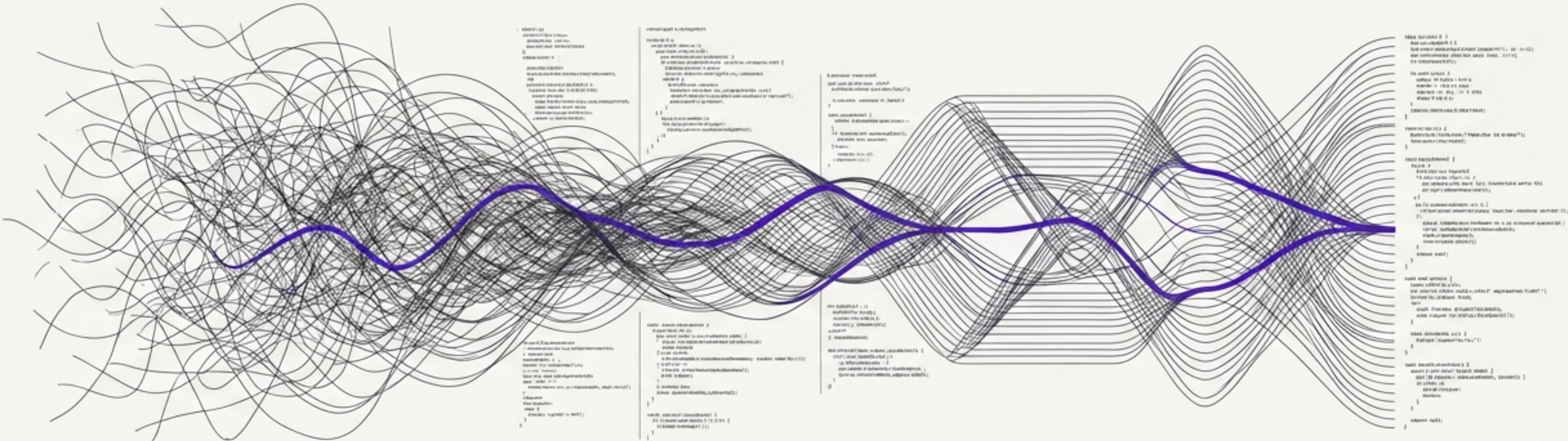
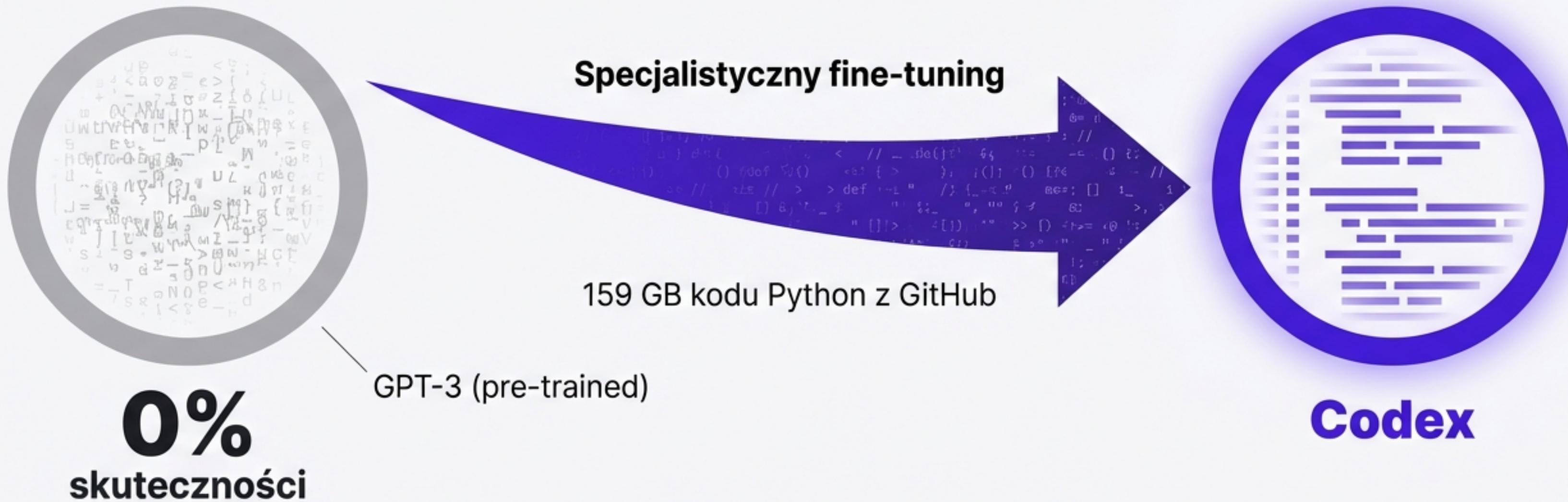


Spełniony Sen: Od Idei do Syntezy Programów



Przepis na Kodującą Umysł: Hipoteza i Trening

Główna hipoteza: Model językowy z rodziny GPT, po specjalistycznym **fine-tuningu** na gigantycznym zbiorze kodu, może osiągnąć biegłość w generowaniu **działającego kodu** z opisów w języku naturalnym (docstringów).



'Wstępnie wytrenowany model GPT' + '**Specjalistyczny fine-tuning na 100 miliardach tokenów kodu**'

Wyzwanie: Jak Zmierzyć "Dobry" Kod?

Ocena kodu różni się fundamentalnie od oceny tłumaczenia tekstu.

Tłumaczenie Tekstu

The cat sat on the mat.



A feline was on the rug.

**Wysoki BLEU score
(podobne znaczenie)**

Generowanie Kodu

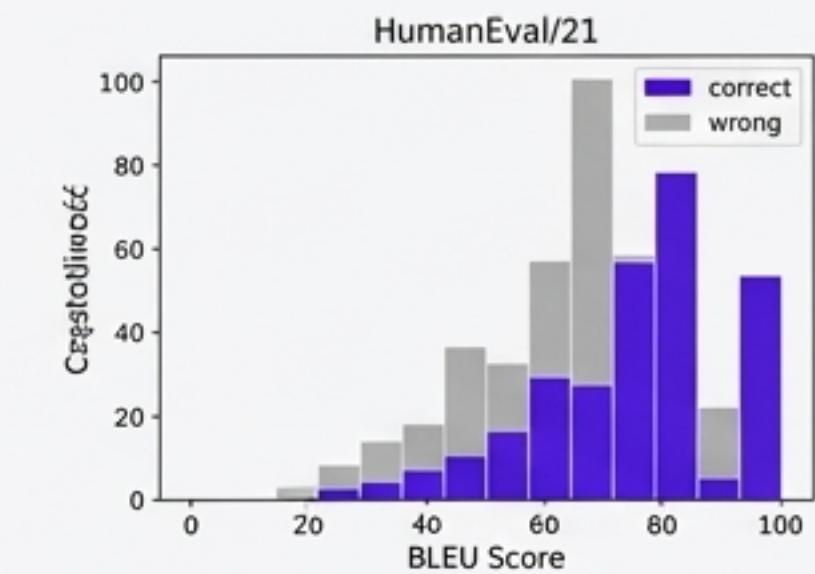
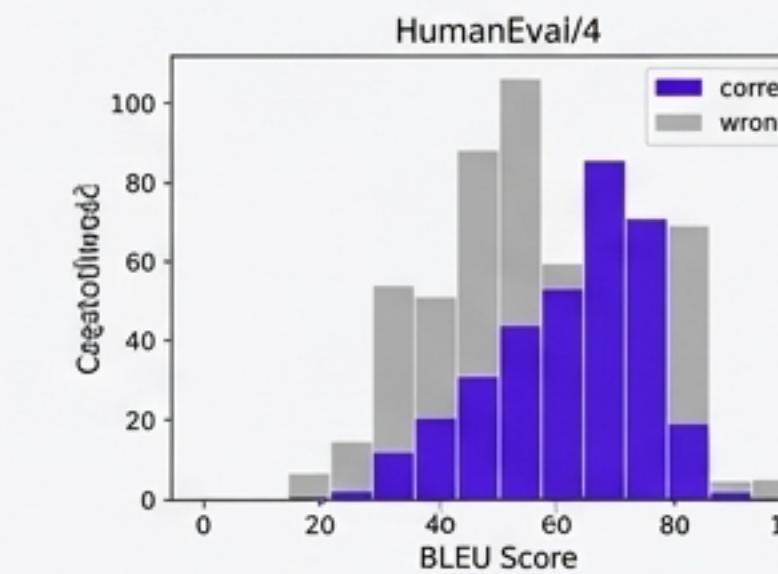
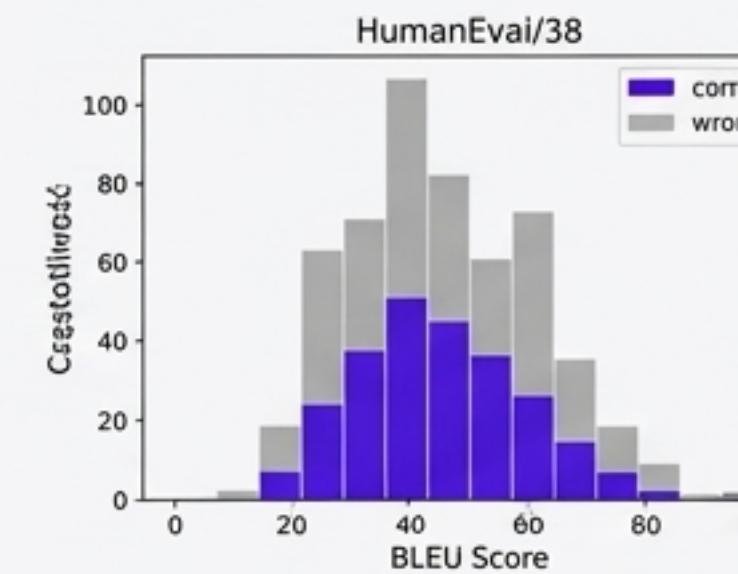
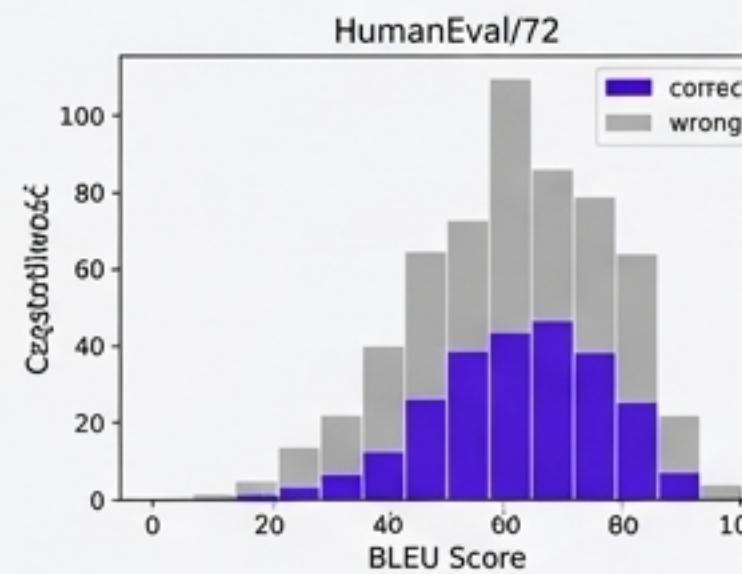
```
def is_even(n):  
    return n % 2 == 0
```



```
def is_even(n):  
    return (n & 1) == 0
```

**Niski BLEU score (różna
składnia), ale 100% ta
sama funkcjonalność**

Mylące wskaźniki: Błędne programy często osiągają wyższy BLEU score niż te poprawne.



Zasada zero-jedynkowa: Kod albo przechodzi wszystkie testy, albo nie. Nie ma punktów za częściowy sukces.

Stworzenie Areny: Benchmark HumanEval

Aby rzetelnie mierzyć poprawność funkcjonalną, zespół OpenAI stworzył od zera dedykowany zestaw testowy.

- **164 ręcznie napisane problemy programistyczne**, stworzone specjalnie na potrzeby tej ewaluacji, aby uniknąć "zanieczyszczenia" danymi treningowymi z internetu.
- **Poziom trudności:** Od prostych operacji po zadania porównywalne z pytaniami na podstawowych rozmowach kwalifikacyjnych.
- **Kompletny zestaw:** Każdy problem posiadał sygnaturę funkcji, docstring (opis w języku naturalnym) oraz zestaw testów jednostkowych (średnio 7.7 testu na 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]
    return [i + 1 for i in l]
```

Ścisła definicja sukcesu: Wygenerowany kod jest uznany za poprawny **tylko i wyłącznie**, jeśli przejdzie **WSZYSTKIE** testy jednostkowe bez wyjątku.

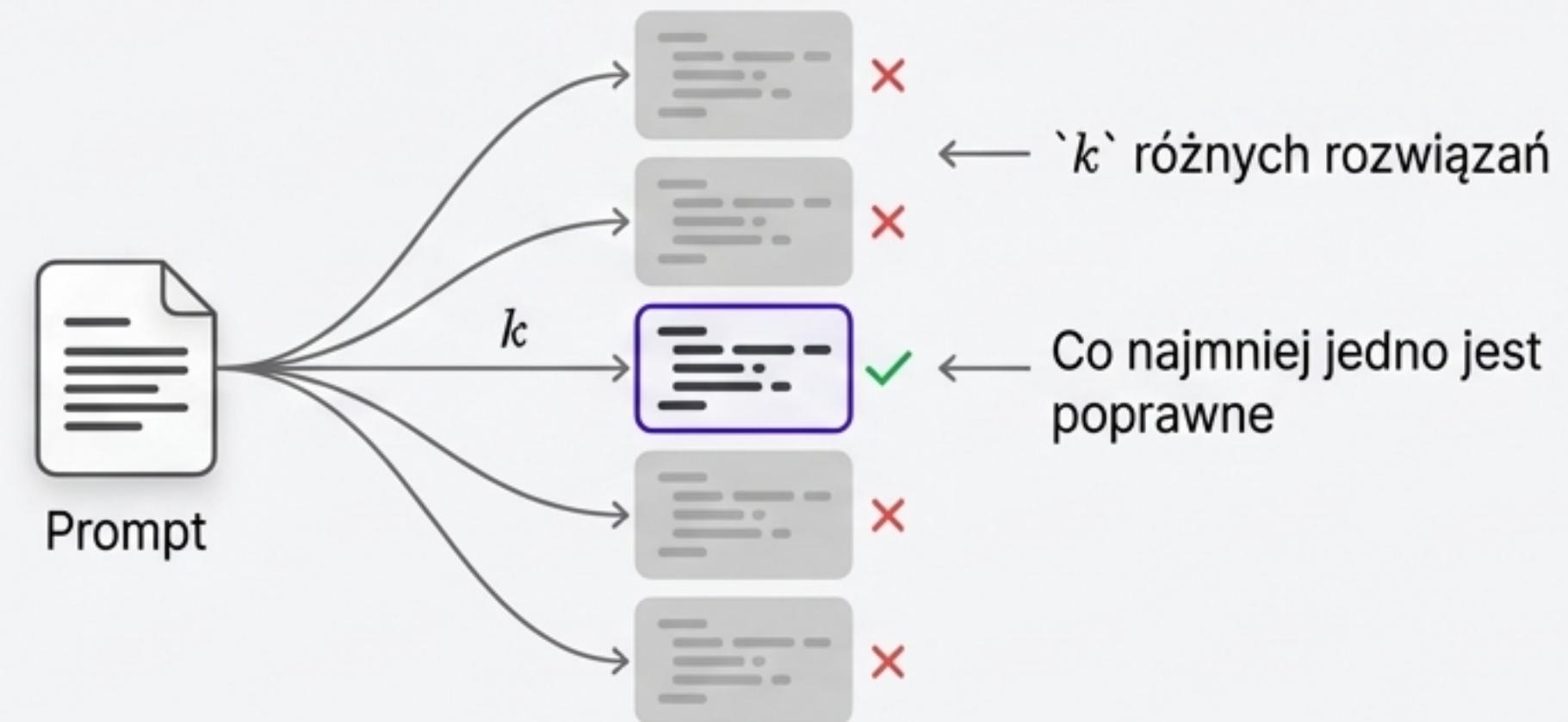
Nowy Standard Oceny: Metryka Pass@k

Tradycyjna ocena 'czy pierwsza próba się udała?' jest niewystarczająca. 'Pass@k' to znacznie mądrzejsze podejście.

pass@k

- **Analogia:** Programista ma ' k ' podejść do rozwiązania problemu. Jaka jest jego szansa na sukces?
- **Co to mierzy?** Nie tylko natychmiastową poprawność, ale zdolność modelu do *odnalezienia* poprawnego rozwiązania.
- **Odwzorczenie ludzkiego procesu:** Programiści rzadko piszą idealny kod za pierwszym razem. 'Pass@k' naśladuje proces iteracji i eksploracji.

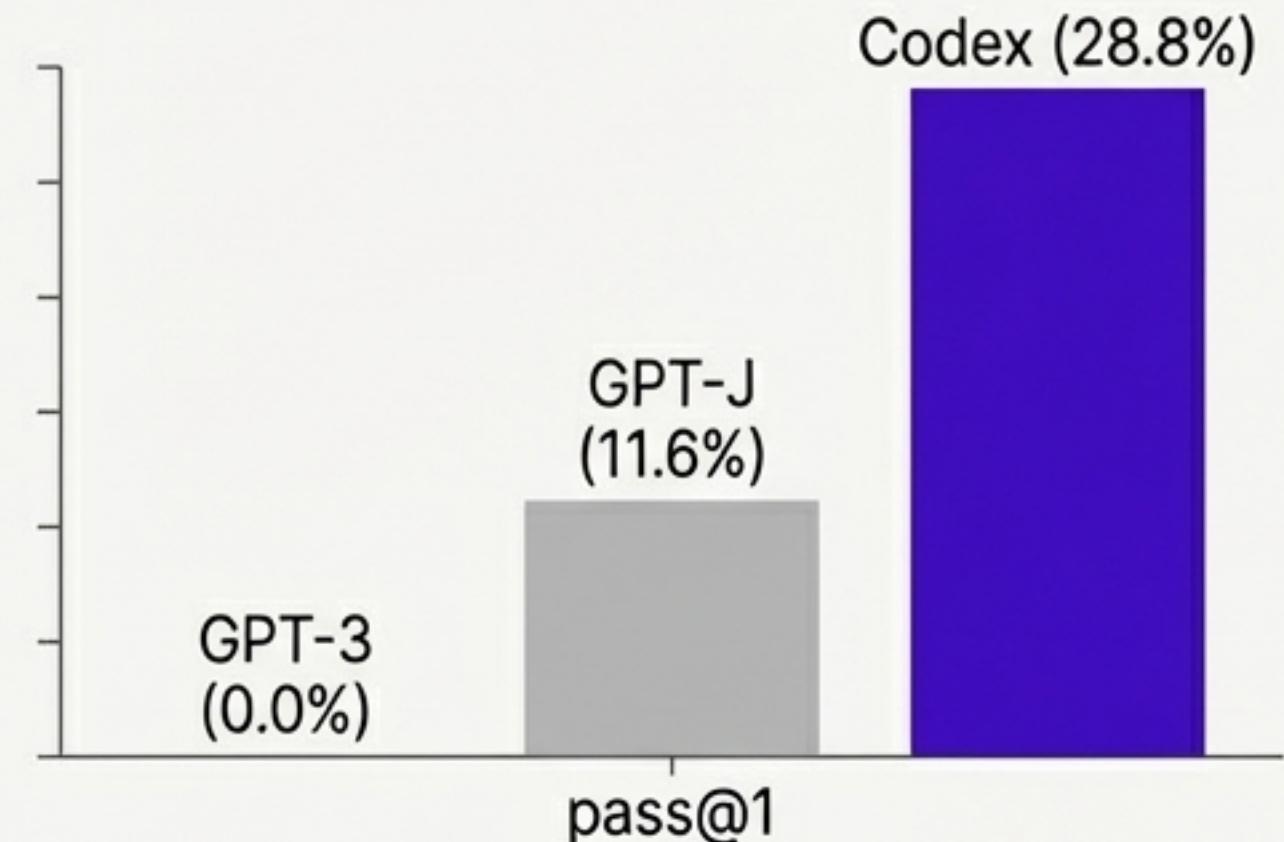
- **Pytanie, na które odpowiada 'pass@k':**
„Jakie jest prawdopodobieństwo, że co najmniej jedna z ' k ' wygenerowanych próbek kodu jest poprawna?”



Przełom w Liczbach: Wyniki na HumanEval

Wyniki jednoznacznie pokazały, że specjalistyczny trening w połączeniu z nową metodą oceny ujawnił prawdziwe zdolności modelu.

Model	pass@1 (Jedna próba)	pass@100 (Sto prób)
GPT-3 (175B)	0.0%	0.0%
GPT-J (6B)	11.6%	27.7%
Codex (12B)	28.8%	72.3%



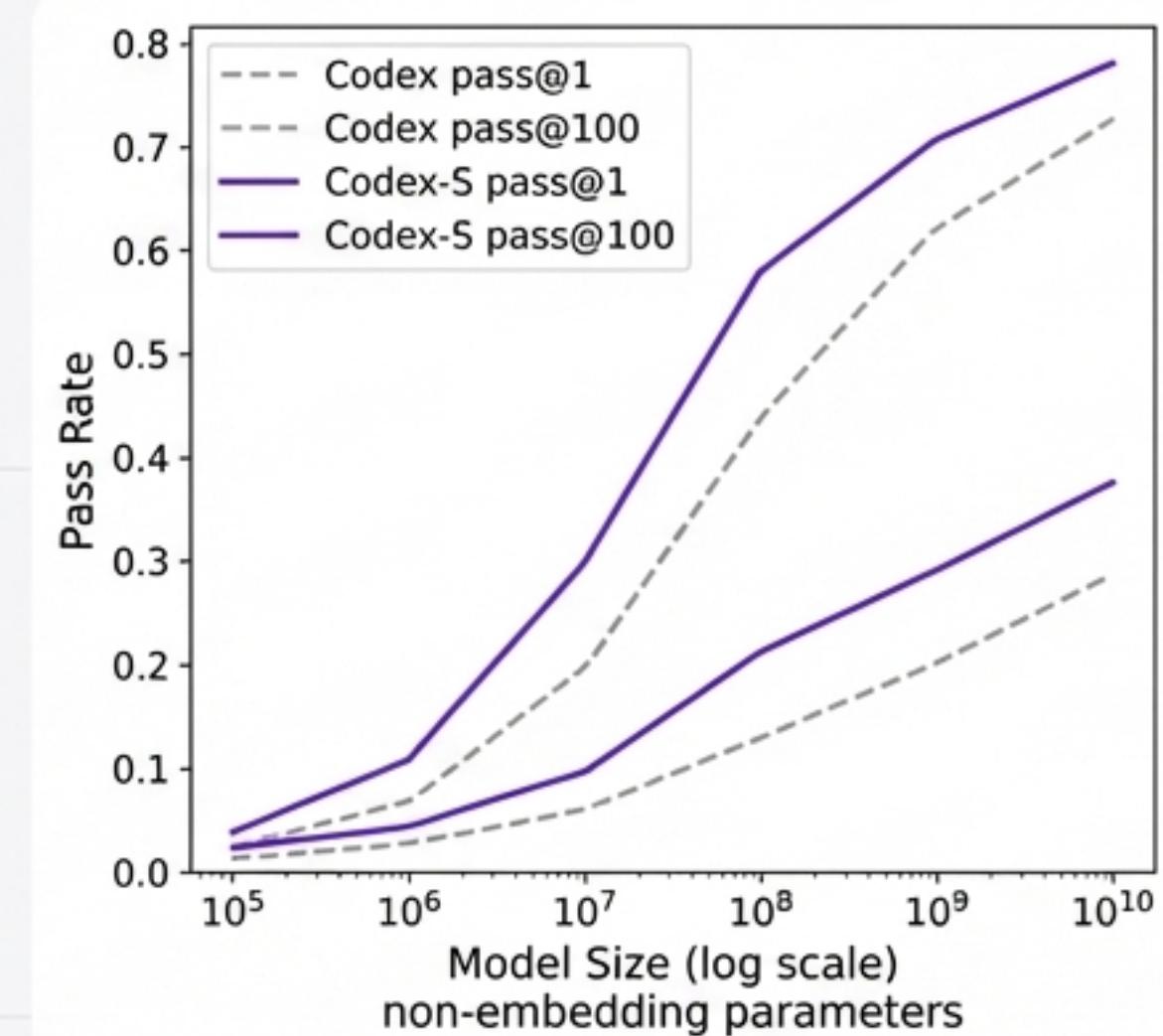
- **Całkowita porażka GPT-3:** Brak specjalizacji uniemożliwił rozwiązywanie jakiegokolwiek problemu.
- **Gigantyczny skok:** Przejście z 11.6% (GPT-J) do **28.8%** (Codex) w `pass@1` to ogromny postęp w tej dziedzinie.
- **Ukryta wiedza:** Wynik **72.3% przy pass@100** sugeruje, że model niemal na pewno "zna" odpowiedź, ale potrzebuje kilku prób, by ją poprawnie sformułować.

Jakość ponad ilość: Dalsza Specjalizacja w Codex-S

Zespół zadał pytanie: co jeśli douczymy model na *idealnych* przykładach? Tak powstał **Codex-S** (Supervised).



Model	pass@1	pass@100
Codex (12B)	28.8%	72.3%
Codex-S (12B)	37.7%	77.5%



Wniosek: Jakość danych treningowych jest kluczowa. Im bardziej dane przypominają docelowy problem, tym lepsze wyniki.

Strategia Generowania: Kreatywność pod Kontrolą

“Temperatura” to kluczowy parametr kontrolujący losowość (kreatywność) generowanych odpowiedzi. Badania pokazały, że optymalna strategia zależy od celu.

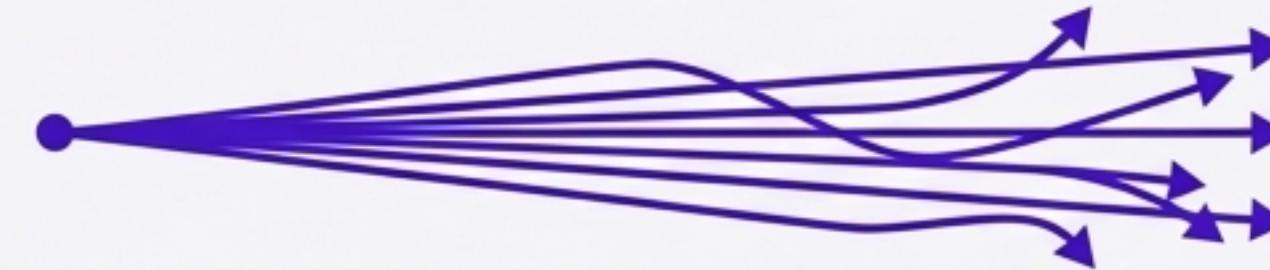
Niska temperatura (np. 0.2)



- Model jest **konserwatywny**. Generuje najbardziej prawdopodobne, “bezpieczne” odpowiedzi.
- **Optymalna dla ‘pass@1’** (gdy liczy się tylko pierwsza, najlepsza próba).

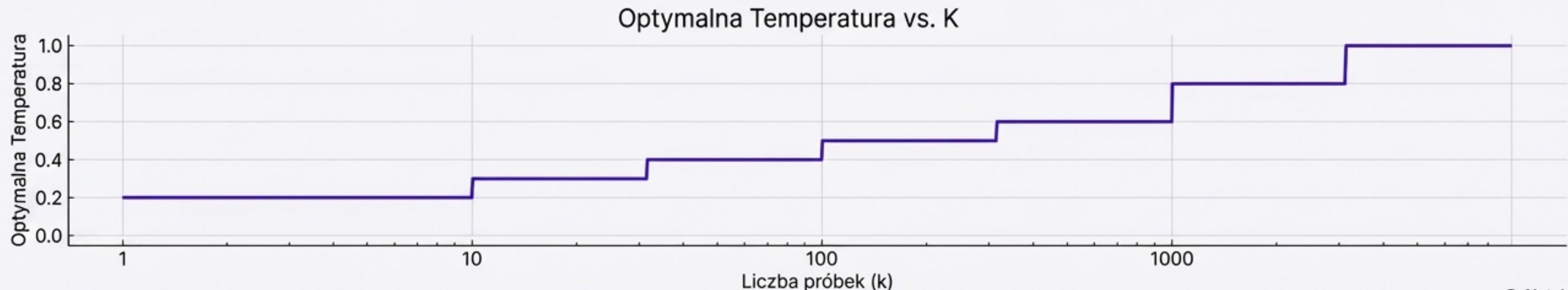
„Daj mi najbardziej niezawodne rozwiązanie.”

Wysoka temperatura (np. 0.8)



- Model jest **kreatywny**. Generuje bardziej zróżnicowane, czasem nieoczywiste rozwiązania.
- **Optymalna dla ‘pass@k’ przy dużym ‘k’ (np. 100)** (gdy chcemy zbadać jak największą liczbę ścieżek).

„Nie trzymaj się utartych szlaków, spróbuj czegoś nieoczekiwanej!”



Świadomość Ograniczeń: Gdzie Codex Nadal Zawodzi?

Mimo przelomowych wyników, Codex nie jest nieomylny. Autorzy rzetelnie wskazują na jego kluczowe słabości.

1. Długie, wieloetapowe instrukcje:

Skuteczność modelu spada wykładniczo z każdym kolejnym krokiem w opisie zadania. Dobrze radzi sobie z prostymi poleceniami, ale gubi się w złożonych specyfikacjach.

2. Problemy z wiązaniem zmiennych (Variable Binding):

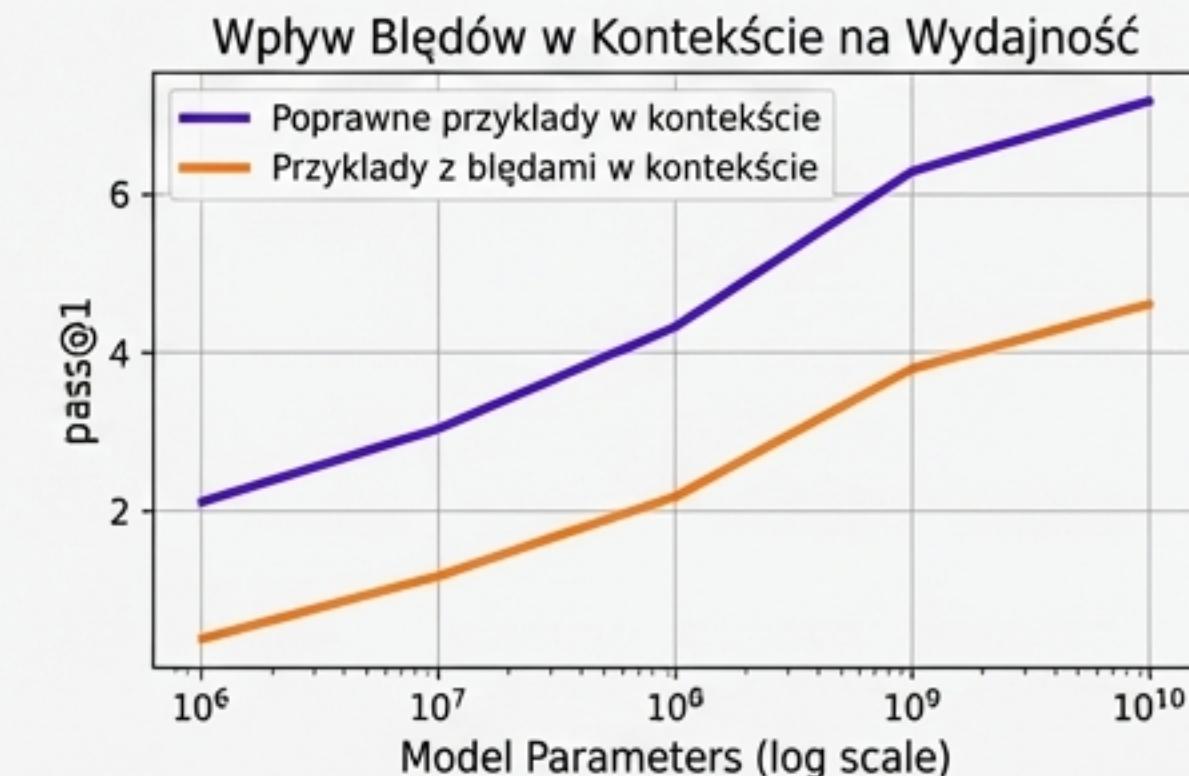
Model potrafi pomylić, która operacja dotyczy której zmiennej, zwłaszcza przy wielu zmiennych w kontekście.

```
# Polecenie: "Dodaj 3 do y, następnie odejmij 4 od x oraz w.  
# Zwróć iloczyn czterech liczb."
```

```
# Błędna generacja Codexa:  
def do_work(x, y, z, w):  
    t = y + 3  
    u = x - 4  
    v = z * w # Błąd: nie odjęto 4 od w  
    return v # Błąd: zwrócono tylko iloczyn dwóch liczb
```

3. Podatność na błędy w kontekście:

Jeśli w podanym kontekście (prompt) znajdują się subtelne błędy, wydajność modelu spada. Model ma tendencję do naśladowania 'złych nawyków'.



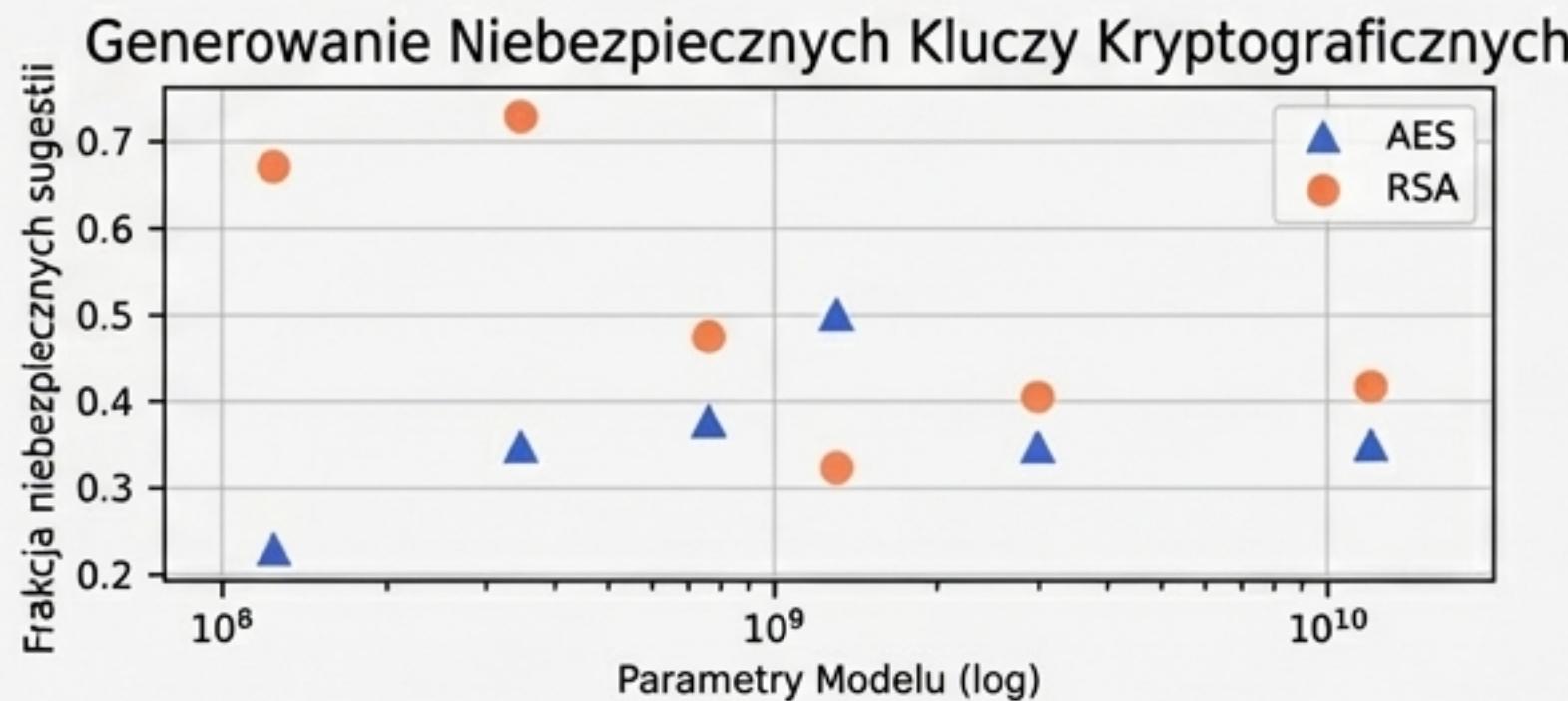
Wielka Moc, Wielka Odpowiedzialność: Wpływ i Ryzyka

Wdrożenie tak potężnej technologii niesie ze sobą nie tylko korzyści, ale i poważne wyzwania, którym autorzy poświęcają obszerny rozdział.

Potencjalne Korzyści



- Zwiększenie produktywności programistów.
- Obniżenie bariery wejścia do świata kodowania.
- Nowe narzędzia edukacyjne i eksploracyjne.



Główne Zidentyfikowane Ryzyka



- **Nadmierne zaufanie (Over-reliance):** Użytkownicy mogą bezkrytycznie akceptować kod, który zawiera subtelne błędy lub luki bezpieczeństwa.
- **Niezgodność z intencją (Misalignment):** Model może powielać błędy obecne w kodzie-kontekście, optymalizując pod kątem statystycznego podobieństwa.
- **Bezpieczeństwo:** Generowanie kodu podatnego na ataki, np. sugerowanie przestarzałych konfiguracji kryptograficznych (klucze RSA < 2048 bitów).
- **Uprzedzenia (Bias):** Model może odtwarzać stereotypy obecne w danych treningowych (np. w komentarzach).

> "Konkluzja:" Codex to potężne narzędzie, ale wymaga ludzkiego nadzoru, krytycznego myślenia i świadomości jego ograniczeń. To asystent, a nie autonomiczny ekspert."