Code generation with agents

Agents && software engineering:
analysis, learnings, practical insights

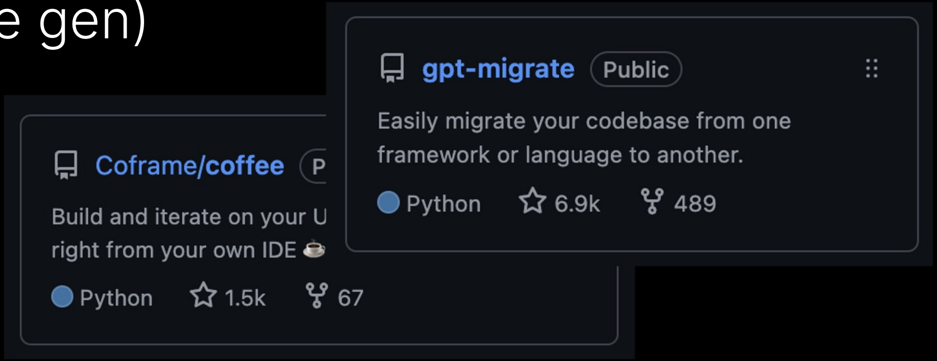
Josh Payne

Agenda

- Intro
- Brief history of AI for code generation
- Benchmarking code gen performance
- Agents x software engineering
- Ephemeral software & environment engineering

Intro

- I'm Josh 👋
- Founder of Coframe (AI for UI optimization + code gen), prev two other companies (one AI-focused)
- Created GPT-Migrate (LLM-powered codebase migration), Coffee (LLM-powered UI code gen)
- Stanford alum 🌲



gpt-migrate Public

Easily migrate your codebase from one framework or language to another.

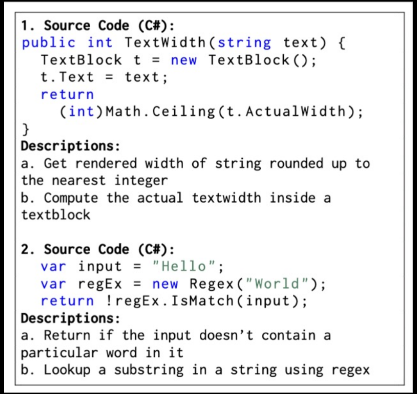
● Python ☆ 6.9k 489

Coframe/coffee P

Build and iterate on your U right from your own IDE ☕

● Python ☆ 1.5k 67

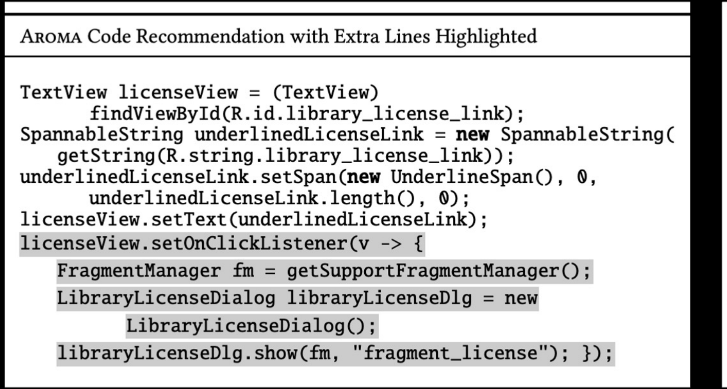
Brief History



```
1. Source Code (C#):
public int TextWidth(string text) {
  TextBlock t = new TextBlock();
  t.Text = text;
  return
    (int)Math.Ceiling(t.ActualWidth);
}
Descriptions:
a. Get rendered width of string rounded up to
the nearest integer
b. Compute the actual textwidth inside a
textblock

2. Source Code (C#):
  var input = "Hello";
  var regEx = new Regex("World");
  return !regEx.IsMatch(input);
Descriptions:
a. Return if the input doesn't contain a
particular word in it
b. Lookup a substring in a string using regex
```

CodeNN (Iyer et al., 2016)

Code summarization

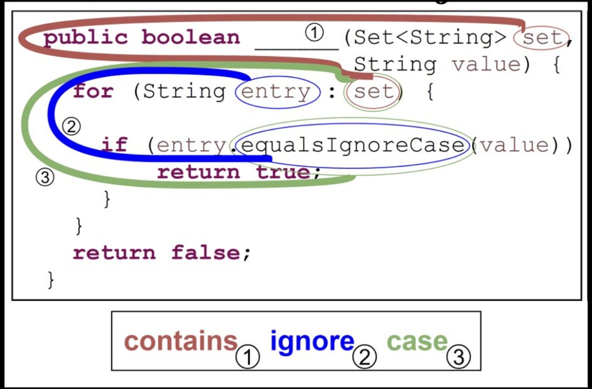


Aroma Code Recommendation with Extra Lines Highlighted

```
TextView licenseView = (TextView)
      findViewById(R.id.library_license_link);
SpannableString underlinedLicenseLink = new SpannableString(
    getString(R.string.library_license_link));
underlinedLicenseLink.setSpan(new UnderlineSpan(), 0,
      underlinedLicenseLink.length(), 0);
licenseView.setText(underlinedLicenseLink);
licenseView.setOnClickListener(v -> {
    FragmentManager fm = getSupportFragmentManager();
    LibraryLicenseDialog libraryLicenseDlg = new
          LibraryLicenseDialog();
    libraryLicenseDlg.show(fm, "fragment_license"); });
```

Aroma (Luan et al, 2019)

Code search (early copilot)



```
public boolean ______① (Set<String> set,
                          String value) {
  for (String entry : set) {
   ②
     if (entry.equalsIgnoreCase(value))
③         return true;
     }
  }
  return false;
}
```

contains① ignore② case③

Code2Seq (Alon et al., 2019)

Better code summarization

(Try it! → https://code2seq.org/)

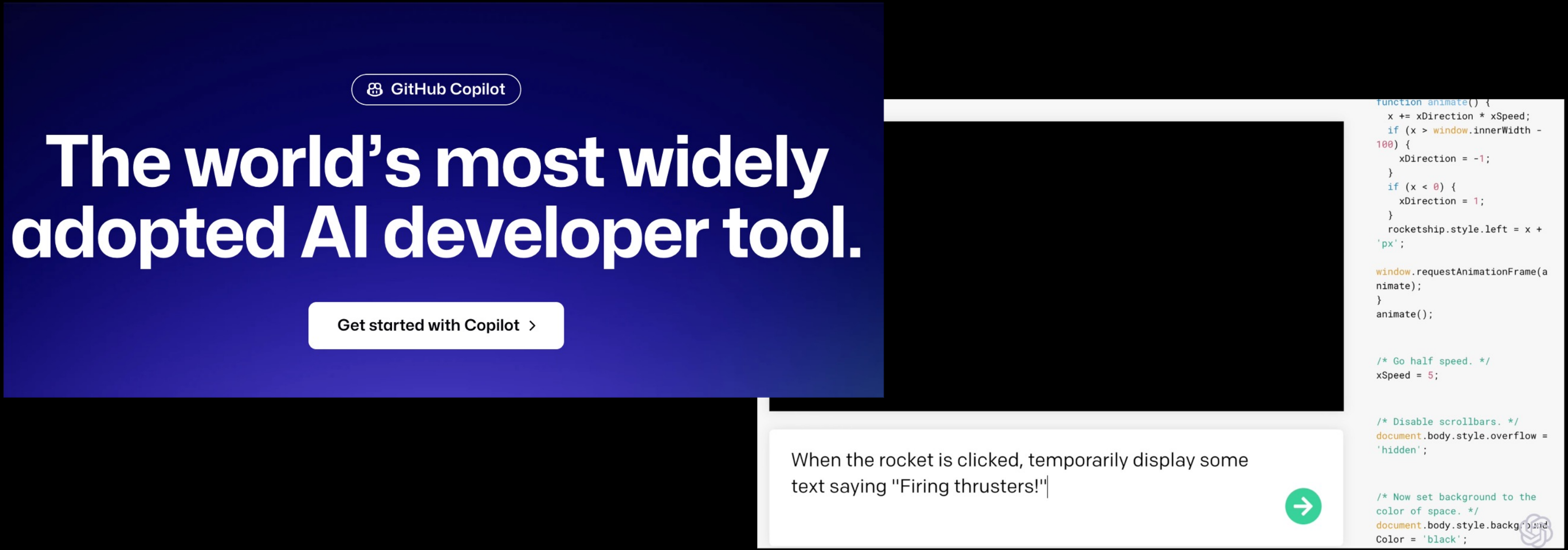
Pre-LLM era:
RNNs and search

Early applications:
GPT-3, Codex, GitHub Copilot

"Oh wow, AI can actually write code now":
GPT-3.5, GPT-4, OSS LLMs

AI x software engineering:
Agents and integrated workflows

Brief History



The world's most widely adopted AI developer tool.

```
function animate() {
  x += xDirection * xSpeed;
  if (x > window.innerWidth -
100) {
    xDirection = -1;
  }
  if (x < 0) {
    xDirection = 1;
  }
  rocketship.style.left = x +
'px';

window.requestAnimationFrame(a
nimate);
}
animate();


/* Go half speed. */
xSpeed = 5;


/* Disable scrollbars. */
document.body.style.overflow =
'hidden';


/* Now set background to the
color of space. */
document.body.style.background
Color = 'black';
```

When the rocket is clicked, temporarily display some text saying "Firing thrusters!"

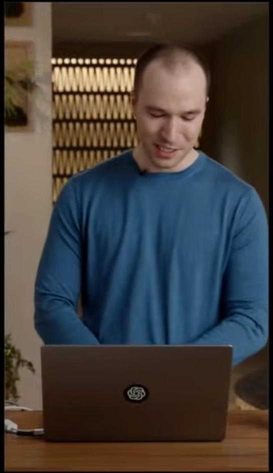
Pre-LLM era:
RNNs and search

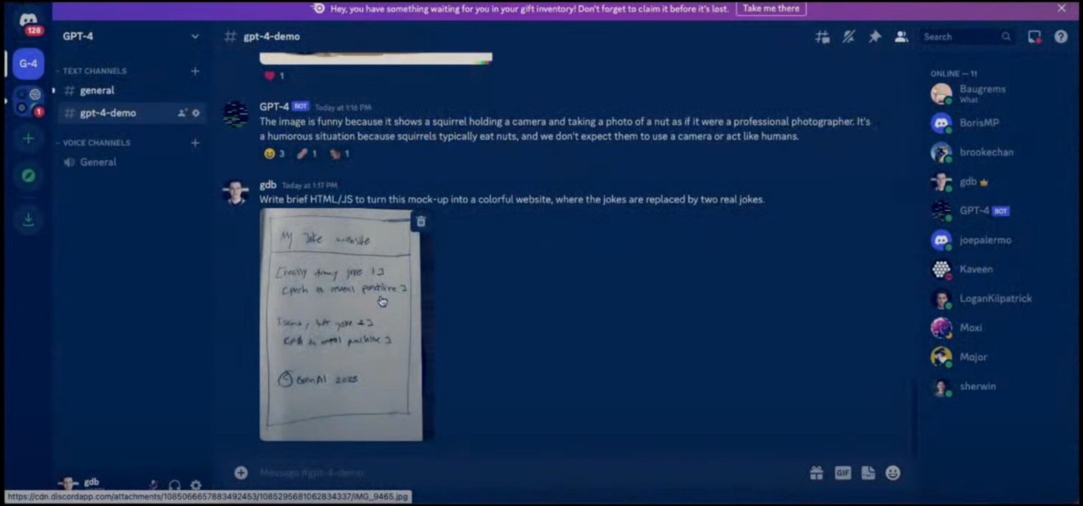
Early applications:
GPT-3, Codex, GitHub Copilot

"Oh wow, AI can actually write code now":
GPT-3.5, GPT-4, OSS LLMs

AI x software engineering:
Agents and integrated workflows

Brief History





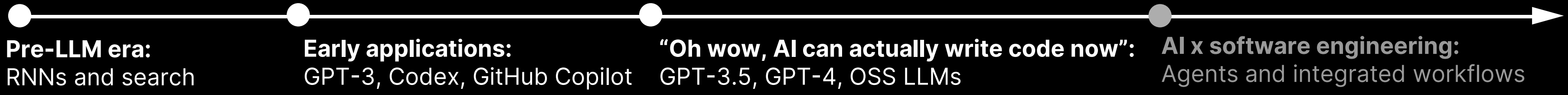
GPT-4 BOT Today at 1:16 PM

The image is funny because it shows a squirrel holding a camera and taking a photo of a nut as if it were a professional photographer. It's a humorous situation because squirrels typically eat nuts, and we don't expect them to use a camera or act like humans.

gdb Today at 1:17 PM

Write brief HTML/JS to turn this mock-up into a colorful website, where the jokes are replaced by two real jokes.

My Joke website
[really funny joke 1]
[push to reveal punchline]
[second, funny joke 2]
[push to reveal punchline]
© GenAI 2023
https://cdn.discordapp.com/attachments/1085066657883492453/1085295681062834337/IMG_9465.jpg



Pre-LLM era:
RNNs and search

Early applications:
GPT-3, Codex, GitHub Copilot

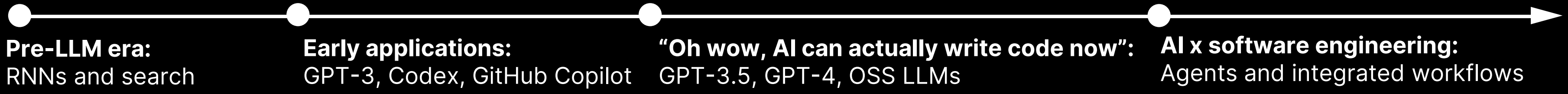
"Oh wow, AI can actually write code now":
GPT-3.5, GPT-4, OSS LLMs

AI x software engineering:
Agents and integrated workflows

Brief History

Now a production-ready category, rapidly maturing

CLAUDE CODE
CODEX
Lovable
replit
Cognition
CURSOR
FACTORY



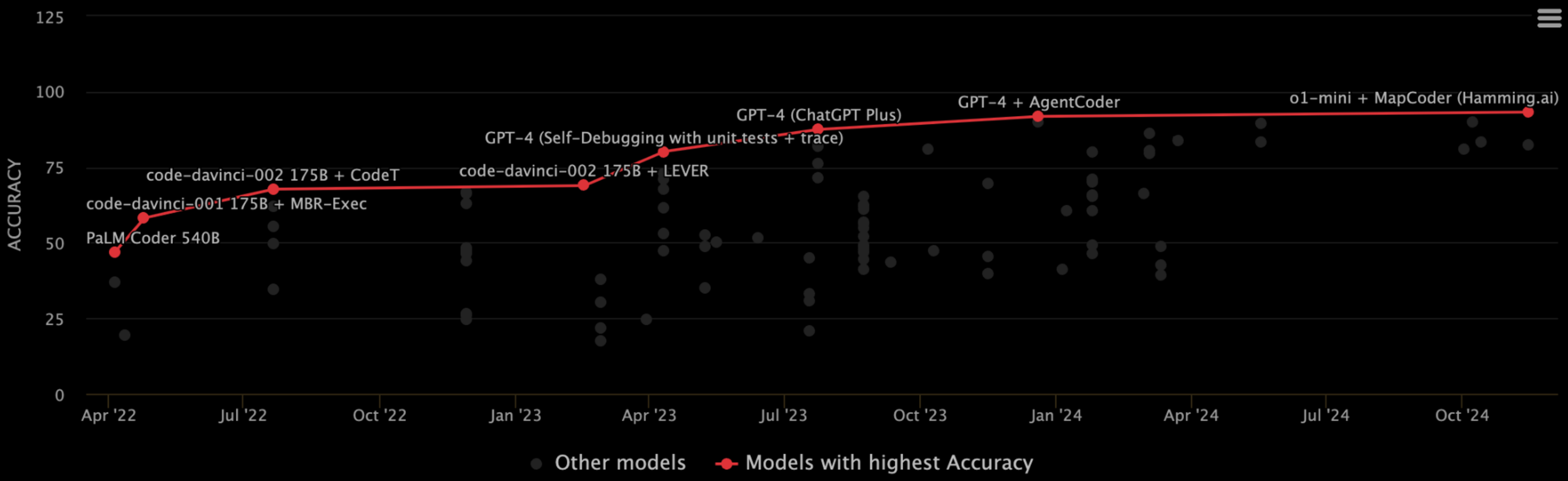
Pre-LLM era:
RNNs and search

Early applications:
GPT-3, Codex, GitHub Copilot

“Oh wow, AI can actually write code now”:
GPT-3.5, GPT-4, OSS LLMs

AI x software engineering:
Agents and integrated workflows

Benchmarking code generation



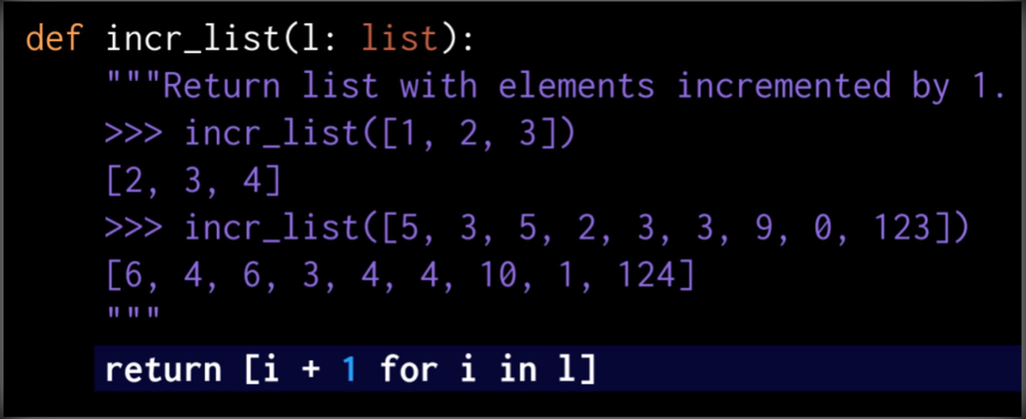
How do we measure this?

1 Academic Benchmarks 2 Competitions 3 Real-world impact

HumanEval (Chen et al., 2021) was for a long time the most widely-recognized research benchmark for code generation.

This paper also introduced **Codex**, the first major code-specific LLM.

HumanEval is 164 handwritten programming problems, each with several unit tests.



```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

The prompt provided to the model is shown with a black background, and a successful model-generated completion is shown in a blue background. To be successful, it must pass the unit tests.

How do we measure this?

1 Academic Benchmarks 2 Competitions 3 Real-world impact

Other early benchmarks:

- **MultiPL-E** is a dataset for evaluating large language models for code generation that supports 18 programming languages. It translates HumanEval problems into other languages.

- **HumanEval-X** consists of 820 high-quality human-crafted data samples, compared with HumanEval's 164.

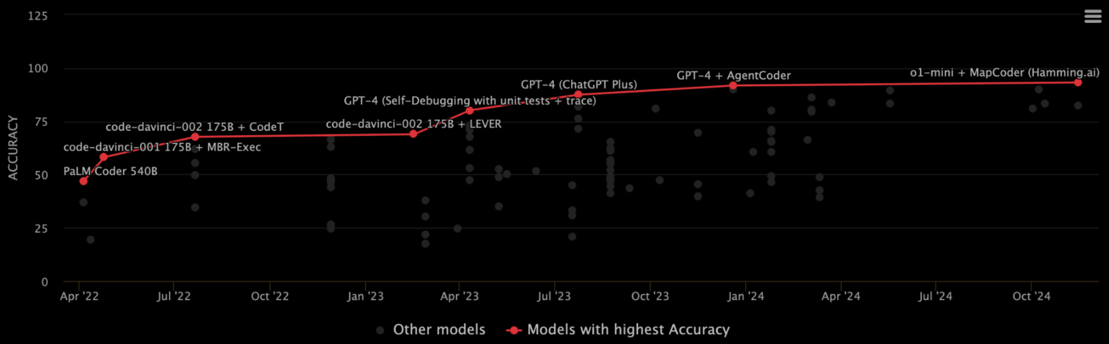
- **MBPP** (Mostly Basic Python Problems) is a dataset of 1000 crowd-sourced Python programming problems.

How do we measure this?

1 Academic Benchmarks 2 Competitions 3 Real-world impact

This benchmark is now almost fully *saturated*: trivial for powerful LLMs such as Claude Opus 4.6 or GPT-5.3-Codex.

Benchmarks are becoming saturated at an increasing rate, and it’s a real research question to come up with new interesting ones.



New Benchmarks

SWE-Bench Pro

Multi-language real-world GitHub issues

Replaces SWE-Bench Verified (Python-only)

GPT-5.3-Codex	**80.0%**
Opus 4.6	80.8%

Terminal-Bench 2.0

Terminal & CLI coding skills

Measures real terminal workflow competency

GPT-5.3-Codex	**77.2%**
Opus 4.6	65.4%

OSWorld-Verified

Multi-modal tasks on real computers

GUI interaction, file management, apps

GPT-5.3-Codex	64.7%
Opus 4.6	**72.7%**

Emerging Benchmarks

OpenRCA

Software root cause failure analysis

SWE-Bench Multilingual

Multi-language extension of SWE-Bench

Cybergym

Cybersecurity vulnerability reproduction

BioPipelineBench

Computational biology pipelines

Tau2-Bench

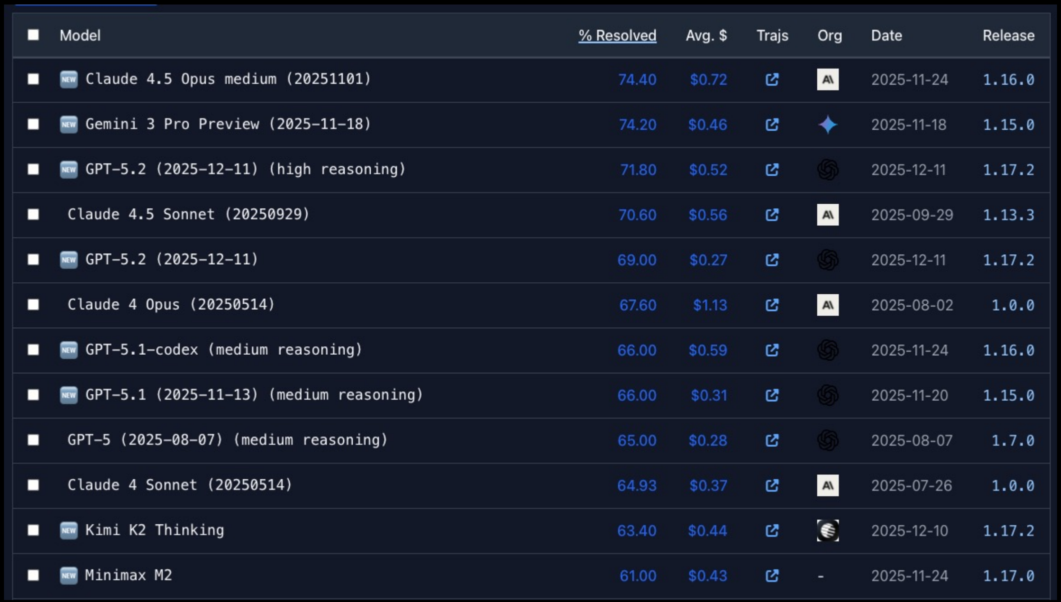
Agentic tool use evaluation

How do we measure this?

1 Academic Benchmarks 2 Competitions 3 Real-world impact

SWE-Bench Verified was the 2025 gold standard.

Now shifting to SWE-Bench Pro (multi-language, contamination-resistant).



□	Model	% Resolved	Avg. $	Trajs	Org	Date	Release
□	NEW Claude 4.5 Opus medium (20251101)	74.40	$0.72	↗	A\	2025-11-24	1.16.0
□	NEW Gemini 3 Pro Preview (2025-11-18)	74.20	$0.46	↗	✦	2025-11-18	1.15.0
□	NEW GPT-5.2 (2025-12-11) (high reasoning)	71.80	$0.52	↗	⊛	2025-12-11	1.17.2
□	Claude 4.5 Sonnet (20250929)	70.60	$0.56	↗	A\	2025-09-29	1.13.3
□	NEW GPT-5.2 (2025-12-11)	69.00	$0.27	↗	⊛	2025-12-11	1.17.2
□	Claude 4 Opus (20250514)	67.60	$1.13	↗	A\	2025-08-02	1.0.0
□	NEW GPT-5.1-codex (medium reasoning)	66.00	$0.59	↗	⊛	2025-11-24	1.16.0
□	NEW GPT-5.1 (2025-11-13) (medium reasoning)	66.00	$0.31	↗	⊛	2025-11-20	1.15.0
□	GPT-5 (2025-08-07) (medium reasoning)	65.00	$0.28	↗	⊛	2025-08-07	1.7.0
□	Claude 4 Sonnet (20250514)	64.93	$0.37	↗	A\	2025-07-26	1.0.0
□	NEW Kimi K2 Thinking	63.40	$0.44	↗	●	2025-12-10	1.17.2
□	NEW Minimax M2	61.00	$0.43	↗	-	2025-11-24	1.17.0

How do we measure this?

1 Academic Benchmarks 2 Competitions 3 Real-world impact

Why are held-out (non-published) benchmarks valuable?

How do we measure this?

1 Benchmark Tasks

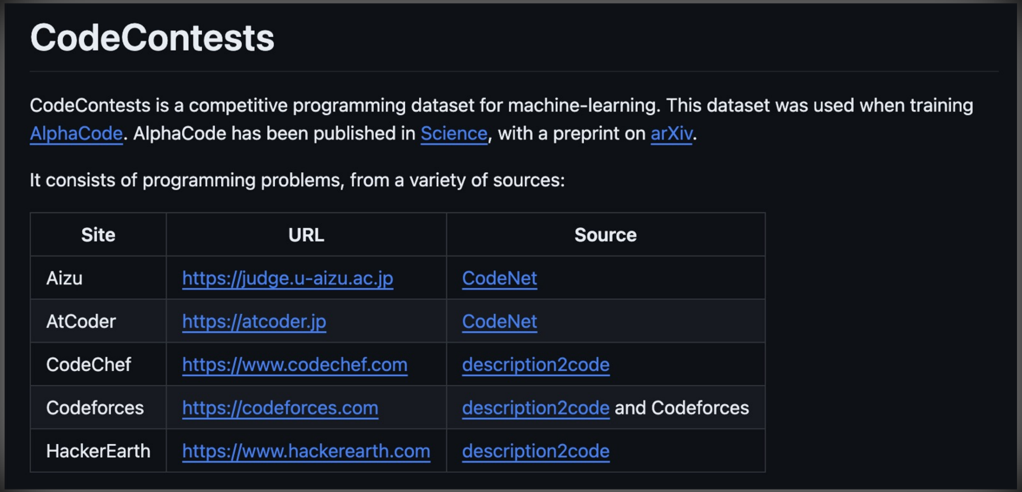
2 Competitions

3 Real-world impact

AlphaCode by DeepMind (Li et al., Dec 2022) created CodeContests, a dataset of compiled competitive programming problems.

Increasingly, datasets from real-world tasks for humans are needed as models approach human-level performance.

Other examples: the LSAT, USMLE, AlphaGeometry (IMO problems)



CodeContests

CodeContests is a competitive programming dataset for machine-learning. This dataset was used when training AlphaCode. AlphaCode has been published in Science, with a preprint on arXiv.

It consists of programming problems, from a variety of sources:

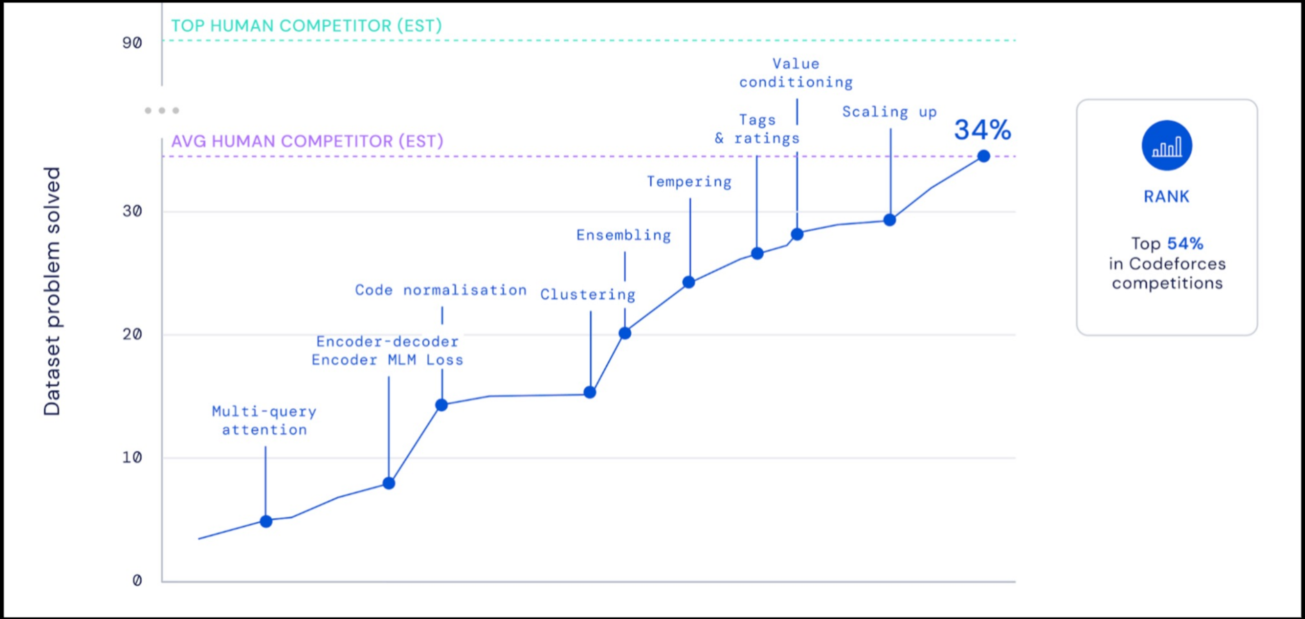
Site	URL	Source
Aizu	https://judge.u-aizu.ac.jp	CodeNet
AtCoder	https://atcoder.jp	CodeNet
CodeChef	https://www.codechef.com	description2code
Codeforces	https://codeforces.com	description2code and Codeforces
HackerEarth	https://www.hackerearth.com	description2code

How do we measure this?

1 Benchmark Tasks

2 Competitions

3 Real-world impact

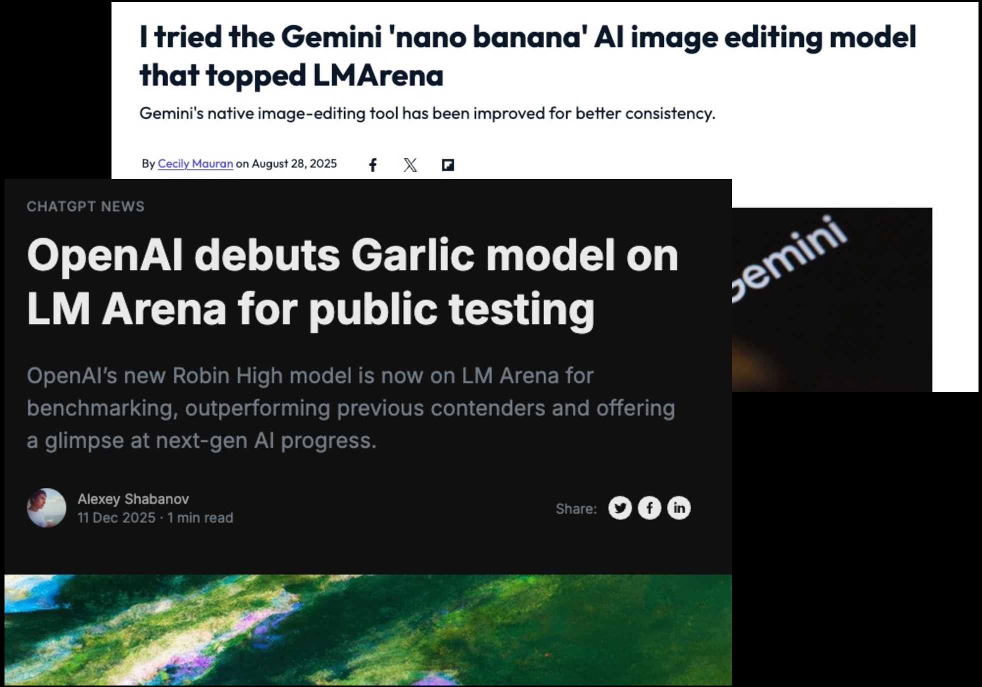


Arenas are becoming a gold standard

Relevant Arenas

Code Arena
WebDev
by Arena (Stanford '20)

Design Arena
Visual design evaluation
by Arcadia Labs



I tried the Gemini 'nano banana' AI image editing model that topped LMArena

Gemini's native image-editing tool has been improved for better consistency.

By Cecily Mauran on August 28, 2025

CHATGPT NEWS

OpenAI debuts Garlic model on LM Arena for public testing

OpenAI's new Robin High model is now on LM Arena for benchmarking, outperforming previous contenders and offering a glimpse at next-gen AI progress.

Alexey Shabanov
11 Dec 2025 · 1 min read

Share:

How do we measure this?

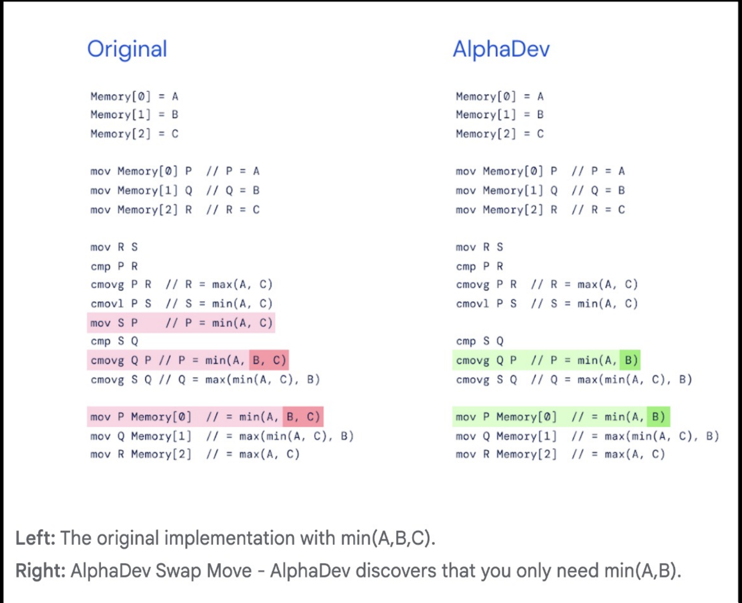
1 Benchmark Tasks 2 Competitions 3 Real-world impact

As models begin to surpass human performance, they will be increasingly measured on impact.

Examples:

AlphaDev (Mankowitz and Michi, June 2023) discovered a faster sorting algorithm for small lists that has now been implemented in the C++ standard lib.

AlphaEvolve (May 2025) found an algorithm to multiply 4x4 complex-valued matrices using 48 scalar multiplications, improving upon Strassen's algorithm.



Original

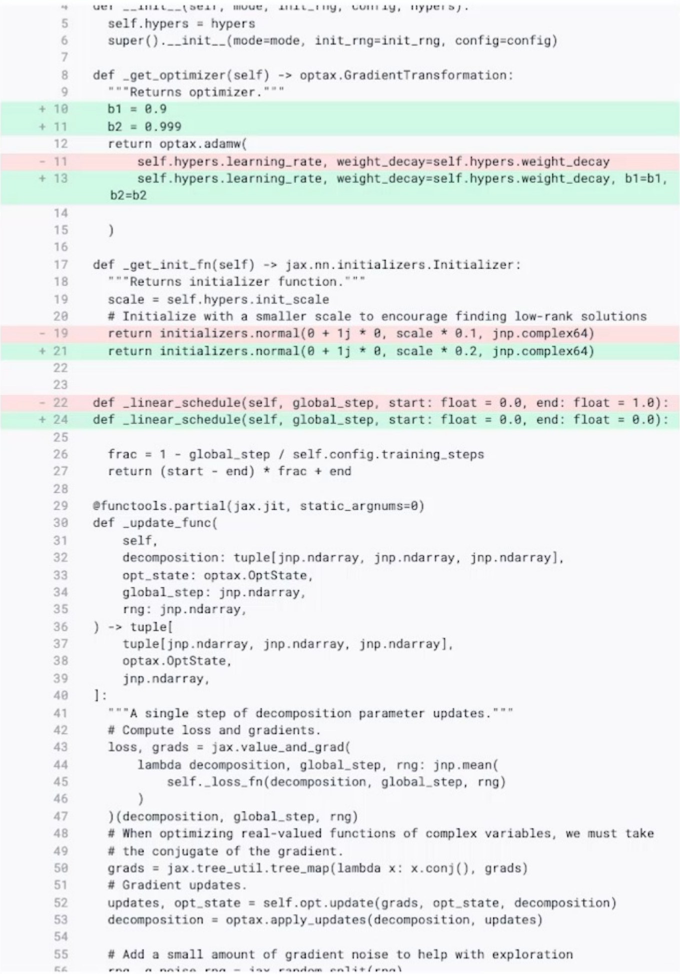
```
Memory[0] = A
Memory[1] = B
Memory[2] = C

mov Memory[0] P  // P = A
mov Memory[1] Q  // Q = B
mov Memory[2] R  // R = C

mov R S
cmp P R
cmovg P R  // R = max(A, C)
cmovl P S  // S = min(A, C)
mov S P    // P = min(A, C)
cmp S Q
cmovg Q P // P = min(A, B, C)
cmovg S Q // Q = max(min(A, C), B)

mov P Memory[0]  // = min(A, B, C)
mov Q Memory[1]  // = max(min(A, C), B)
mov R Memory[2]  // = max(A, C)
```

AlphaDev

```
Memory[0] = A
Memory[1] = B
Memory[2] = C

mov Memory[0] P  // P = A
mov Memory[1] Q  // Q = B
mov Memory[2] R  // R = C

mov R S
cmp P R
cmovg P R  // R = max(A, C)
cmovl P S  // S = min(A, C)

cmp S Q
cmovg Q P  // P = min(A, B)
cmovg S Q  // Q = max(min(A, C), B)

mov P Memory[0]  // = min(A, B)
mov Q Memory[1]  // = max(min(A, C), B)
mov R Memory[2]  // = max(A, C)
```

Left: The original implementation with min(A,B,C).

Right: AlphaDev Swap Move - AlphaDev discovers that you only need min(A,B).



```
    def __init__(self, mode, init_rng, config, hypers):
      self.hypers = hypers
      super().__init__(mode=mode, init_rng=init_rng, config=config)

    def _get_optimizer(self) -> optax.GradientTransformation:
      """Returns optimizer."""
+     b1 = 0.9
+     b2 = 0.999
      return optax.adamw(
-         self.hypers.learning_rate, weight_decay=self.hypers.weight_decay
+         self.hypers.learning_rate, weight_decay=self.hypers.weight_decay, b1=b1, b2=b2

      )

    def _get_init_fn(self) -> jax.nn.initializers.Initializer:
      """Returns initializer function."""
      scale = self.hypers.init_scale
      # Initialize with a smaller scale to encourage finding low-rank solutions
-     return initializers.normal(0 + 1j * 0, scale * 0.1, jnp.complex64)
+     return initializers.normal(0 + 1j * 0, scale * 0.2, jnp.complex64)


-   def _linear_schedule(self, global_step, start: float = 0.0, end: float = 1.0):
+   def _linear_schedule(self, global_step, start: float = 0.0, end: float = 0.0):

      frac = 1 - global_step / self.config.training_steps
      return (start - end) * frac + end

    @functools.partial(jax.jit, static_argnums=0)
    def _update_func(
        self,
        decomposition: tuple[jnp.ndarray, jnp.ndarray, jnp.ndarray],
        opt_state: optax.OptState,
        global_step: jnp.ndarray,
        rng: jnp.ndarray,
    ) -> tuple[
        tuple[jnp.ndarray, jnp.ndarray, jnp.ndarray],
        optax.OptState,
        jnp.ndarray,
    ]:
      """A single step of decomposition parameter updates."""
      # Compute loss and gradients.
      loss, grads = jax.value_and_grad(
          lambda decomposition, global_step, rng: jnp.mean(
              self._loss_fn(decomposition, global_step, rng)
          )
      )(decomposition, global_step, rng)
      # When optimizing real-valued functions of complex variables, we must take
      # the conjugate of the gradient.
      grads = jax.tree_util.tree_map(lambda x: x.conj(), grads)
      # Gradient updates.
      updates, opt_state = self.opt.update(grads, opt_state, decomposition)
      decomposition = optax.apply_updates(decomposition, updates)

      # Add a small amount of gradient noise to help with exploration
      rng, g_noise_rng = jax.random.split(rng)
```

Iteration 15

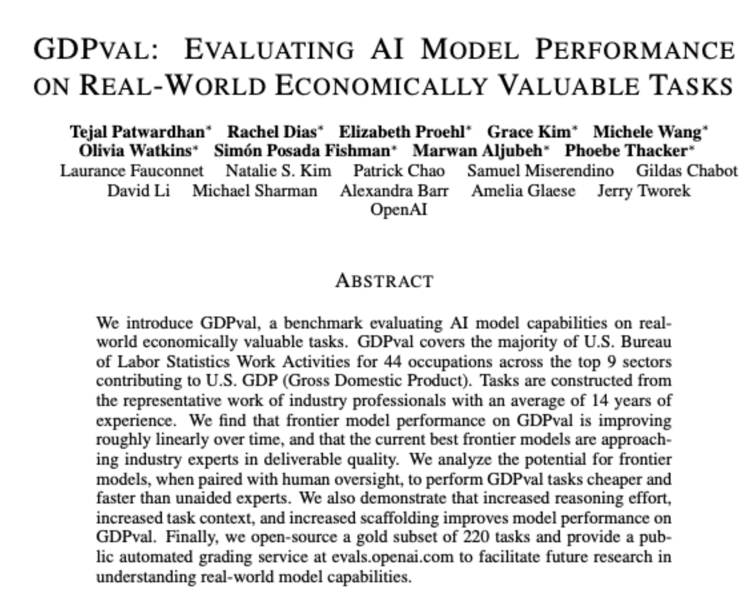
How do we measure this?

1 Benchmark Tasks 2 Competitions 3 Real-world impact

AGI is going to be increasingly measured by economic productivity.

When Cognition launched Devin, a key point was that it could solve real Upwork challenges. Since then, coding agents have proliferated: Claude Code, OpenAI Codex, Cursor, OpenHands, and many more.

This remains a moving, but very reasonable, goalpost.



GDPval: Evaluating AI Model Performance on Real-World Economically Valuable Tasks

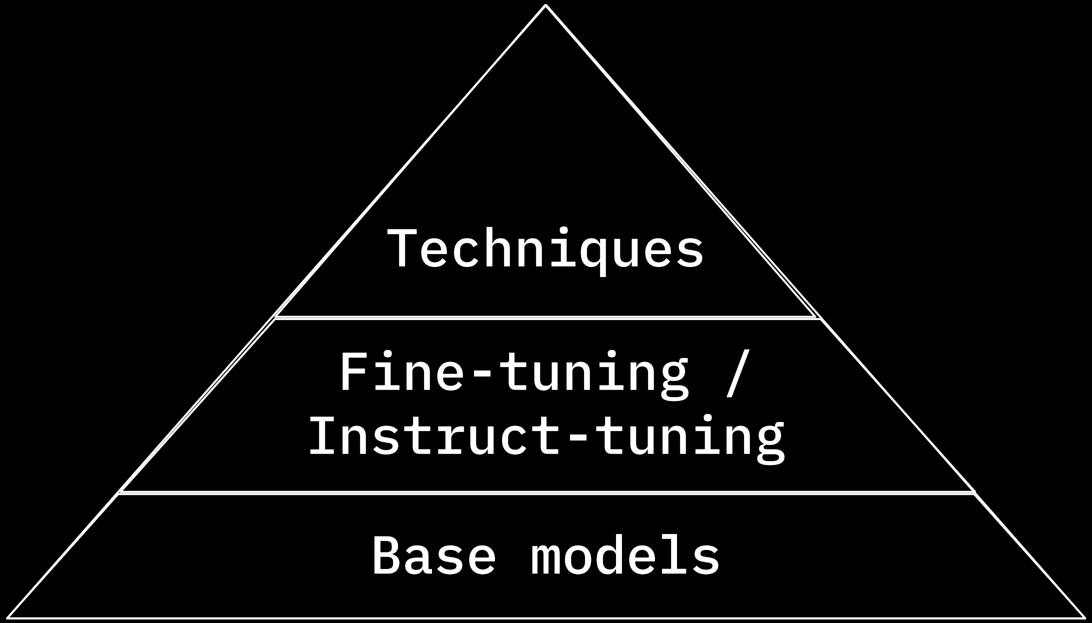
Tejal Patwardhan* **Rachel Dias*** **Elizabeth Proehl*** **Grace Kim*** **Michele Wang*** **Olivia Watkins*** **Simón Posada Fishman*** **Marwan Aljubeh*** **Phoebe Thacker***
Laurance Fauconnet Natalie S. Kim Patrick Chao Samuel Miserendino Gildas Chabot
David Li Michael Sharman Alexandra Barr Amelia Glaese Jerry Tworek
OpenAI

Abstract

We introduce GDPval, a benchmark evaluating AI model capabilities on real-world economically valuable tasks. GDPval covers the majority of U.S. Bureau of Labor Statistics Work Activities for 44 occupations across the top 9 sectors contributing to U.S. GDP (Gross Domestic Product). Tasks are constructed from the representative work of industry professionals with an average of 14 years of experience. We find that frontier model performance on GDPval is improving roughly linearly over time, and that the current best frontier models are approaching industry experts in deliverable quality. We analyze the potential for frontier models, when paired with human oversight, to perform GDPval tasks cheaper and faster than unaided experts. We also demonstrate that increased reasoning effort, increased task context, and increased scaffolding improves model performance on GDPval. Finally, we open-source a gold subset of 220 tasks and provide a public automated grading service at evals.openai.com to facilitate future research in understanding real-world model capabilities.

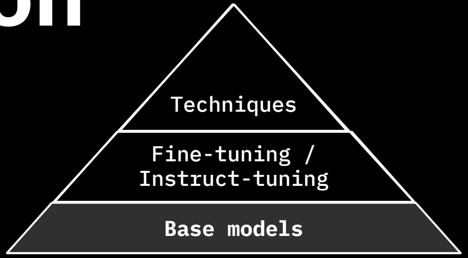
Benchmarking code generation

✓ Benchmark: SWEBench-Verified



Benchmarking code generation

✓ Benchmark: SWEBench-Verified



Base Models are the GPTs and Claudes of the world: not fine-tuned for a particular task.

Open LLMs

Weights are open, easy to do custom tuning and experimentation

- Kimi-K2.5
- DeepSeek-V3.2
- Qwen 2.5 / Qwen-3-Coder
- Llama 4 (Maverick)

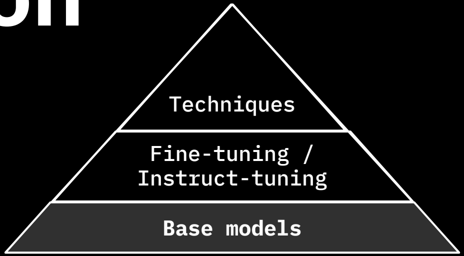
Closed LLMs

Weights are closed, tuning and experimentation are limited

- GPT-5.3-Codex
- Gemini 3 Pro
- Claude Opus 4.6
- Grok 3

Benchmarking code generation

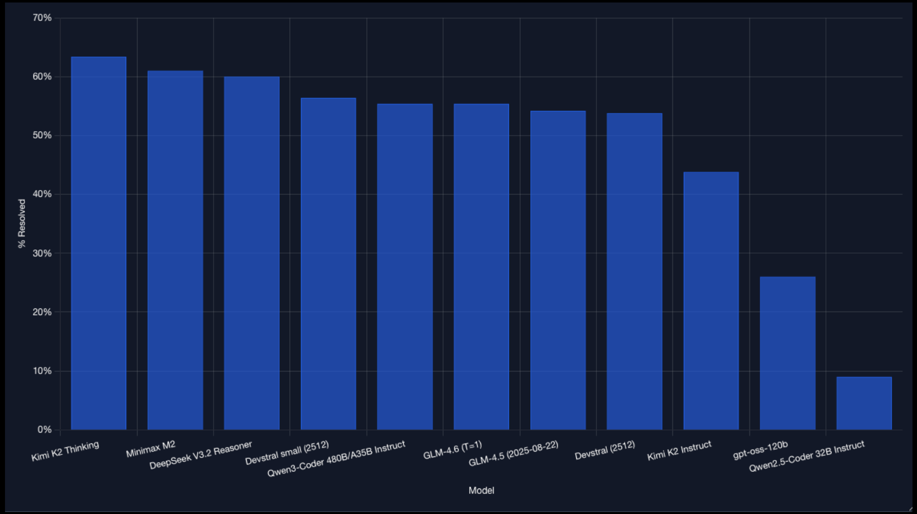
✓ Benchmark: SWEBench-Verified

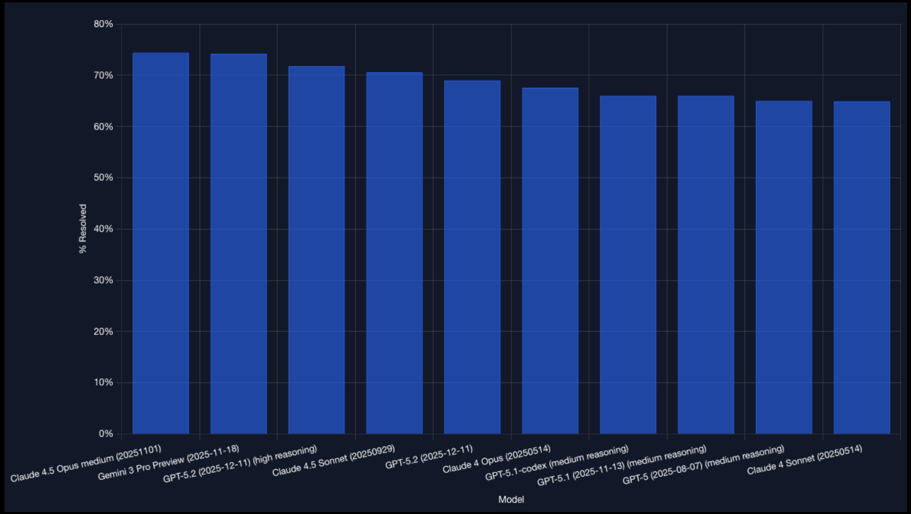


Base Models are the GPTs and Llamas of the world: not fine-tuned for a particular task.

Open: Kimi-K2.5, 76.8%

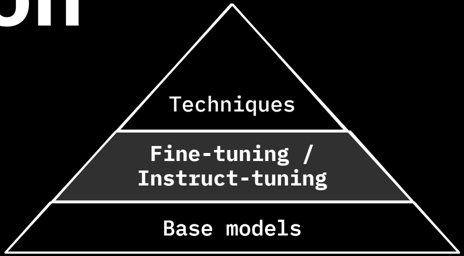
Closed: Claude 4.6 Opus, 80.8%





Benchmarking code generation

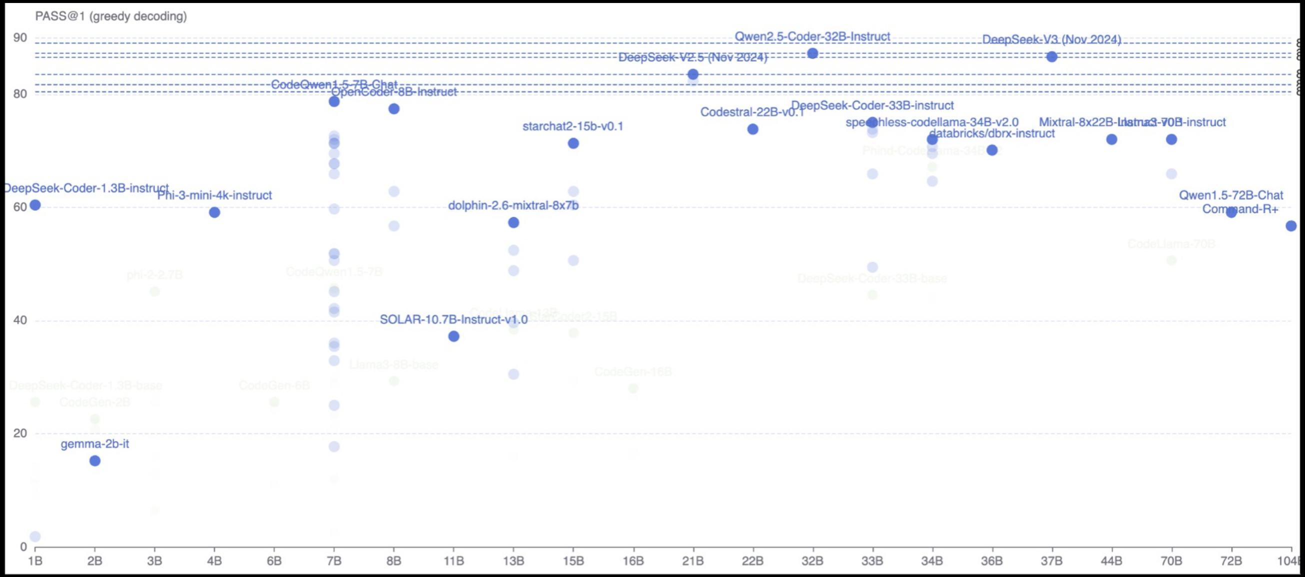
✓ Benchmark: SWEBench-Verified


Techniques

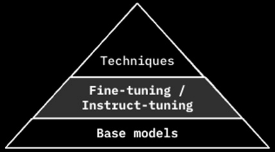
Fine-tuning / Instruct-tuning

Base models

Instruct-tuned models are models that are fine-tuned with instructions: in this case, for code.



Benchmarking code generation



✓ Benchmark: HumanEval

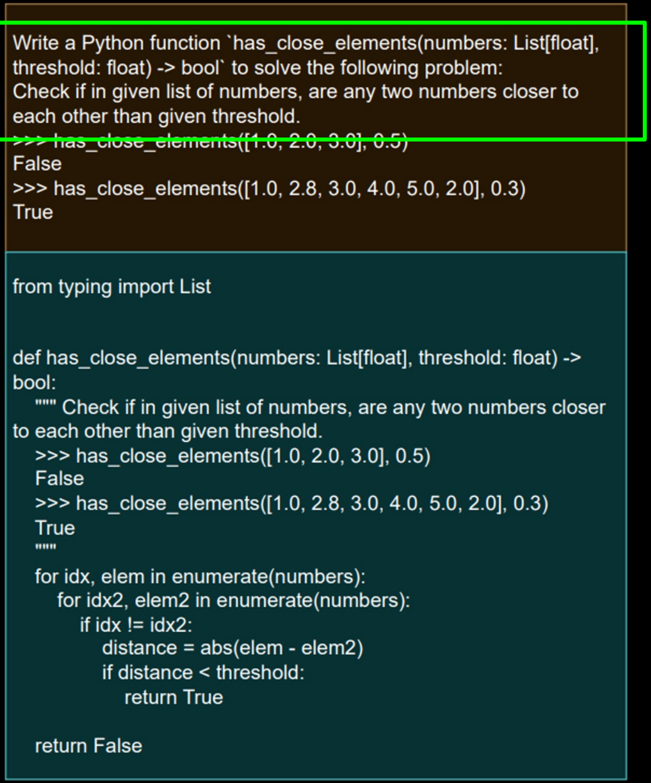
Instruct-tuned models are models that are fine-tuned with instructions: in this case, for code.

Instruct-tuning involves a prompt which contains an instruction, and a response. Including the instruction is important for the model to know how to understand new instructions at inference time.

Example: Synthesis

🟫 **Model Input**

🟦 **Target Output**

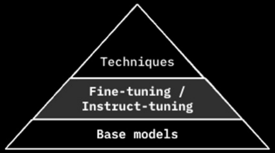


Write a Python function `has_close_elements(numbers: List[float], threshold: float) -> bool` to solve the following problem:
Check if in given list of numbers, are any two numbers closer to each other than given threshold.

```
>>> has_close_elements([1.0, 2.0, 3.0], 0.5)
False
>>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
True
```

```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = abs(elem - elem2)
                if distance < threshold:
                    return True

    return False
```

Benchmarking code generation



✓ Benchmark: HumanEval

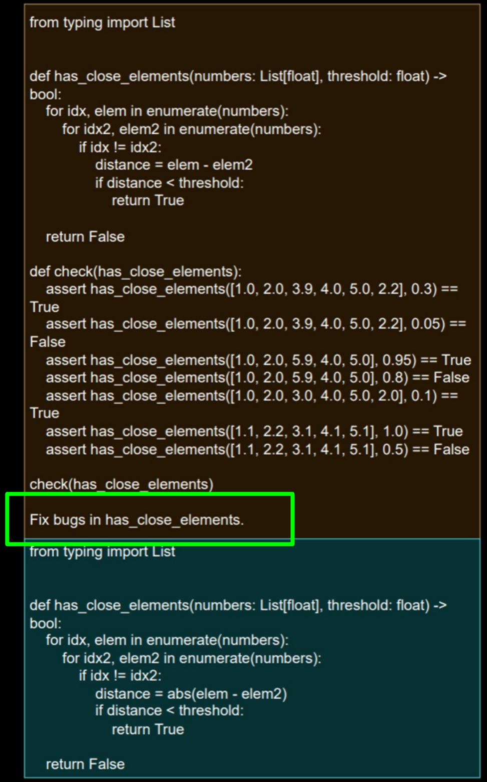
Instruct-tuned models are models that are fine-tuned with instructions: in this case, for code.

Instruct-tuning involves a prompt which contains an instruction, and a response. Including the instruction is important for the model to know how to understand new instructions at inference time.

Example: Fix a bug

🟫 **Model Input**

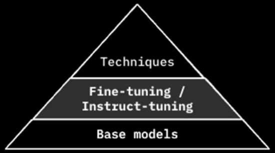
🟦 **Target Output**



```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) ->
bool:
    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = elem - elem2
                if distance < threshold:
                    return True

    return False

def check(has_close_elements):
    assert has_close_elements([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.3) ==
True
    assert has_close_elements([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.05) ==
False
    assert has_close_elements([1.0, 2.0, 5.9, 4.0, 5.0], 0.95) == True
    assert has_close_elements([1.0, 2.0, 5.9, 4.0, 5.0], 0.8) == False
    assert has_close_elements([1.0, 2.0, 3.0, 4.0, 5.0, 2.0], 0.1) ==
True
    assert has_close_elements([1.1, 2.2, 3.1, 4.1, 5.1], 1.0) == True
    assert has_close_elements([1.1, 2.2, 3.1, 4.1, 5.1], 0.5) == False

check(has_close_elements)

Fix bugs in has_close_elements.
```

```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) ->
bool:
    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = abs(elem - elem2)
                if distance < threshold:
                    return True

    return False
```

Benchmarking code generation



✓ Benchmark: HumanEval

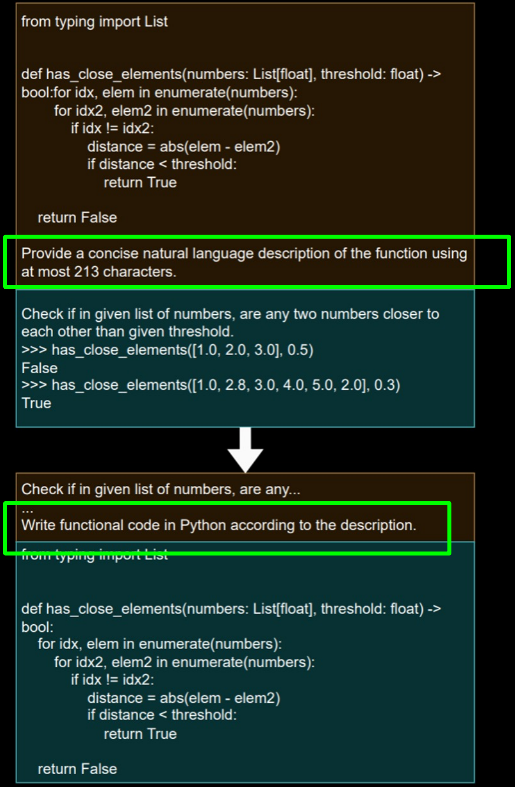
Instruct-tuned models are models that are fine-tuned with instructions: in this case, for code.

Instruct-tuning involves a prompt which contains an instruction, and a response. Including the instruction is important for the model to know how to understand new instructions at inference time.

Example: Explain code

🟫 **Model Input**

🟦 **Target Output**



```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) ->
bool:for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = abs(elem - elem2)
                if distance < threshold:
                    return True

    return False
```

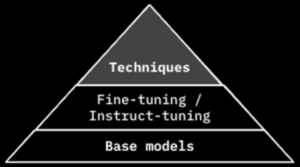
Provide a concise natural language description of the function using at most 213 characters.

```
Check if in given list of numbers, are any two numbers closer to
each other than given threshold.
>>> has_close_elements([1.0, 2.0, 3.0], 0.5)
False
>>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
True
```

```
Check if in given list of numbers, are any...
...
Write functional code in Python according to the description.
```

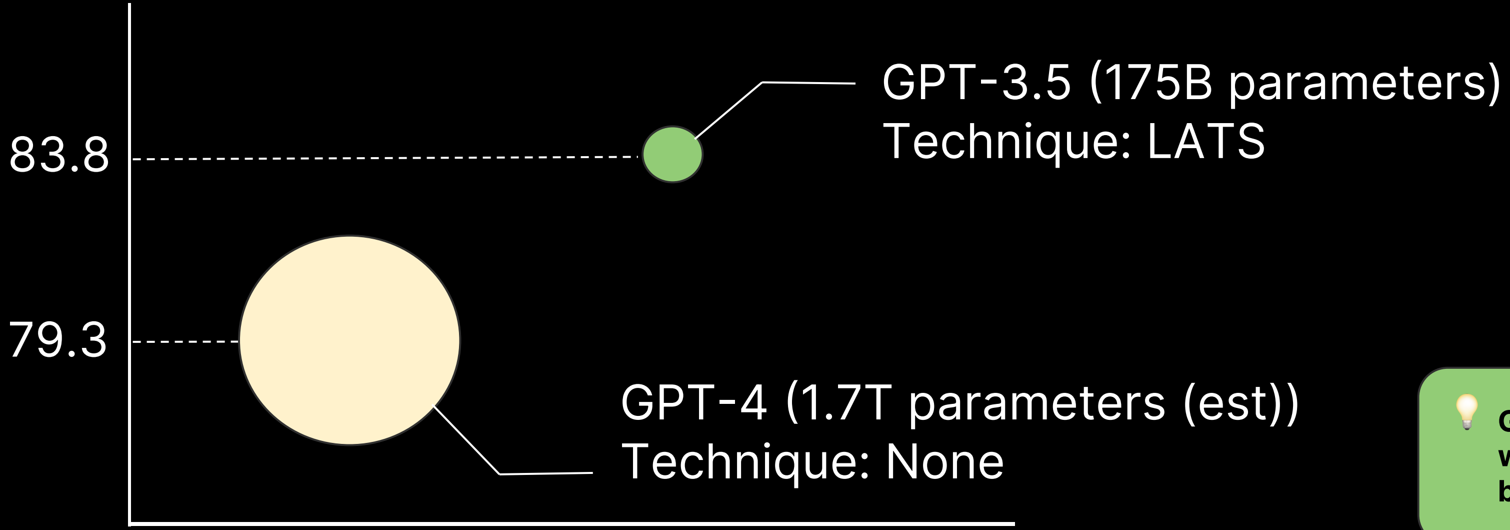
```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) ->
bool:
    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = abs(elem - elem2)
                if distance < threshold:
                    return True

    return False
```

Benchmarking code generation



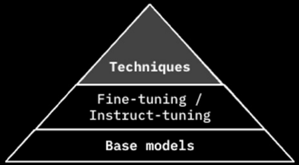
✓ Benchmark: HumanEval

Technique can make all the difference. This is broadly broken down into **reasoning methods** and **decision-making methods**.



💡 **GPT-3.5 beats GPT-4 with LATS, despite being 10x smaller!**

Benchmarking code generation

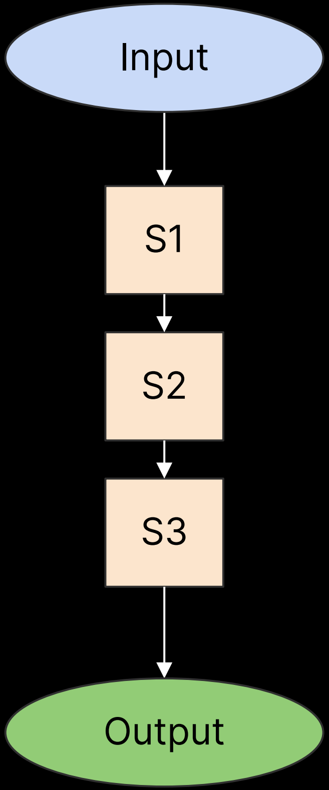


Chain-of-Thought

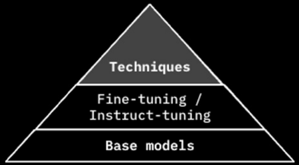
Reasoning Method

Chain-of-Thought (CoT) prompts LLMs to sequentially generate reasoning steps from input to output. It was first introduced in PaLM: Scaling Language Modeling with Pathways. (Chowdhery, Catasta et al., 2022)

However, it suffers from error propagation as the chain length increases.



Benchmarking code generation

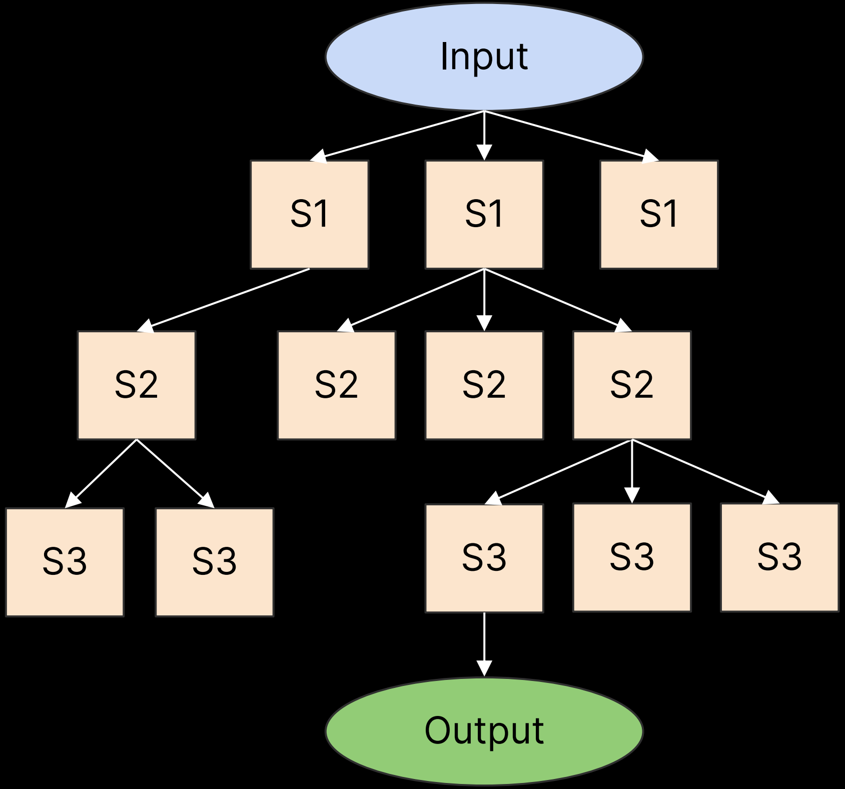


Tree-of-Thoughts

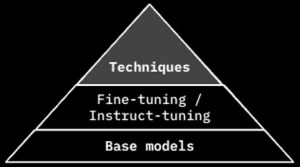
Reasoning Method

Tree-of-Thoughts (ToT) extends CoT by exploring multiple reasoning paths using search algorithms like BFS and DFS. (Yao et al., May 2023)

That said, it is limited by relying solely on the LLM's internal knowledge.



Benchmarking code generation

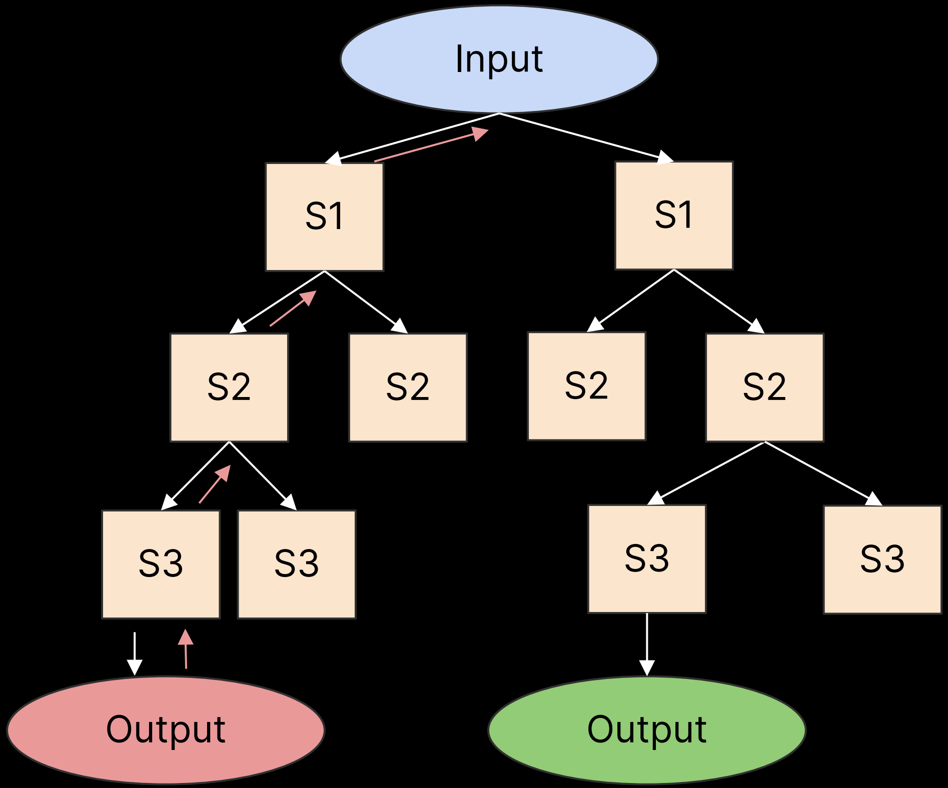


Reasoning via Planning

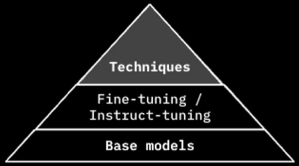
Reasoning Method

Reasoning via Planning (RAP) (Hao et al., October 2023) uses Monte Carlo Tree Search for planning chains of reasoning.

However, it also lacks external feedback.



Benchmarking code generation

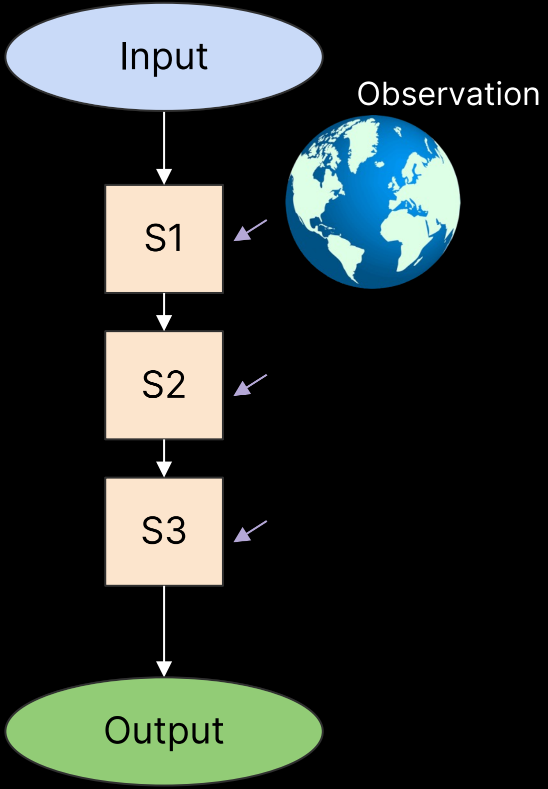


ReAct

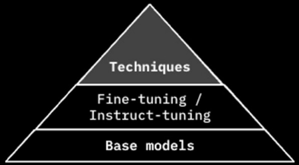
Decision-making method

ReAct prompts LLMs with alternating actions and observations for decision-making in interactive environments. (Yao et al., March 2023)

However, it greedily follows one trajectory and cannot adapt.



Benchmarking code generation

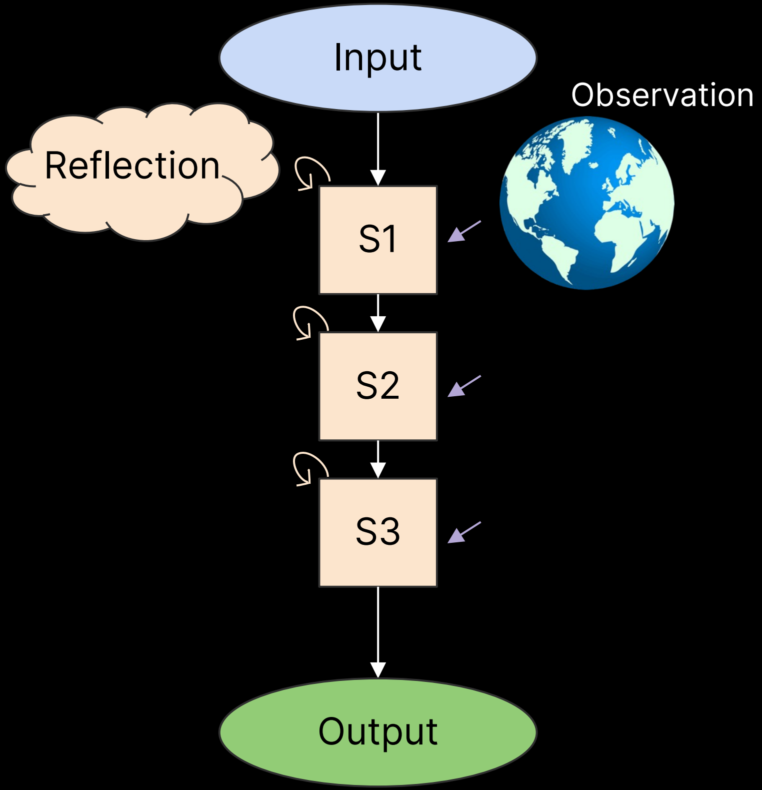


Reflexion

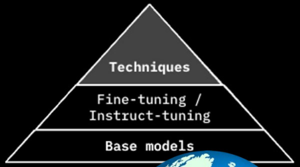
Decision-making method

Reflexion adds self-reflection to ReAct. This improves overall performance by allowing the LLM more time to think through the problem, similar to CoT. (Shinn et al., October 2023)

However, it does not consider alternative options at each step.



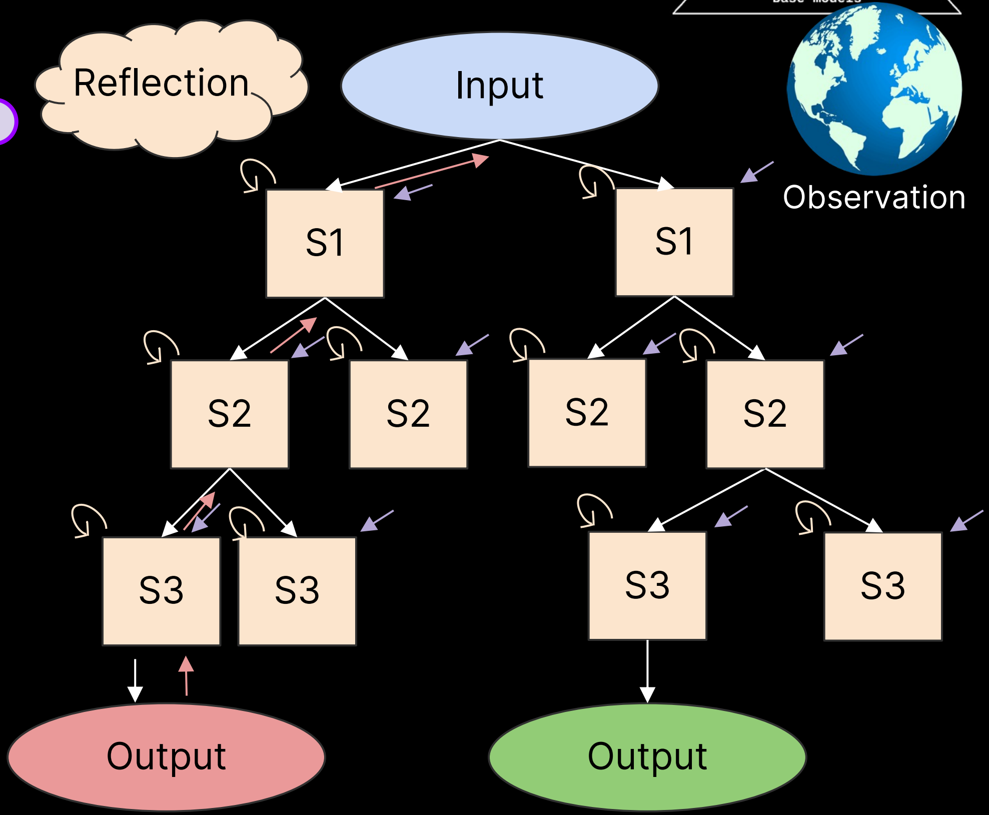
Benchmarking code generation



Language Agent Tree Search

Decision-making method **Reasoning Method**

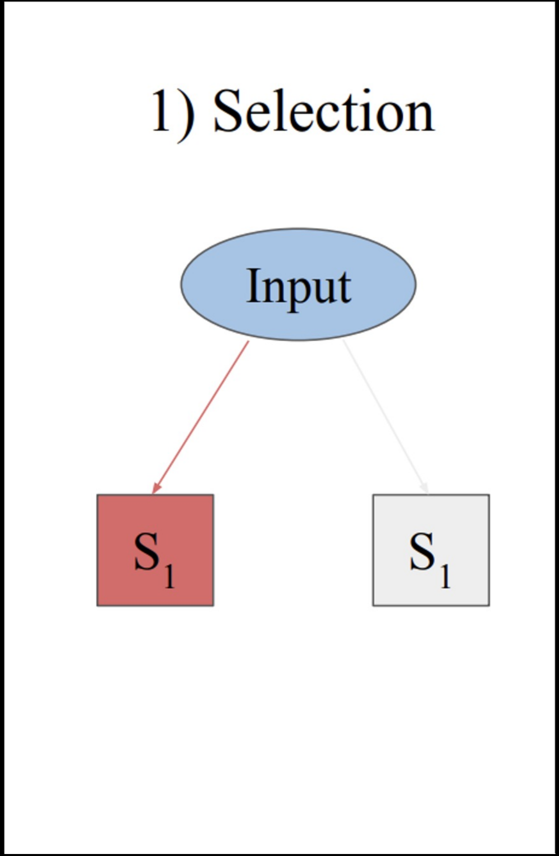
LATS unifies the strengths of both reasoning and decision-making methods through principled search, while overcoming limitations via environmental feedback and self-reflection. (Zhou et al., December 2023)



LATS Steps

Selection

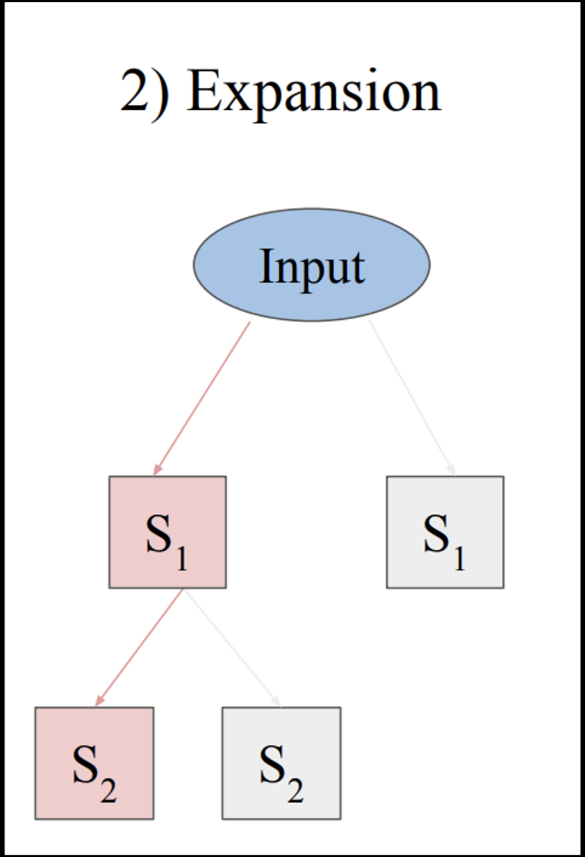
Select a node to travel to using the score we'll talk about.



LATS Steps

Expansion

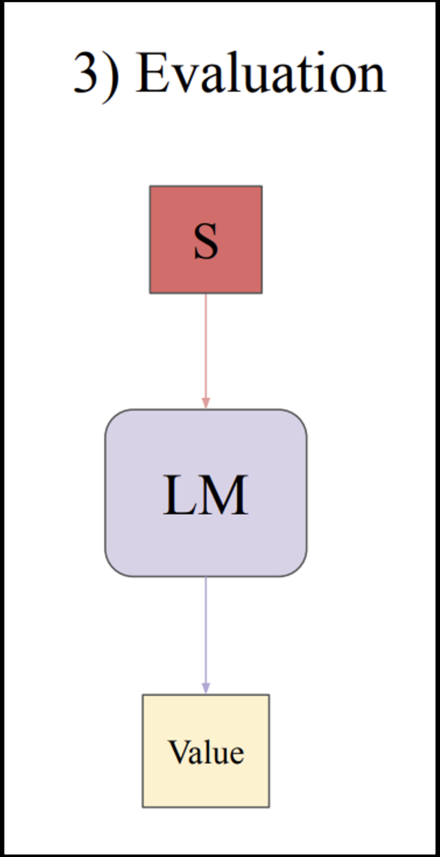
After selecting a node, the second operation expands the tree by sampling n actions from pθ, as described in the prior section.



LATS Steps

Evaluation

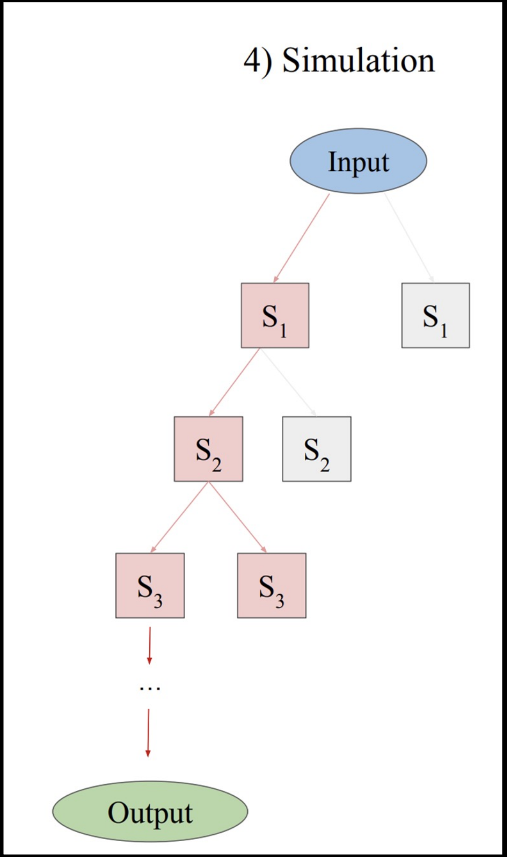
Assigns a scalar value to each new child node to be used for selection and backpropagation. This value effectively quantifies the agent's progress in task completion, steering the agent towards the most promising branch.



LATS Steps

Simulation

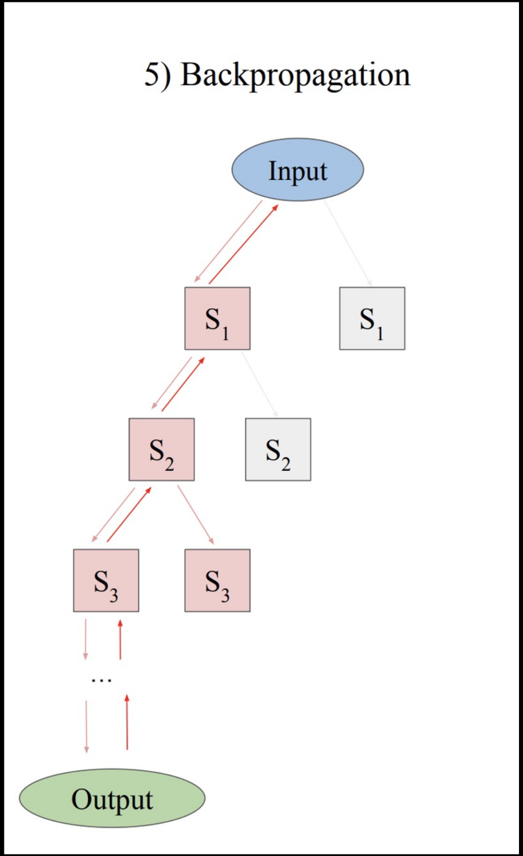
Expands the currently selected node until a terminal state is reached. At each depth level we sample and evaluate nodes with the same operations, but prioritize nodes of highest value.



LATS Steps

Backpropagation

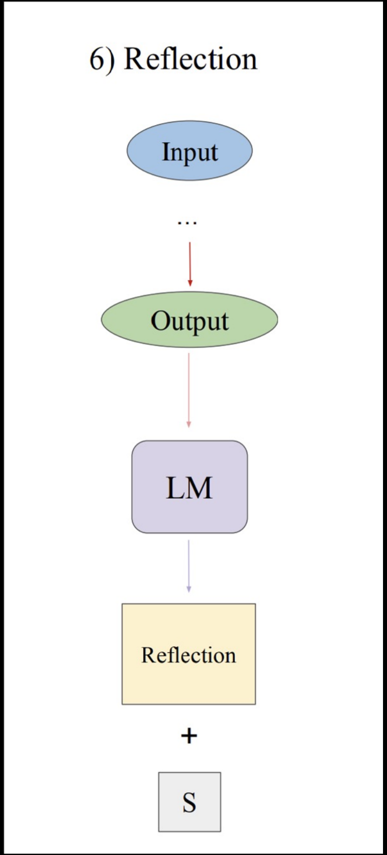
Updates the values of the tree based on the outcome of a trajectory.



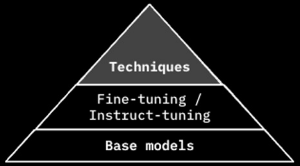
LATS Steps

Reflection

Upon encountering an unsuccessful terminal node, pθ is prompted with the trajectory and final reward to provide a verbal self-reflection that summarizes the errors in the reasoning or acting process and proposes superior alternatives.



Benchmarking code generation



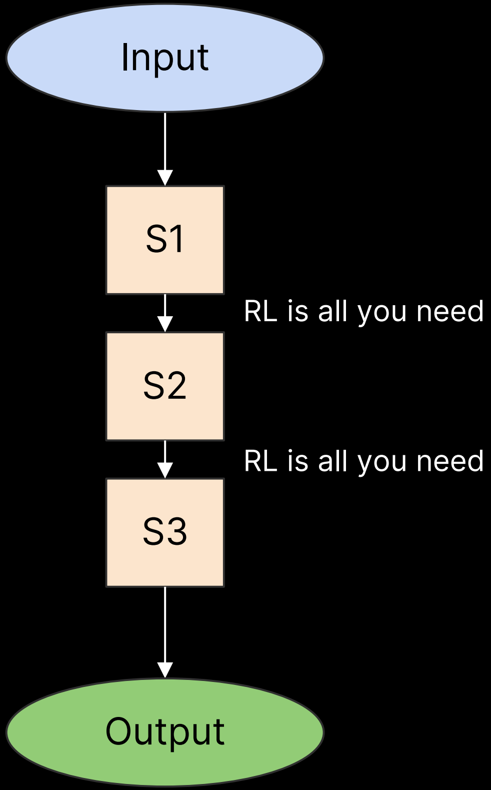
RL-driven reasoning in token space

Reasoning Method

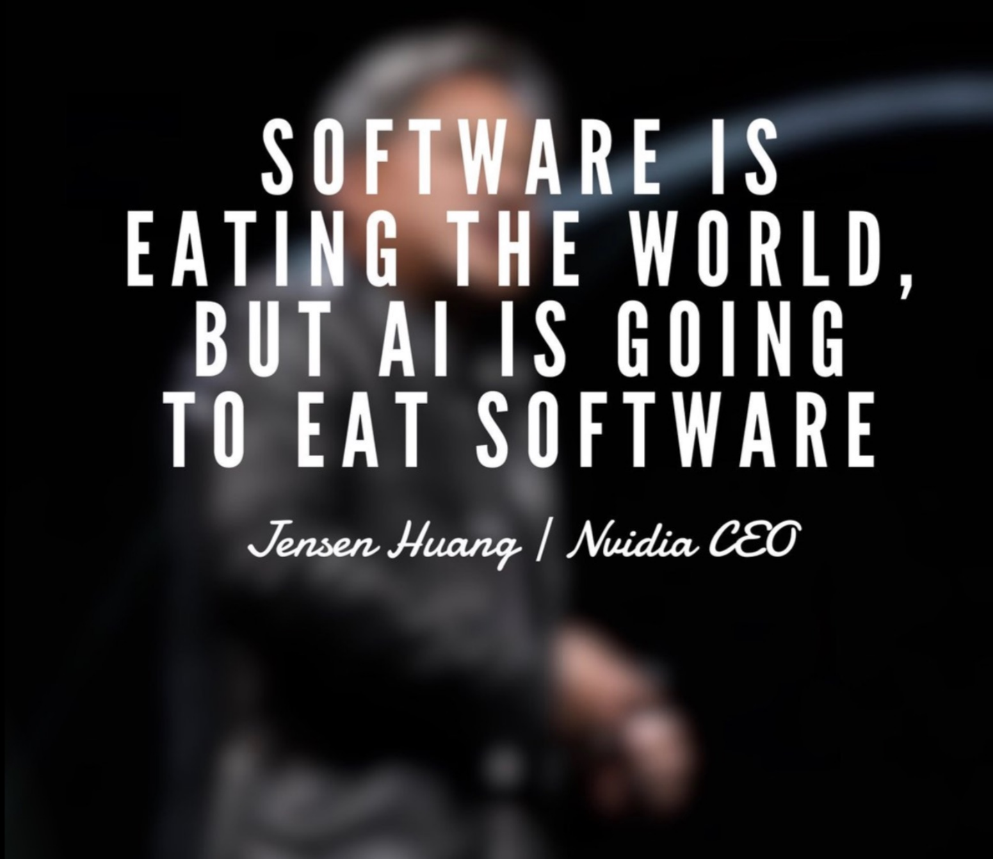
OpenAI's o-series (o1 through o4) pioneered RL-driven reasoning in token space.

This approach has been widely adopted: Claude's extended thinking, DeepSeek R1, Gemini's thinking mode.

"Bitter lesson" for agent techniques?



Applications and agents

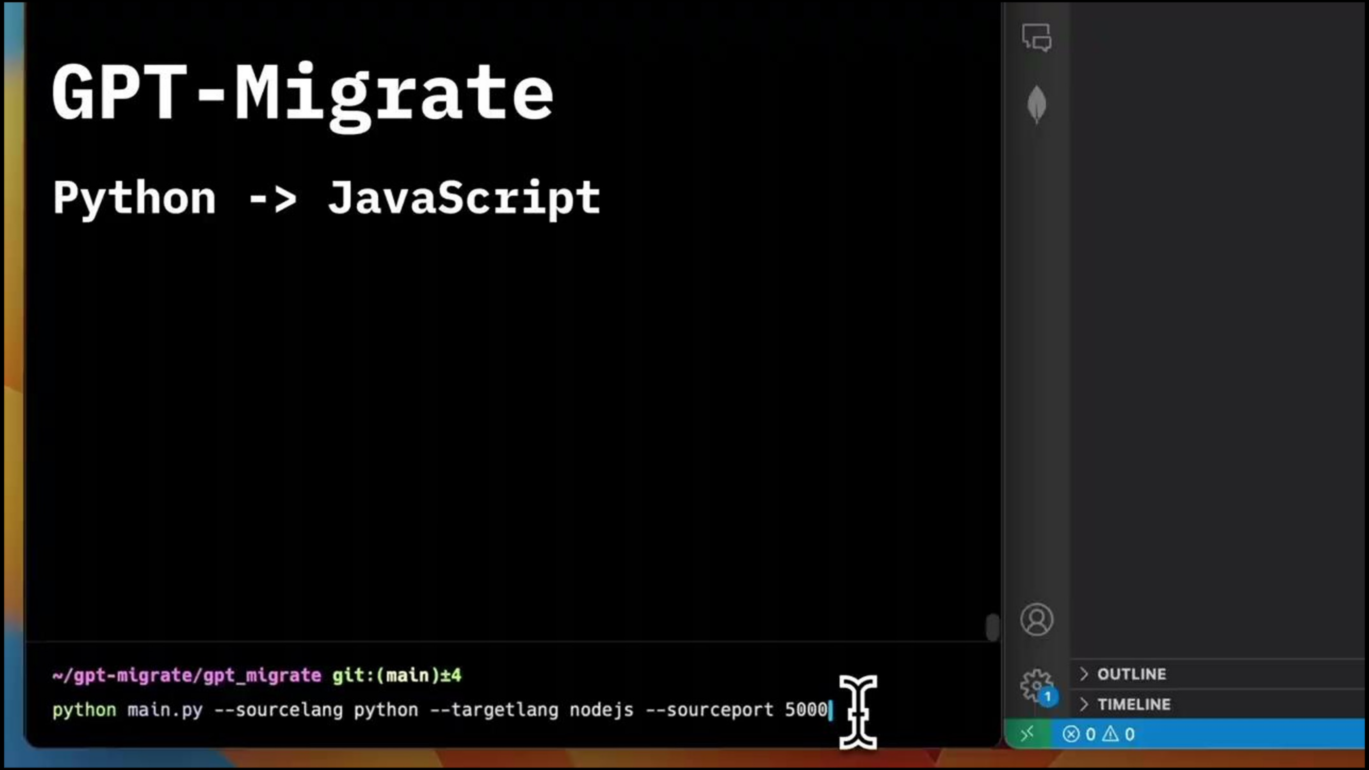


SOFTWARE IS EATING THE WORLD, BUT AI IS GOING TO EAT SOFTWARE

Jensen Huang | Nvidia CEO

Applications and agents

Deep dive: GPT-Migrate



GPT-Migrate

Python -> JavaScript

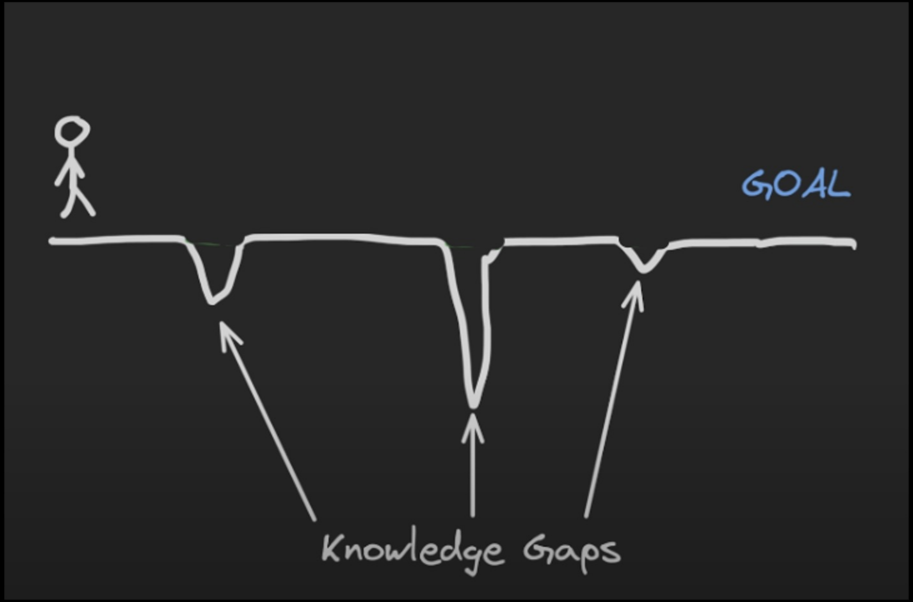
```
~/gpt-migrate/gpt_migrate git:(main)±4
python main.py --sourcelang python --targetlang nodejs --sourceport 5000
```

AI x Software Engineering

- Using code generation wisely
- Multi-agent management
- Harness-driven development

AI x Software Engineering

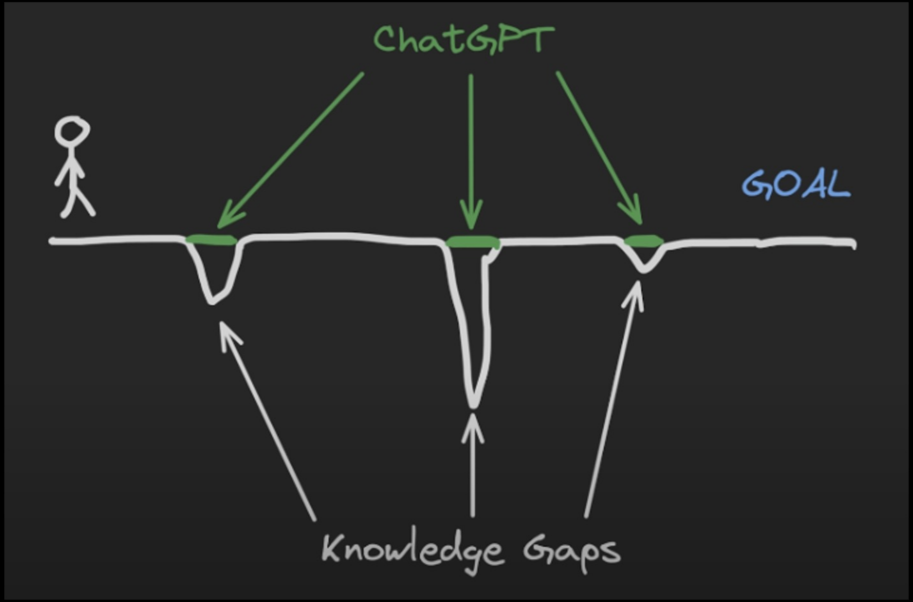
Using code generation wisely



Credit to [Joshua Morony]

AI x Software Engineering

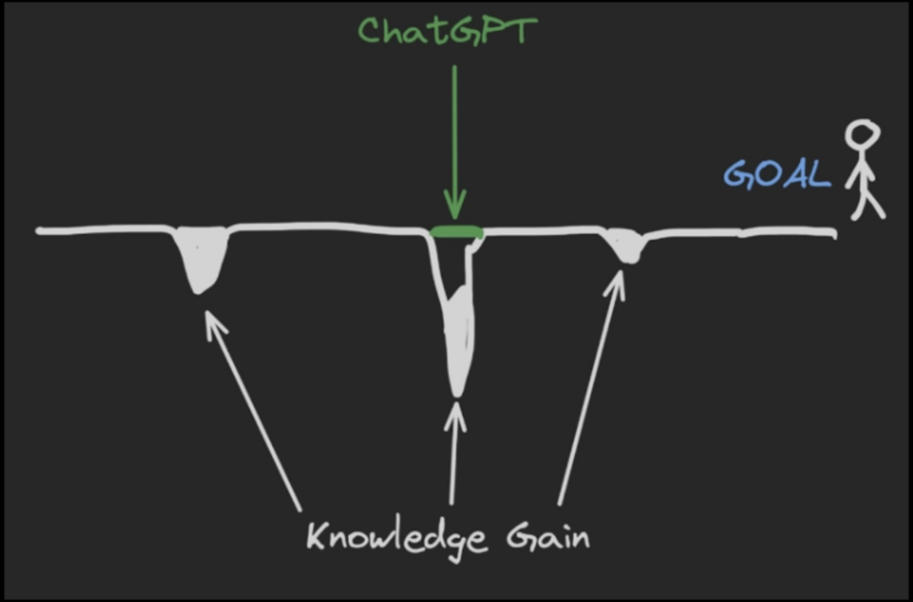
Using code generation wisely



Credit to Joshua Morony

AI x Software Engineering

Using code generation wisely

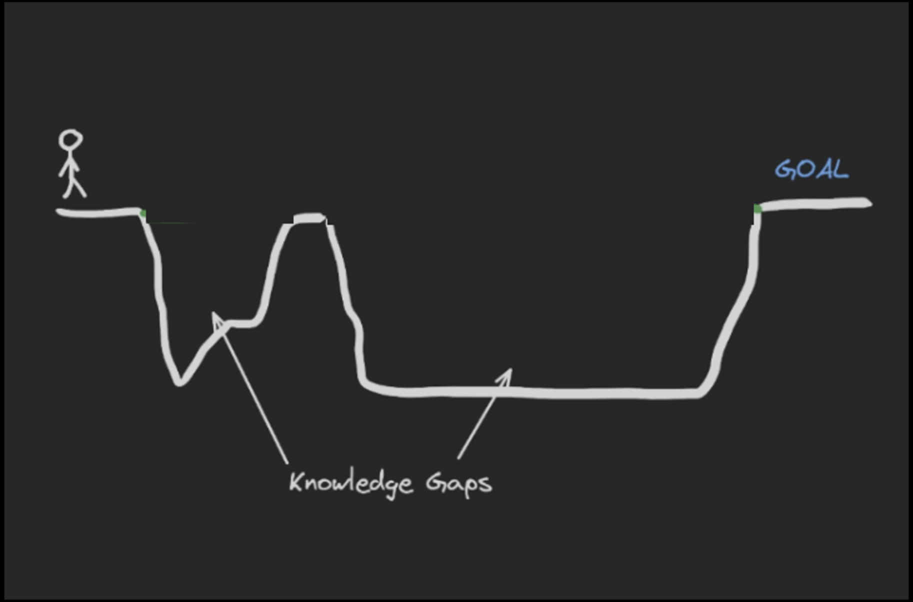


✓

Credit to Joshua Morony

AI x Software Engineering

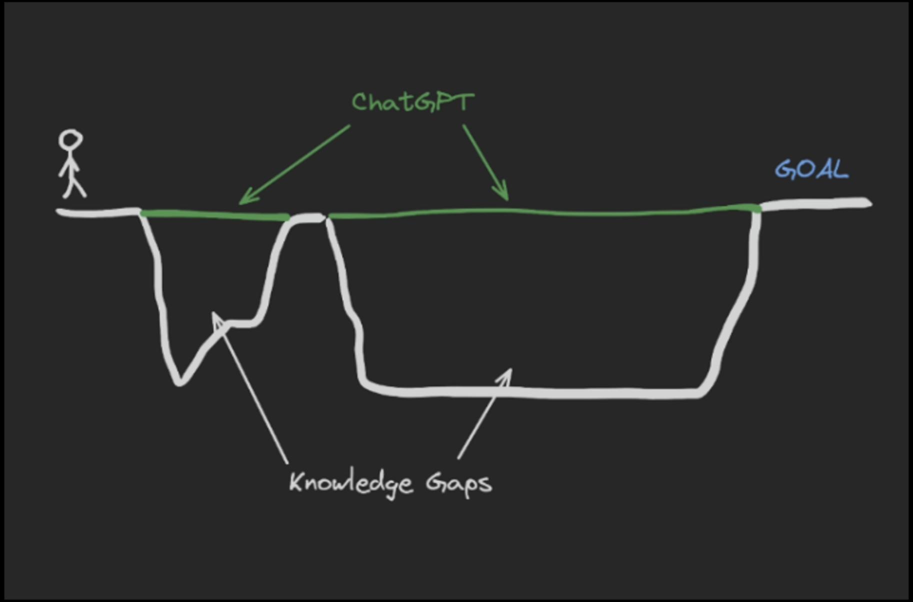
Using code generation wisely



Credit to Joshua Morony

AI x Software Engineering

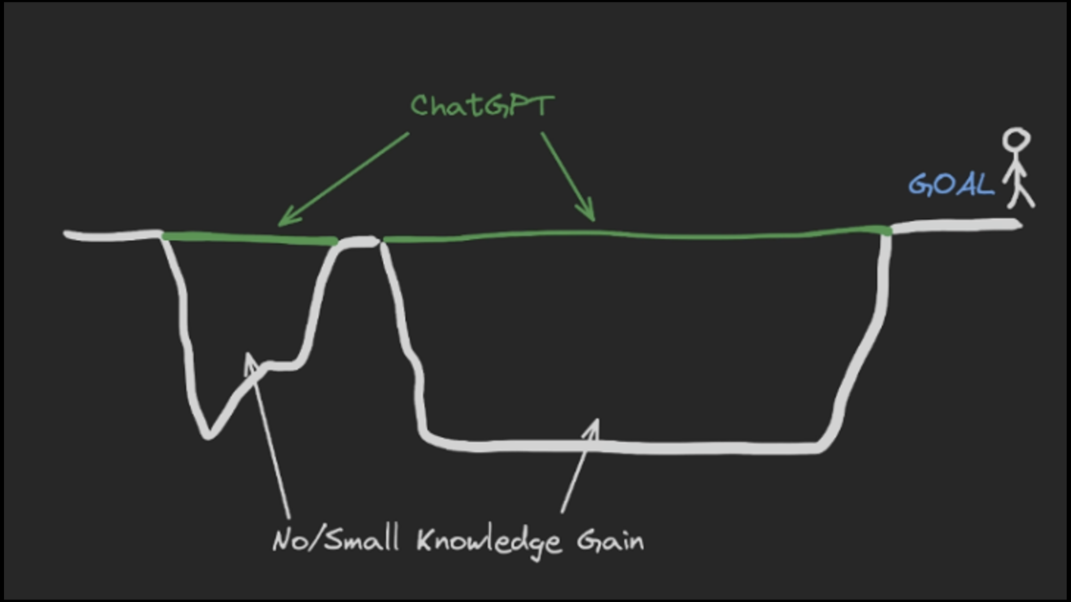
Using code generation wisely



Credit to Joshua Morony

AI x Software Engineering

Using code generation wisely



❌

Credit to Joshua Morony

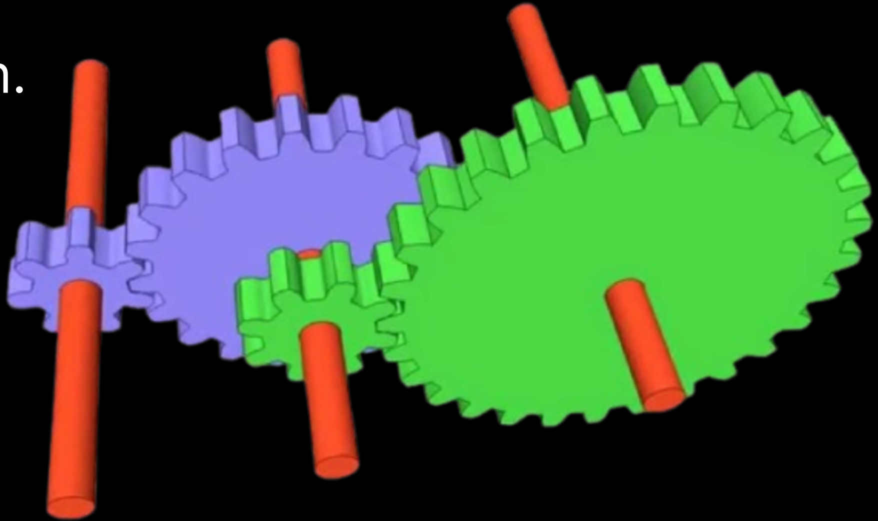
AI x Software Engineering

Using code generation wisely

HOWEVER: agents provide an incredibly powerful way to give you the knowledge you need.

Move up the layers of abstraction.

Get leverage.



AI x Software Engineering

Using code generation wisely

HOWEVER: agents provide an incredibly powerful way to give you the knowledge you need.

Move up the layers of abstraction.

Get leverage.

But not more than
you can handle.

❌

AI x Software Engineering

Using code generation wisely

Why?

- Learning is important
- Understanding your code is important
- Maintainability and knowledge transfer is important
    - Fully agent-written projects tend to produce "spaghetti code", still. IT WORKS! But it's spaghetti: confusing to understand and maintain.

AI x Software Engineering

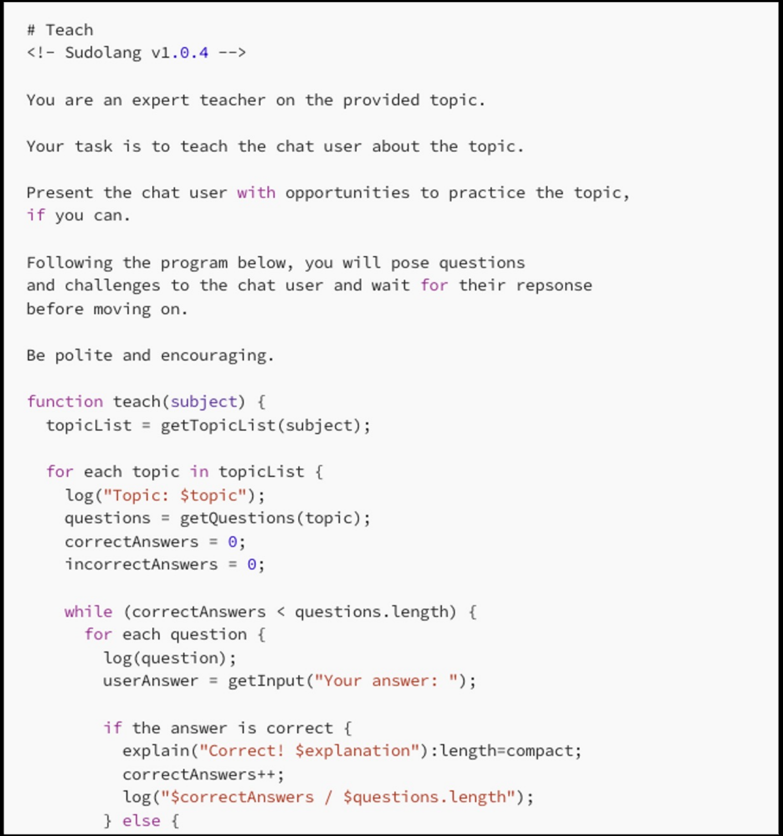
Prompting for code gen -- exercise

Sudolang is a natural language constraint-based programming pseudolanguage, with an LLM as the interpreter. What?

More simply, it combines natural language elements and simple coding conventions for better prompting.

SudoLang prompts can often be written with 20% - 30% fewer tokens than natural language.

The expressiveness and precision helps when writing code, as well as when “programming” the LLM to serve as an application itself.



```
# Teach
<!- Sudolang v1.0.4 -->

You are an expert teacher on the provided topic.

Your task is to teach the chat user about the topic.

Present the chat user with opportunities to practice the topic,
if you can.

Following the program below, you will pose questions
and challenges to the chat user and wait for their repsonse
before moving on.

Be polite and encouraging.

function teach(subject) {
  topicList = getTopicList(subject);

  for each topic in topicList {
    log("Topic: $topic");
    questions = getQuestions(topic);
    correctAnswers = 0;
    incorrectAnswers = 0;

    while (correctAnswers < questions.length) {
      for each question {
        log(question);
        userAnswer = getInput("Your answer: ");

        if the answer is correct {
          explain("Correct! $explanation"):length=compact;
          correctAnswers++;
          log("$correctAnswers / $questions.length");
        } else {
```

Coding Agent Setup

1 Configure

Use agents.md / CLAUDE.md as living documentation. Check into git. Update often. Ask the agent to update it.

2 Plan

Start in planning mode. Iterate on the plan before writing code. Good plans enable one-shot execution.

3 Verify

Give agents feedback loops: tests, linters, hooks. Verification is the single biggest quality multiplier.

Custom slash commands for repeated workflows
Pre-configure permissions in settings
Connect to Slack, issue trackers, error monitoring

Verification feedback loops → 2-3x output quality

Vibe Engineering

Shipping 30x+ faster with coding agents

1 Get started

This is a crucial competitive advantage.
A sword is only as good as its wielder.

2 Clarity is everything

Think before prompting. Know how the model will interpret you. Use collaborative planning.

3 Run agents in parallel

Push to 5+ simultaneous agents. **IMPORTANT SPEED BOOST.**
Use git worktrees or just let them share a tree.

4 TDD and beyond

TDD → test harnesses → ephemeral apps.
Spin up parallel databases to simulate state. **ANOTHER IMPORTANT SPEED BOOST.**

Vibe Engineering

Shipping 30x+ faster with coding agents

5 **Don’t be afraid to restart**

Better foundations = higher potential. Rebuilding with better context is often faster and better.

6 **Treat software as living**

Ship it, adapt rapidly. Small bugs are <1 min fixes. Doesn’t apply to infra/security.

7 **Know your models**

Build intuition about strengths and weaknesses.
Stay current — guidance from months ago is outdated.

8 **Guard quality**

The boiling frog: AI can let quality drift gradually if you let it. Maintain standards vigilantly. Agents.md / Claude.md will help!

The agents.md Pattern

Agent configuration as living documentation

What to Include

- Project architecture & context
- Build & test commands
- Code style & conventions
- Security boundaries
- Deployment & CI/CD steps

How to Maintain

- Check into version control
- Update multiple times per week
- Ask the agent to update it
- Nest configs in monorepos
- Keep separate from READMEs

When the agent makes a mistake, document it → it won't repeat it

Anatomy of a Great agents.md

Lessons from 2,500+ repositories (GitHub)

Commands

Executable commands first.
npm test, pytest -v,
bun run typecheck

Testing

Test commands, coverage expectations, CI/CD plan location

Project Structure

Stack with versions, key directories, file locations.

"React 18 + TypeScript + Vite"

Code Style

Show examples, not descriptions.
Good vs bad code snippets.
Examples are a cheat code.
You can point it to existing repos.

Git Workflow

Branch naming, commit format, PR conventions, review criteria

Boundaries

Three tiers:
Always do / Ask first / Never do

Vagueness fails.
"Helpful coding assistant" → bad.
"Test engineer for React components following these examples" → good.

Example: CLAUDE.md

From Anthropic’s claude-code-action (real production config)

```
# Commands
bun test                 # Run tests
bun run typecheck        # TypeScript checks
bun run format           # Format with prettier

# What This Is
A GitHub Action that lets Claude respond
to @claude mentions on issues/PRs...

# Things That Will Bite You
- Strict TypeScript: noUnusedLocals is on
- Discriminated unions for GitHub context
- Token lifecycle: revocation is in always()
- Integration tests are in a separate repo

# Code Conventions
- Runtime is Bun, not Node
- Imports don't need .js extensions
- GitHub API calls use retry logic
```

What makes it great

✓ Commands up front

Agent knows exactly how to build, test, and format

✓ Architecture context

Explains what the project is and how the pieces connect

✓ Gotchas documented

"Things that will bite you" prevents repeated mistakes

✓ Conventions explicit

Runtime, import style, retry patterns — no ambiguity

✓ Concise

~40 lines. Not a novel.
Just what the agent needs.

Ephemeral Software

The next step beyond test-driven development

4 Simulation Environments

Full sandbox environments with real services. Agents simulate real users on the other side.

3 Ephemeral Applications

Throwaway apps built purely for testing: load-test botnets, input playgrounds, API trigger apps. Entirely delete-able.

2 AI Tests (LLM-as-Judge)

Unit tests where the evaluator is an LLM. Structured test format. Critical for AI apps.

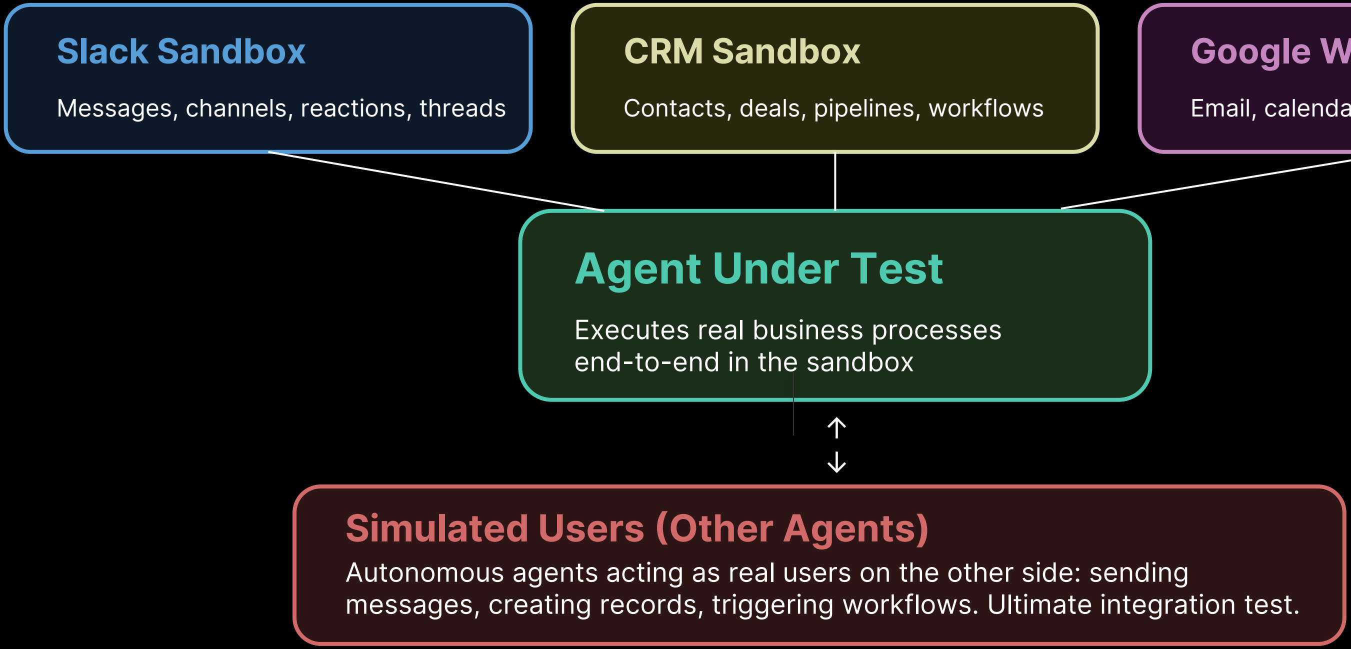
1 Unit Tests (TDD)

Base layer. LLMs love structure. "Write tests, don't stop until passing" -- can run overnight like ML training jobs.

Increasing complexity & realism

Environment Engineering: Example

Robust environments are the foundation for agentic software



Slack Sandbox

Messages, channels, reactions, threads

CRM Sandbox

Contacts, deals, pipelines, workflows

Google Workspace

Email, calendar, drive, docs

Agent Under Test

Executes real business processes end-to-end in the sandbox

↑
↓

Simulated Users (Other Agents)

Autonomous agents acting as real users on the other side: sending messages, creating records, triggering workflows. Ultimate integration test.

The Parallel

Unit tests
↓
AI tests
↓
Ephemeral apps
↓
Simulation envs
↓
Agent-as-user

The ultimate test: can the agent do the job when other agents are the users?

Questions

josh@coframe.com