GENOME 569

Class 3: Vibecoding for Bioinformatics workflows

Coding LLMs

What is an LLM?

A Large Language Model is trained on massive amounts of text.
It learns to predict: given everything so far, what word comes next?

It fills in what's most likely to follow:

```
The mitochondria is the powerhouse of the cell
The patient presented with elevated white blood cell count
```

It doesn't "understand" anything - it has learned statistical patterns of which words follow which. Powerful, but also the source of every mistake it makes

What is a coding LLM?

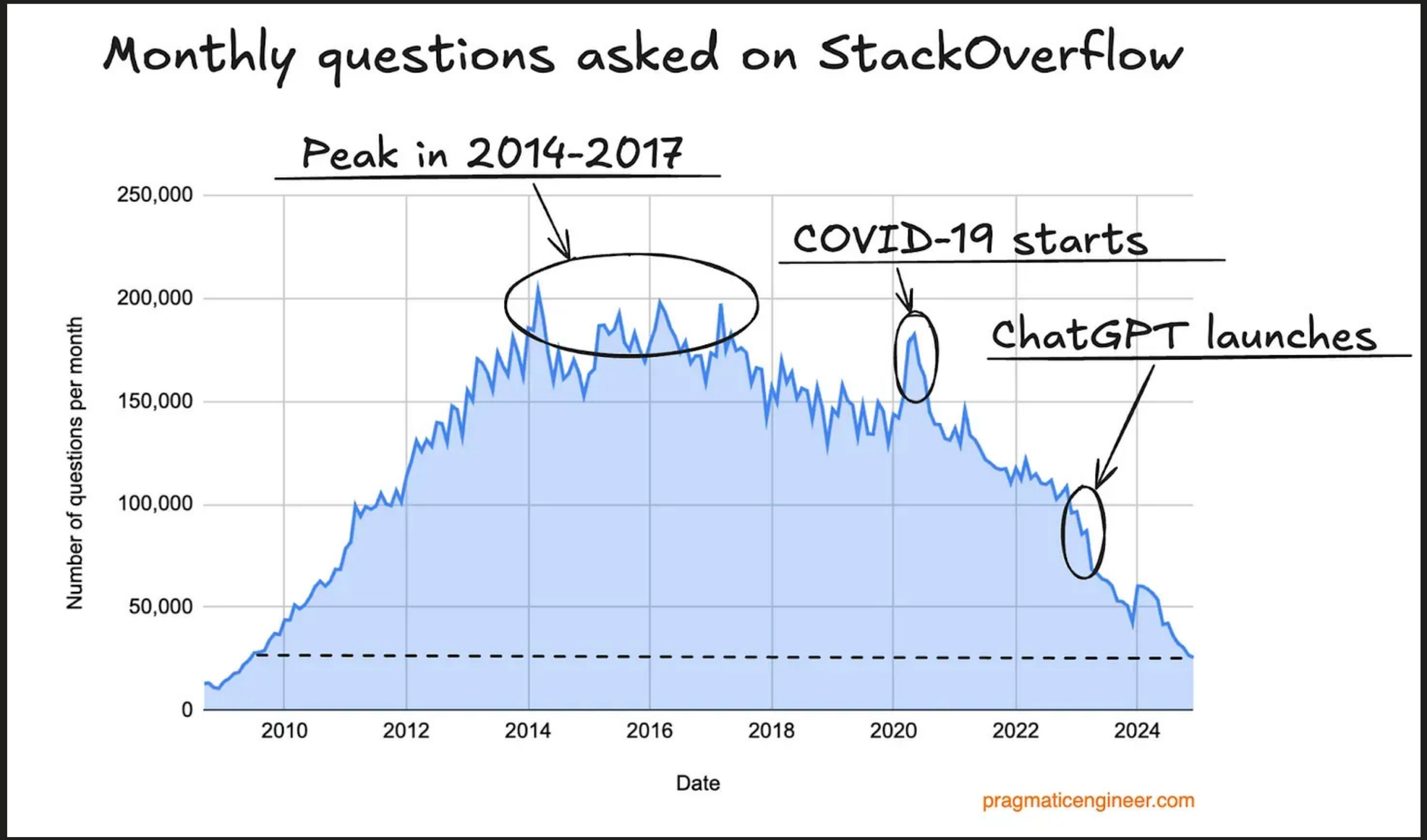
Same idea, but trained on billions of lines of code.
You write a comment or start typing, it predicts the rest:

```
# Sort BAM and count reads per gene
def count_reads(bam, bed):
    sorted = sort_bam(bam)
    return intersect(sorted, bed)
```

White = you typed Green = LLM predicted

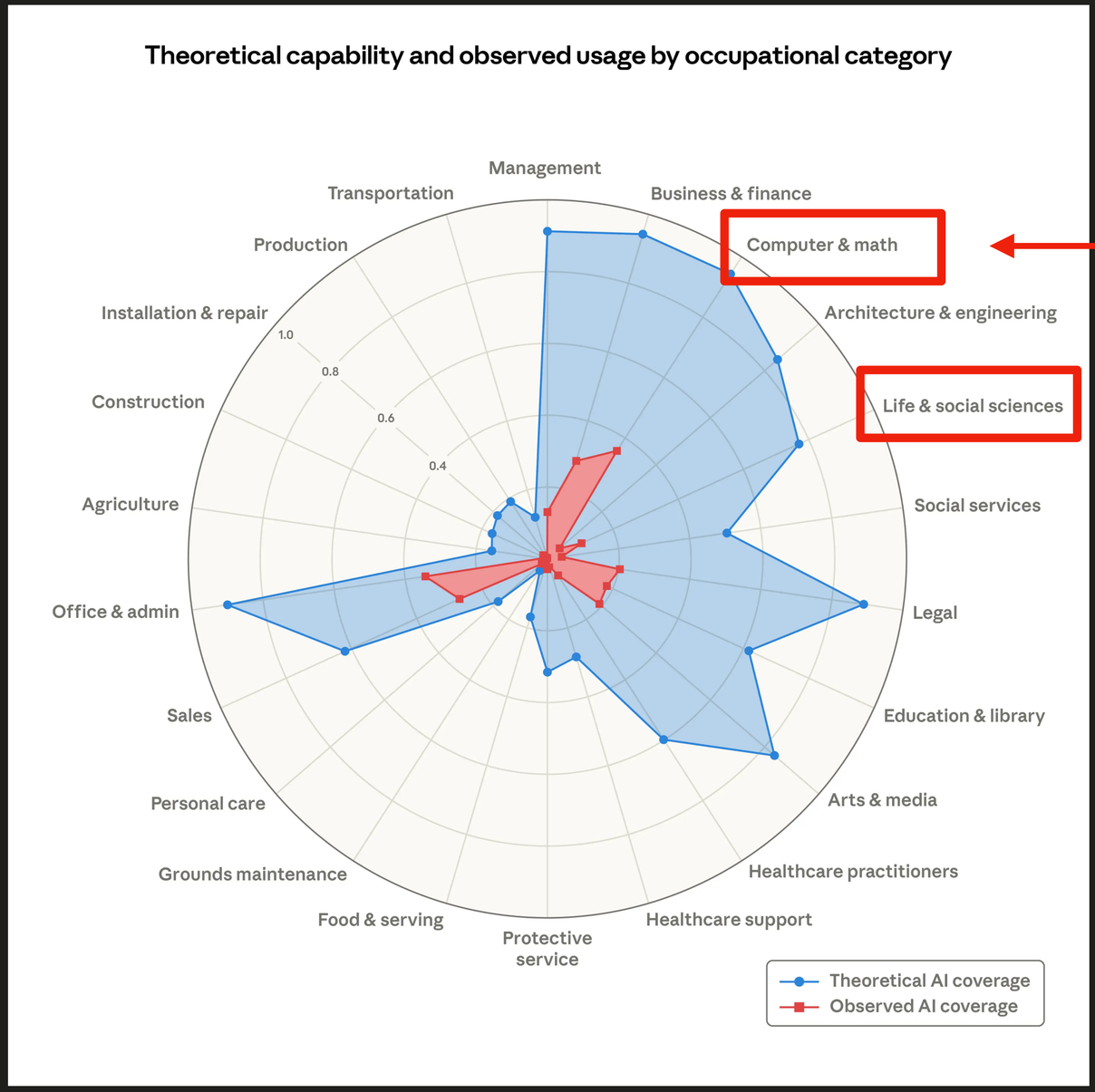
Great at: Python, R, bash, standard bioinformatics tools
Unreliable for: niche packages, novel logic, new versions
Always confident, sometimes wrong

Coding LLMs are transforming software engineering



Credit: Gergely Orosz

LLMs have potential to transform the life sciences



Credit: Anthropic

VS Code and Copilot

Three ways to use Copilot

Ghost text (Tab to accept): suggests completions inline as you type. Great for finishing functions and boilerplate

Chat (sidebar): ask questions, request explanations, generate whole functions. Like a TA who read every Stack Overflow post

Agent mode: describe a task, Copilot writes code, runs it, reads errors, fixes them in a loop. A junior programmer on autopilot

<Live demo>

Copilot has usage limits

Every interaction consumes a request from a **monthly quota**.
When it runs out, Copilot slows down until the quota resets

1 prompt in chat, planning, or agent mode = 1 request

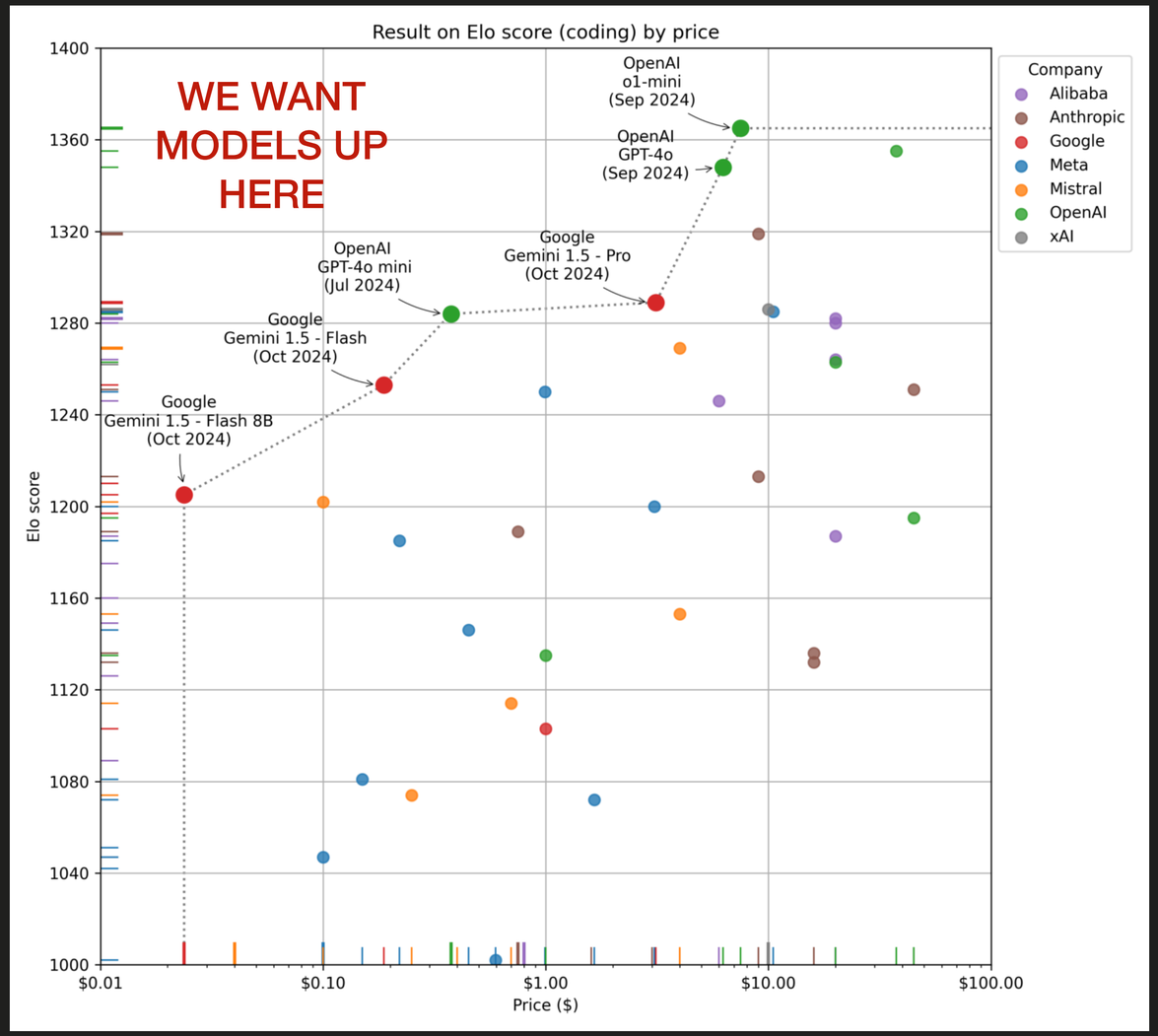
Check your quota: click the Copilot icon in the VS Code status bar

Be thoughtful: a good prompt the first time saves you requests
and gets better results

Consider a cheaper model!

Coding LLMs vary in capability and cost

Benchmark (ELO Score)



Price ($)

Credit: Towards Data Science

Try it: Copilot as tutor

Paste this STAR command into Copilot Chat and ask what each argument does:

```
STAR \
  --runThreadN 8 \
  --genomeDir $INDEX \
  --genomeLoad LoadAndKeep \
  --readFilesIn $INPUT/$FILE \
  --readFilesCommand zcat \
  --outSAMtype BAM Unsorted \
  --outSAMstrandField intronMotif
```

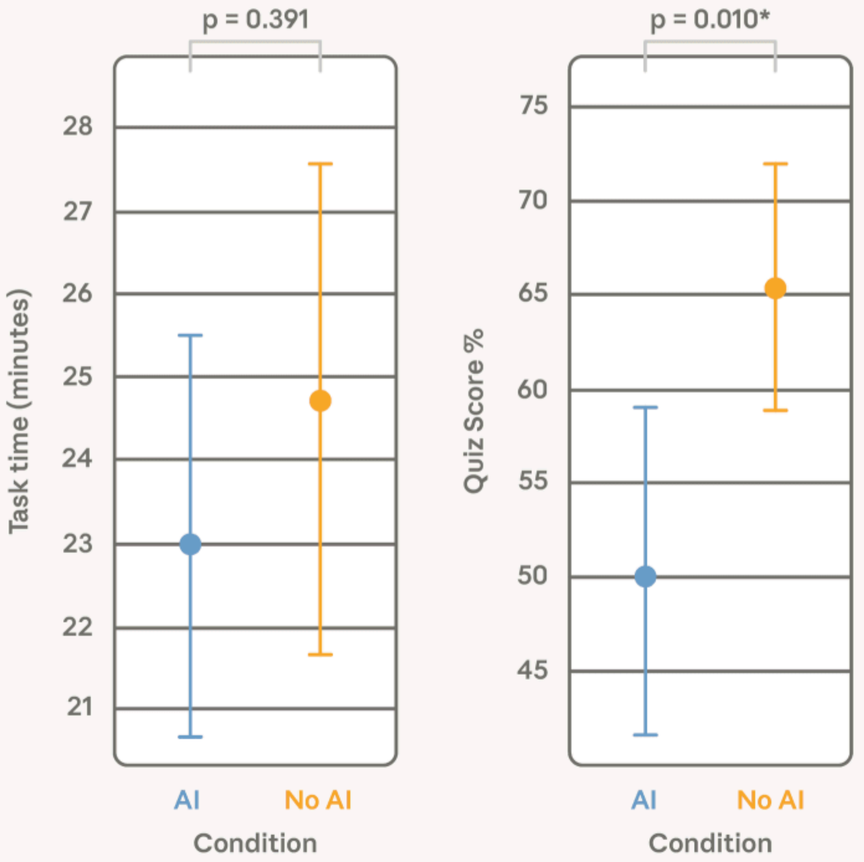
Follow-up:

- Why LoadAndKeep instead of LoadAndRemove?
- What would I change to output a sorted BAM?
- Check Copilot's answers against the STAR manual!

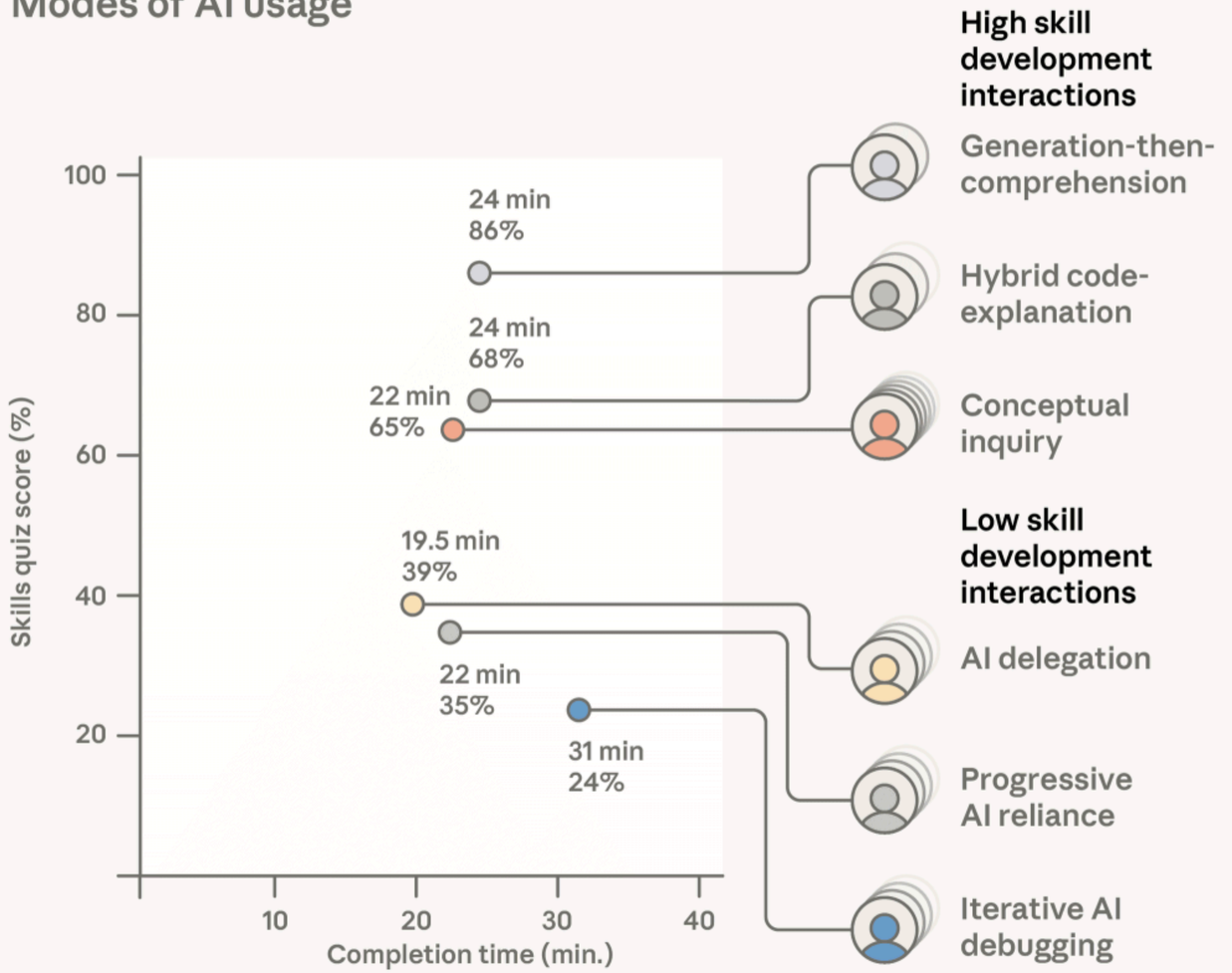
Pitfalls and opportunities

How AI assistance impacts coding speed and skill formation

AI assistance: treatment effect on coding speed and knowledge score



Modes of AI usage



Anthropic

Use AI as tutor, not ghostwriter

Ask it to **explain unfamiliar code** in a lab pipeline.
"What does this awk command do?" is a great use case

Ask it to **generate commented code**, then study the comments

Ask it to review YOUR code and **explain what you got wrong** or **suggest improvements**.
This is one of the most underused features

The brain rot test

If you can't explain every line to a labmate, you don't understand it well enough to have it in your project

If you always accept completions without reading them, you'll build a codebase you can't debug or defend at your thesis committee

The goal: use AI to go **faster on things you understand**, and to **learn things you don't**. Never to avoid understanding.

Common failure modes

Hallucinated APIs: confidently calls functions that don't exist

Doesn’t understand conventions: mixing 0-based (BED, Python) and 1-based (SAM, R) indexing. LLMs often get this wrong

Plausible but wrong stats: clean-looking code that uses the wrong test, wrong correction, or wrong assumption

Try it: spot the coordinate bug

Ask Copilot: Write a script that looks up a gene in a BED file
and uses samtools view to extract reads at that locus

```
I   11953512   11961984   WBGene00002004   255   -
```

BED is 0-based. samtools expects 1-based: I:11953513-11961984
Did Copilot add 1 to the start? Many LLMs don't!

This is a silent failure: the code runs, the output looks fine,
but the region is off by one base

How to write good prompts

Load context into your prompt

Tell it: language, framework, input format, expected output

Specify constraints: "use samtools and bedtools",
"input is a BAM from STAR", "output in 6-column BED"

Include a small example of your input data when possible -
this dramatically reduces hallucination

Try it: bad vs good prompts

Try this prompt:

```
Write a script to count reads per gene
```

Now try this one:

```
Write a bash script that takes a sorted BAM
and a 6-column BED as arguments, uses bedtools
intersect -wa -c to count reads per gene.
Output: chrom, start, stop, gene, strand, count
```

Decompose, constrain, iterate

Break complex tasks into **small, testable steps**.
Don't ask for an entire pipeline - ask for one step at a time

The first output is a draft. Paste error messages back,
ask "what edge cases does this miss?", ask it to refactor

For agents: **define "done" before you start**.
Agents with vague goals wander and produce bloated code. **Use plan mode** to come up with a strategy together!

Try it: decompose a script

Don't ask for this all at once: "Write a shell script that aligns, sorts, and counts reads per gene"

Instead, one function at a time:

```
Prompt 1: bash function align_reads: takes a
.fastq, runs STAR, outputs unsorted BAM

Prompt 2: function sort_bam: samtools sort

Prompt 3: function count_genes: bedtools
intersect -wa -c with genes.bed
```

Test each function before moving on!
Then combine into a loop. Compare to all-at-once

Verifying LLM output

The verification stack

Read the code line by line. Non-negotiable for anything going into a publication or your project repo

Write unit tests with known inputs and expected outputs. Or ask the LLM to write the tests, then verify those

Run on a small test dataset where you can manually check. Compare against established tools or published benchmarks

What is a unit test?

Runs your code with known input, checks the output matches.
Remember the coordinate bug? A test would catch it:

```
def bed_to_region(chrom, start, end):
    return f"{{chrom}}:{{start+1}}-{{end}}"

def test_bed_to_region():
    assert bed_to_region(
        "I", 11953512, 11961984
    ) == "I:11953513-11961984"
```

Think of unit tests as **positive controls**, and **negative controls** for your pipeline

Run with: pytest test_convert.py
In R, same idea: testthat with test_that() and expect_equal()
If the +1 breaks later, the test catches it instantly

Try it: ask for tests

Prompt without tests:

```
Write a Python function that converts BED
coordinates to a samtools region string
```

Same prompt, but ask for tests:

```
Write a Python function that converts BED
coords to samtools region. BED is 0-based,
samtools is 1-based. Include pytest tests
using: I, 11953512, 11961984
```

Try both. Which gets the coordinate conversion right?

Asking for tests forces the LLM to think through expected behavior - and you get tests for free

Silent failures in bioinformatics

The **most dangerous bugs**: code runs without errors but produces wrong results. LLMs are especially prone to these

Always run a positive control end-to-end. Version control everything. Note when code was AI-generated

Discussion about Phase I

Any problems?

What packages did CoPilot use?

What did you have to do “the old fashioned way?”

THE END

For now