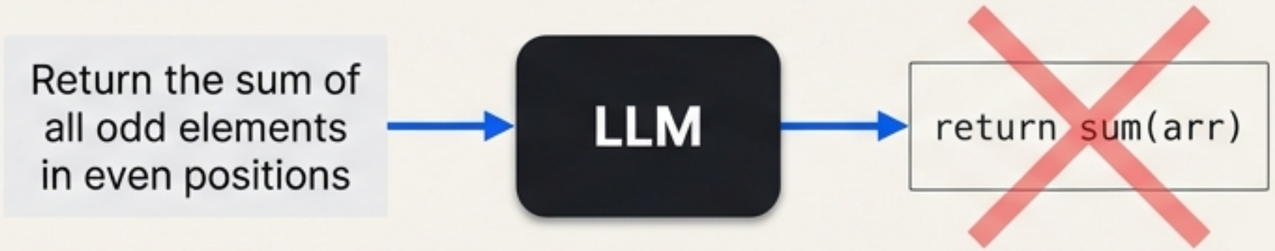
Evaluating Large Language Models Trained on Code

Mark Chen, Jerry Tworek, Heewoo Jun, et al. | OpenAI | arXiv 2021

The Goal: Turning Natural Language into Working Code

- Program synthesis—generating code from specifications—is a classic challenge in computer science.
- Recent Large Language Models (LLMs) like GPT-3 can generate code, but it's it's often syntactically correct yet functionally wrong.
- On a new, rigorous benchmark of Python problems, **GPT-3 solves 0%**.
- **The key question**: Can we train an LLM to not just write code, but to write ***functionally correct*** code?



We Need a Better Ruler: Measuring Functional Correctness

Traditional metrics like BLEU score are poor for code, as many different programs can be correct.

The Solution: Functional correctness. Code is judged correct if it passes a set of unit tests, just like in test-driven development.

The HumanEval Dataset:

- 164 original, hand-written programming problems.
- Covers algorithms, language comprehension, and math.
- Each problem has multiple unit tests (average of 7.7 per problem).

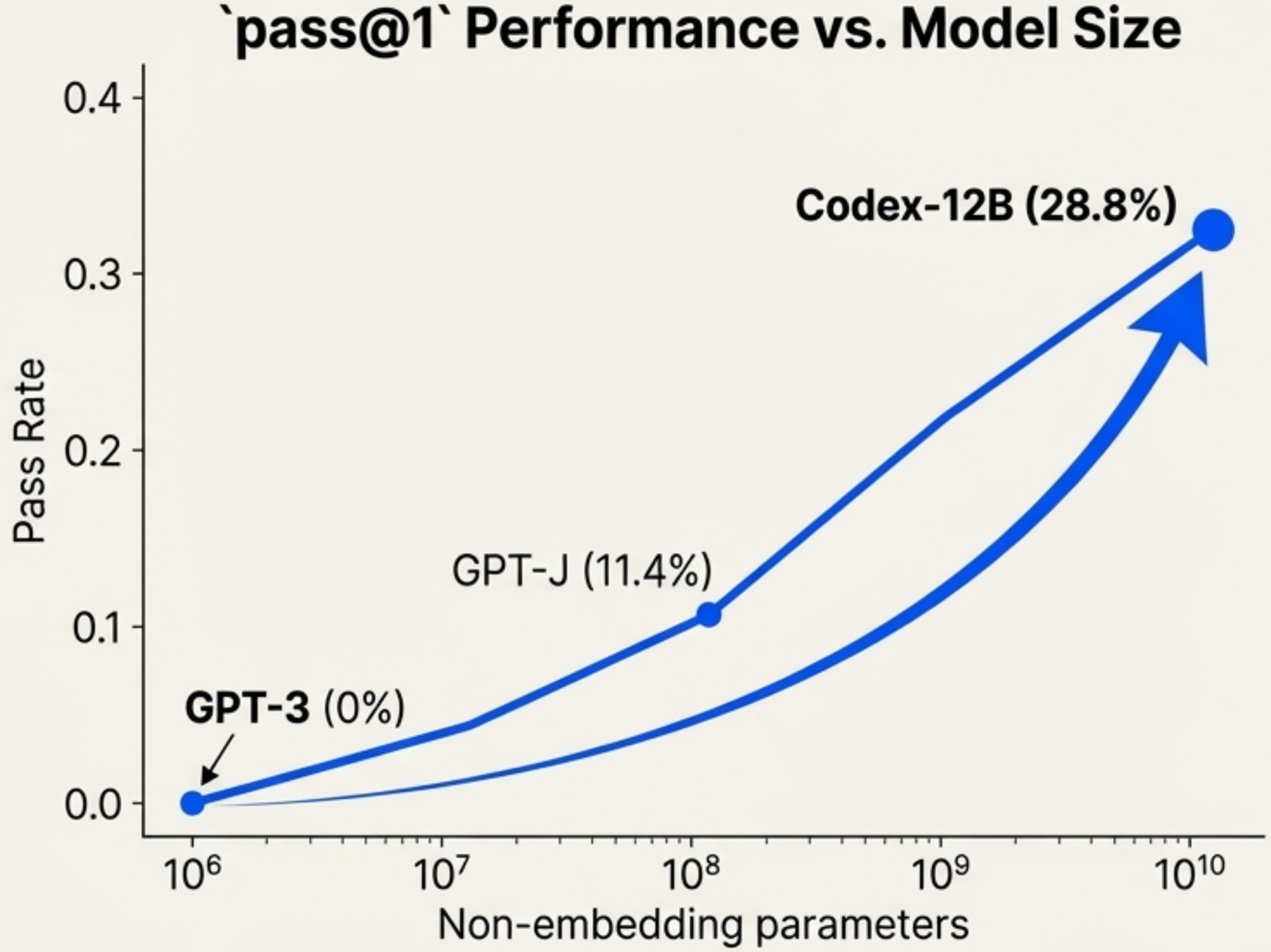
```
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

A GPT Model Specialized for Code

- **Model**: An architecture based on the GPT language model.
- **Training Data**: Fine-tuned on a dataset collected from 54 million public GitHub repositories.
- **Scale**: The final filtered dataset contained 159 GB of unique Python files.
- **Hypothesis**: Training on a massive corpus of real-world code will teach the model the patterns and logic required for program synthesis.

Codex Solves 28.8% of Problems on the First Try

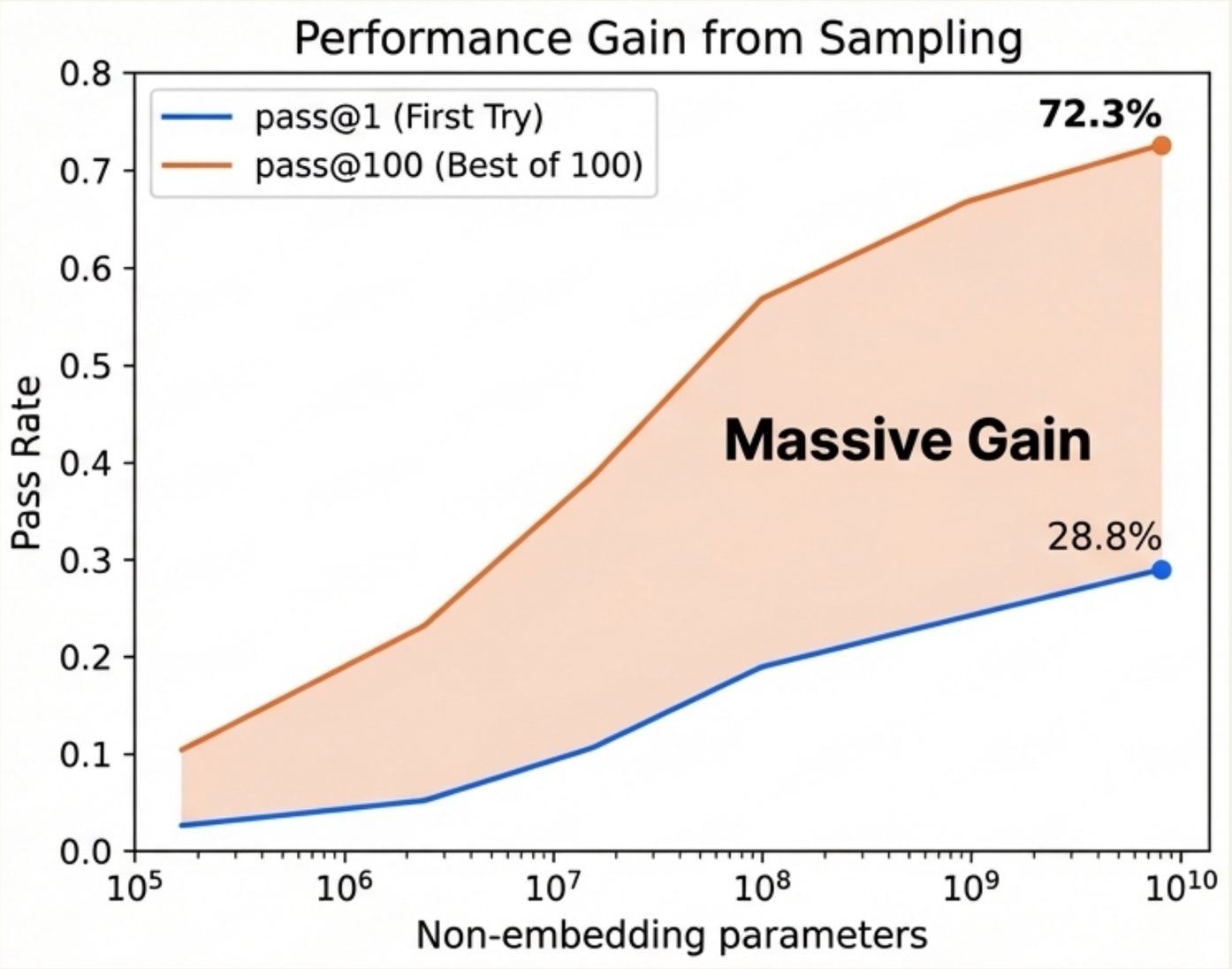
The largest Codex model (12B parameters) achieves **28.8%** **`pass@1`** on HumanEval. This is a monumental jump from GPT-3's 0% and significantly outperforms other contemporary models like GPT-J (11.4%). Performance scales predictably with model size, a trend also seen in natural language models.



Generating More Samples is a Surprisingly Effective Strategy

Instead of one attempt, what if we generate multiple solutions (`k` samples) and check if *any* are correct?

- This is measured by **`pass@k`**: the probability of getting at least one correct solution in `k` attempts.
- With 100 samples (pass@100`), the Codex-12B model solves **72.3%** of the problems.
- This simulates a developer trying multiple approaches until one works.



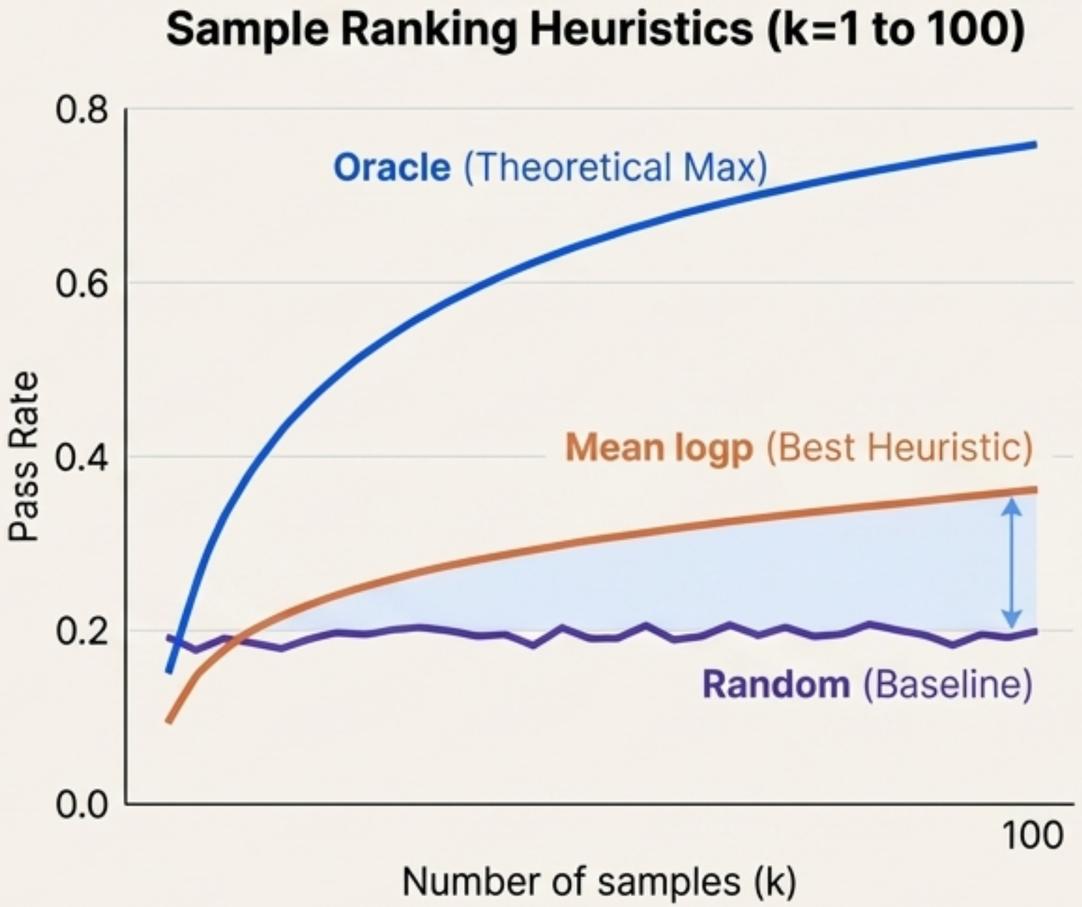
Finding the Needle in the Haystack Without an Oracle

In practice, we don't have unit tests to check every sample. How do we pick the best one?

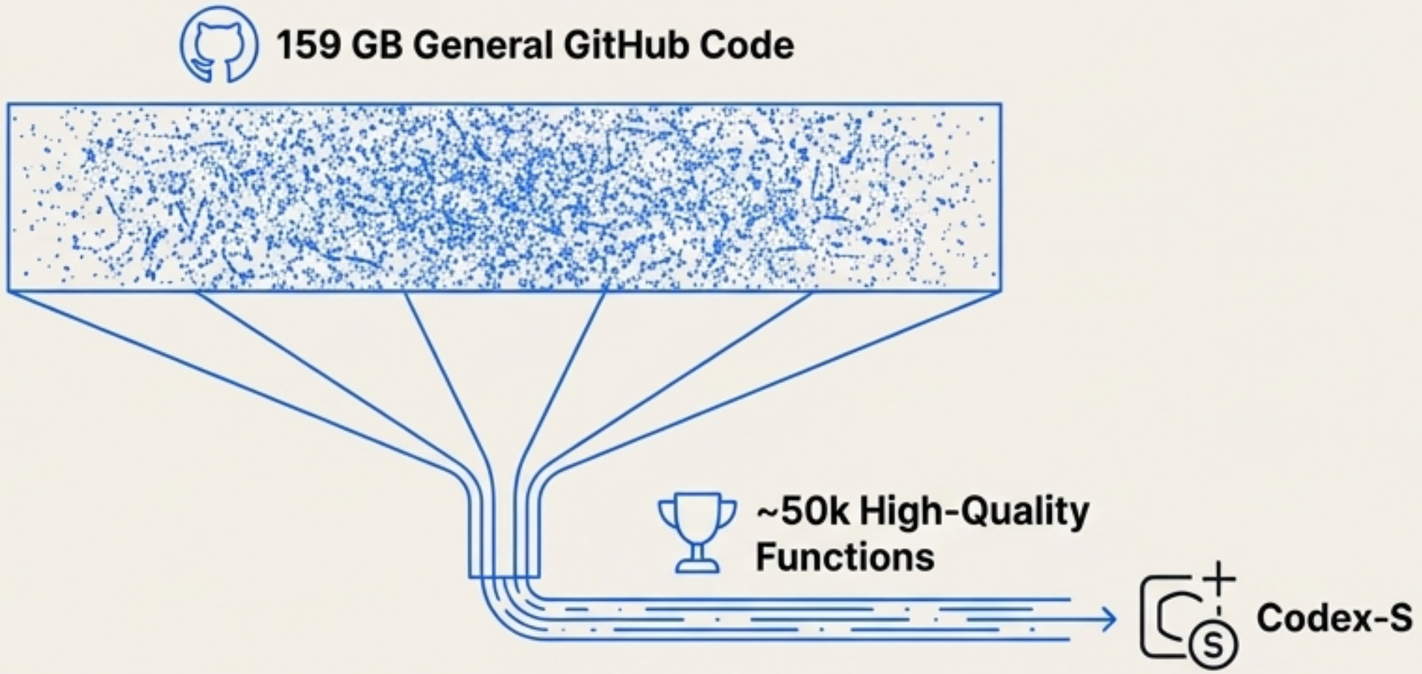
✍️ **Heuristic**: Select the sample with the highest **mean log probability** (the one the model is most “confident” in).

This simple ranking method significantly outperforms random selection.

- For Codex-12B, it boosts performance from what would be a random guess to solving **~35%** of problems, getting much closer to the oracle's performance.



From General Code to Perfect Functions: Meet Codex-S



The initial GitHub dataset is noisy and contains more than just standalone functions.

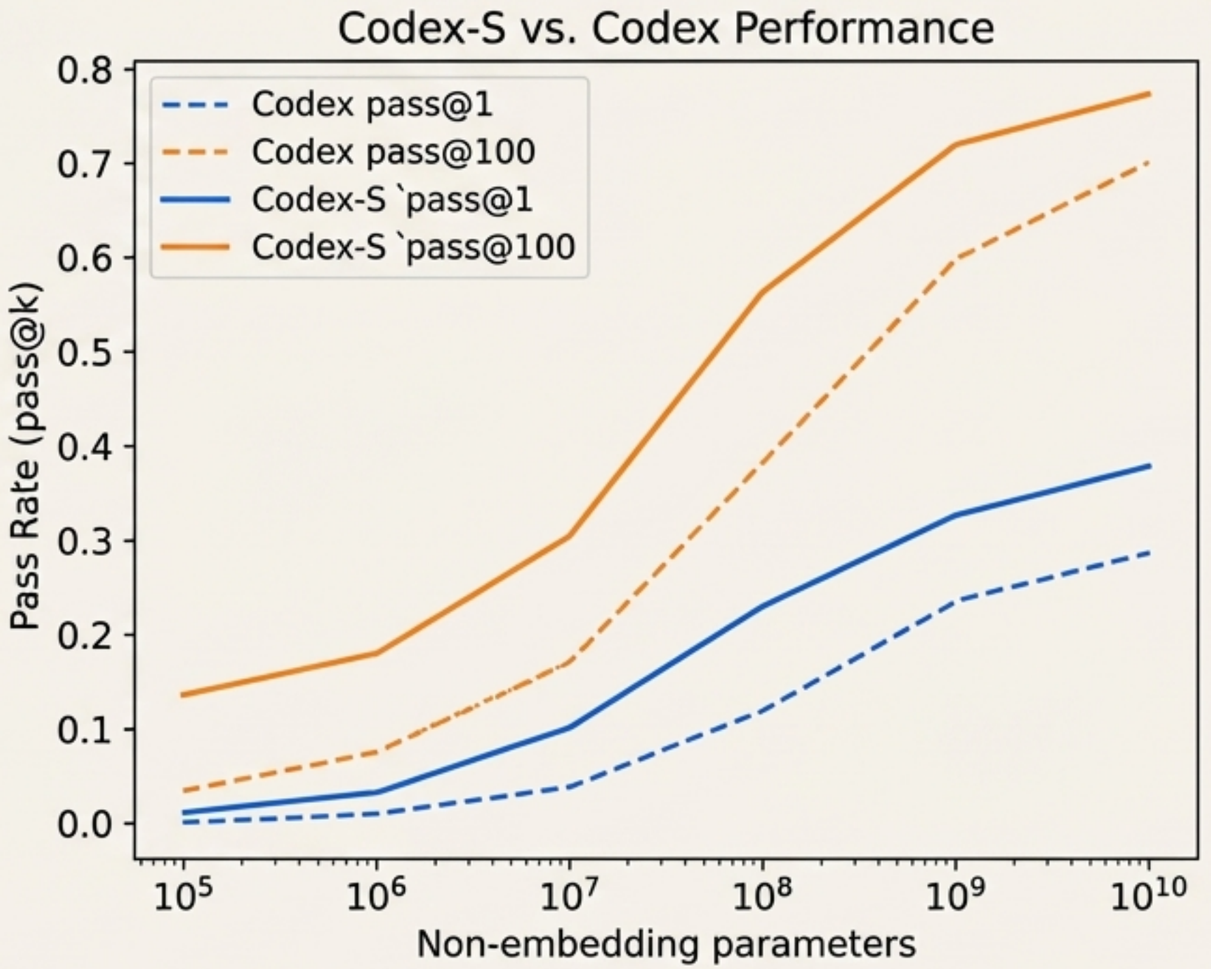
Hypothesis: Fine-tuning on a curated dataset of high-quality, correct functions will improve performance on the specific synthesis task.

Codex-S: The original Codex, further fine-tuned on a dataset created from:
1. Solutions from competitive programming websites.
2. Functions extracted from open-source projects with continuous integration (CI) tests.

Supervised Fine-Tuning Delivers Major Gains

Codex-S significantly outperforms the base Codex across all model sizes.

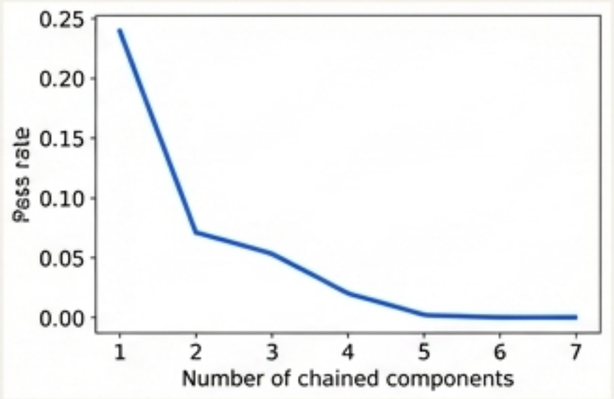
- The 12B Codex-S model achieves **37.7% pass@1** (up from 28.8%).
- With 100 samples, it solves **77.5%** of problems.
- The refinement makes the model much more parameter-efficient: a 300M Codex-S model outperforms a 2.5B base Codex model.



The Model Is Powerful, But Not Perfect

Long Chains of Reasoning

Performance degrades exponentially as the number of sequential steps in a prompt increases.



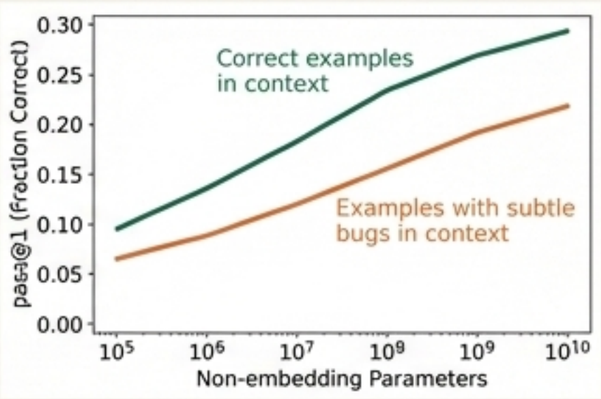
Variable Binding

Can fail to correctly bind operations to variables, especially with complex prompts.

```
do_work(x,y,z,w)
u = x - 4
```

Misalignment

If a prompt contains subtle bugs, the model is more likely to generate buggy code, learning 'bad habits'.



With Great Power Comes Great Responsibility

Deploying powerful code generation models requires careful consideration of their broader impacts.

- **Over-reliance:** Users, especially novices, may trust incorrect or insecure code. Human oversight is essential.
- **Security:** The model can generate insecure code (e.g., using weak cryptography) by learning from patterns in public data.
- **Bias:** Can generate code and comments that reflect societal biases present in the training data.
- **Economic Impacts:** Potential to increase programmer productivity, but also to change the nature of software-related jobs.

A New Era for Human-Computer Collaboration

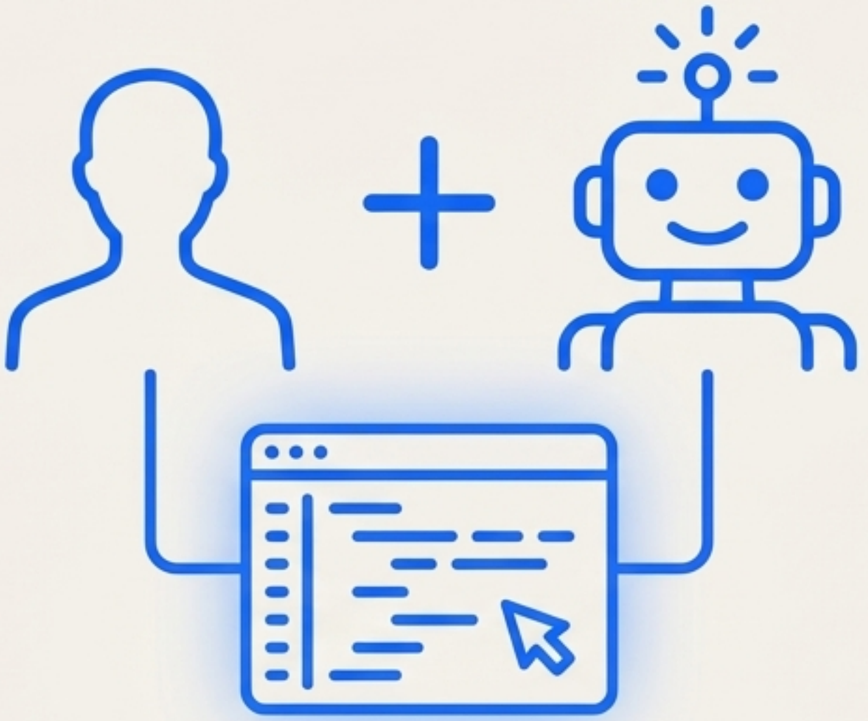
Codex is a major leap in program synthesis, moving from 0% to >70% success on a challenging benchmark.

Key Drivers of Success:

1. “Massive scale of code-specific training data.”
2. “A rigorous evaluation framework (HumanEval).”
3. “The effectiveness of generating and ranking multiple samples.”

This research powers tools like **GitHub Copilot**, changing how developers write software.

Future work lies in improving reliability and ensuring safe, beneficial deployment.



From Code Generation to Code Collaboration