Systems Research Challenges Workshop 2026

10th Anniversary Keynote

Has AI Just Made Us All Irrelevant?

Mark Little, IBM/Red Hat

Meet AI, AI Can Do Anything

How LLMs Are Trained & Work

Training the LLM

Books • Websites • Articles • Code • More

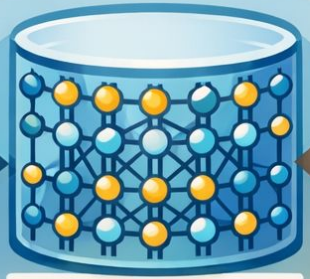
Pattern Recognition

Learning Language Patterns

Model Optimization

Adjusting Weights

Training



Large Language Model

Inference

Using the LLM

User Prompt

Processing & Generation

Understanding & Generating Text

Model Output

Answers, Essays, Code & More

Applications

Chatbots

Search

Translation

Content Creation

The Architecture vs. Syntax Divide

AI excels at writing code, but designing and building Systems requires something more

WHAT AI DOES WELL

Syntax-Level Tasks

Writes individual functions

Generates self-contained code blocks quickly

Generates syntax quickly

Produces boilerplate and standard patterns at speed

Pattern-matches from training data

Leverages vast repositories of existing code

WHAT AI STRUGGLES WITH

Architecture-Level Challenges

Complex system design

Designing distributed architectures with many interacting parts

Avoiding conflicts with existing patterns

New code must align with established architectural decisions

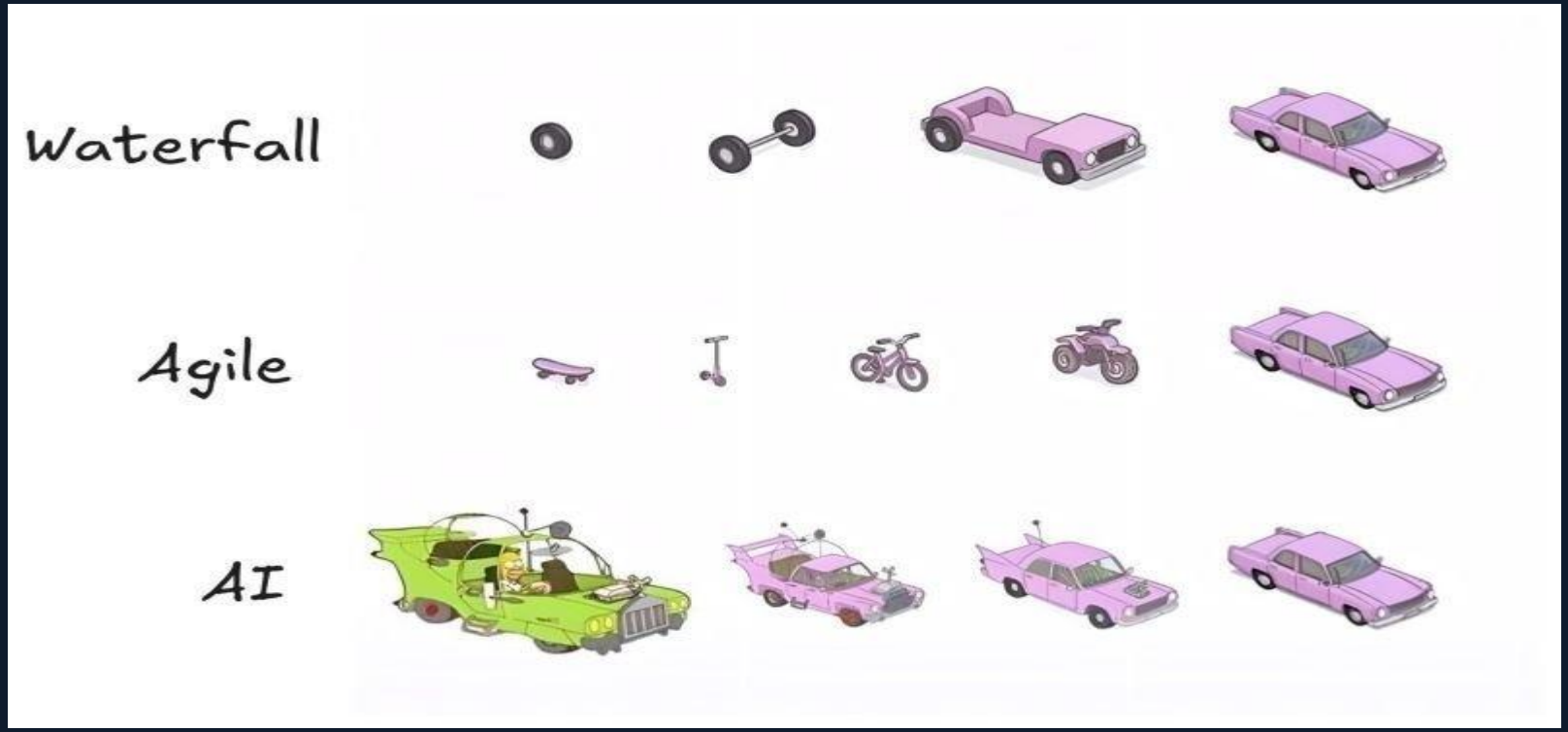
Preventing bloated codebases

Avoiding unnecessarily repeated logic across modules

Long-term maintainability

Bug fixing, support and feature enhancements over years

Context is important!



https://x.com/mrdoob/status/2040586168132366449

AI Knows How to Program, But Not Why

Cannot Understand Your Pain Points

AI cannot attend meetings, read a room, or understand user frustrations and the real problems behind their requests.

Cannot Predict User Impact

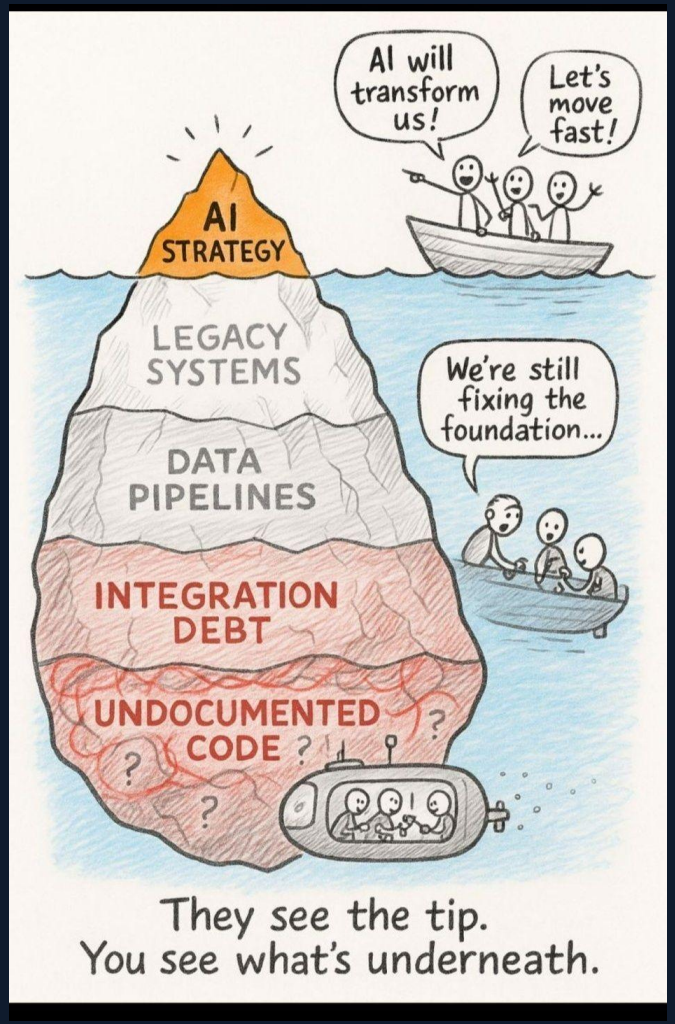
AI cannot predict how a new feature might impact a specific user demographic or anticipate how real people will interact with changes.

Cannot Assess Existing Functionality

AI cannot assess how changes will ripple through an existing codebase, potentially breaking dependencies and destabilising proven systems.

Operates on Patterns, Not Understanding

AI operates purely on statistical pattern matching: it has no strategic awareness, or domain intuition.



https://x.com/kmcnam1/status/2040112471134851513/photo/1

When AI Code Goes Wrong, Who Is Accountable?

No Consequences on the AI for AI Failures

If an AI-written script crashes a bank's server or leaks user data, the AI doesn't face consequences, leaving organizations exposed with no clear accountability.

No Ethical Framework (The Dark Star Effect)

AI lacks an ethical framework for decision-making. It cannot weigh moral implications, social consequences, or the human impact of the code it generates.

Biased Training Data

AI may inadvertently use biased data during code generation, embedding discriminatory patterns or skewed logic into critical business systems without detection. Example: Framework Bias.

Intellectual Property Risks

AI may violate intellectual property rights when generating code, reproducing licensed or copyrighted code fragments, creating legal liability for the organization.

Distributed Systems Are **Complex**

High-Performance Messaging

Pub/sub systems, event streaming and real-time message brokers requiring sub-millisecond latency

Databases & Persistence

ACID transactions, replication, sharding and query optimization across distributed data stores

Caching Subsystems

Cache invalidation strategies, multi-layer caching and consistency guarantees under high throughput

Distributed Nature

Consensus protocols, fault tolerance, service orchestration and five-nines uptime guarantees

There Isn't Enough (Good) Training Data

Up to 45%

of AI-generated code snippets contain **security vulnerabilities** or reference non-existent libraries (hallucinations)

2,000–10,000

minimum **high-quality labeled examples** needed for useful fine-tuning of an AI model

20,000–100,000+

curated examples plus evaluation sets needed for **high-trust deployment** in production

The situation is significantly worse for application infrastructure code: domain-specific training data for high-performance messaging, distributed databases and caching systems is extremely scarce.

Three Reasons Human Experts Are Essential

1

Infrastructure Is Deeply Complex

Application infrastructure is more complex than other types of software, designing, building, integrating and supporting it requires expert knowledge and experience.

2

Insufficient Training Data

There is not enough training data to enable low-error, AI-driven coding of application infrastructure, the domain-specific data simply doesn't exist at scale.

3

High Cost of Errors

The cost of an error can be very high as application infrastructure is critical, failures can cascade across entire systems and organizations.

Expert Developers and Researchers Translate Problems into Solutions

Bridging the gap between user needs and technical execution requires human judgment at every stage.

1

Understand the Why

Expert developers understand the reasoning behind the code: the goals, constraints and human context that no model can infer on its own.

2

Translate Requirements

They convert human problems into precise technical requirements, structuring ambiguity into actionable architecture, APIs and data models.

3

Make Trade-offs

They make informed trade-offs, balancing performance, cost, security and maintainability in ways impossible for AI operating on statistical patterns alone.

→ AI can write code, but only humans can decide *what to build, why it matters and what to sacrifice.*

What Would It Take?

> "For an LLM to be trustworthy at expert level in designing high-performance distributed databases with ACID + five-nines uptime, you are realistically looking at **50,000–500,000+** extremely high-quality, expert-level training examples. And even then, you **should not trust the model** without strong human review, formal verification, and automated testing. This is **one of the hardest software domains that exists.**"

ChatGPT assessment on training requirements

50K–500K+
TRAINING EXAMPLES REQUIRED

Expert-Level
QUALITY STANDARD PER EXAMPLE

Still Not Enough
HUMAN REVIEW STILL ESSENTIAL

Why We Should Use AI

Boost Productivity

Automate code reviews, debugging and testing, freeing 50-70% time for innovative system design.

Accelerate Discovery

Analyze massive logs/traces, uncover performance bottlenecks, predict system failures faster.

Efficiency Gains

Efficiency Gains: Optimize compilers, schedulers and distributed systems simulations at scale; cut prototyping costs.

Interdisciplinary Insights

Integrate hardware metrics, network data for resilient cloud/edge architectures.

Remember these?

Compilers

“Developers won't understand computer architecture.”

IDEs

“Developers will become reliant on ‘Intellisense’ and forget syntax.”

Garbage Collection

“If you don’t manually free() your memory, you’re building bloated, inefficient software and you’ll never understand how a heap actually works.”

StackOverflow & Copy/Paste

“Developers are bypassing the ‘productive struggle’. Instead of debugging a problem to understand the root cause, they’re just finding the highest-voted answer and moving on without learning why it worked.”

THE REAL QUESTION FOR 2026

Can We **Trust** This Code in Production**?**

In 2026, the question has shifted from "Can AI code?" to **"Can we trust this code in production?"**

Lightning-Fast Syntax

AI can generate syntax at lightning speed, producing code faster than any human developer.

Missing Critical Judgment

But it lacks contextual judgment, architectural vision and accountability for the code it produces.

Only Experts Deliver All Three

Only expert human developers provide contextual judgment, architectural vision, and true accountability.

Areas of AI Systems Research

Hallucinations & Reliability

LLMs generate plausible but inaccurate info, especially in specialized domains like law or medicine.

Reasoning & Generalization

Struggle with complex reasoning, planning, abstraction, and out-of-distribution scenarios.

Scalability & Efficiency

High energy/compute demands; inference bottlenecks in memory, interconnects, and long contexts.

Data & Bias Issues

Data exhaustion, imbalances, domain mismatches; challenges in continual learning and knowledge updating.