num_heads_kv_embeddings_embeddings_embeddings_embeddings_embeddings": 128, "num_heads_kv_embeddings_embeddings_embeddings_embeddings_embeddings_embeddings": 128,
59     },
60     "gpt4o_mini": {
61         "vocab_size": 50257, "vocab_size_embeddings": 50257, "vocab_size_embeddings_embeddings": 50257, "vocab_size_embeddings_embeddings_embeddings": 50257,
62         "num_layers": 32, "num_heads": 128, "num_heads_kv": 128, "num_heads_kv_embeddings": 128, "num_heads_kv_embeddings_embeddings": 128,
63         "num_heads_kv_embeddings_embeddings_embeddings": 128, "num_heads_kv_embeddings_embeddings_embeddings_embeddings": 128,
64         "num_heads_kv_embeddings_embeddings_embeddings_embeddings_embeddings": 128, "num_heads_kv_embeddings_embeddings_embeddings_embeddings_embeddings_embeddings": 128,
65     },
66 }
67
68 class KCache(metaclass=ABCMeta):
69     """Abstract class for KCache"""
70     def __init__(self, num_batches, max_seq_length, num_heads, head_dim, dtype=torch.float16):
71         super().__init__()
72         self.num_batches = num_batches
73         self.max_seq_length = max_seq_length
74         self.num_heads = num_heads
75         self.head_dim = head_dim
76         self.dtype = dtype
77         self.register_buffer('k_cache', torch.zeros((self.num_batches, self.max_seq_length, self.num_heads, self.head_dim), dtype=dtype))
78         self.register_buffer('v_cache', torch.zeros((self.num_batches, self.max_seq_length, self.num_heads, self.head_dim), dtype=dtype))
79         self.register_buffer('k_val', torch.zeros((self.num_batches, self.max_seq_length, self.num_heads, self.head_dim), dtype=dtype))
80         self.register_buffer('v_val', torch.zeros((self.num_batches, self.max_seq_length, self.num_heads, self.head_dim), dtype=dtype))
81
82     def update(self, input_pos, k_val, v_val):
83         # input_pos must be [0, k_val.shape[2], ..., k_val.shape[2]-1]
84         assert input_pos.shape[0] == k_val.shape[2]
85
86         self.k_cache[:input_pos.shape[0], :input_pos.shape[1], :, :self.head_dim] = k_val
87         self.v_cache[:input_pos.shape[0], :input_pos.shape[1], :, :self.head_dim] = v_val
88
89         self.k_val[:input_pos.shape[0], :input_pos.shape[1], :, :self.head_dim] = k_val
90         self.v_val[:input_pos.shape[0], :input_pos.shape[1], :, :self.head_dim] = v_val
```

**First, why do I want to make LLMs  
code like me and my team?**

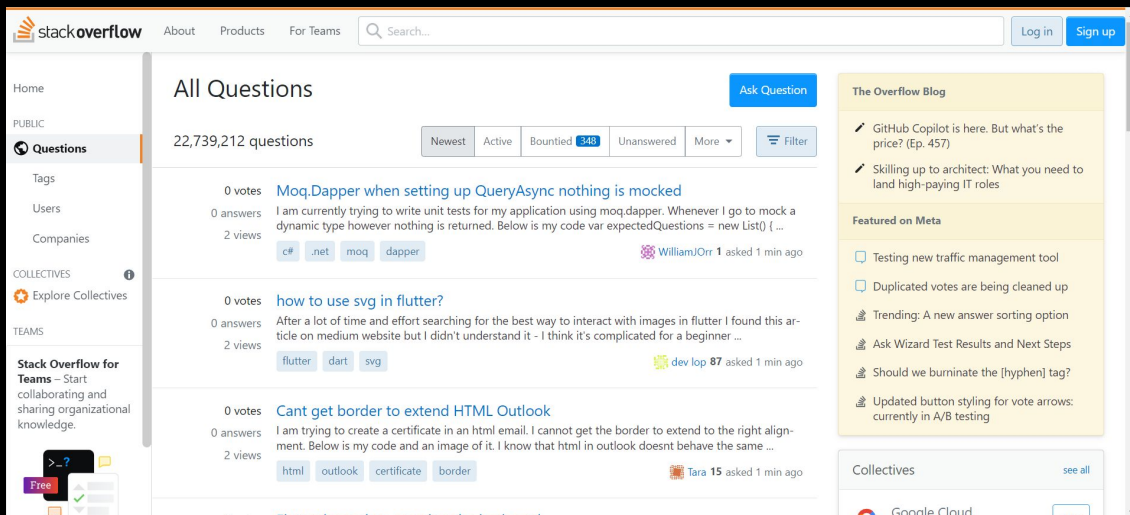


**As developers, we want to experience flow state**



# Getting stuck disrupts our flow state

This is why so many of us are excited about software development copilots



# But bad / wrong suggestions disrupt flow state too

I worry our Copilot is leaving some passengers behind.



Blog post by

 Josh  
Collinsworth

## A real-world example: my simple component

Recently, I set out to build a component to help me generate footnotes on this site. You know; the kind that shows up as a tiny link in some text, and that when clicked, jumps you to the bottom of the page for an accompanying annotation. <sup>1</sup>

Just links doing link things. Good old-fashioned HTML.

But for this dead-simple task, *GitHub Copilot wanted me to add a JavaScript click handler*. Something like this, instead:

```
Svelte
<script>
  const handleClick = (e) => {
    e.preventDefault()
    const target = document.getElementById('#footnote-1')
    target.focus()
  }
</script>

<a href="#" on:click={handleClick}>1</a>
```

I hope any good developer would immediately spot this as categorically bad code.

**Okay, but what is  
development data?**



# Dev data = how you build software

Data on the stuff that happens in between Git commits

Created as a by-product of using LLMs while coding

```
$ cd .continue
$ cd dev_data
$ ls
accept_reject_diff.jsonl      new_session.jsonl
autocomplete.jsonl           select_context_item.jsonl
chat.jsonl                    step_run.jsonl
context_used.jsonl           suggestions.json
feedback.jsonl               tokens_generated.jsonl
model_use.jsonl
$ █
```



# How to use your development data

## Step 1

Collect your  
dev data and  
look at it

## Step 2

Improve the  
compound  
AI system

## Step 3

Improve the  
Large Language  
Models (LLMs)



# How to use your development data

## Step 1

**Collect your  
dev data and  
look at it**

## Step 2

**Improve the  
compound  
AI system**

## Step 3

**Improve the  
Large Language  
Models (LLMs)**



**Collect your dev data and look at it**



# Collect your development data and look at it

```
$ cd dev_data
$ ls
accept_reject_diff.jsonl      chat.jsonl                  feedback.jsonl              new_session.jsonl          step_run
.jsonl                       tokens_generated.jsonl
autocomplete.jsonl          context_used.jsonl         model_use.jsonl            select_context_item.jsonl  suggesti
ons.json
$ cat autocomplete.jsonl
{"time":24,"completion":>("(\n\n)","prompt":"<| fim_begin | >
\Code Llama",\n
\deepseek-coder:1.3b-instruct-q8_0": \DeepSeek Coder 1.3B",\n
\magicoder:7b-s-cl-q8_0": \Magicoder 6.7B",\n
\HuggingFaceH4/zephyr-7b-beta": \Zephyr 7B",\n
\bloke-
deepseek-coder-gptq-4bit-32g-actorder_True": \DeepSeek Coder",\n
\lunademo": \Unknown",\n
\dolphin-2.5-mixtral-8x7
b": \Dolphin Mixtral 8x7B",\n
\ggml": \Unknown",\n
\deepseek-coder:33b-instruct-q8_0": \DeepSeek Coder 33B",\n
\starcoder:1b": \StarCoder 1B",\n
\togethercomputer/llama-2-7b-chat": \Llama 2 7B",\n
\gemini-pro": \Gem
ini Pro",\n
\codebooga": \Codebooga 34B",\n
}\n\n if original_name not in models:\n return \N/A" else:\n
return models[original_name]\n\ndef get_data_from_bigquery():\n client = bigquery.Client.from_service_account_json(\autod
ebug-719c31b032c6.json")\n\n query_job = client.query("SELECT * FROM `autodebug.dbt_nsesti.tokens_generated` \n
LIMIT 1000000"")\n\n results = query_job.result() # Waits for job to complete\n\n model_tokens_count = {}\n\n for row in
results:\n model = convert_model_name(row.model)\n if model == \N/A":\n model_tokens_count.setdefault(\Other
r", 0)\n model_tokens_count[\Other"] += row.tokens\n #print("{} needs to be added to the models list".format(
row.model))\n else:\n model_tokens_count.setdefault(model, 0)\n model_tokens_count[model] += row.tokens\n
\n for key, value in sorted(model_tokens_count.items(), key=lambda item: item[1], reverse=True):\n print("{},".format
(value, key))\n # print("{} generated {} tokens".format(key, value))\n print("<| fim_hole | >\n\nif __name__ == \"_main__\":
\n get_data_from_bigquery(<| fim_end | >,"modelProvider":ollama","modelName":deepseek-coder:1.3b-base","completionOptions":{"stop
":["<| fim_begin | >","<| fim_hole | >","<| fim_end | >","/"]},"cacheHit":false,"accepted":true)
{"time":27,"completion":>("(\n\n)","prompt":"<| fim_begin | >
\Code Llama",\n
\deepseek-coder:1.3b-instruct-q8_0": \DeepSeek Coder 1.3B",\n
\magicoder:7b-s-cl-q8_0": \Magicoder 6.7B",\n
\HuggingFaceH4/zephyr-7b-beta": \Zephyr 7B",\n
\bloke-
deepseek-coder-gptq-4bit-32g-actorder_True": \DeepSeek Coder",\n
\lunademo": \Unknown",\n
\dolphin-2.5-mixtral-8x7
b": \Dolphin Mixtral 8x7B",\n
\ggml": \Unknown",\n
\deepseek-coder:33b-instruct-q8_0": \DeepSeek Coder 33B",\n
\starcoder:1b": \StarCoder 1B",\n
\togethercomputer/llama-2-7b-chat": \Llama 2 7B",\n
\gemini-pro": \Gem
ini Pro",\n
\codebooga": \Codebooga 34B",\n
}\n\n if original_name not in models:\n return \N/A" else:\n
return models[original_name]\n\ndef get_data_from_bigquery():\n client = bigquery.Client.from_service_account_json(\autod
ebug-719c31b032c6.json")\n\n query_job = client.query("SELECT * FROM `autodebug.dbt_nsesti.tokens_generated` \n
LIMIT 1000000"")\n\n results = query_job.result() # Waits for job to complete\n\n model_tokens_count = {}\n\n for row in
results:\n model = convert_model_name(row.model)\n if model == \N/A":\n model_tokens_count.setdefault(\Other
r", 0)\n model_tokens_count[\Other"] += row.tokens\n #print("{} needs to be added to the models list".format(
row.model))\n else:\n model_tokens_count.setdefault(model, 0)\n model_tokens_count[model] += row.tokens\n
\n for key, value in sorted(model_tokens_count.items(), key=lambda item: item[1], reverse=True):\n print("{},".format
(value, key))\n # print("{} generated {} tokens".format(key, value))\n print("<| fim_hole | >\n\nif __name__ == \"_main__\":
\n get_data_from_bigquery(<| fim_end | >,"modelProvider":ollama","modelName":deepseek-coder:1.3b-base","completionOptions":{"stop
```

# How to use your development data

## Step 1

**Collect your  
dev data and  
look at it**

## Step 2

**Improve the  
compound  
AI system**

## Step 3

**Improve the  
Large Language  
Models (LLMs)**



# How to use your development data

## Step 1

Collect your  
dev data and  
look at it

## Step 2

**Improve the  
compound  
AI system**

## Step 3

Improve the  
Large Language  
Models (LLMs)



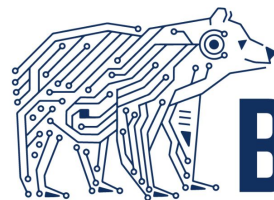
**Improve the compound AI system**



# Software dev copilots are compound AI systems

Software development AI systems today include many components

- “Chat” model
- “Tab” model
- “Embeddings” model
- Local context engine
- Server context engine
- Filtering engine
- etc.



## BAIR

BERKELEY ARTIFICIAL INTELLIGENCE RESEARCH

[Subscribe](#) [About](#) [Archive](#) [BAIR](#)

### The Shift from Models to Compound AI Systems

*Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, Ali Ghodsi*

*Feb 18, 2024*

AI caught everyone’s attention in 2023 with Large Language Models (LLMs) that can be instructed to perform general tasks, such as translation or coding, just by prompting. This naturally led to an intense focus on models as the primary ingredient in AI application development, with everyone wondering what capabilities new LLMs will bring. As more developers begin to build using LLMs, however, we believe that this focus is rapidly changing: **state-of-the-art AI results are increasingly obtained by compound systems with multiple components, not just monolithic models.**



# Provide clear and comprehensive instructions

Give me code for a component that helps me generate footnotes on my personal blog

Here is a simple JavaScript code snippet that can be used to generate footnotes on a blog. This code assumes that you have footnotes marked in your blog posts with a specific HTML tag, like `<sup>`.

```
window.onload = function() {
  var count = 0;
  var footnotes = document.getElementsByTagName('sup');
  var container = document.createElement('div');
  container.id = 'footnotes';
  var heading = document.createElement('h2');
  heading.innerHTML = 'Footnotes';
  container.appendChild(heading);
  for (var i = 0; i < footnotes.length; i++) {
    count++;
    var footnote = footnotes[i];
    footnote.id = 'footnote' + count;
    var backlink = document.createElement('a');
    backlink.innerHTML = count;
    backlink.href = '#footnote' + count;
    var item = document.createElement('p');
    item.appendChild(backlink);
    item.innerHTML += ' ' + footnote.innerHTML;
    container.appendChild(item);
    footnote.innerHTML = '<a href="#footnote' + count + '">' + count + '</a>';
  }
  document.body.appendChild(container);
}
```

This script will generate a list of footnotes at the bottom of the page, each with a backlink to the original reference in the text.

Note: You will need to include this script in the HTML file of your blog. Also, this is a very basic implementation and might not work perfectly for all cases. You might need to adjust it according to your specific needs.

Ask a follow-up

+ Add Context

↵ Enter

New Session (36 L)

VS.

Give me code for a component that helps me generate footnotes on my personal blog using two anchor tags and nothing else

Here's a simple example of a component that generates footnotes using two anchor tags:

```
<!DOCTYPE html>
<html>
<head>
<style>
.footnote {
font-size: smaller;
text-align: left;
margin-top: 1em;
border-top: 1px solid #ccc;
padding-top: 1em;
}
</style>
</head>
<body>

<p>
Here is some text with a footnote<sup><a href="#footnote1" id="ref1">1</a></sup>.
</p>

<div class="footnote" id="footnote1">
<sup><a href="#ref1">1</a></sup> Here is the text of the footnote.
</div>

</body>
</html>
```

In this example, we have a paragraph of text with a superscript number that links to the footnote at the bottom of the page. The footnote itself is in a `div` with a class of `footnote`, and it includes a superscript number that links back to the original reference in the text.

You can add as many footnotes as you need by following this pattern, just make sure to give each footnote and reference a unique ID.

Ask a follow-up

+ Add Context

↵ Enter

New Session (36 L)

# Add a system message with instructions that should always be followed

How can I kill a running process?

There are several ways to kill a running process in Linux. Here are a few methods:

1. Using the `kill` command:

The `kill` command is used to send a signal to a process. By default, it sends the `TERM` signal, which asks the process to terminate. If the process doesn't respond to this signal, you can use the `-9` option to send the `KILL` signal, which forces the process to terminate immediately.

```
kill PID
```

Replace `PID` with the process ID of the process you want to kill.

2. Using the `killall` command:

The `killall` command sends a signal to all processes with the specified name. By default, it sends the `TERM` signal.

```
killall process_name
```

Replace `process_name` with the name of the process you want to kill.

3. Using the `pkill` command:

The `pkill` command is similar to `killall`, but it allows you to specify a pattern to match against the process name.

```
pkill process_name
```

Replace `process_name` with the name or pattern of the process(es) you want to kill.

4. Using the `xkill` graphical tool:

If you're using a graphical interface, you can use the `xkill` tool to select a window and kill the process that owns it.

5. Using the `htop` or `top` command:

These commands display a list of running processes and allow you to kill processes by pressing `F9` and selecting the process to kill.

Remember, killing a process can disrupt its operation and may lead to data loss or other undesirable effects. Always make sure you're certain that you want to kill a process before doing so.

Ask a follow-up  
+ Add Context ↵ Enter

New Session (⌘ L)

vs.

```
{  
  "title": "Deepseek Coder Instruct (33B)",  
  "model": "deepseek-ai/deepseek-coder-33b-instruct",  
  "apiKey": "70df16b0e8391058bc1baf2e7805d7858e3696b57",  
  "completionOptions": {},  
  "provider": "together",  
  "systemMessage": "My operating system is MacOS"  
},
```

How can I kill a running process?

You can kill a running process in MacOS using the `kill` command. Here's how you can do it:

# Automatically filter for obviously bad suggestions and ask for a new suggestion

## Examples

- Block suggestions matching public code
  - Ensure only certain libraries are used
  - Make sure suggestions pass your linter
  - etc.
- 

# Improve how context from your codebase + software development lifecycle is retrieved and used

@Codebase where are the transformer configurations defined?

Context Used

The transformer configurations are defined in the `ModelArgs` class within the `model.py` file (lines 20-48). Specifically, the `from_name` class method suggests that there is a dictionary called `transformer_configs` that contains different configuration options for transformers, which are accessed by a given `name` key. However, the actual `transformer_configs` dictionary is not shown in the provided code snippets, so it is likely defined elsewhere in the codebase. To learn more about the specific configurations, one would need to look for the definition of the `transformer_configs` dictionary, which could be in a separate configuration file or within the same `model.py` file but outside of the provided code snippets.

Ask a follow-up

+ Add Context

Enter

New Session (⌘ L)

## Jira Issues

Type '@jira' to reference the conversation in a Jira issue. Make sure to include your own Atlassian API Token.

```
{
  "name": "issue",
  "params": {
    "domain": "company.atlassian.net",
    "email": "someone@somewhere.com",
    "token": "ATATT..."
  }
}
```

## Issue Query

By default, the following query will be used to find issues:

```
assignee = currentUser() AND resolution = Unresolved order by updated DESC
```

You can override this query by setting the `issueQuery` parameter.

## Built-in Context Providers

- Git Diff
- Terminal
- Documentation
- Open Files
- Codebase Retrieval
- Folders
- Exact Search
- File Tree
- Google
- GitHub Issues
- Jira Issues
- Code Outline
- Code Highlights
- PostgreSQL
- Database Tables
- Requesting Context Providers
- Building Your Own Context Provider

# Select the right model for the job

## “Chat” model

- Typically 30B+ parameters
- Highest quality responses
- Often run on server or used via an API endpoint
- Examples: GPT-4, DeepSeek Coder 33B, Claude 3, Code Llama 70B, etc.

## “Tab” model

- Typically 1-15B parameters
- Quality vs. latency tradeoffs
- Often run locally or on server
- Examples: Codex, StarCoder 2, Replit Code, etc.

# How to use your development data

## Step 1

Collect your  
dev data and  
look at it

## Step 2

**Improve the  
compound  
AI system**

## Step 3

Improve the  
Large Language  
Models (LLMs)



# How to use your development data

## Step 1

Collect your  
dev data and  
look at it

## Step 2

Improve the  
compound  
AI system

## Step 3

**Improve the  
Large Language  
Models (LLMs)**

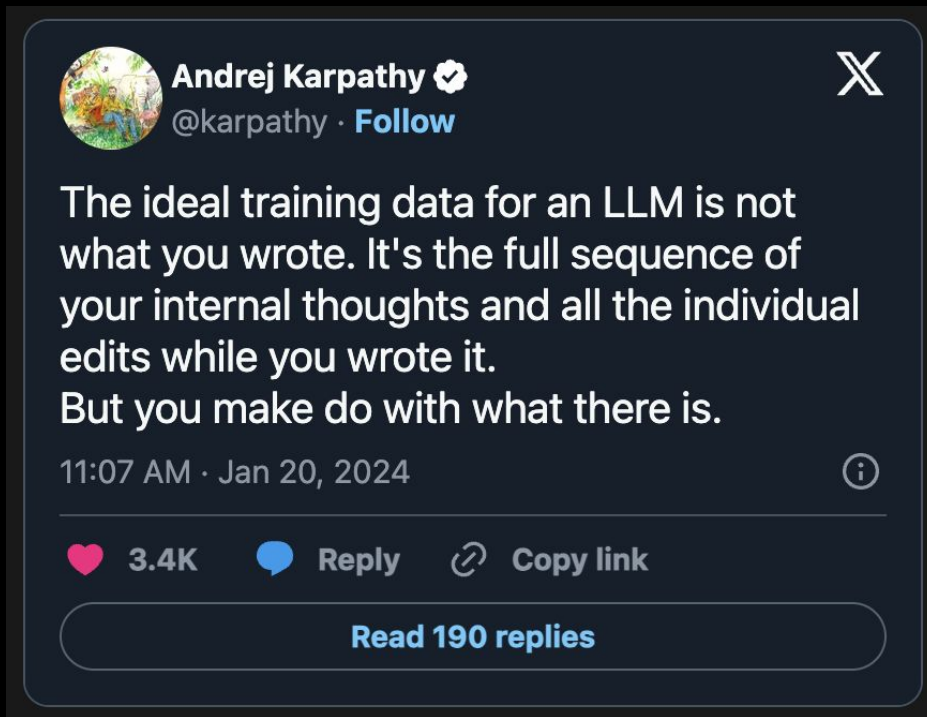


**Improve the LLMs**









# The ideal data for an LLM






A screenshot of a tweet from Andrej Karpathy. The tweet text discusses the ideal training data for an LLM, stating it's not just what you wrote but the full sequence of internal thoughts and edits. The tweet has 3.4K likes and 190 replies. The interface includes a profile picture, name, handle, follow button, and interaction icons for likes, replies, and link copying.

 **Andrej Karpathy**   
@karpathy · [Follow](#) 

The ideal training data for an LLM is not what you wrote. It's the full sequence of your internal thoughts and all the individual edits while you wrote it.  
But you make do with what there is.

11:07 AM · Jan 20, 2024 

---

 3.4K  Reply  Copy link

[Read 190 replies](#)

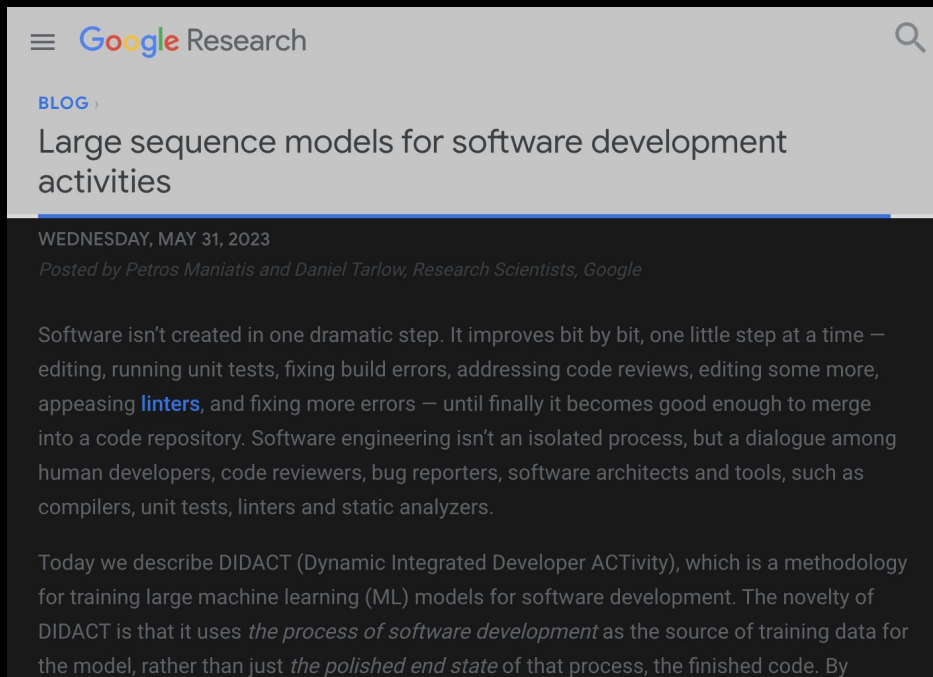
## By-product of using LLMs → close to ideal data

When you use LLMs while coding, you create development data that shows

- The step-by-step process a developer takes to complete a task
- The context a developer uses to decide what to do at each step
- Natural language that explains the reasoning behind the steps



# Google is already using their development data



The image is a screenshot of a Google Research blog post. At the top left, there is a hamburger menu icon followed by the text 'Google Research'. At the top right, there is a magnifying glass search icon. Below the header, the word 'BLOG' is written in blue with a right-pointing arrow. The main title of the post is 'Large sequence models for software development activities'. Below the title, the date 'WEDNESDAY, MAY 31, 2023' is displayed. Underneath the date, the author information reads 'Posted by Petros Maniatis and Daniel Tarlow, Research Scientists, Google'. The main body of the post contains two paragraphs of text. The first paragraph describes the iterative nature of software development, mentioning activities like editing, testing, and reviewing, and lists tools like linters, compilers, and static analyzers. The second paragraph introduces the 'DIDACT' methodology, explaining that it uses the entire software development process as training data for machine learning models, rather than just the final code.

Google Research

BLOG ›

## Large sequence models for software development activities

WEDNESDAY, MAY 31, 2023


Posted by Petros Maniatis and Daniel Tarlow, Research Scientists, Google

Software isn't created in one dramatic step. It improves bit by bit, one little step at a time — editing, running unit tests, fixing build errors, addressing code reviews, editing some more, appeasing [linters](#), and fixing more errors — until finally it becomes good enough to merge into a code repository. Software engineering isn't an isolated process, but a dialogue among human developers, code reviewers, bug reporters, software architects and tools, such as compilers, unit tests, linters and static analyzers.

Today we describe DIDACT (Dynamic Integrated Developer ACTivity), which is a methodology for training large machine learning (ML) models for software development. The novelty of DIDACT is that it uses *the process of software development* as the source of training data for the model, rather than just *the polished end state* of that process, the finished code. By

# So what development data is helpful now?

## Examples

- Tab-autocomplete accepted / rejected suggestions
  - /edit accepted / rejected suggestions
  - Thumbs up / down on chat responses
  - The “apply this code” button
  - Manual edits 1 min, 1 hour, 1 day later
  - What results from RAG are used in the response
  - etc.
- 

# Use fine-tuning to improve existing LLMs

Domain-specific instructions + hundreds of GPU hours

```
"""Pipeline to load slack into duckdb."""
from typing import List

import dlt
from pendulum import datetime
from slack import slack_source

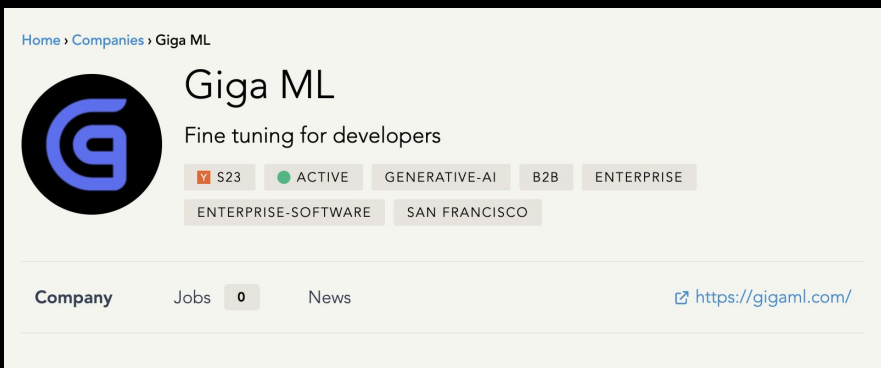
def load_all_resources(start_date) -> None:
    """Load all resources from slack without any selection of channels."""
    for channel in slack_source.get_all_channels():
        load_channel(channel, start_date)

"""Pipeline to load slack into duckdb."""
from typing import List


import dlt
from pendulum import datetime
from slack import slack_source



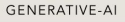


def load_all_resources(start_date) -> None:
    """Load all resources from slack without any selection of channels."""
    pipeline = dlt.pipeline(
        pipeline_name="slack", destination="duckdb", dataset_name="slack_data"
    )
```

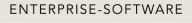
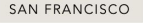
dltHub fine-tuned StarCoder 2 on their codebase, docs, accepted tab autocomplete data, etc.



Home > Companies > Giga ML

 **Giga ML**  
Fine tuning for developers

Company Jobs **0** News <https://gigaml.com>

GigaML is fine-tuning StarCoder 2 on accepted tab autocomplete data

# Use domain-adaptive continued pre-training to improve open-source LLMs

Billions of tokens of relevant company data + thousands of GPU hours

## ChipNeMo: Domain-Adapted LLMs for Chip Design

Mingjie Liu<sup>1</sup>, Teodor-Dumitru Ene<sup>1</sup>, Robert Kirby<sup>1</sup>, Chris Cheng<sup>1</sup>, Nathaniel Pinckney<sup>1</sup>, Rongjian Liang<sup>1</sup>  
Jonah Alben, Himyanshu Anand, Sammitra Banerjee, Ismet Bayraktaroglu, Bomita Bhaskaran, Bryan Catanzaro  
Arjun Chaudhuri, Sharon Clay, Bill Dally, Laura Dang, Parikshit Deshpande, Siddhant Dhodhi, Sameer Halepate  
Eric Hill, Jianshang Hu, Sumit Jain, Bruce Khailany, George Kokai, Kishor Kunal, Xiaowei Li  
Charley Lind, Hao Liu, Stuart Oberman, Sujet Omar, Sreedhar Pratty, Jonathan Raiman, Ambar Sarkar  
Zhengjiang Shao, Hanfei Sun, Pratik P Suthar, Varun Tej, Walker Turner, Kaizhe Xu, Haoxing Ren  
NVIDIA

**Abstract**—ChipNeMo aims to explore the applications of large language models (LLMs) for industrial chip design. Instead of directly deploying off-the-shelf commercial or open-source LLMs, we instead adopt the following domain adaptation techniques: custom tokenizers, domain-adaptive continued pretraining, supervised fine-tuning (SFT) with domain-specific instructions, and domain-adapted retrieval models. We evaluate these methods on three selected LLM applications for chip design: an engineering assistant chatbot, EDA script generation, and bug summarization and analysis. Our results show that these domain adaptation techniques enable significant LLM performance improvements over general-purpose base models across the three evaluated applications, enabling up to 5x model size reduction with similar or better performance on a range of design tasks. Our findings also indicate that there's still room for improvement between our current results and ideal outcomes. We believe that further investigation of domain-adapted LLM approaches will help close this gap in the future.

and report generation, and bug triage. In this study, we focus on these three specific LLM applications: an **engineering assistant chatbot** for GPU ASIC and Architecture design engineers, which understands internal HW designs and is capable of explaining complex design topics; **EDA scripts generation** for two domain specific tools based on Python and Tcl for VLSI timing analysis tasks specified in English; **bug summarization and analysis** as part of an internal bug and issue tracking system.

Although general-purpose LLMs trained on vast amounts of internet data exhibit remarkable capabilities in generative AI tasks across diverse domains (as demonstrated by Bubeck et al. in [9]), recent work such as BloombergGPT [10] and BioMedLLM [11] demonstrate that domain-specific LLM models can outperform a general purpose model on domain-specific tasks. In the hardware design domain, [6], [12] showed that open-

## 2 Code Llama: Specializing Llama 2 for code

### 2.1 The Code Llama models family

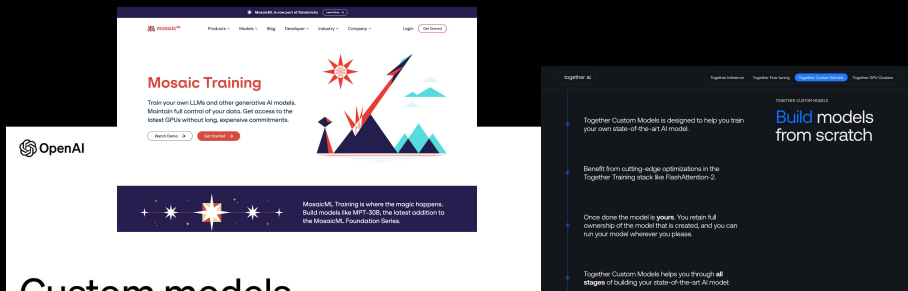
**Code Llama.** The CODE LLAMA models constitute foundation models for code generation. They come in four model sizes: 7B, 13B, 34B and 70B parameters. The 7B, 13B and 70B models are trained using an infilling objective (Section 2.3), and are appropriate to be used in an IDE to complete code in the middle of a file, for example. The 34B model was trained without the infilling objective. All CODE LLAMA models are initialized with LLAMA 2 model weights and trained on 500B tokens from a code-heavy dataset (see Section 2.2 for more details), except CODE LLAMA 70B which was trained on 1T tokens. They are all fine-tuned to handle long contexts as detailed in Section 2.4.

How ChipNeMo was  
created by Nvidia

How Code Llama  
was created by Meta

# Pre-train your own LLM from scratch

Trillions of tokens of Internet data + company data + millions of GPU hours



**OpenAI**

### Mosaic Training

Train your own LLMs and other generative AI models. Maintain full control of your data. Get access to the latest GPUs without long, expensive commitments.

MosaicML Training is where the magic happens. Build models like LLaMA, the latest addition to the MosaicML Foundation Series.

**Together AI**

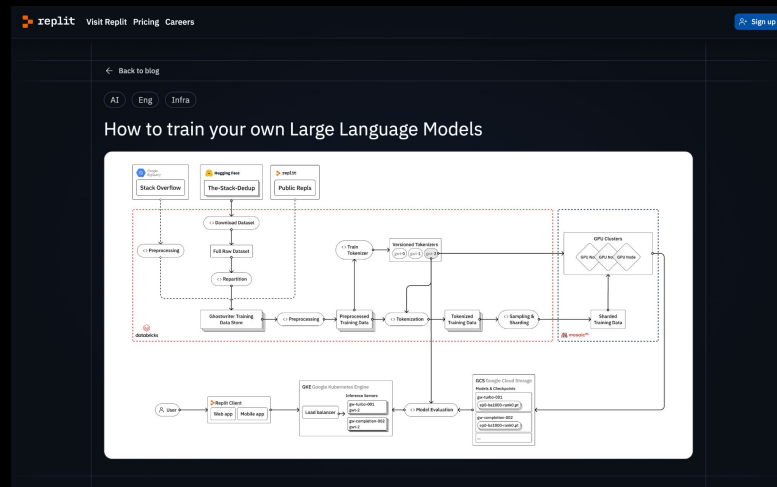
### Build models from scratch

- Together Custom Models is designed to help you train your own state-of-the-art AI model.
- Benefit from cutting-edge optimizations in the Together Training stack like FlashAttention-2.
- Once done the model is yours. You retain full ownership of the model that is created, and you can run your model whenever you please.
- Together Custom Models helps you through all stages of building your state-of-the-art AI model.

The Custom Models program gives selected organizations an opportunity to work with a dedicated group of OpenAI researchers to train custom GPT-4 models to their specific domain. This includes modifying every step of the model training process, from doing additional domain specific pre-training, to running a custom RL post-training process tailored for the specific domain. Organizations will have exclusive access to their custom models. This program is particularly applicable to domains with extremely large proprietary datasets—billions of tokens at minimum.

## Custom models

## Information



**replit** Visit Replit Pricing Careers

### How to train your own Large Language Models

The diagram illustrates the LLM training pipeline, starting with data collection and preprocessing, followed by training on a GPU cluster, and finally evaluation and deployment. Key components include: Stack Overflow, The Stack-DevOps, Public Repla, Preprocessing, Full-Row Dataset, Repartition, OpenAI Training Data Store, Proprietary Training Data, Tokenization, Tokenized Training Data, Sampling & Streaming, and Shared Training Data. The pipeline is supported by infrastructure like AWS, GCP, and Azure, and involves various roles such as ML Engineers, Data Engineers, and DevOps Engineers.

Replit trained their own model

OpenAI, MosaicML, Together, etc. will help you train your own custom model

# How to use your development data

## Step 1

Collect your  
dev data and  
look at it

## Step 2

Improve the  
compound  
AI system

## Step 3

Improve the  
Large Language  
Models (LLMs)





# TL;DR: Dev data can be used to automate even more

2402.09171v1 [cs.SE] 14 Feb 2024

## Automated Unit Test Improvement using Large Language Models at Meta

Nadia Alshahwan\*  
Jubin Chheda  
Anastasia Finegenova  
Beliz Gokkaya  
Mark Harman  
Inna Harper  
Alexandru Marginean  
Shubho Sengupta  
Eddy Wang  
Meta Platforms Inc.,  
Menlo Park, California, USA

### ABSTRACT

This paper describes Meta's TestGen-LLM tool, which uses LLMs to automatically improve existing human-written tests. TestGen-LLM verifies that its generated test classes successfully clear a set of filters that assure measurable improvement over the original test suite, thereby eliminating problems due to LLM hallucination. We describe the deployment of TestGen-LLM at Meta test-a-thons for the Instagram and Facebook platforms. In an evaluation on Reels and Stories products for Instagram, 75% of TestGen-LLM's test cases built correctly, 37% passed reliably, and 25% increased coverage. During Meta's Instagram and Facebook test-a-thons, it improved 11.5% of all classes to which it was applied, with 73% of its recommendations being accepted for production deployment by Meta software engineers. We believe this is the first report on industrial scale deployment of LLM-generated code backed by such assurances of code improvement.

### KEYWORDS

Unit Testing, Automated Test Generation, Large Language Models, LLMs, Genetic Improvement.

### 1 INTRODUCTION

As part of our overall mission to automate unit test generation for Android code, we have developed an automated test class improver, TestGen-LLM. TestGen-LLM uses two of Meta's<sup>1</sup> Large Language Models (LLMs) to extend existing human-written, Kotlin test classes by generating additional test cases that cover previously missed corner cases, and that increase overall test coverage. TestGen-LLM is an example of Assured Offline LLM-Based Software Engineering (Assured Offline LLMSE) [6].

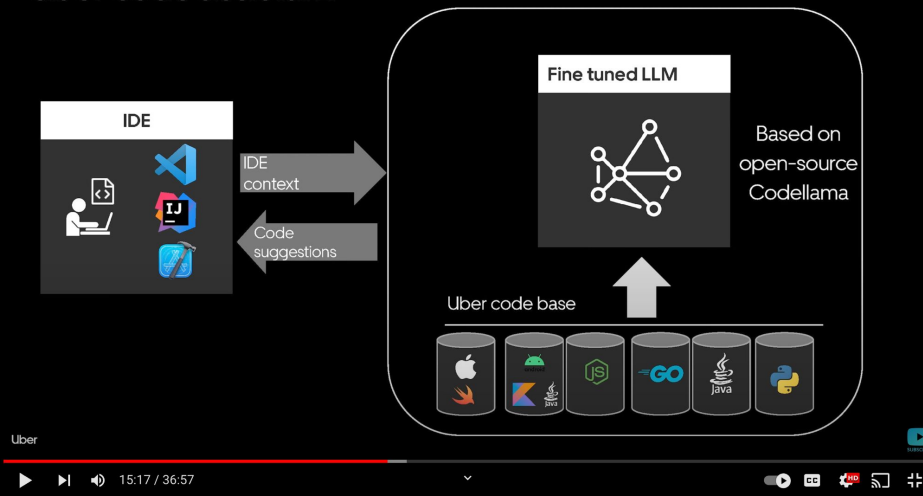
That is, unlike other LLM-based code and test generation techniques, TestGen-LLM uses Assured Offline LLMSE to embed the language models, as a service, in a larger software engineering workflow that ultimately recommends fully formed software improvements rather than smaller code snippets. These fully-formed code improvements are backed by verifiable guarantees for improvement and non-regression of existing behavior. A filtration process discards any test case that cannot be guaranteed to meet the assurances.

The filtration process can be used to evaluate the performance of a particular LLM, prompt strategy, or choice of hyper-parameters.

From Myth to Legend: How Generative AI can Supercharge Productivity to Create 10x Developers



## Uber code assistant



# Thanks!

We are at the beginning on this journey :)

Lots more R&D to come!

We are hiring



# Appendix





ChatGPT



GitHub Copilot



Build your own



AI

## GitHub teases Copilot enterprise plan that lets companies customize for their codebase

The plan will cost \$39/month and will arrive in February, 2024

Paul Sawers @psawers / 9:10 AM PST • November 8, 2023

 Comment



 Image Credits: CFOTO/Future Publishing / Getty Images

GitHub today announced plans for an enterprise subscription tier that will allow companies to fine-tune its Copilot pair-programmer based on their internal codebase.

July 3, 2023

# What GitHub Copilot Lacks: Fine-tuning on Your Private Code

Jonathan Ma, Prem Nair, Douglas Chen, Nick Moy

Share this post



## Contents

[Hallucinations and Why Fine-tuning Matters](#)

[Experimental Setup: Fine-tuned Codeium vs GitHub Copilot](#)

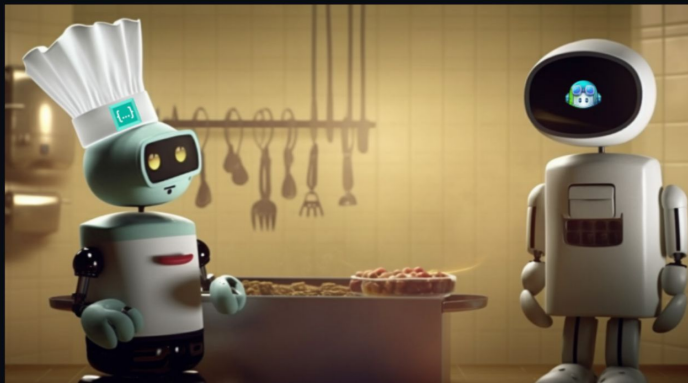
[Results: Fine-tuned Codeium Outperforms GitHub Copilot](#)

[Task 1: Using the Right Class Given Comment](#)

[Task 1 Take #2: Using the Right Class Given Comment](#)

[Task 2: Populating Arguments](#)

TL;DR We prove that fine-tuning Codeium's generic base code model on unseen code leads to substantial, observable improvements in suggestion quality over other tools such as GitHub Copilot.



[← Back to blog](#)

# Introducing SafeCoder

Published August 22, 2023

[Update on GitHub](#)

[jeffboudier](#)  
Jeff Boudier



[philschmid](#)  
Philipp Schmid

☰ Today we are excited to announce SafeCoder - a code assistant solution built for the enterprise.

The goal of SafeCoder is to unlock software development productivity for the enterprise, with a fully compliant and self-hosted pair programmer. In marketing speak: “your own on-prem GitHub copilot”.

Before we dive deeper, here’s what you need to know:

- SafeCoder is not a model, but a complete end-to-end commercial solution
- SafeCoder is built with security and privacy as core principles - code never leaves the VPC during training or inference
- SafeCoder is designed for self-hosting by the customer on their own infrastructure
- SafeCoder is designed for customers to own their own Code Large Language Model

# How Cody understands your codebase

Alex Isken, Corey Hill February 15, 2024

We often say that [Cody](#) uses a deep understanding of your codebase to help you write and understand code faster. Meanwhile, we're frequently asked: How does Cody understand *my* codebase? And what exactly does that mean?

In building Cody, we found that Large Language Models (LLMs), when applied to specific use cases, are only as good as the context they're given. For Cody, those use cases are writing, understanding, and answering questions about code. Cody needs the context of a user's codebase to do those things effectively.

This blog unpacks why context matters and how we've built Cody Enterprise to use the right context to help you write and understand code.

## Why does context matter?

When it comes to AI chat and asking questions of an LLM, you can think of questions loosely falling into two categories as they relate to context:

- General coding questions that *are not* dependent on a user's context
- Specific questions that *are* dependent on a user's existing code/context



## copilot-explorer

### Copilot Internals | [thakkarparth007.github.io](https://thakkarparth007.github.io)

Github Copilot has been incredibly useful to me. It can often magically read my mind and make useful suggestions. The thing that surprised me the most was its ability to correctly "guess" functions/variables from surrounding code – including from other files. This can only happen, if the copilot extension sends valuable information from surrounding code to the Codex model. I was curious about how it worked, so I decided to take a look at the source code.

In this post, I try to answer specific questions about the internals of Copilot, while also describing some interesting observations I made as I combed through the code. I will provide pointers to the relevant code for almost everything I talk about, so that interested folks can take a look at the code themselves.

#### Overview

This post is organized as follows:

- [Reverse Engineering preview](#)
- [Copilot: a 10,000 feet view](#)
- [Secret Sauce 1: Prompt engineering](#)
  - [What does a prompt look like?](#)
    - [How is the prompt prepared? A code walkthrough.](#)
      - [A close look at Snippet Extraction](#)
- [Secret Sauce 2: Model Invocation](#)
  - [Inline/GhostText](#)
    - [Preventing poor requests via Contextual Filter](#)
  - [Copilot Panel](#)
  - [Don't show unhelpful completions](#)
- [Secret Sauce 3: Telemetry](#)
  - [Question 1: How is the 40% number measured?](#)
  - [Question 2: Does telemetry data include code snippets?](#)
  - [Important Update](#)
- [Other random tidbits](#)
  - [Enabling verbose logging](#)
- [Onwards](#)
- [Links](#)

#### Reverse Engineering preview