IMEN574/IDSC574 Programing for Data Science

5. **Modern IDE**

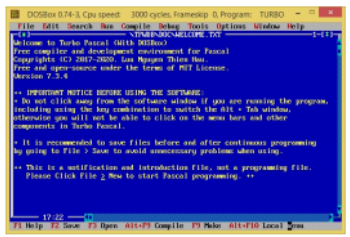
Spring 2026

Instructor: **Sangwoo Mo**

Why

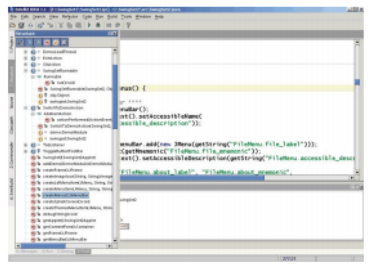
- IDE (Integrated Development Environment)
  - All-in-one workspaces for software development containing editor, compiler, debugger, and more
  - Most development work is done there so it’s a natural form factor for AI enhancement
  - In their evolution, there’s always a see-saw between functionality consolidation and developer customization

A brief history



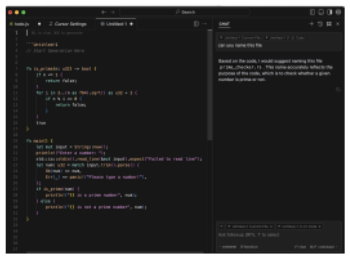
Release of Turbo Pascal, first true IDE

1983



Intellij IDEA released with advanced contextual code navigation, refactoring, code completion

2001



Cursor released, one of the first widely used AI native IDEs

2023

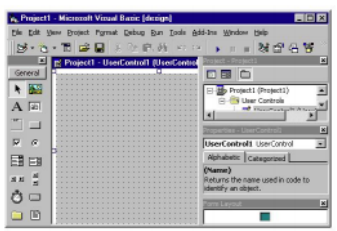
1980

1997

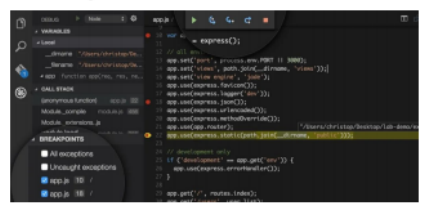
2015

2030

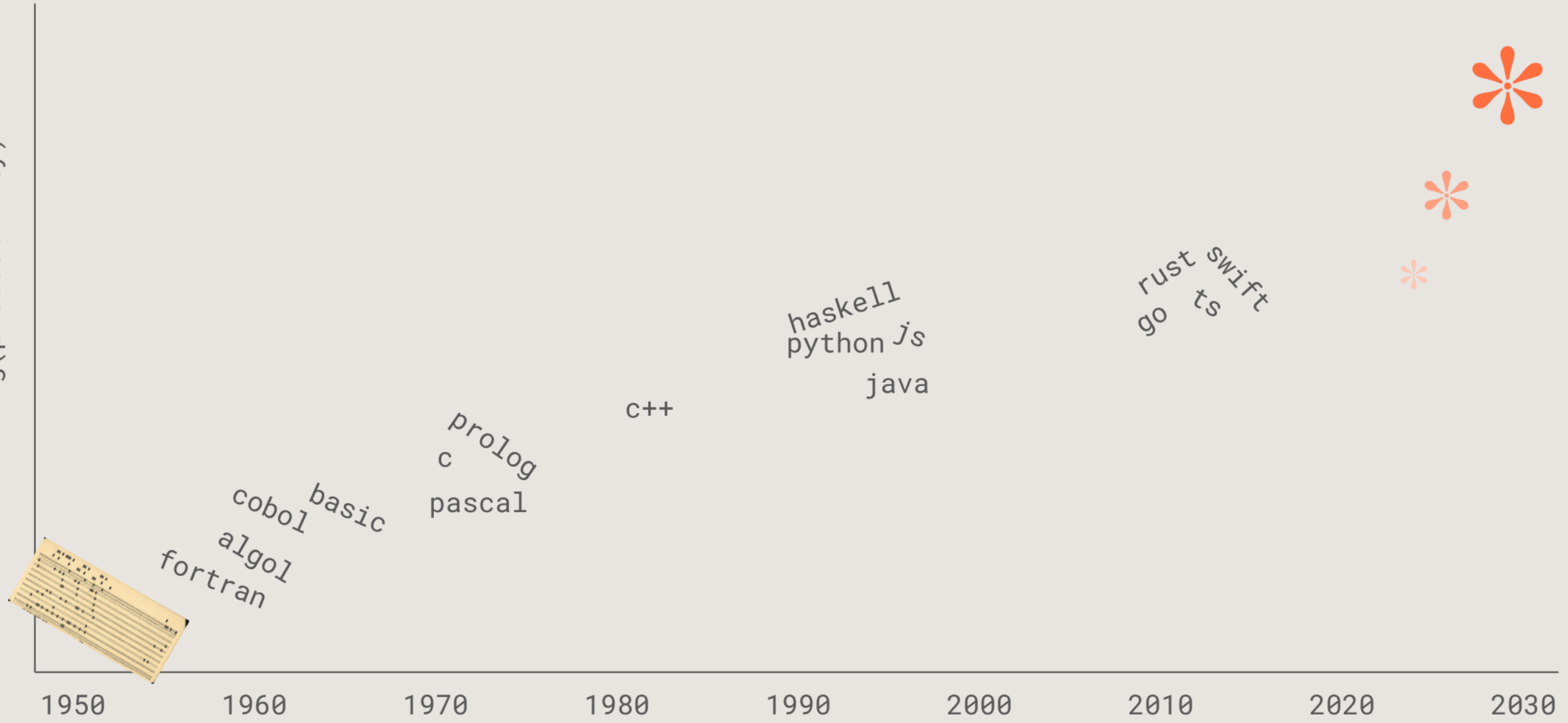
Microsoft Visual Studio released, offering advanced debugging capabilities for the C++/Visual Basic language



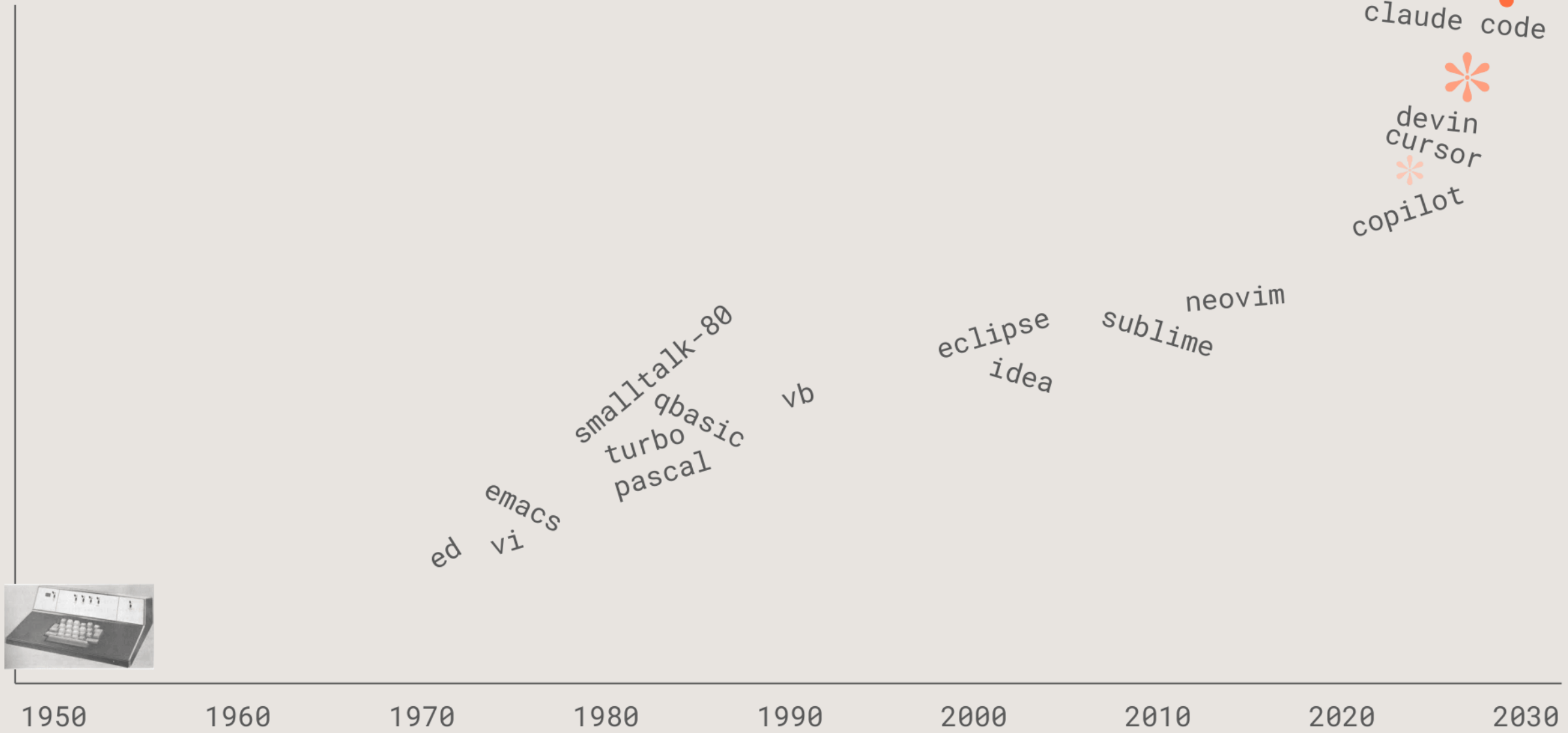
Microsoft VSCode released offering lightweight editor with highly extensible ecosystem



programming language productivity is
increasing exponentially, driven by ai

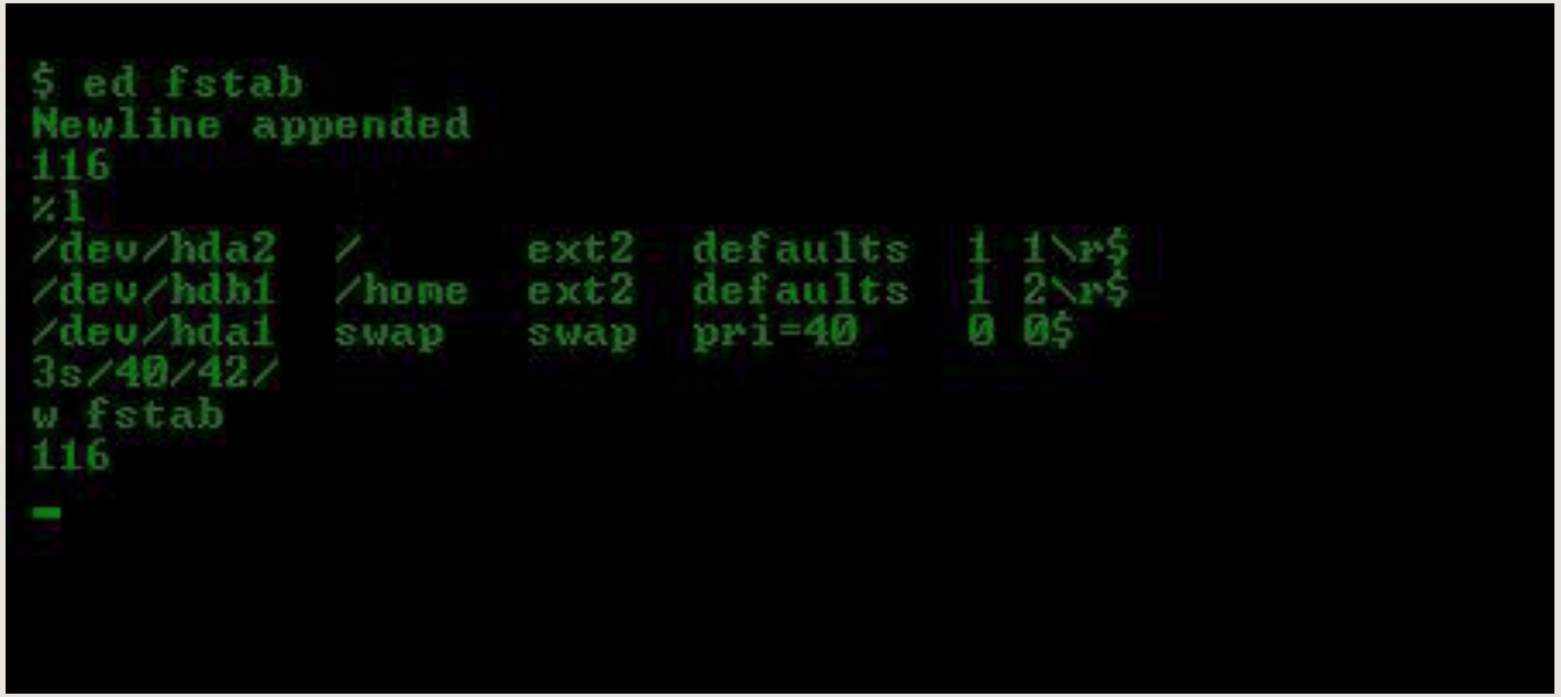


ide productivity is following a similar exponential, also driven by ai

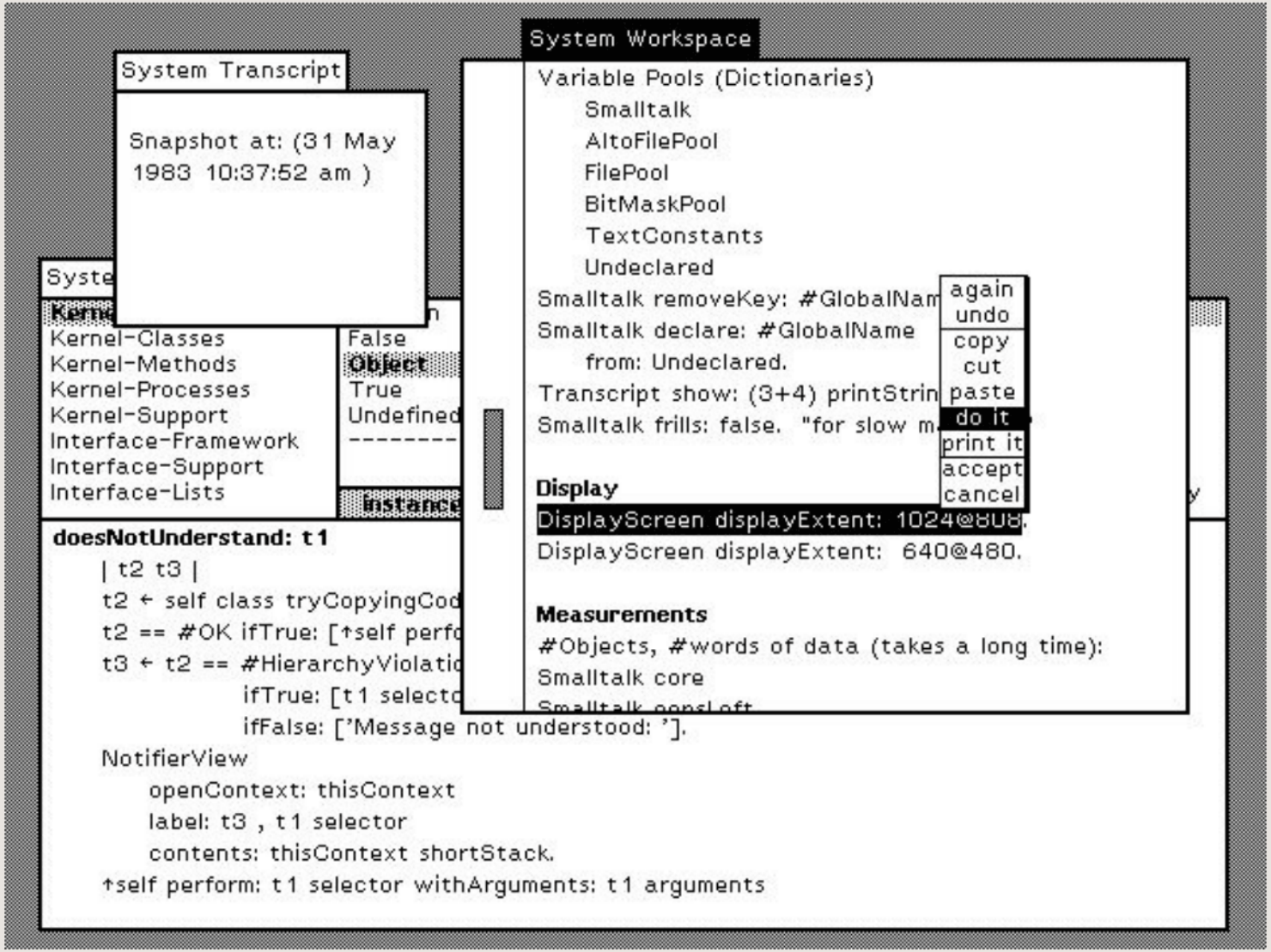




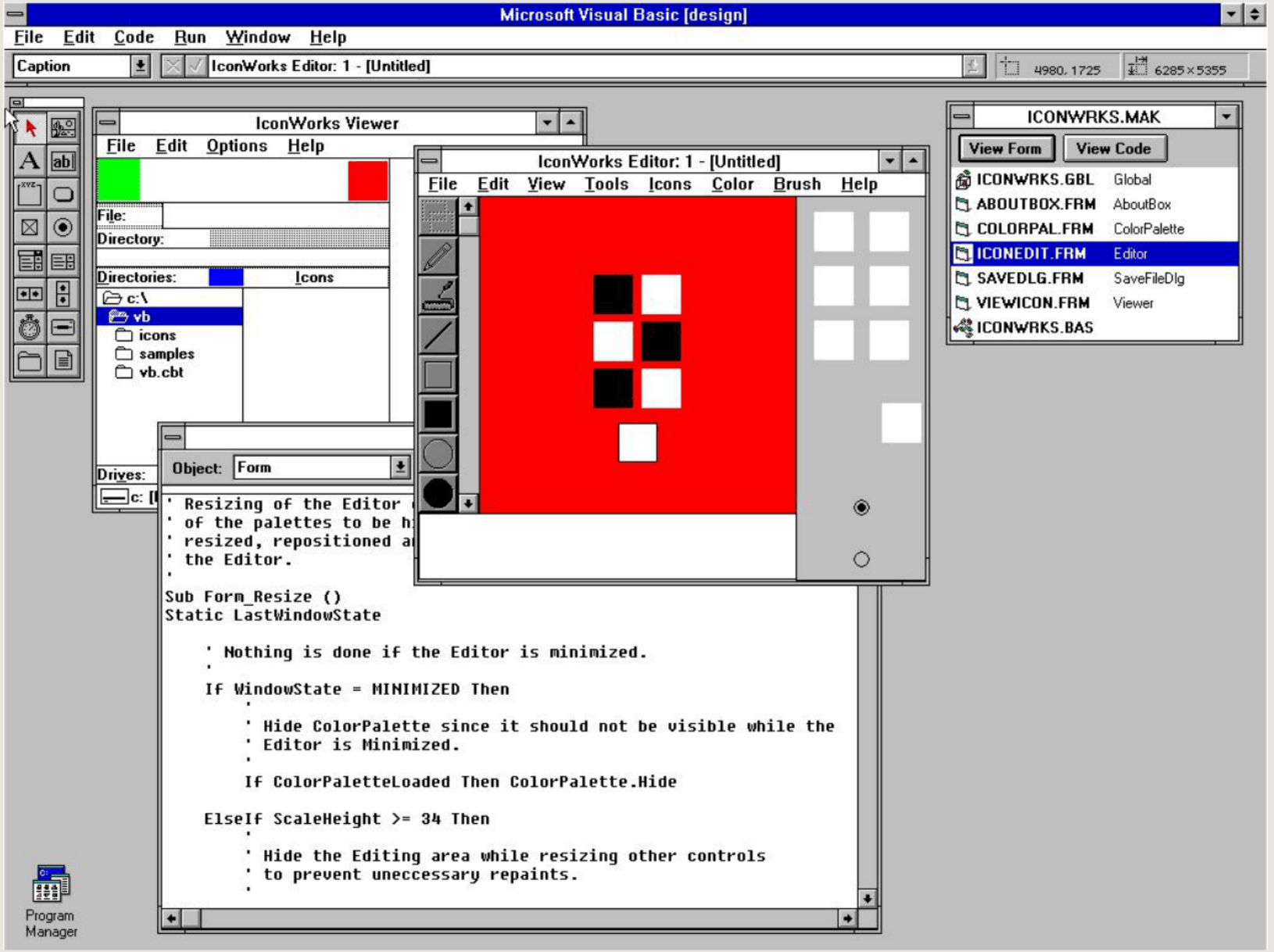
ibm 029 (1964)


$ ed fstab
Newline appended
116
%l
/dev/hda2 / ext2 defaults 1 1\r$
/dev/hdb1 /home ext2 defaults 1 2\r$
/dev/hda1 swap swap pri=40 0 0$
3s/40/42/
w fstab
116

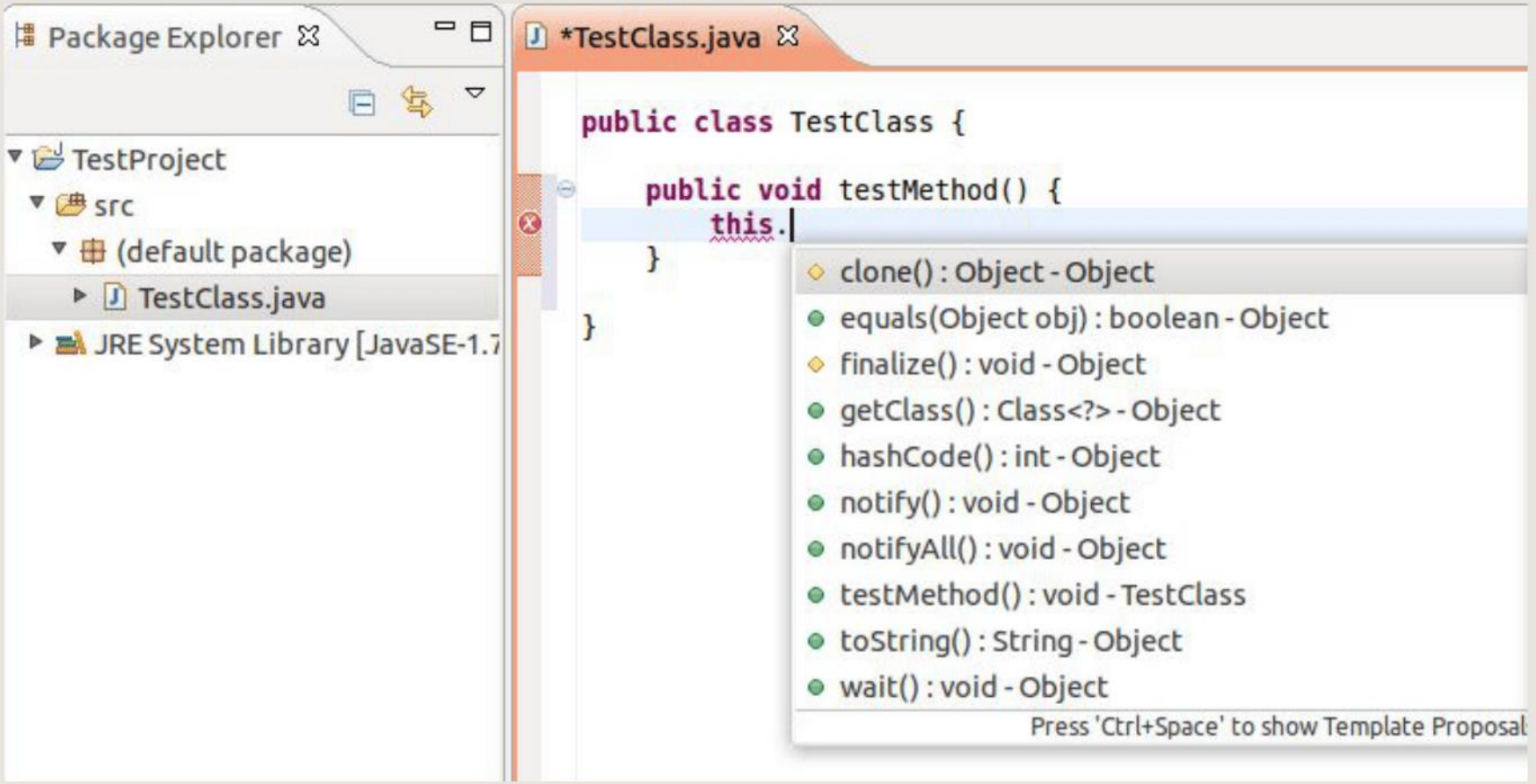
ed (1969)



smalltalk-80 (1980)



visual basic (1991)



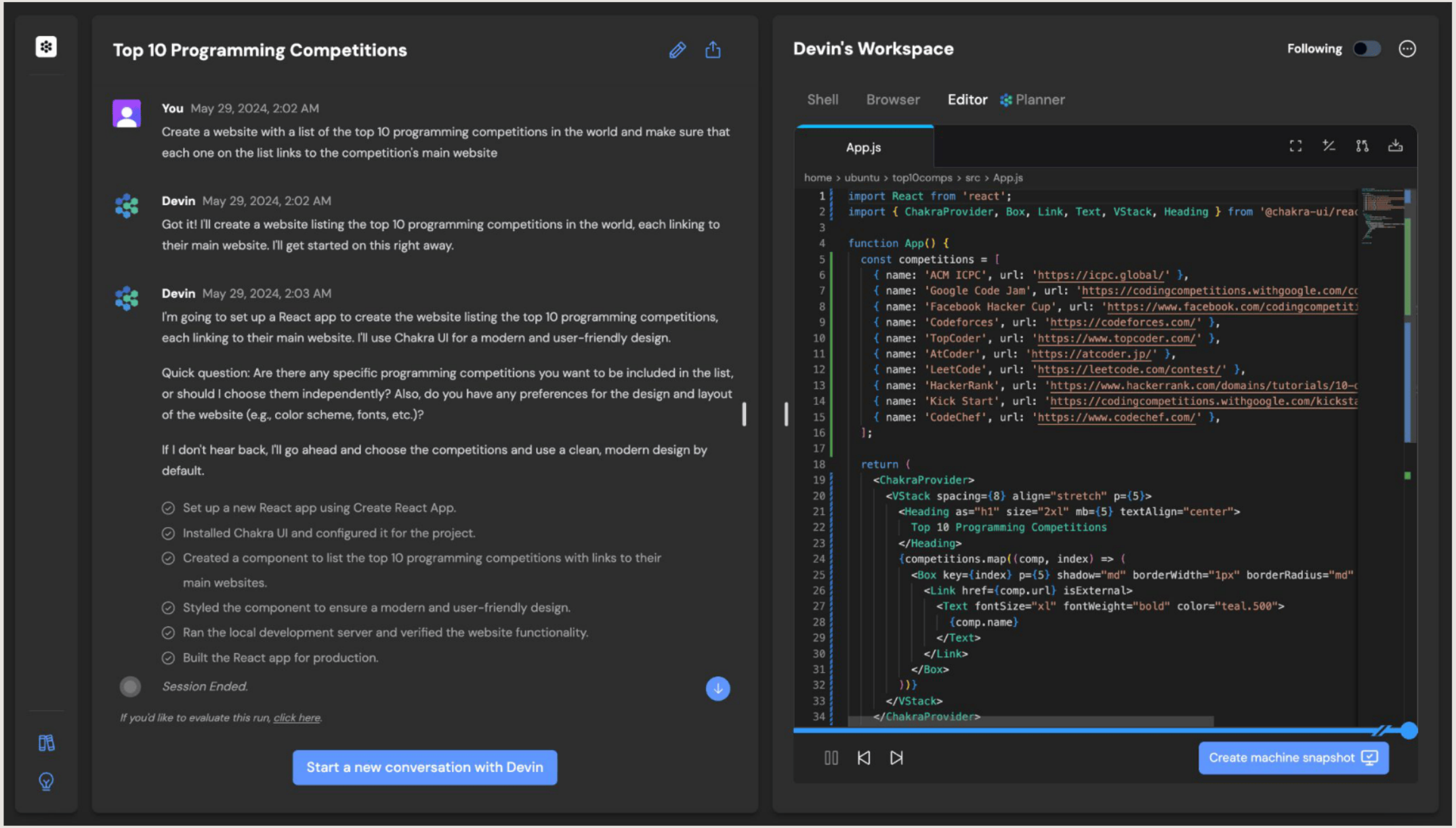
eclipse (2001)

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

class Sequence(nn.Module):
    def __init__(self):
        super(Sequence, self).__init__()
        self.lstm1 = nn.LSTMCell(1, 51)
        self.lstm2 = nn.LSTMCell(51, 51)
        self.linear = nn.Linear(51, 1)

    def forward(self, input, future = 0):
        outputs = []
        h_t = torch.zeros(input.size(0), 51, dtype=torch.double)
```

copilot (2021)

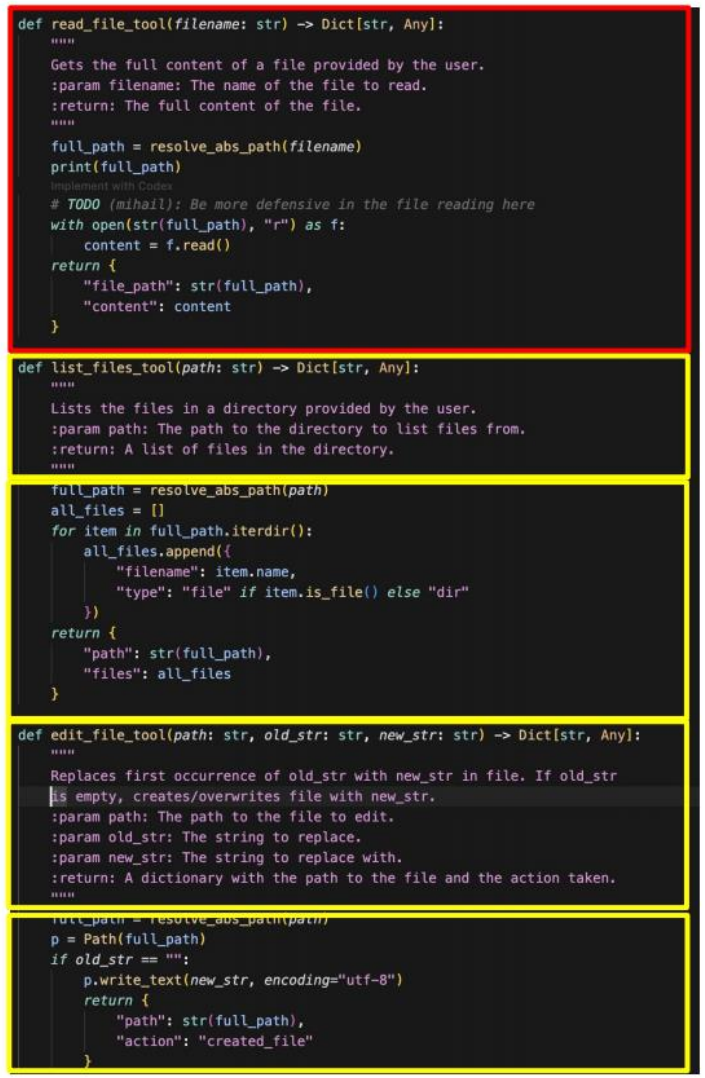

Top 10 Programming Competitions
You May 29, 2024, 2:02 AM
Create a website with a list of the top 10 programming competitions in the world and make sure that each one on the list links to the competition's main website
Devin May 29, 2024, 2:02 AM
Got it! I'll create a website listing the top 10 programming competitions in the world, each linking to their main website. I'll get started on this right away.
Devin May 29, 2024, 2:03 AM
I'm going to set up a React app to create the website listing the top 10 programming competitions, each linking to their main website. I'll use Chakra UI for a modern and user-friendly design.
Quick question: Are there any specific programming competitions you want to be included in the list, or should I choose them independently? Also, do you have any preferences for the design and layout of the website (e.g., color scheme, fonts, etc.)?
If I don't hear back, I'll go ahead and choose the competitions and use a clean, modern design by default.
Set up a new React app using Create React App.
Installed Chakra UI and configured it for the project.
Created a component to list the top 10 programming competitions with links to their main websites.
Styled the component to ensure a modern and user-friendly design.
Ran the local development server and verified the website functionality.
Built the React app for production.
Session Ended.
If you'd like to evaluate this run, click here.
Start a new conversation with Devin
Devin's Workspace
Following
Shell
Browser
Editor
Planner
App.js
home > ubuntu > top10comps > src > App.js
import React from 'react';
import { ChakraProvider, Box, Link, Text, VStack, Heading } from '@chakra-ui/reac
function App() {
const competitions = [
{ name: 'ACM ICPC', url: 'https://icpc.global/' },
{ name: 'Google Code Jam', url: 'https://codingcompetitions.withgoogle.com/c
{ name: 'Facebook Hacker Cup', url: 'https://www.facebook.com/codingcompetit
{ name: 'Codeforces', url: 'https://codeforces.com/' },
{ name: 'TopCoder', url: 'https://www.topcoder.com/' },
{ name: 'AtCoder', url: 'https://atcoder.jp/' },
{ name: 'LeetCode', url: 'https://leetcode.com/contest/' },
{ name: 'HackerRank', url: 'https://www.hackerrank.com/domains/tutorials/10-
{ name: 'Kick Start', url: 'https://codingcompetitions.withgoogle.com/kicksta
{ name: 'CodeChef', url: 'https://www.codechef.com/' },
];
return (
<ChakraProvider>
<VStack spacing={8} align="stretch" p={5}>
<Heading as="h1" size="2xl" mb={5} textAlign="center">
Top 10 Programming Competitions
</Heading>
{competitions.map((comp, index) => (
<Box key={index} p={5} shadow="md" borderWidth="1px" borderRadius="md"
<Link href={comp.url} isExternal>
<Text fontSize="xl" fontWeight="bold" color="teal.500">
{comp.name}
</Text>
</Link>
</Box>
))}
</VStack>
</ChakraProvider>
Create machine snapshot

devin (2024)

Usage

- Bread-and-butter modes
  - Inline
  - Function
  - Single-file
  - Multi-file
- True AI-native
  - Background agents
  - MCP
  - Learn memories
  - Bugbot (PR review)

How an AI IDE works under-the-hood



```python
def read_file_tool(filename: str) -> Dict[str, Any]:
    """
    Gets the full content of a file provided by the user.
    :param filename: The name of the file to read.
    :return: The full content of the file.
    """
    full_path = resolve_abs_path(filename)
    print(full_path)
    Implement with Codex
    # TODO (mihail): Be more defensive in the file reading here
    with open(str(full_path), "r") as f:
        content = f.read()
    return {
        "file_path": str(full_path),
        "content": content
    }
```

```python
def list_files_tool(path: str) -> Dict[str, Any]:
    """
    Lists the files in a directory provided by the user.
    :param path: The path to the directory to list files from.
    :return: A list of files in the directory.
    """
    full_path = resolve_abs_path(path)
    all_files = []
    for item in full_path.iterdir():
        all_files.append({
            "filename": item.name,
            "type": "file" if item.is_file() else "dir"
        })
    return {
        "path": str(full_path),
        "files": all_files
    }
```

```python
def edit_file_tool(path: str, old_str: str, new_str: str) -> Dict[str, Any]:
    """
    Replaces first occurrence of old_str with new_str in file. If old_str
    is empty, creates/overwrites file with new_str.
    :param path: The path to the file to edit.
    :param old_str: The string to replace.
    :param new_str: The string to replace with.
    :return: A dictionary with the path to the file and the action taken.
    """
    full_path = resolve_abs_path(path)
    p = Path(full_path)
    if old_str == "":
        p.write_text(new_str, encoding="utf-8")
        return {
            "path": str(full_path),
            "action": "created_file"
        }
```

Tab-complete

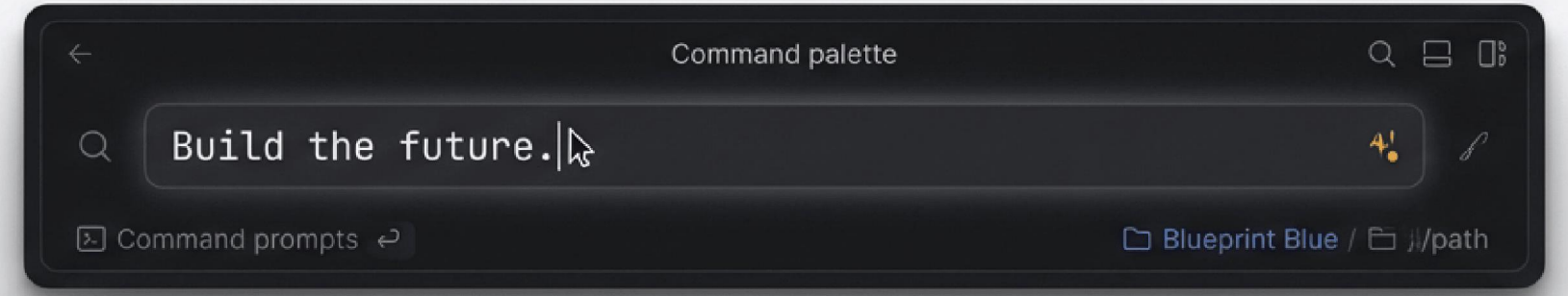
- Small context window around current code is encrypted
- Server receives and runs infilling LLM
- Suggestion sent back and displayed

Chat

- Store code chunks as embeddings in semantic index on server (obfuscated filename + code)
- Any query retrieves most relevant chunks and feeds as context into LLM
- IDE regularly re-index code chunks and syncs embeddings
- Chunk diffs are computed via Merkle trees for efficient updates

Cursor: The AI-First Code Editor

From writing code to directing code.

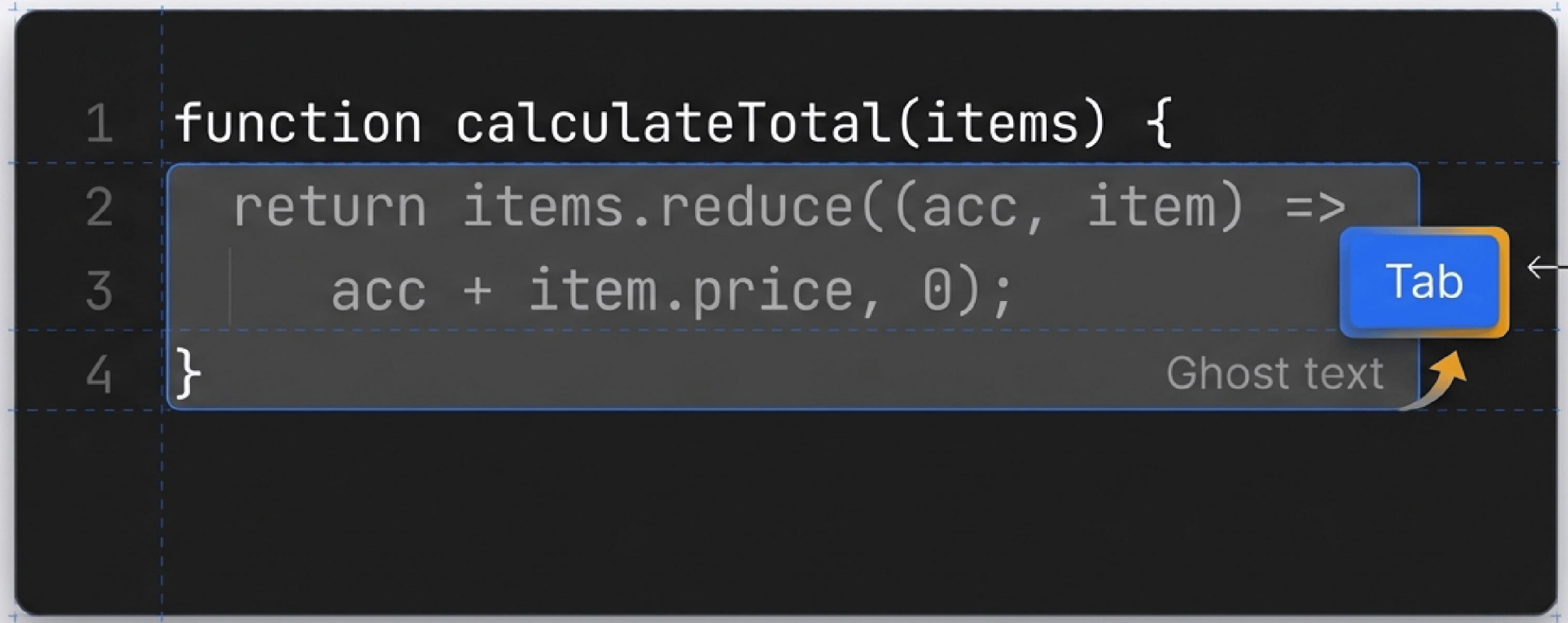


command palette // AI-assisted development // build 2024.10

NotebookLM

Tab to Predict (Copilot++)

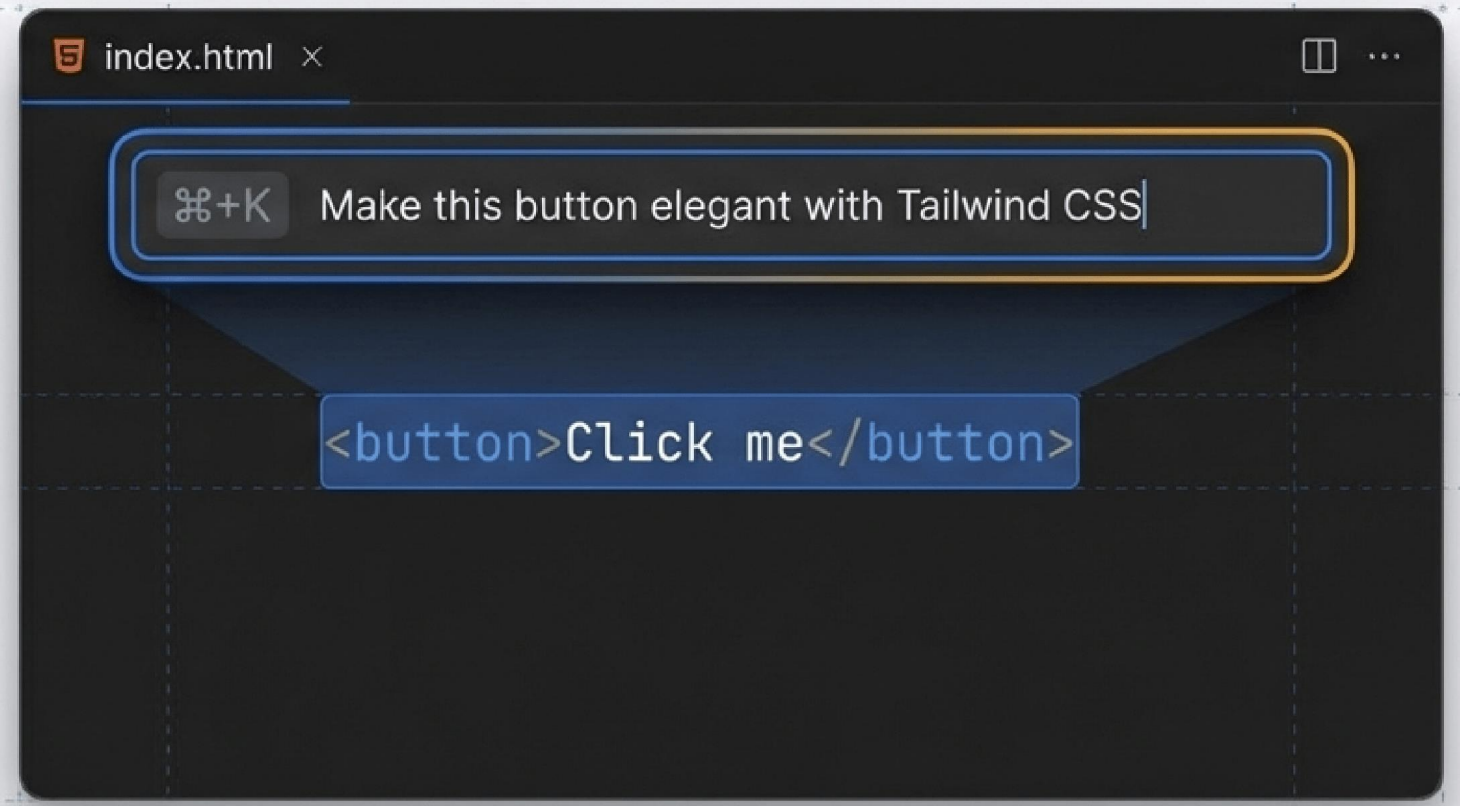
Predicts your next change, not just your next word.



NotebookLM

Cmd+K: Just Ask

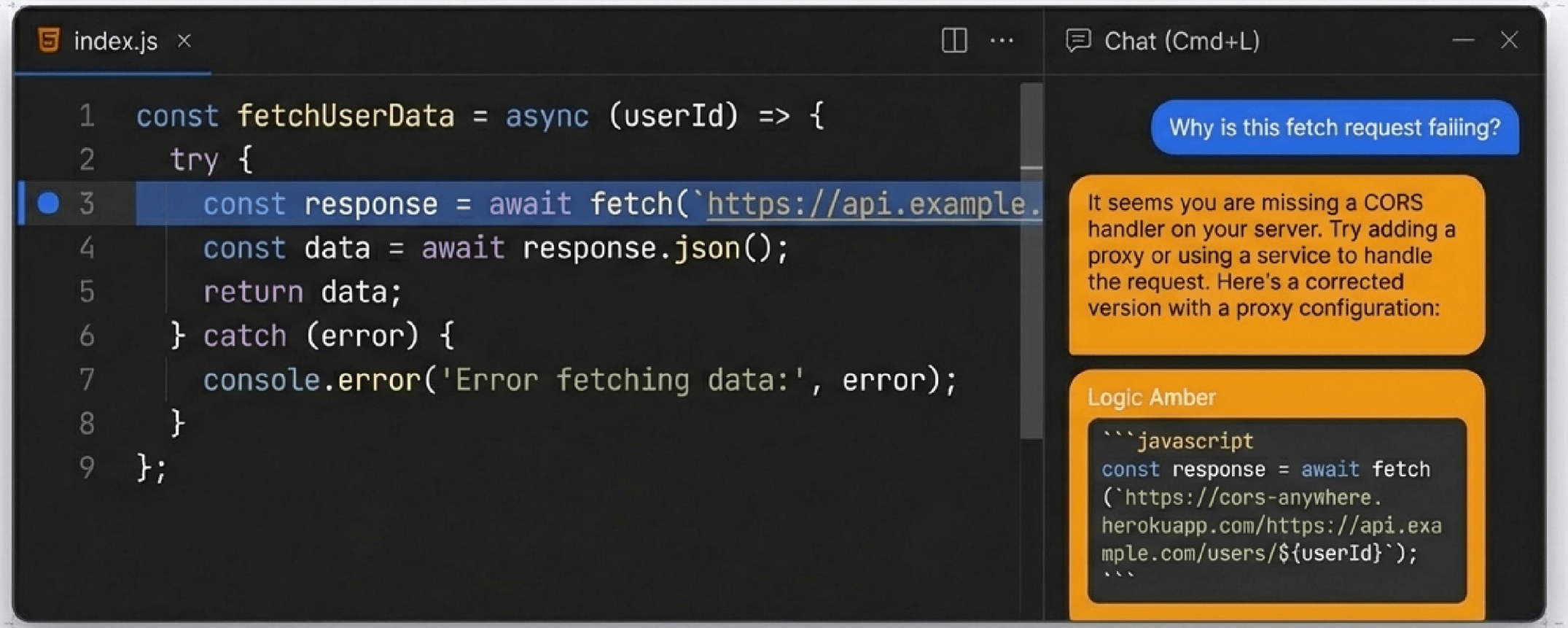
Generate new code or edit existing lines in-place using natural language. No need to leave the keyboard.



NotebookLM

Cmd+L: Your Technical Sidekick

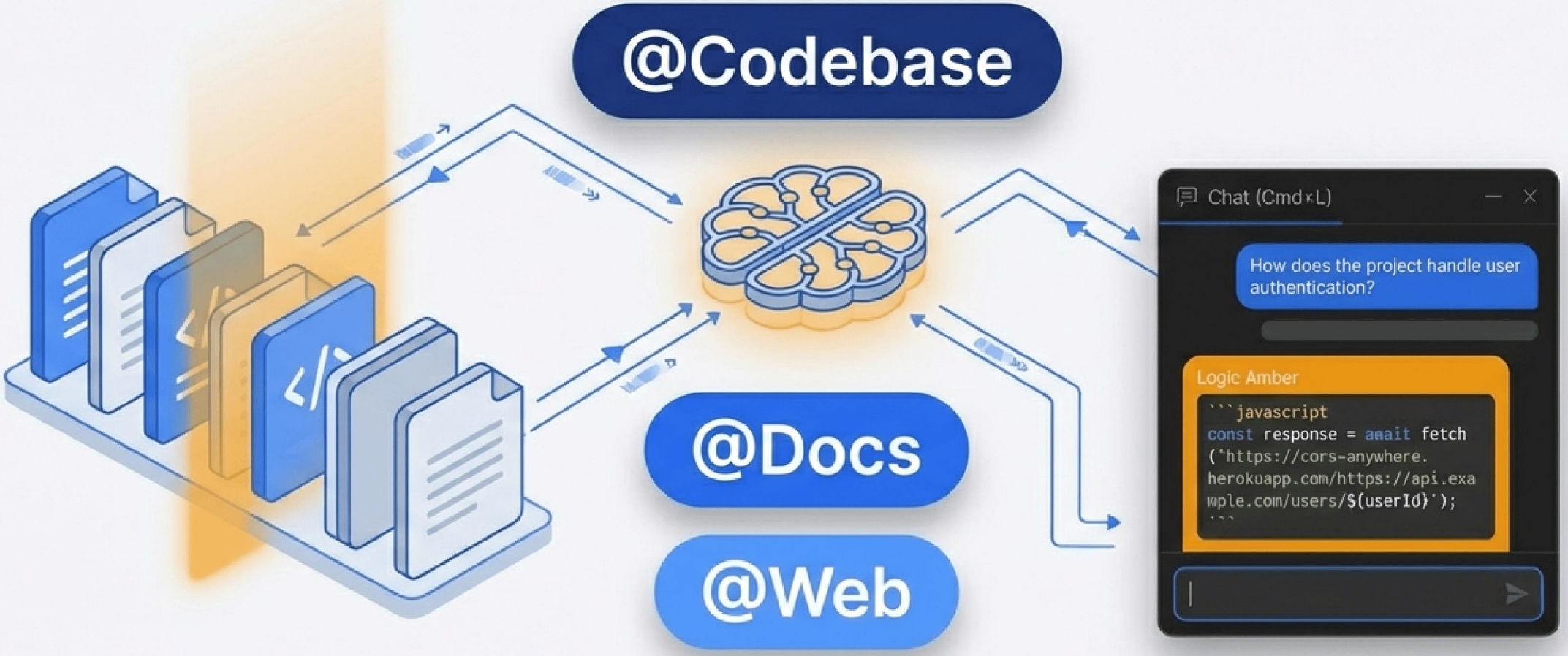
A dedicated chat interface for debugging, Q&A, and understanding complex logic.



NotebookLM

The Killer Feature: Codebase Indexing

Cursor computes embeddings for your local files to understand the entire context of your project.



NotebookLM

Iteration: "Make it Elegant"

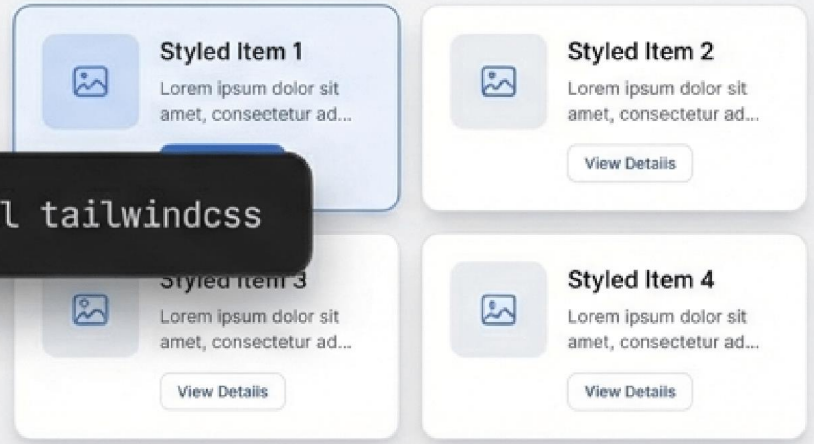
Refactoring and styling with natural language prompts.

Raw HTML

- Basic Item 1
- Basic Item 2
- Basic Item 3
- Basic Item 4

```
> Run Terminal Command: npm install tailwindcss
```

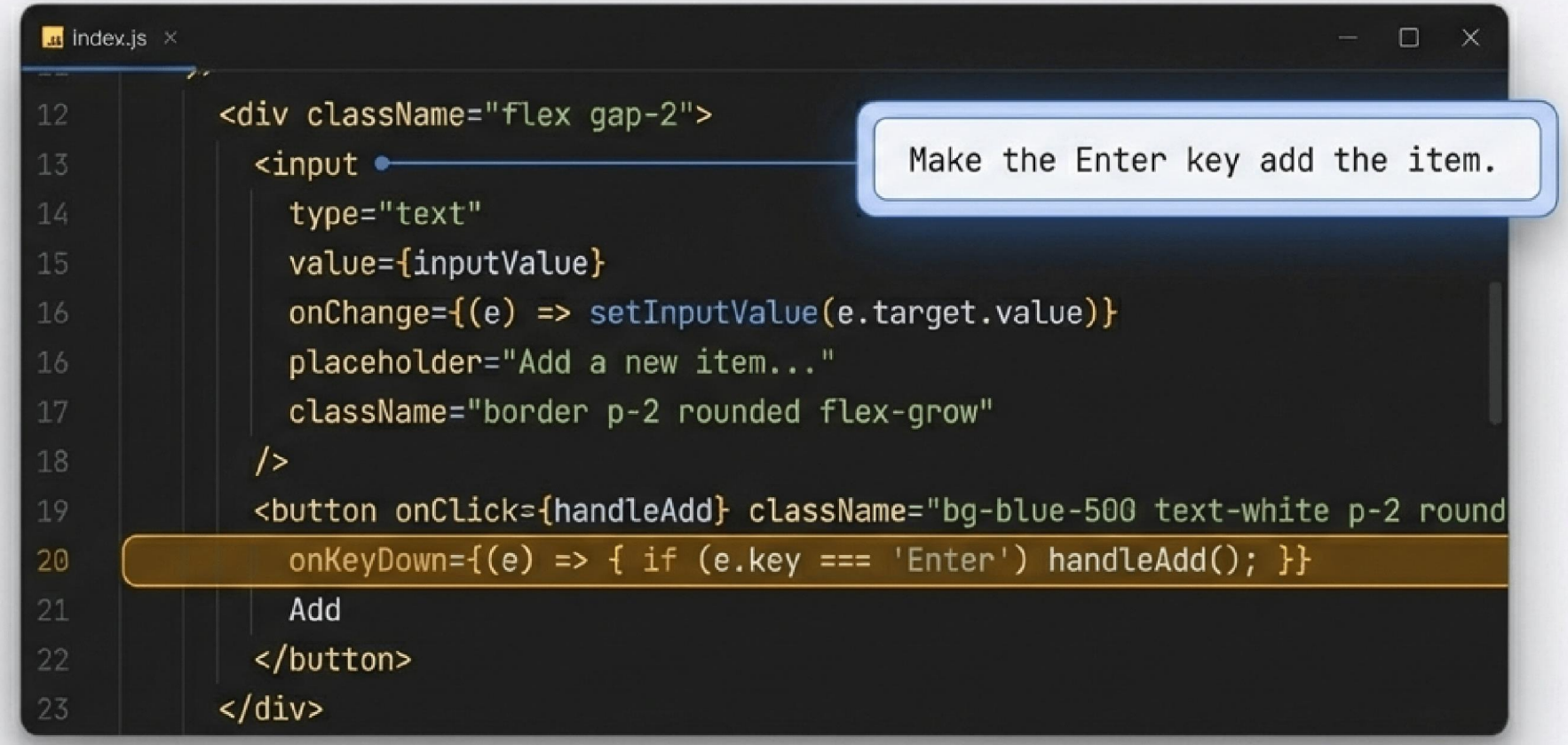
With Tailwind CSS



NotebookLM

Debugging with Natural Language

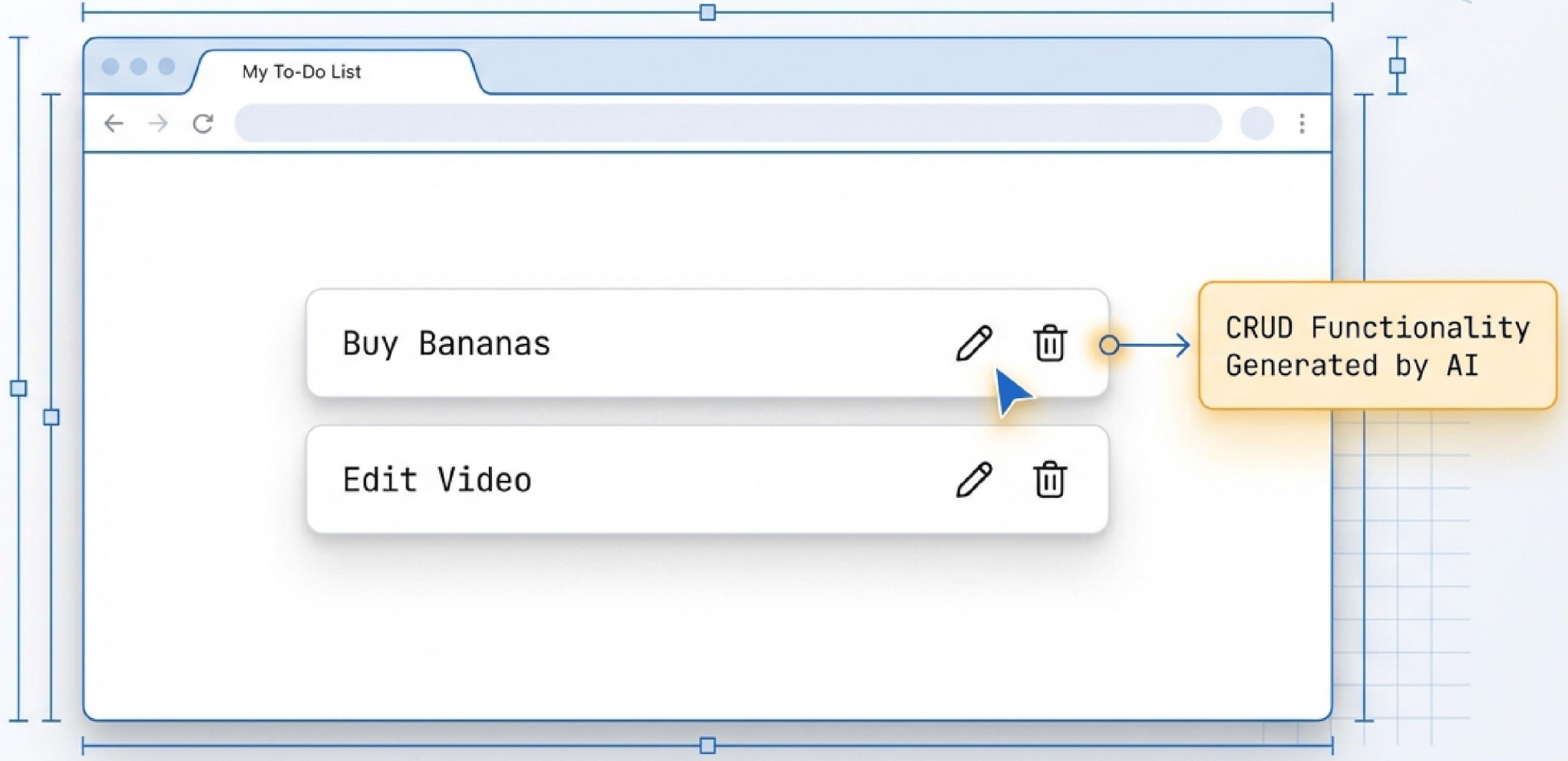
Fixing logic errors by describing the desired outcome, not the syntax.



NotebookLM

Extending Functionality

The AI understands the connection between UI buttons and backend logic.



NotebookLM

Models & Modes

Switch between top-tier models and enable Agent mode for autonomous codebase changes.



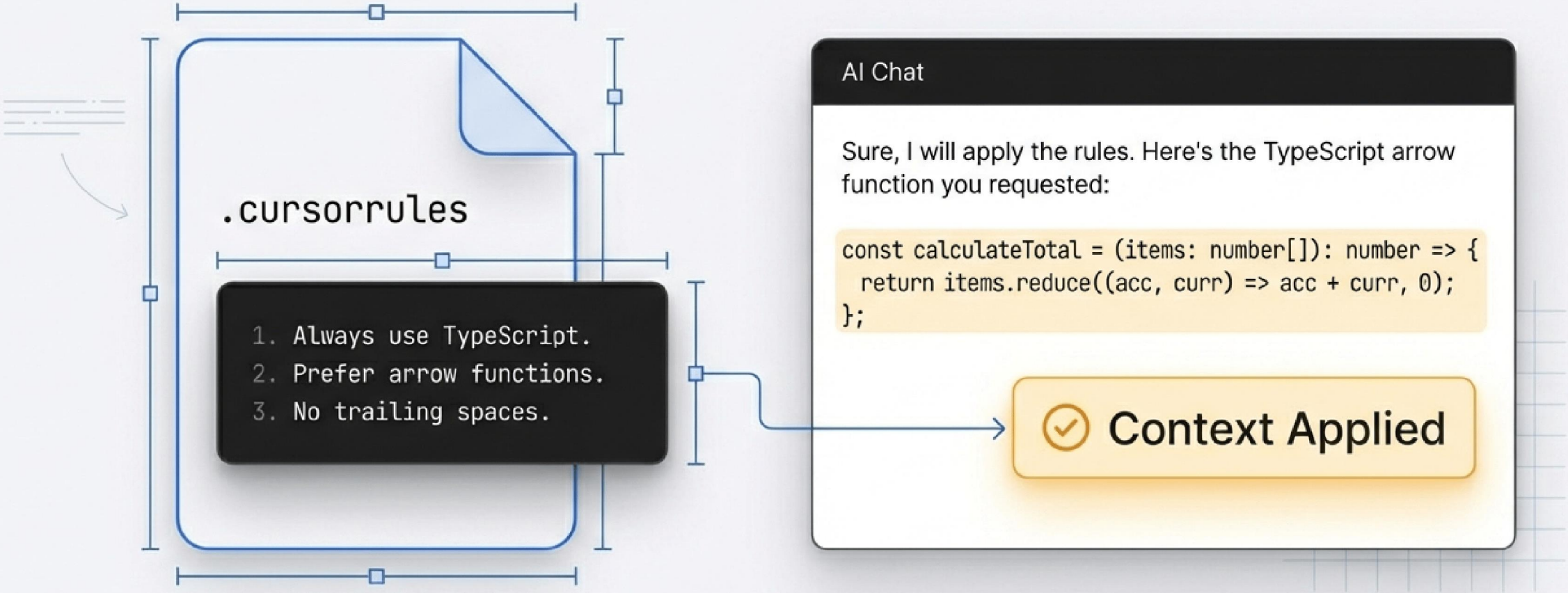
Selected Model: Claude 3.5 Sonnet.

Agent Mode: Active. Autonomous changes enabled.

NotebookLM

Power User: Rules & Context

Enforce coding standards and behaviors with `.cursorrules` configuration.



NotebookLM

Best practices

- For simple changes you don’t have to be too thoughtful about prompting
- For more complex tasks, you’re going to become a product manager
    - Carefully crafted specs doc

Best practices

- **Goal**
  - What is the purpose of the change
- **Definitions**
  - What prereqs does the LLM need to know about the problem
- **Plan**
  - High-level implementation breakdown
- **Source files being changed**
  - What parts of the codebase are relevant and why
- **Test cases**
  - How will testing be done
- **Edge cases**
  - What special cases need to be accounted for
- **Out-of-scope**
  - What should *not* be changed
- **Extensions**
  - What changes will be relevant later so the LLM can future-proof its design and not take shortcuts

Best practices

- Optimize your codebase so that a human and an agent could understand what's going on
- Much of LLM confusion comes from trying to finish a task with a messy repo as context
- Provide optimal context for LLMs by describing
  - Repo orientation
  - File structure
  - Setup and environment
  - Best practices
  - Code style
  - Access patterns
  - APIs and contracts
  - *All of this should be thoroughly documented*
  - **Tip:** a monorepo design in your repo is highly encouraged

Best practices

- Help LLM navigate your codebase with agent configurations
  - claude.md
    - *`CLAUDE.md` is a special file that Claude automatically pulls into context when starting a conversation. This makes it an ideal place for documenting: common bash commands, core files and utility functions, code style guidelines, testing instructions.*
  - cursorrules
  - AGENTS.md
    - Open format
  - llms.txt
    - Provide that navigation guidance for LLMs scraping the web
  - **Note:** The agents won't always adhere to these descriptions/directives. They are intended as guidance.

Samples

```
# Sample AGENTS.md file

## Dev environment tips
- Use `pnpm dlx turbo run where <project_name>` to jump to a package instead of scanning with `ls`.
- Run `pnpm install --filter <project_name>` to add the package to your workspace so Vite, ESLint, and TypeScript can see it.
- Use `pnpm create vite@latest <project_name> -- --template react-ts` to spin up a new React + Vite package with TypeScript checks ready.
- Check the name field inside each package's package.json to confirm the right name—skip the top-level one.

## Testing instructions
- Find the CI plan in the .github/workflows folder.
- Run `pnpm turbo run test --filter <project_name>` to run every check defined for that package.
- From the package root you can just call `pnpm test`. The commit should pass all tests before you merge.
- To focus on one step, add the Vitest pattern: `pnpm vitest run -t "<test name>"`.
- Fix any test or type errors until the whole suite is green.
- After moving files or changing imports, run `pnpm lint --filter <project_name>` to be sure ESLint and TypeScript rules still pass.
- Add or update tests for the code you change, even if nobody asked.

## PR instructions
- Title format: [<project_name>] <Title>
- Always run `pnpm lint` and `pnpm test` before committing.
```

Agenda

1. Overview of the AI tooling landscape
2. Synchronous vs. asynchronous tools
3. The 2025 coding workflow
    - When to hand-off from sync to async?
    - How to combine tools like Devin & Windsurf
4. Where are we headed?

Three Eras of AI Coding Tools

Local Development ⇒ Collaborative Cloud Agents

1

GitHub Copilot: speed up coding

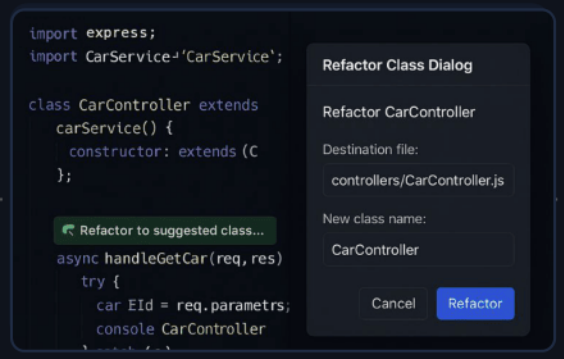
Code Completion



2

AI IDEs: single-player task completion

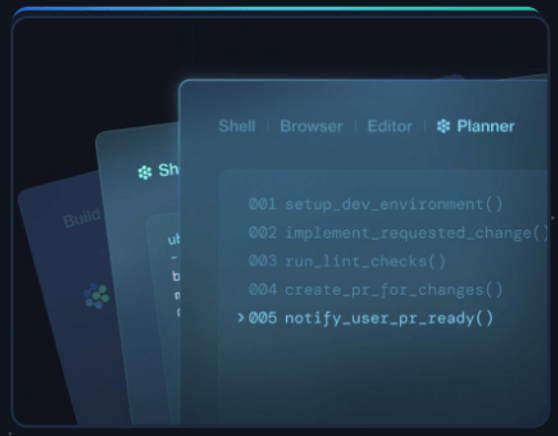
IDE Automation



3

AI agents: scale workflows in parallel

AI Software Engineer



Three Eras of AI Coding Tools

Local Development ⇒ Collaborative Cloud Agents

1 **~10% efficiency gain**

GitHub Copilot: speed up coding

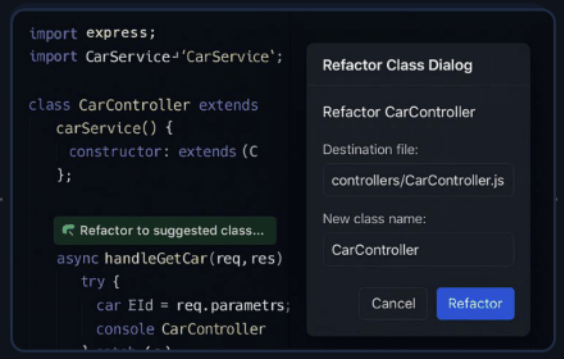
Code Completion



2 **~20% efficiency gain**

AI IDEs: single-player task completion

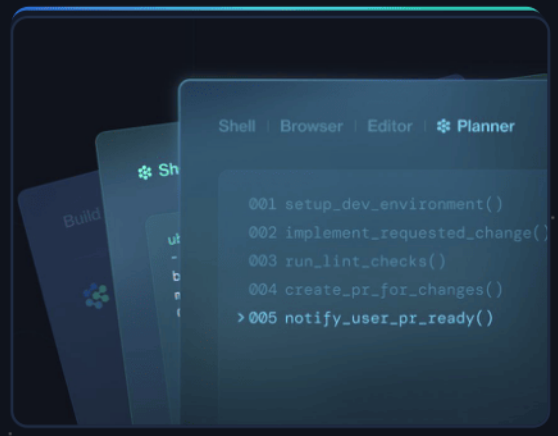
IDE Automation



3 **6-12x efficiency gain**

AI agents: scale workflows in parallel

AI Software Engineer



Cloud, Asynchronous

Local, Synchronous

GitHub Copilot · CURSOR · windsurf

Synchronous vs. Asynchronous

sync:
single-threaded, human-in-the-loop, your attention is focused on one task
=> AI agent works for 20 seconds - 1.5 minutes

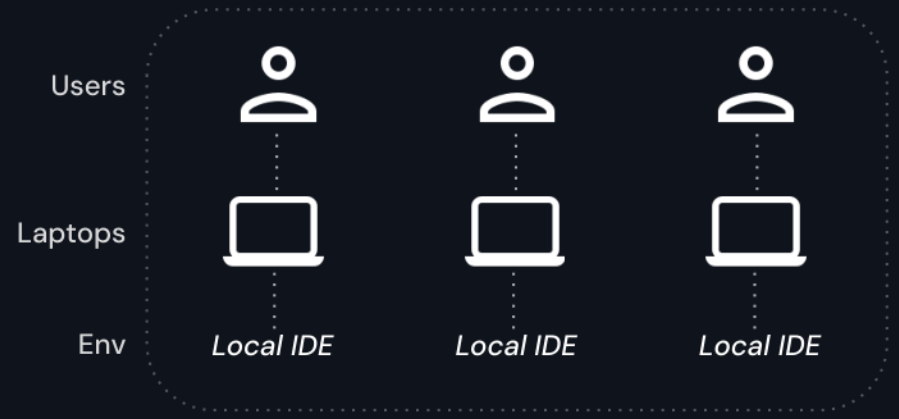
async:
multi-threaded, human delegates to AI, switches attention between multiple tasks
=> AI agent works for 10 minutes - multiple hours

	Sync	Async
Local	Windsurf CURSOR GitHub Copilot	CLAUDE CODE
Cloud	DeepWiki	Devin Codex

Cloud + Async enables 10x parallelism

Local AI IDEs

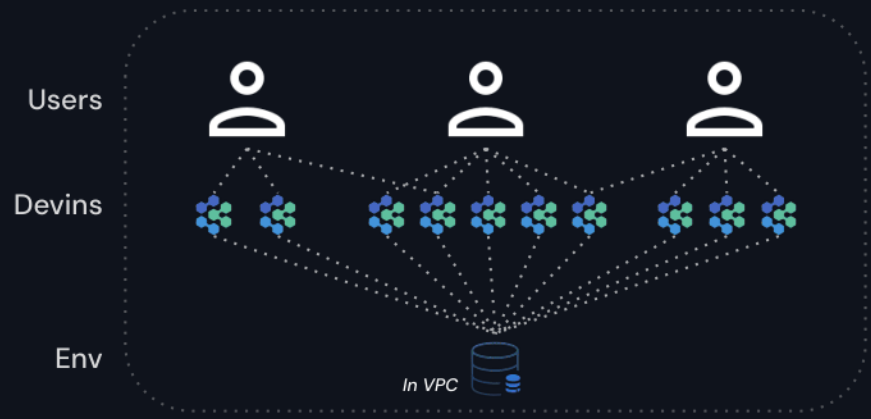
Boost individual speed.



Local • Synchronous • 1-to-1 • Isolated Knowledge

Cloud AI Agents

Unlimited Devins for parallel capacity.



Cloud • Asynchronous • 1-to-Many • Organizational Knowledge

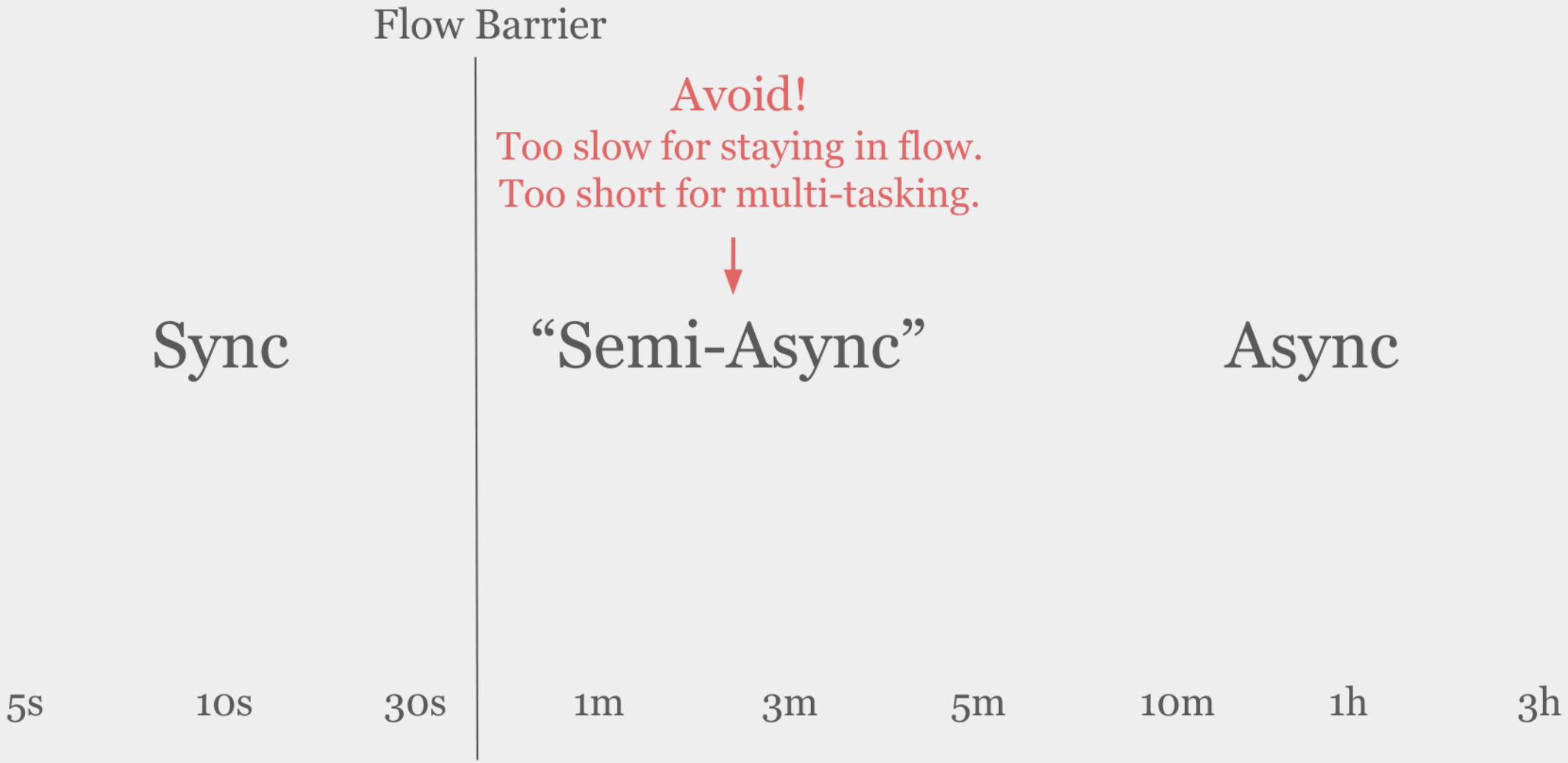
Using async agents is a hard but learnable skill

Managing async agents can unlock 10x gains…
…but most people use sync agents.

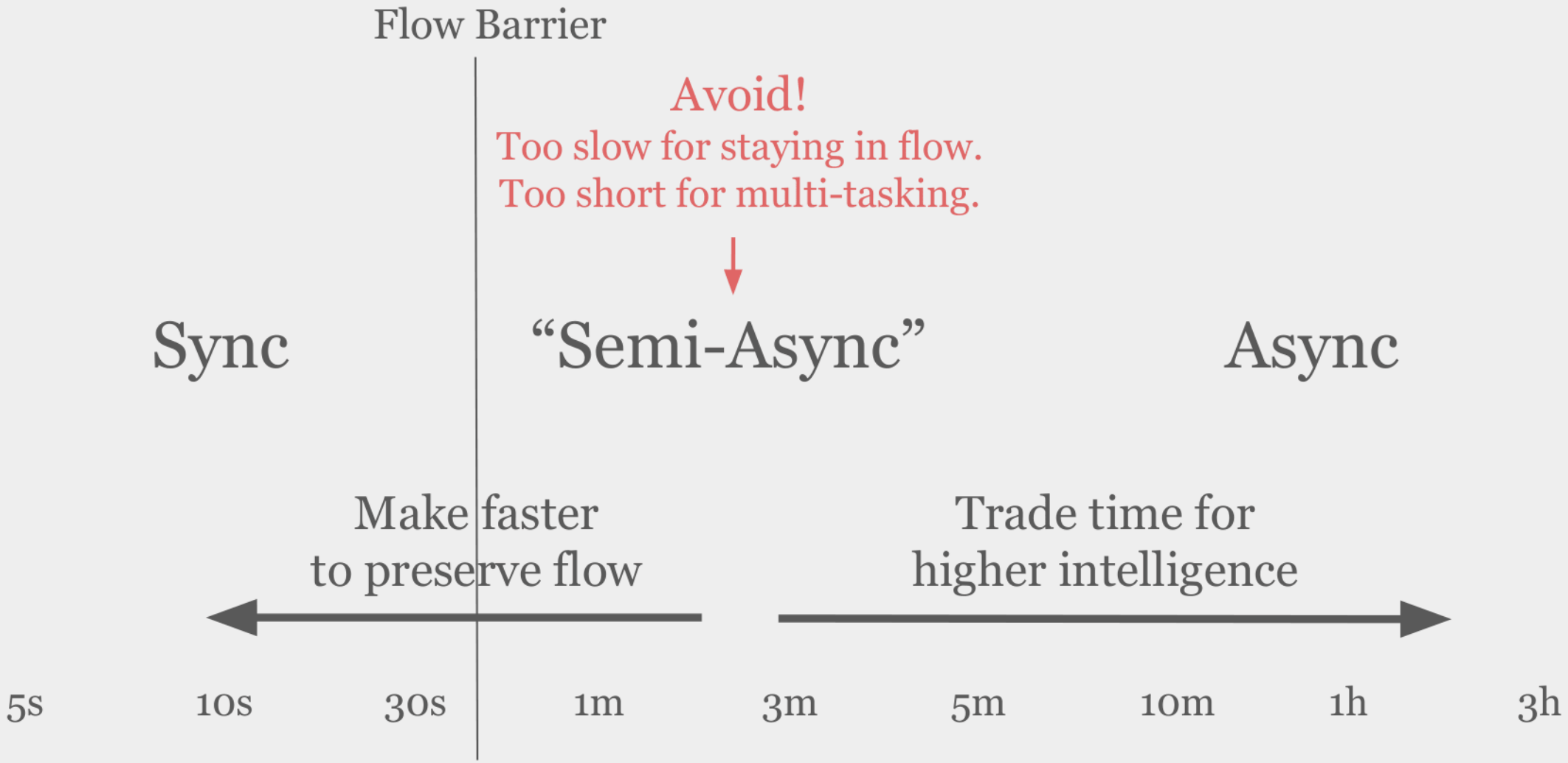
Why?

- Turns out management & delegation is a difficult skill to master – whether it's humans or agents.
- Requires ability to cycle between multiple tasks and quickly understanding new context

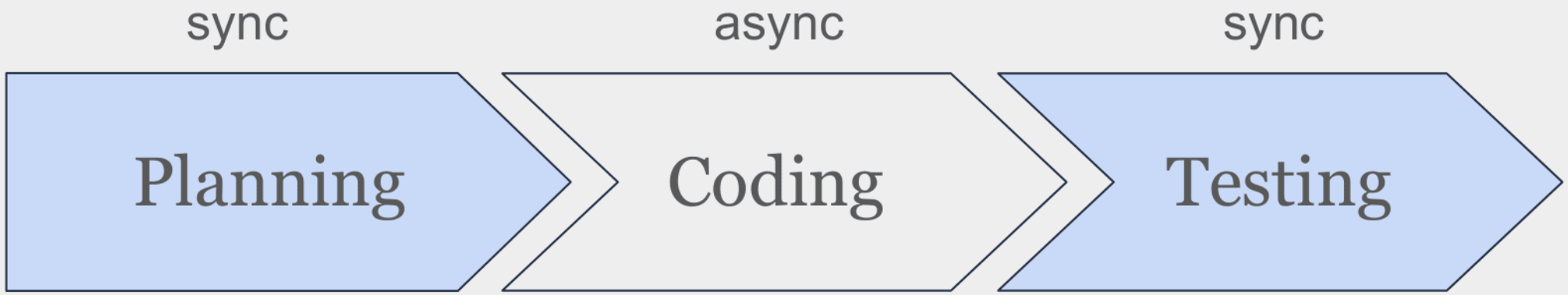
Semi-Async: The awkward middle



Semi-Async: The awkward middle




Planning
Coding
Testing

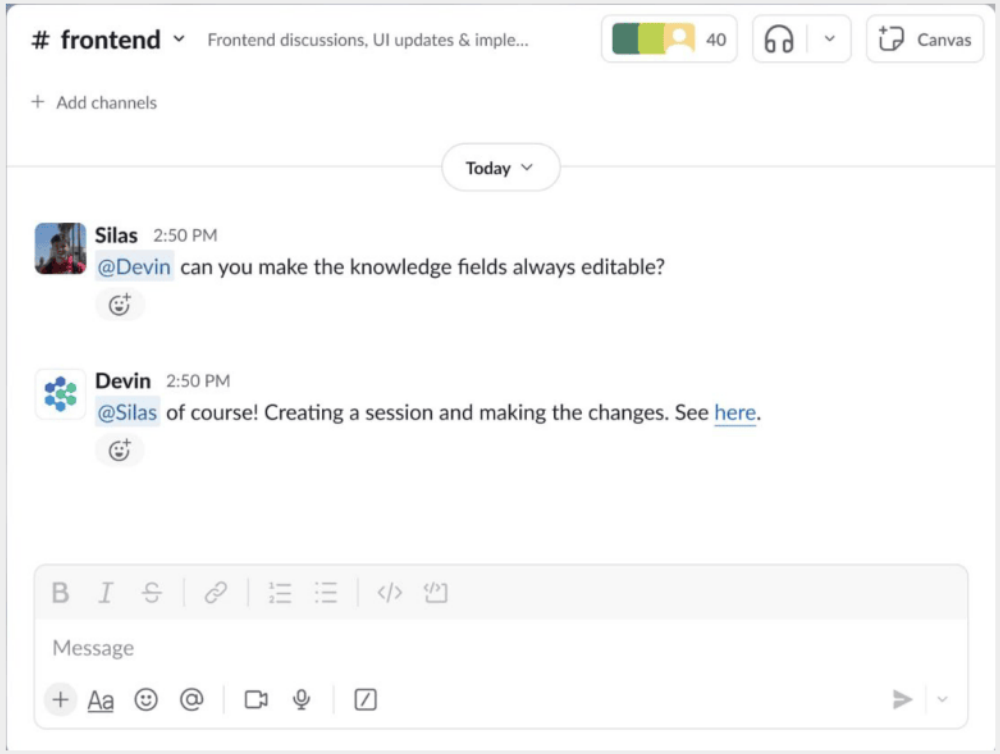

sync
async
sync
Planning
Coding
Testing

Planning with Windsurf & Devin

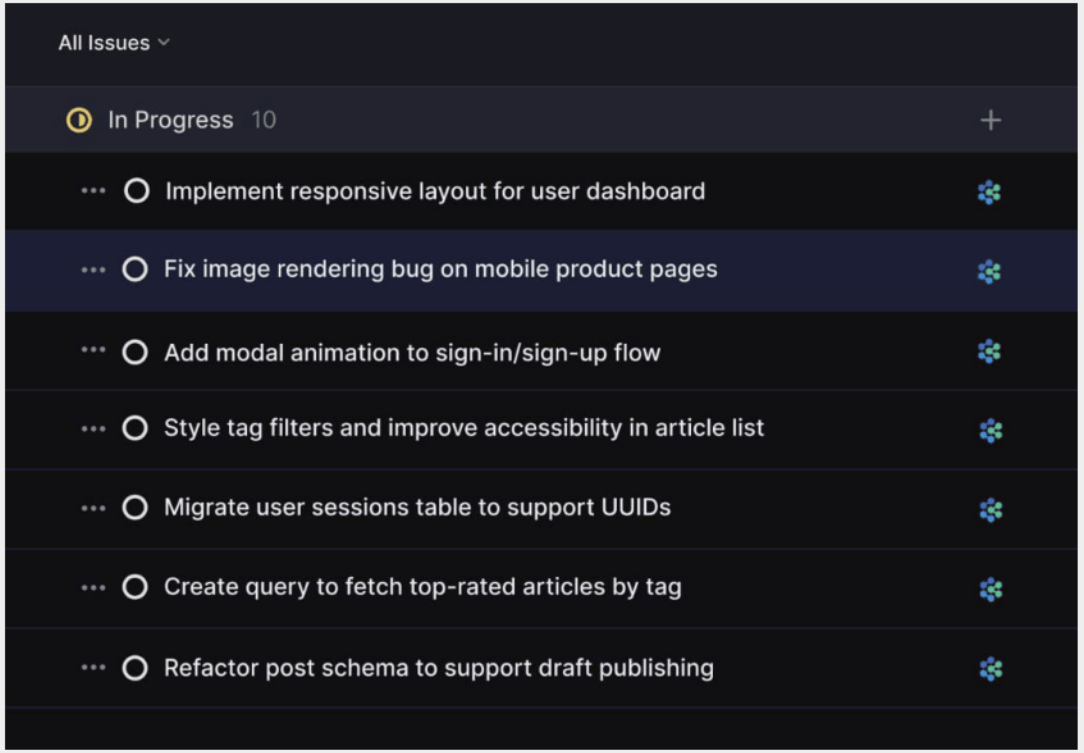
1. DeepWiki
2. Ask Devin
3. Codemaps
4. DeepWiki in Windsurf

Delegate the **coding** to the agent

slack



Linear



Testing the agent's changes

Common workflow:
1. Delegate task to Devin (async)
2. Test & refine changes in Windsurf (sync)

Future outlook:
If async agents could test autonomously, the leverage increases.
This is slowly starting to become a reality.

Where are we headed?

Today:

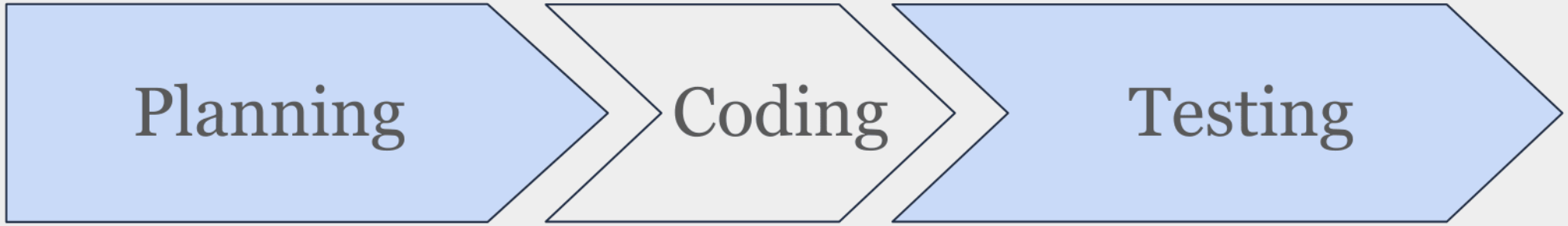


Where are we headed?

Today:



Future:



Where are we headed?

1. The human engineer as the agent manager
    a. Leveraging sync tools to solve the most difficult problems
    b. Leveraging async tools to achieve 10x leverage
2. Valuable skills for the future:
    a. Delegation & multi-threading
    b. Code reading
    c. Planning, scoping, architecting