AI Can Code.
So Why Should You Learn CS?

Dr. Bernard Nongpoh
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati

World Creativity and Innovation Day
St. Edmund's College, Shillong
April 18, 2026

If AI can write code...

. . . why are you here?

AI-generated code

```
def sort_and_search(arr, target):
  # AI-generated: sorts array, binary searches
  arr = sorted(arr)
  lo, hi = 0, len(arr) - 1
  while lo <= hi:
    mid = (lo + hi) // 2
    if arr[mid] == target: return mid
    elif arr[mid] < target: lo = mid + 1
    else: hi = mid - 1

  return -1
```

Looks correct. Runs fine.

But is it actually correct?

Is it **secure**?

Can you **prove** it?

Let's start from the beginning.

What is Generative AI?

The Core Idea

Generative AI = **autocomplete on steroids**

How it “generates”

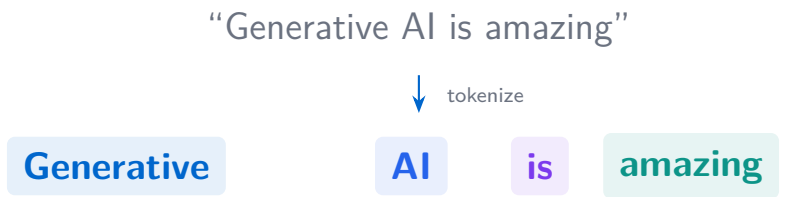
The cat sat on the --> **mat**

mat	72%
floor	15%
roof	8%
table	5%

It predicts the **most probable** next word.

Then the next. Then the next. One word at a time.

Jargon Decoded — Tokens



Tokens = the **atoms** of language for AI

Words, sub-words, or characters that the model reads and writes.

Jargon Decoded — Parameters

Each knob = one parameter. A model has **billions** of these.

Parameters = **knobs the model tunes** during training

Think of it this way:

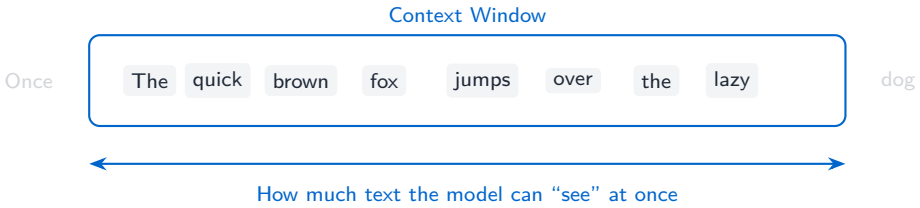
GPT-5 ≈ **1.8 Trillion** parameters

Llama 4 ≈ **400 Billion** parameters

More parameters ≈ more patterns the model can learn

Source: BuildFastWithAI, "Best AI Models, SiliconAngle, Apr. 2026

Jargon Decoded — Context Window



Like a **reading window** that slides over text

GPT-5: 128K tokens ≈ 300 pages | Qwen 3.5: 262K tokens ≈ 650 pages

Source: BentoML, “The Best Open-Source LLMs in 2026,” Apr. 2026

Jargon Decoded — Temperature

Temperature = 0.1

(Focused / Predictable)

Temperature = 1.5

(Creative / Random)

Temperature = how **creative vs predictable** the output is

Low = safe, repetitive | High = surprising, risky

The Transformer

The engine behind every frontier model

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Source: Vaswani et al., "Attention Is All You Need," NeurIPS 2017

The Frontier Models

April 2026

Closed Source

- GPT-5.4 OpenAI
- Claude Opus 4.6 Anthropic
- Gemini 3.1 Pro Google
- Grok 4.1 xAI

VS

Open Source

- Llama 4 Meta
- DeepSeek V3.2 DeepSeek
- Qwen 3.5 Alibaba
- Mistral Large 3 Mistral

Source: BuildFastWithAI, Apr. 2026; Digital Bricks, "The Age of Frontier Intelligence," Jan. 2026

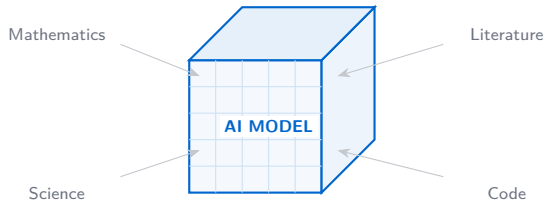
The gap between them?

$\sim$3 months

Open-source models now trail frontier closed models by only about three months on average.

Source: Epoch AI, cited in BentoML, "The Best Open-Source LLMs in 2026," Apr. 2026

The Knowledge Cube



A single artifact containing **all human knowledge available on the web**.

It feels like an oracle. But it's not.

The Crack in the Cube

Hallucinations
Confidently wrong

Subtle Bugs
Looks right, isn't

Security Flaws
Invisible dangers

The model doesn't **understand** correctness.

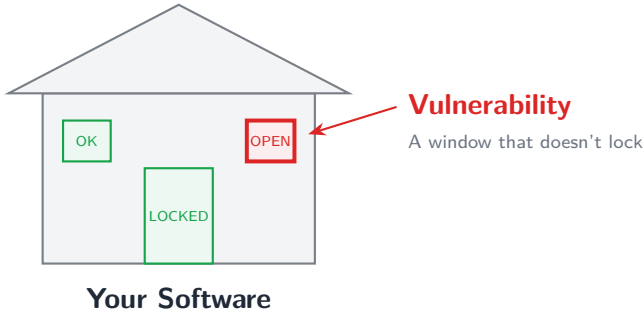
It predicts **plausibility**.

What happens when plausible-looking code

has a security vulnerability?

What is a Vulnerability?

Vulnerability = A Flaw in Software



A vulnerability is a **weakness** or **flaw** in software that *could* be used to cause harm.

Common Vulnerability Types

Buffer Overflow

Writing data beyond allocated memory

Race Condition

Two processes clash over shared resource

Logic Error

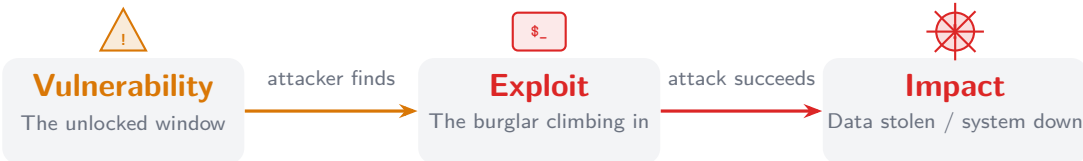
Code does the wrong thing "correctly"

These are **unintentional** flaws. The developer didn't mean to leave them.

But attackers *hunt* for exactly these.

What is an Exploit?

Exploit = Weaponizing the Flaw



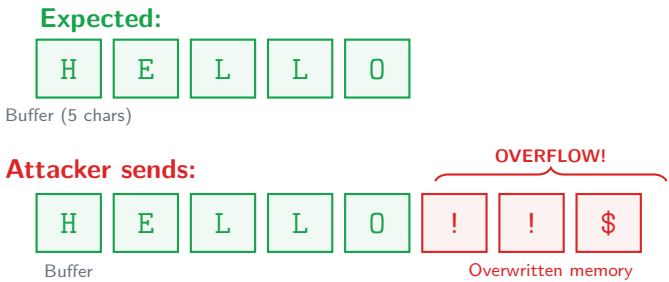
The vulnerability is the **unlocked window**.

The exploit is the **burglar climbing through it**.

The impact is **everything they steal**.

Example: Buffer Overflow

The classic vulnerability



Data spills beyond its boundary → attacker can **inject code**

What is a Zero-Day?

Zero-Day Vulnerability

A zero-day is a vulnerability **unknown to defenders**.

Zero time to prepare. **Zero** patches available.

NOW IMAGINE THIS . . .

What if an AI could find AND exploit zero-days automatically?

APRIL 7, 2026

Claude Mythos Preview

Anthropic

What Mythos Preview Did

❀ Found zero-days in **every major OS**

● Found zero-days in **every major browser**

✻ Found a **27-year-old** bug in OpenBSD

➪ Chained **4 vulnerabilities** into one browser exploit

✦ All done **autonomously**. No human guidance.

Source: Anthropic Red Team, "Claude Mythos Preview," red.anthropic.com, Apr. 2026

How It Works

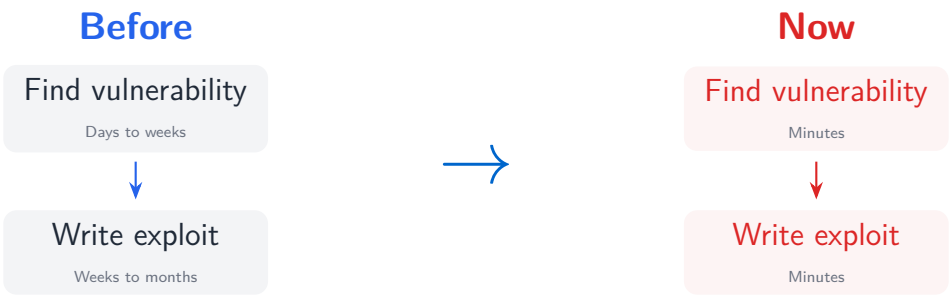
Surprisingly simple setup

Reads source → forms hypotheses → runs software

→ uses debuggers → produces proof-of-concept exploit

Source: Anthropic Red Team, “Claude Mythos Preview,” red.anthropic.com, Apr. 2026

The New Reality



Thousands of vulnerabilities found. Fewer than **1%** patched so far.

Source: Tom's Hardware, "Claude Mythos Preview sparks race to fix critical bugs," Apr. 2026

Project Glasswing

Anthropic's response: use AI to defend, not just attack

$100M in credits + $4M in direct donations to open-source security

Source: Anthropic, "Project Glasswing," anthropic.com/glasswing, Apr. 2026

If AI can find vulnerabilities in every OS and browser...

How do we write software we can actually trust?

Testing

Testing

"I tried many inputs and it worked"

✓ ✓ ✓ ✓ ✓ ✓ ✗ missed!

Tested 1000 inputs. **All passed.**

Input 1001 **crashes the system**.

Proving

Formal Verification

"I proved it works for ALL inputs, mathematically"

✓ Every possible input. Every edge case. **Proven.**

Analogy:

Testing = checking every door in a building is locked

Proving = proving the **blueprint makes break-in impossible**

Testing vs Proving

Testing

Checks **some** inputs

Can miss edge cases

"Probably works"

Easier but weaker

Formal Verification

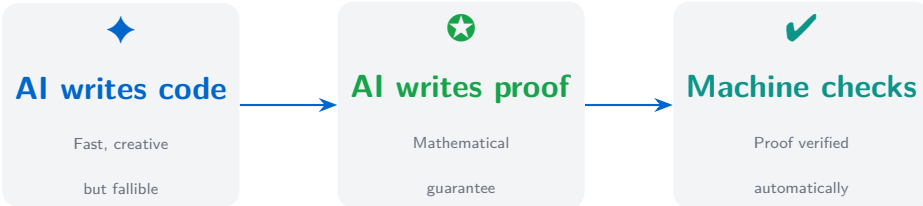
Proves for **all** inputs

Covers every edge case

"Mathematically correct"

Harder but **bulletproof**

The Future: AI + Formal Verification



Result: Code that is **proven correct**. Not just tested.

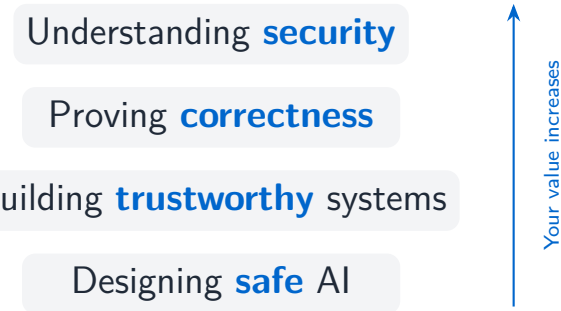
Source: VentureBeat, "Lean4: The New Competitive Edge," Dec. 2025

AI raises the **floor**.

Anyone can generate code now.

But it doesn't raise the ceiling.

The Ceiling Is Yours



These are the skills AI **cannot** replace.

Because they require **understanding**, not just generation.

Your career in CS
has never been more important.

The world doesn't need more code generators.

It needs people who can **prove the code is right**.

Thank You

Questions?