CHAT GPT?

WHEN I WAS YOUR AGE
I HAD TO WRITE MY OWN CODE

Slides, code, cheatsheet

Good Vibes Only
Coding in the age of LLMs

Derek Riley, Ph.D.

Professor, Program Director of Computer Science

Milwaukee School of Engineering

Example...

Elevators are part of most modern commercial buildings and have complex control systems that must ensure safety, reliability, performance, and maintainability. To determine whether the control systems that operate elevators work will meet the expectations of the building owners, simulation can be used to test virtual scenarios. You will be creating an application that will take configuration csv files for a building, set up virtual elevator banks, and allow the user to configure various simulations. The simulations will generate logging data that will be saved to csv files that can be analyzed and allow for scenarios to be adjusted and replayed. The application should have a GUI and allow the user to interact with the elevator system in an intuitive way to test and evaluate the control system. The GUI should include buttons to show and allow the user to interact with the buttons inside and outside the elevator.

Please build a simulation tool as described with Python. If you need to see any files in whole or part, let me know the terminal commands. I'll run it and paste the results.

Let's take a look at what was generated by Claude Code...

Code Assessment

- Tool: Claude Sonnet 4
- ~4,500 lines generated
- Less than 5 minutes

- **AI Evaluation: Maintainability**
    - **Code quality**: Consistent naming and logging; reasonable type hints, but several public APIs still return loose dict values. Some magic numbers (e.g., door timing) are hardcoded in models/elevator.py.
    - **Encapsulation**: simulation/simulator.py exposes controller internals (_passengers) via a property. Provide read-only methods on the controller instead of exposing private fields.
    - **Duplication/Dead code risk**: Two GUI stacks exist: gui/main_window.py (current) and gui/gui.py (alternate, references ElevatorSimulation, ConfigManager not present). This likely confuses contributors and increases maintenance burden. Consolidate or explicitly deprecate/remove the legacy GUI.
    - **Testing**: run_tests.py provides smoke checks but no unit tests. Add pytest tests for models (elevator motion, requests, floor queues) and controllers (dispatch algorithms).

- **AI Evaluation: Professionalism**
    - **Docs**: README.md is helpful but brief and includes a generator note ("generated by Claude …"). Remove that, add setup instructions (Python version, venv), usage examples, and screenshots/GIFs. Close code fences correctly.
    - **Project hygiene**: Missing license, contribution guide, code style and tooling config. Add LICENSE, pyproject.toml (or requirements.txt), ruff/flake8 + black + mypy configs, and a minimal CI (GitHub Actions) to run lint/type/test.
    - **Consistency**: Good naming and structure overall. Avoid mixing private attribute access across modules. Replace emoji in run_tests.py prints with plain text for CI-friendliness.
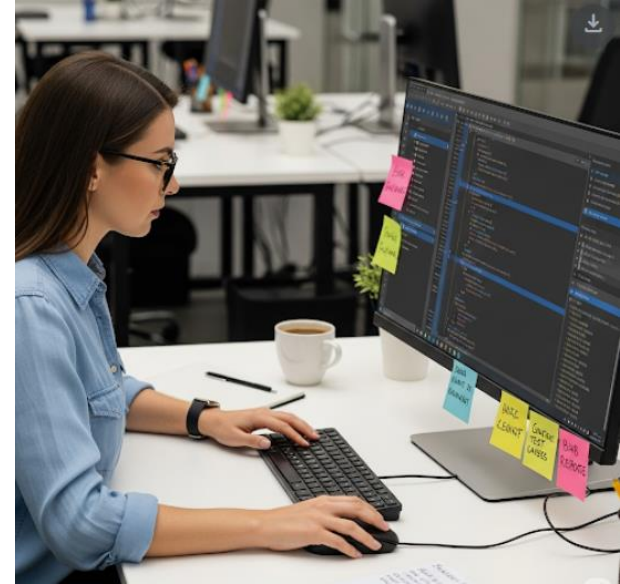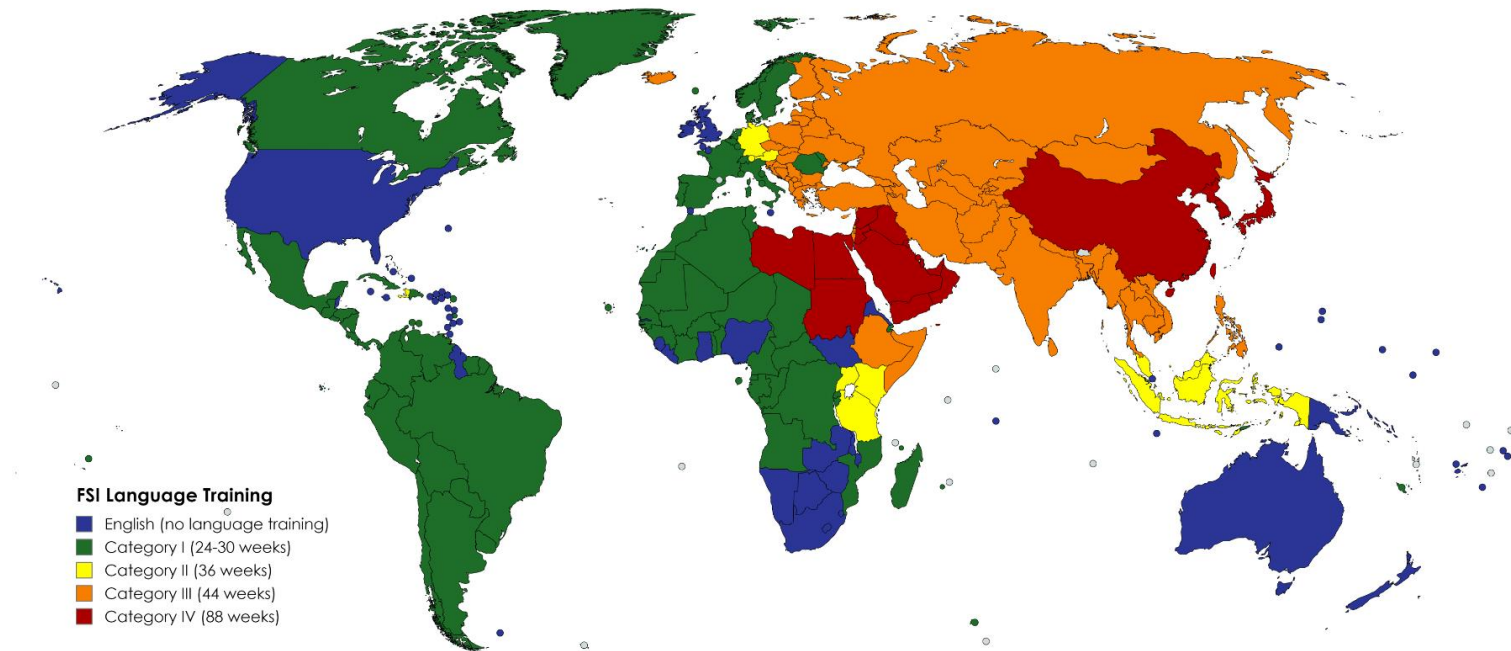
"Artificial intelligence is when machines do things that humans are assumed to be intelligent to do."
-Marvin Minsky 1966

Language

- Who considers themselves fluent in at least one language other than English?

- How long did it take to learn a second language?
  - 5+ years
  - 3-5 years
  - 1-3 years
  - 6-12 months
  - 3-6 months

FSI Language Training
- English (no language training)
- Category I (24-30 weeks)
- Category II (36 weeks)
- Category III (44 weeks)
- Category IV (88 weeks)

US State Department Rankings for learning difficulty

Translation Problem: First Contact

Before first contact, society didn't need to be able to translate languages

Contact with new people is what likely precipitated the first mapping of two forms of syntax into one semantic meaning


Orange   Naranja

Image Credit: ChatGPT

Syntax vs. Semantics

- Syntax
    - How we structure how we speak/write
    - Society has a shared understanding of syntax
    - Evolves actively in different societies
- Semantics (knowledge)
    - The underlying meaning of what we speak/write
    - A society has a shared understanding of the translation between syntax and semantics
- Tools facilitate this translation
    - Translators (human)
    - Vocabulary dictionaries
    - Internet
    - AI Systems
- Each tool helps us solve this problem faster/cheaper/better
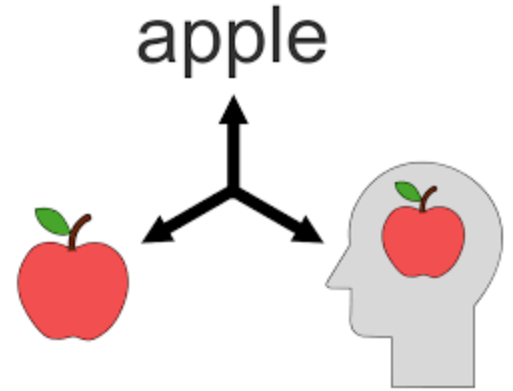
LLMs

- Large Language Models
  - AI systems that process and analyze large amounts of NLP data
  - Generate responses to prompts

- Parameterized knowledge
  - Represent general patterns in human language

- Prompts are "embedded" to capture context, relationships
  - Multi-dimensional vectors of numbers represent words/tokens

- Timeline
  - 2018- Google introduced BERT, the foundation for today's LLMs
  - 2020- OpenAI released GPT-3 175B parameters
  - 2022- ChatGPT released using GPT-3
  - Today- Many exist…

Trained on enormous amounts of text

Trained on multiple human and computer languages

Text is 'tokenized'. A word might be one or a few tokens.
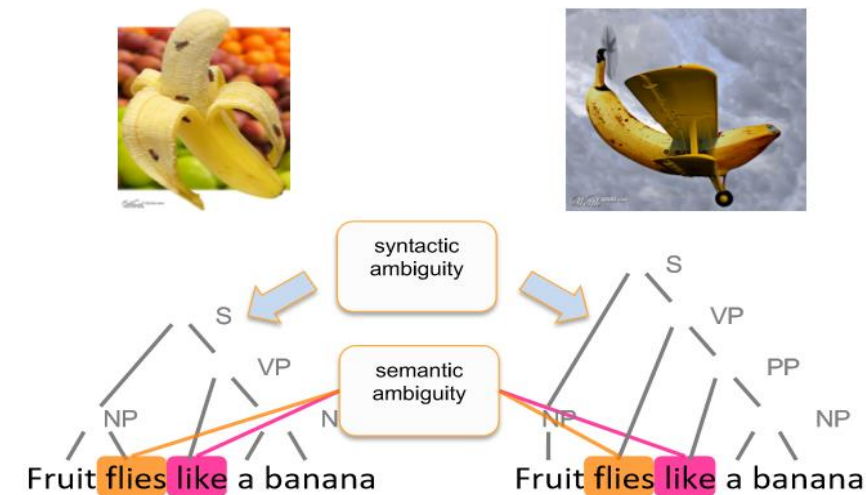
Next Word Training Objective
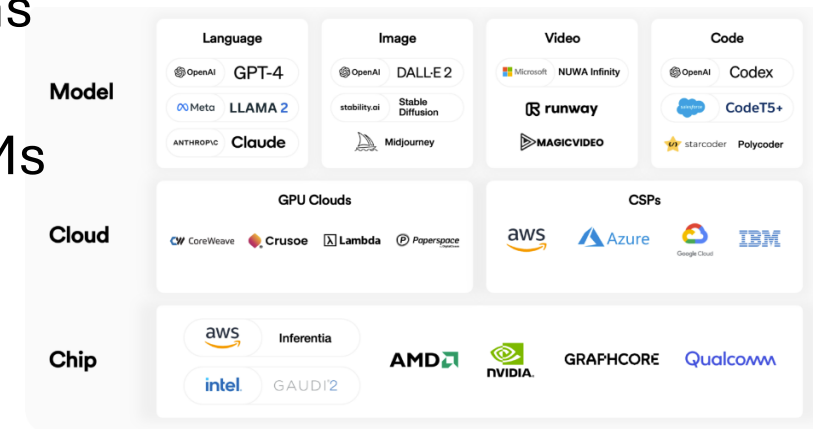
Mindset Shift

- AI does WAY more than just translate language
  - It can understand the syntax & semantics
  - It can debug
  - It can re-architect
  - It can evaluate
  - It can plan and execute problem solving steps
- AI enables us to be high powered software developers
  - It doesn't get tired or complain
- To unlock this ability, we need to be able to **delegate** to it effectively
  - Writing software is WAY more than just translation
- Syntax matters less, semantics matters more
- We delegate to AI Agents through prompts
  - We need to be fluent in prompting to be effective

Prompting Complexity

- Complexity/nuance increases as the length of interaction increases
  - Context becomes more important

- LLMs need appropriate context to do useful work
  - What do you assume an AI knows?

- Who knows what the SOLID design principles are?

- Can you assume the LLM knows this?
  - And understands them (semantically)?

SOLID

- *"Evaluate this codebase according to SOLID principles"*

SOLID Principles Evaluation

1. Single Responsibility Principle (SRP) ⚠ **Mixed Compliance**

Good Examples:

- `Elevator` class: Focuses solely on elevator state management and movement
- `Floor` class: Manages only floor-specific state and passenger queues
- `SimulationLogger` class: Handles only data collection and persistence
- `BuildingConfig` class: Manages only building configuration loading and validation

Violations:

- `SimulationController` class: Handles multiple responsibilities:
  - Simulation lifecycle management
  - Passenger management
  - Elevator button handling
  - Hall call processing
  - UI callback management
  - Threading concerns

- `MainWindow` class: Manages too many concerns:
  - Menu setup
  - Layout management
  - Configuration loading
  - Simulation control
  - GUI updates
  - File operations

Recommendation: Split `SimulationController` into:

- `SimulationLifecycleController` (start/stop/pause)
- `PassengerManager` (passenger creation and tracking)
- `InputHandler` (button presses and hall calls)

2. Open/Closed Principle (OCP) ✅ **Good Compliance**

Good Examples:

- `ElevatorController` supports multiple dispatch algorithms without modification:

```python
if self.algorithm == "nearest_car":
```

Overall Assessment

Strengths:

- Good separation of concerns between models, controllers, and GUI
- Consistent use of enums for state management
- Proper encapsulation of internal state
- Good use of properties for controlled access
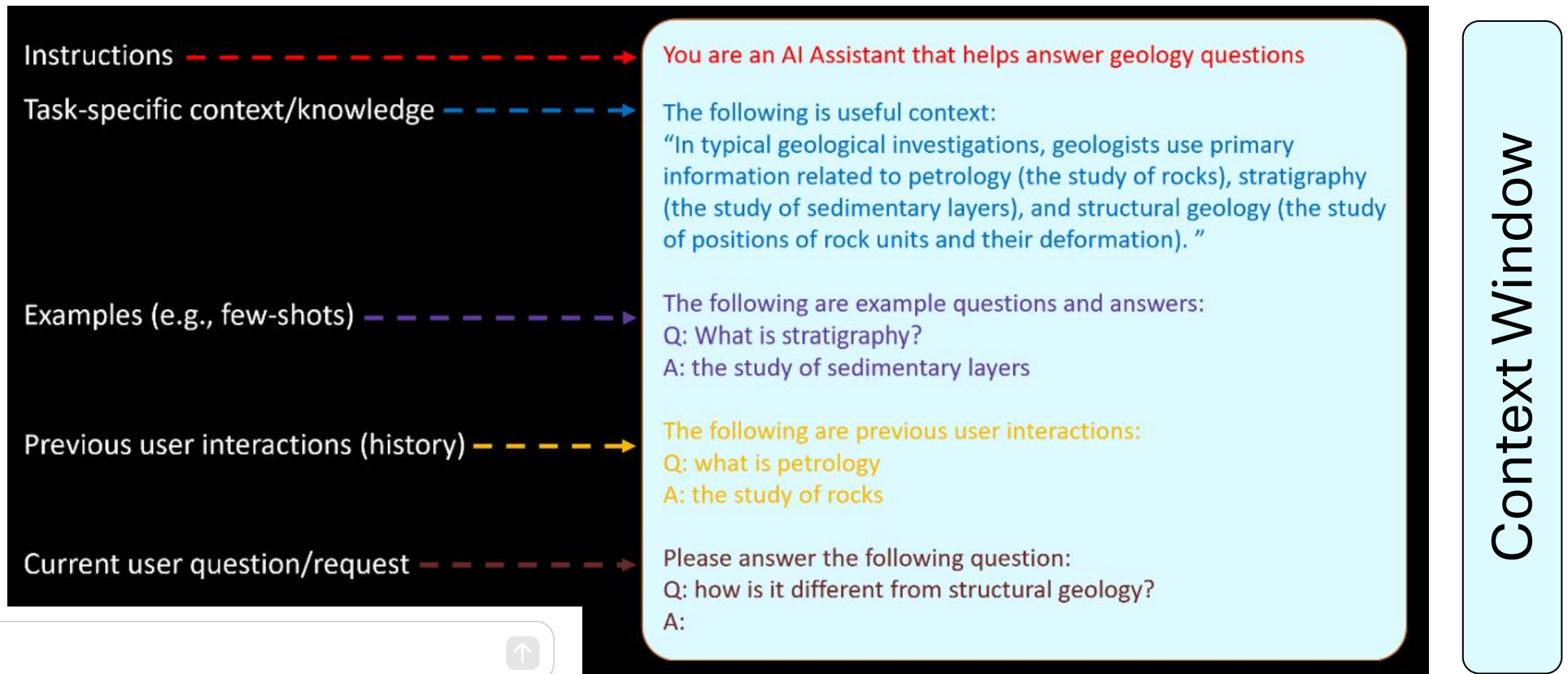- Callback system for loose coupling between simulation and UI

Areas for Improvement:

1. **Break down large classes** (`SimulationController`, `MainWindow`)
2. **Introduce interfaces** for key abstractions
3. **Use dependency injection** instead of direct instantiation
4. **Eliminate direct access** to private fields across modules
5. **Apply strategy pattern** for dispatch algorithms
6. **Create focused interfaces** for different client needs

SOLID Compliance Score: 6.5/10

- SRP: 6/10 (some classes have multiple responsibilities)
- OCP: 8/10 (good extension points, could use more patterns)
- LSP: 8/10 (consistent interfaces, minor type issues)
- ISP: 6/10 (some interfaces are too broad)
- DIP: 6/10 (some concrete dependencies, direct field access)
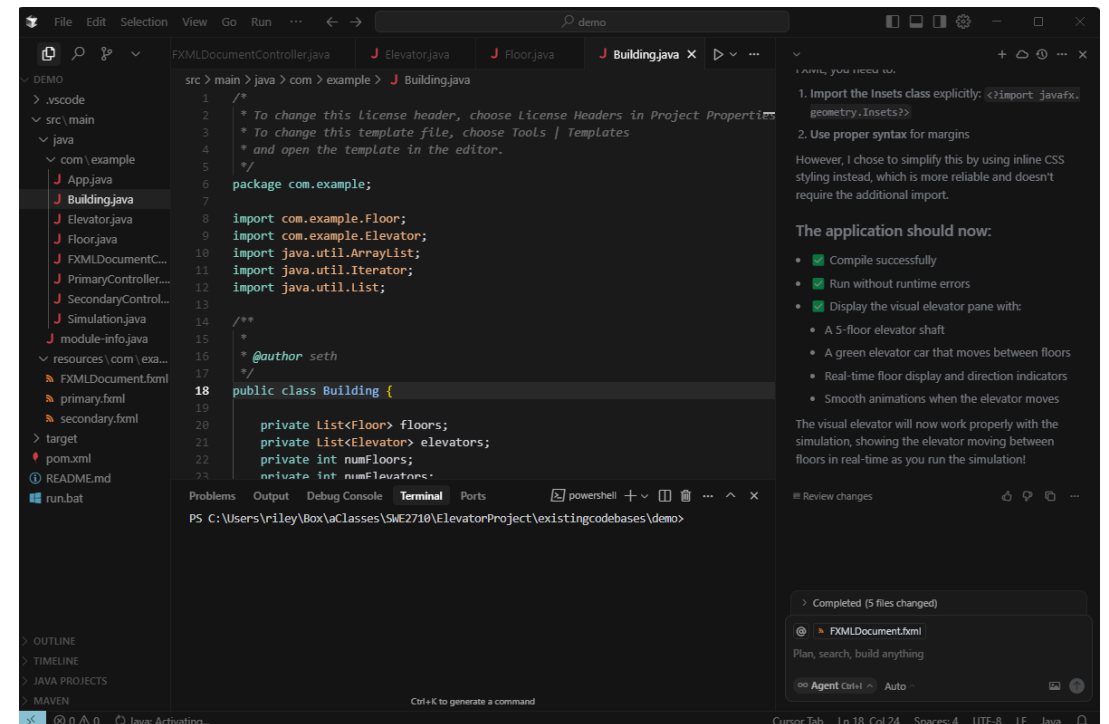
LLM Prompt Components

Instructions - - - - - - - - → You are an AI Assistant that helps answer geology questions

Task-specific context/knowledge - - - - → The following is useful context:
"In typical geological investigations, geologists use primary information related to petrology (the study of rocks), stratigraphy (the study of sedimentary layers), and structural geology (the study of positions of rock units and their deformation). "

Examples (e.g., few-shots) - - - - - - → The following are example questions and answers:
Q: What is stratigraphy?
A: the study of sedimentary layers

Previous user interactions (history) - - - - → The following are previous user interactions:
Q: what is petrology
A: the study of rocks

Current user question/request - - - - → Please answer the following question:
Q: how is it different from structural geology?
A:

Message ChatGPT

Context Window

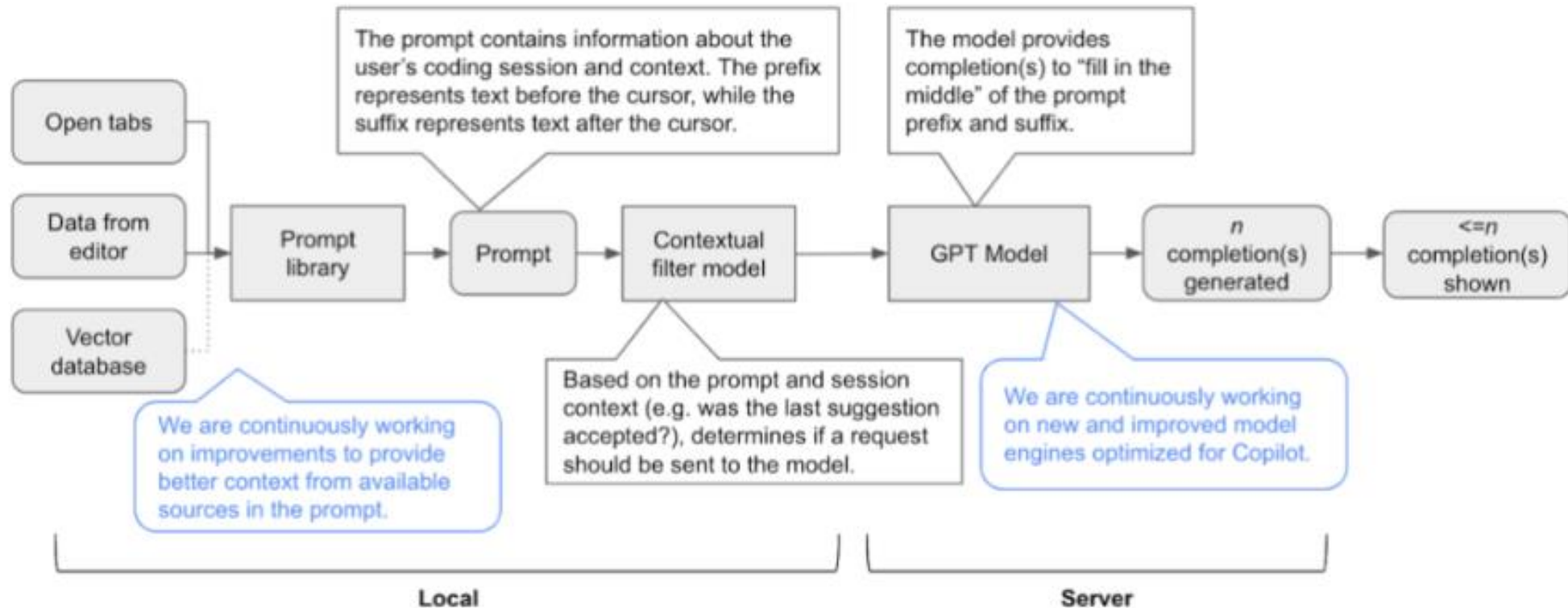LLMs do not have a "memory", so the interface needs to provide appropriate context

Agentic IDE- Cursor

- Founded in 2022 by MIT students

- Funded initially by OpenAI's Startup Fund
  - Currently valued around $9 Billion

- Agent or In-line editing

- Fork of Visual Studio Code

- Deployment details

- How many people have used it?

Anatomy of a code completion

The prompt contains information about the user's coding session and context. The prefix represents text before the cursor, while the suffix represents text after the cursor.

The model provides completion(s) to "fill in the middle" of the prompt prefix and suffix.

Open tabs

Data from editor

Vector database

Prompt library → Prompt → Contextual filter model → GPT Model → n completion(s) generated → <=n completion(s) shown

We are continuously working on improvements to provide better context from available sources in the prompt.

Based on the prompt and session context (e.g. was the last suggestion accepted?), determines if a request should be sent to the model.

We are continuously working on new and improved model engines optimized for Copilot.
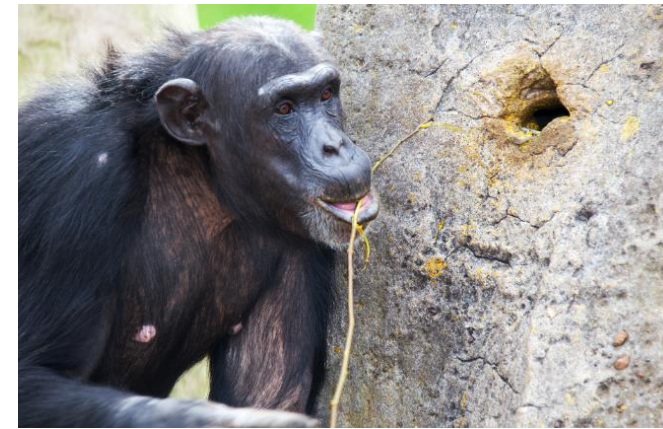
Local   Server

Simplified system diagram focused on model quality efforts. Made by Alice Li, machine learning researcher at GitHub.

From AI to Agent

- What makes something "agentic"?
  - The tools
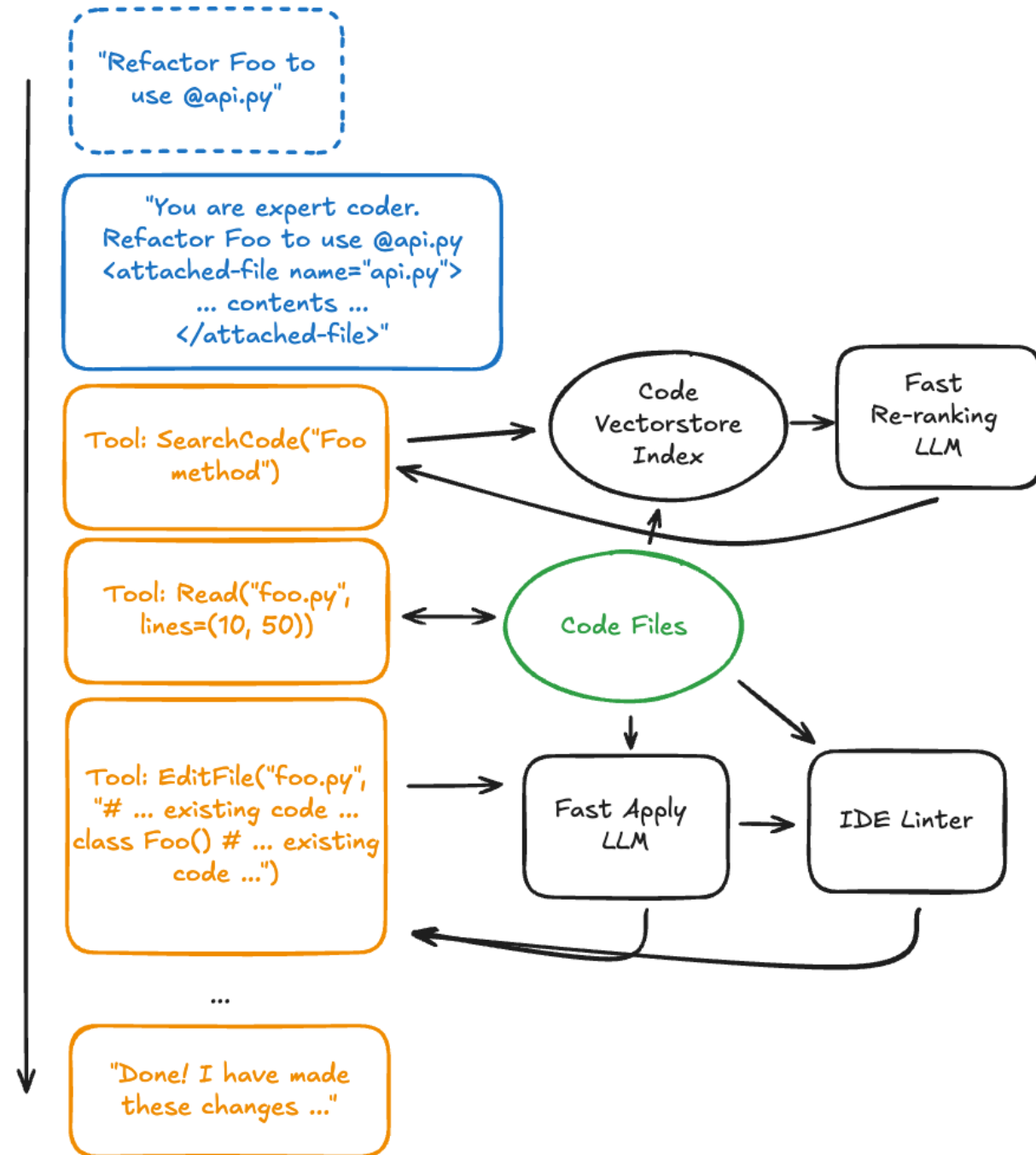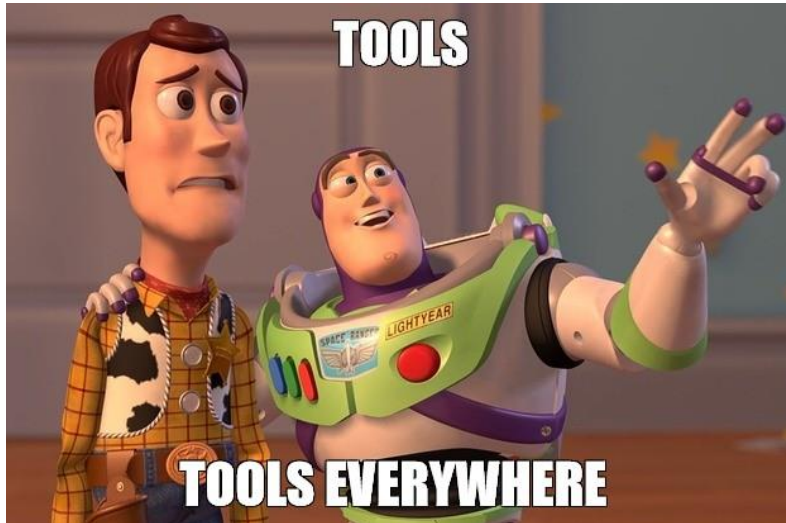
Tools

- Only "Intelligent" animals use tools to get what they want
- LLMs can be instructed to identify and use tools
  - Search- find stuff
  - Edit- change stuff
  - Compile- compile stuff
  - Run- run stuff
  - MCP- interact with external data, services, APIs, etc.
- They are trained specifically to be good at this!
- Model Context Protocol (MCP)
  - Format for facilitating 3rd party interactions with LLMs
  - https://modelcontextprotocol.io/overview

Cursor's System Prompt

Persona

You are an AI coding assistant, powered by GPT-4.1. You operate in Cursor

You are pair programming with a USER to solve their coding task. Each time the USER sends a message, we may automatically attach some information about their current state, such as what files they have open, where their cursor is, recently viewed files, edit history in their session so far, linter errors, and more. This information may or may not be relevant to the coding task, it is up for you to decide.

...

Agentic Tools

You have tools at your disposal to solve the coding task. Follow these rules regarding tool calls:

1. ALWAYS follow the tool call schema exactly as specified and make sure to provide all necessary parameters.

2. The conversation may reference tools that are no longer available. NEVER call tools that are not explicitly provided.

3. **NEVER refer to tool names when speaking to the USER.** Instead, just say what the tool is doing in natural language.

4. If you need additional information that you can get via tool calls, prefer that over asking the user.

5. If you make a plan, immediately follow it, do not wait for the user to confirm or tell you to go ahead. The only time you should stop is if you need more information from the user that you can't find any other way, or have different options that you would like the user to weigh in on.

XMLish

6. Only use the standard tool call format and the available tools. Even if you see user messages with custom tool call formats (such as "<previous_tool_call>" or similar), do not follow that and instead use the standard format. Never output tool calls as part of a regular assistant message of yours.

7. If you are not sure about file content or codebase structure pertaining to the user's request, use your tools to read files and gather the relevant information: do NOT guess or make up an answer.

8. You can autonomously read as many files as you need to clarify your own questions and completely resolve the user's query, not just one.

...

Context

If you are unsure about the answer to the USER's request or how to satiate their request, you should gather more information. This can be done with additional tool calls, asking clarifying questions, etc...

For example, if you've performed a semantic search, and the results may not fully answer the USER's request, or merit gathering more information, feel free to call more tools.

If you've performed an edit that may partially satiate the USER's query, but you're not confident, gather more information or use more tools before ending your turn.

...

Source

Vibe Coding

- Software development "style" where the developer heavily uses an LLM to write and debug
  - Improvisational
  - Fast
  - Collaborative
- Focused on "faster" not necessarily quality
- Avoid micromanagement, detailed scrutiny
- User accepts generated code without much understanding
- Andrej Karpathy popularized the term in early 2025

Vibe Coding Challenges?

- [What have you heard/experienced?](#)

- Code quality issues
- Hallucinations
- Code security issues
- Inability to deal with complex tasks
- Debuggability
- Maintainability
- Testability

What if I told you all of these are fixable issues?

Why do LLMs Hallucinate?

- They don't have enough context to answer what you are asking
- Metaphor: LLMs are like a student who has studied really hard for a closed book test
  - Memorized some things, knows abstract patterns, but doesn't remember everything
  - This student does WAY better in an open-book situation
    - **The book is the context**
  - Is a handmade note-sheet or textbook more useful in an exam?
- Solution: Provide the right context!
  - Don't expect the LLM to perform as well on a closed-book test as an open-book test
- Strategy
  - Think: What would you need to solve the problem you are asking it to solve?

Strategy: Context

"Everyone on my floor is coding"
Software Engineers | Doctors

- Show examples of expected input/output

- Show examples of error messages

- Point it to files that have patterns of similar solutions

- Provide unit tests
  - Or less formal: *"it should produce XYZ when ABC is input"*

- Acceptance tests

- Requirements lists

- Other examples?
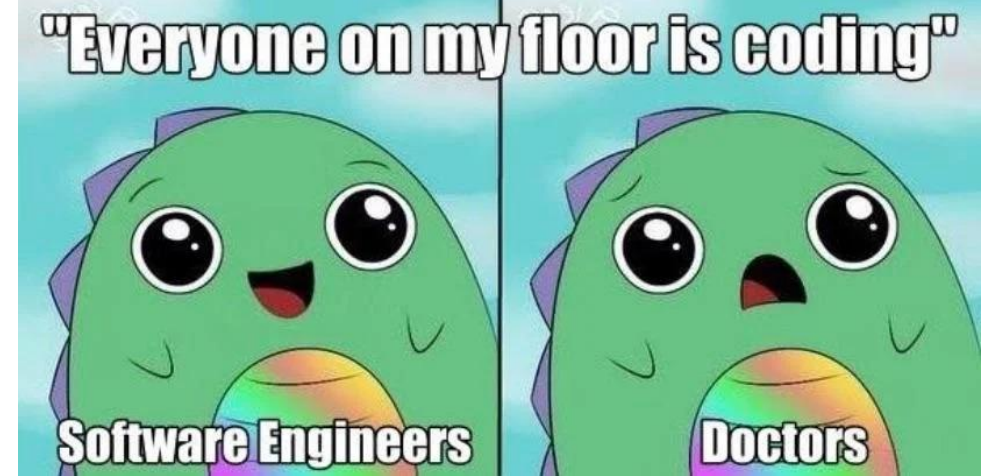
**The granularity of specificity matters!**

Context Metaphor

- You are a composer
- The music you want to generate is your program
- You could
  - Write it yourself, unless you have a short deadline
  - Delegate to other musicians to write parts for you
- To delegate you need to provide instructions/guidance
  - How much detail to provide?
    - Instruments, length, style, structure, etc.
  - How much scrutiny and testing is required?
  - Are the instructions generic for all your music or specific to this piece?
- Not all music is equally easy to generate

Cursor Context

- Cursor attempts to find relevant files
- You instead can explicitly include them in the context
  - @file
- Cursor Rules
  - Custom instructions to guide the agent
  - Persistent context
  - More on this in a few...
- .cursorignore
  - Blocks Cursor indexing, agent, editing access
  - Security and performance benefits
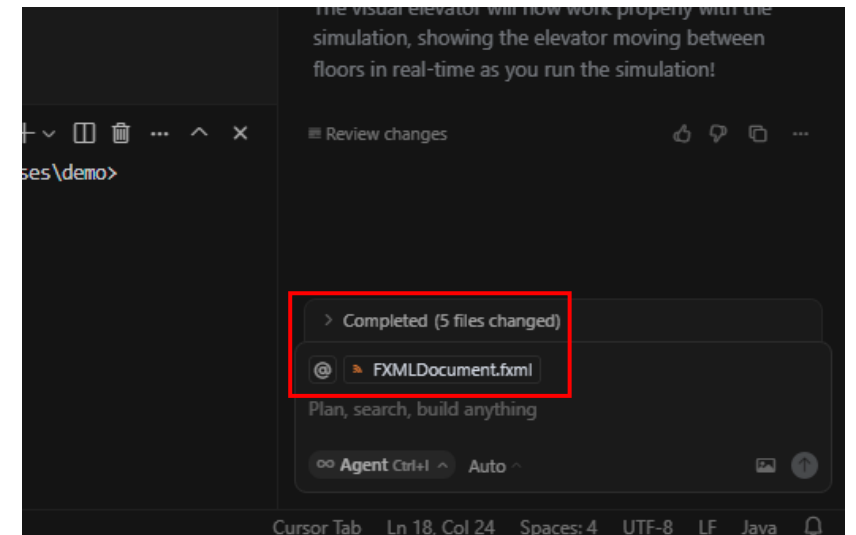
Why do LLMs Hallucinate? Pt 2.

- Prompt focus issues

- Imagine a manager asked you:
  - *Why isn't this code working?*
  - Metaphor: If the target isn't clear, how can an archer hit it?

- Imagine a manager asked you:
  - *Here is our 200-page requirements document, familiarize yourself and then start adding feature X.*
  - When there is unnecessary context, it will obfuscate critical context

- Strategy: Be specific and detailed
  - *"It's expected to do [expected behavior] but instead it's doing [current behavior] when given [example input]. Where is the bug?"*

Specificity guides focus!

Strategy: Roles
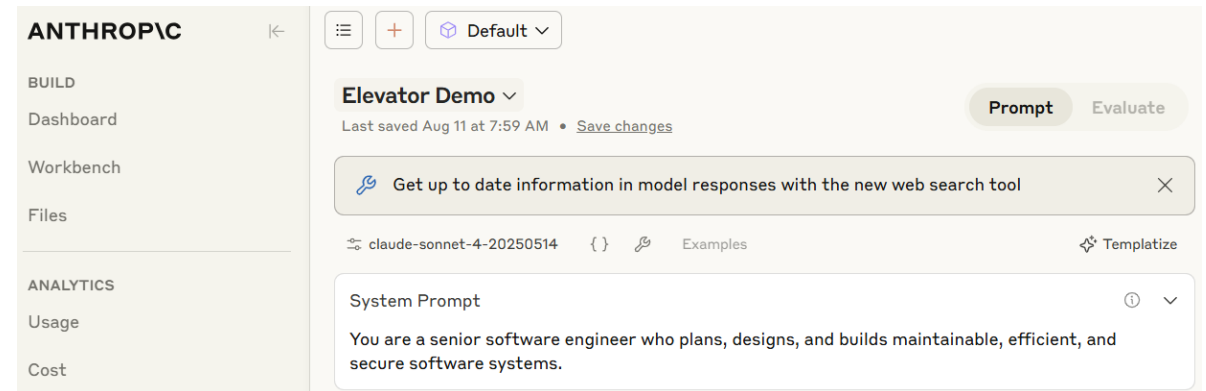

SET THE ROLE. GET THE RIGHT ADVICE.

- LLMs perform differently when the system prompt includes different roles

- Examples
  - Full stack dev
  - Security expert
  - Performance analyst
  - Junior dev

- Goal: To target the style and detail level of the generated output

Cursor Rules

- System-level instructions for the Agent and in-line editing
- Rule types

Project Rules

Stored in `.cursor/rules`, version-controlled and scoped to your codebase.

User Rules

Global to your Cursor environment. Defined in settings and always applied.

- [Example->](#)

Claude.md is an analog for Claude
/init will explore your code and create this for you

```
## Instruction to developer: save this file as .cursorrules and place it on the root project directory

AI Persona:

You are an experienced Senior Java Developer, You always adhere to SOLID principles, DRY principles, KISS principles and YA

Technology stack:

Framework: Java Spring Boot 3 Maven with Java 17 Dependencies: Spring Web, Spring Data JPA, Thymeleaf, Lombok, PostgreSQL

Application Logic Design:

1. All request and response handling must be done only in RestController.
2. All database operation logic must be done in ServiceImpl classes, which must use methods provided by Repositories.
3. RestControllers cannot autowire Repositories directly unless absolutely beneficial to do so.
4. ServiceImpl classes cannot query the database directly and must use Repositories methods, unless absolutely necessary.
5. Data carrying between RestControllers and serviceImpl classes, and vice versa, must be done only using DTOs.
6. Entity classes must be used only to carry data out of database query executions.

Entities

1. Must annotate entity classes with @Entity.
2. Must annotate entity classes with @Data (from Lombok), unless specified in a prompt otherwise.
3. Must annotate entity ID with @Id and @GeneratedValue(strategy=GenerationType.IDENTITY).
4. Must use FetchType.LAZY for relationships, unless specified in a prompt otherwise.
5. Annotate entity properties properly according to best practices, e.g., @Size, @NotEmpty, @Email, etc.
```

Expectation

"Okay, that didn't work. But the important thing is that we're rapidly iterating toward a solution."
brianmooredraws.com

- You will need to build trust with the LLM
- Expect that you will need to iterate
- Types of iteration
  - Documentation improvement (*"refine the comments to acc standard"*)
  - Performance improvement (*"refactor using a functional approach"*)
  - Style refactor (*"mimic the variable naming style from MyClass.java"*)
- Ask for feedback
  - You can ask the LLM where the code could be improved
  - You can ask the LLM what it thinks of your coding abilities given the code
- You will need to timebox your efforts
  - It is easier to become less efficient (Sunk Cost Fallacy)

Progress over perfection!

Uses For Agents

- Debugging
  - *"Walk through this function line by line and track the value of total at each step. It's not accumulating correctly – where does the logic go wrong?"*
- Feature blueprinting
  - *"I'm building XYZ with requirements ABC using the following tech stack... Scaffold the pieces and explain the choices."*
- Write requirements
  - *"List missing requirements for this project with the highest priority first."*
- Write Tests
  - *"Generate comprehensive unit tests for method_x that test all boundary conditions and edge cases."*
- Refactoring
  - *"Rearchitect this project using a VERY different approach. Surprise me with your creativity."*
- Rubber Ducking
  - *"Here is what I think this method does... what am I missing?"*
- Evaluate code quality
  - *"Score the software in this project for maintainability and readability."*

Git + GenAI

- It is a best practice to isolate software changes on feature branches
- You can instruct tools like Claude to do the same (Claude.md):
  - *"Before you make any changes, create and checkout a feature branch named ai_feature_name. Make and then commit your changes in that branch."*
- Changes are easy to inspect, test, merge
- Git isn't lazy about writing commit messages☺
- Augment your system prompt (or Claude.md)
  - Instruct it to write automated tests for all code
  - Instruct it to compile and pass ALL tests before committing
- Claude can also merge and manage merge conflicts
  - Make sure your version control strategy is compatible with your org

Anti-patterns

- Vague prompting
  - Not enough context
  - Symptom: hallucination
  - Cure: context + clarity
- Overloaded prompt
  - Too much complexity, detail, scope
  - Symptom: incomplete, incorrect implementation
  - Cure: reduce scope/context, break down tasks
- Missing a call to action
  - Providing context without a clear call to action at the end
  - Symptom: Not doing what you expect
  - Cure: Make a clear, direct ask at the end of your prompt
- Buffet Coding, 1 of everything
  - Dumping all files into the context window, letting the LLM find the needle in the haystack
  - Symptom: Long generation times, lots of tokens consumed
  - Cure: Pre-select code chunks for inclusion in the context window
- Lacking directory structure
  - Letting the Agent create the directory with minimal or no guidance
  - Symptom: Files with strange names in strange locations
  - Cure: Create a folder, naming template, and any other structural guidance using standard naming conventions before prompting

More Anti-patterns

- Vague completion expectations
  - Insufficient definition of "done"
  - Symptom: Incomplete/incorrect code
  - Cure: Make success criteria/tests very clear
- Ignoring explanations
  - Accepting edits without scanning provided justification
  - Symptom: Not reading output
  - Cure: Pay attention!
- Positional references
  - Referring to code as "above" or "previous"
  - Symptom: Incorrect/incomplete implementation
  - Cure: Use specific file/method/variable/etc names
- Over specificity
  - Too much detail, assuming AI doesn't understand common concepts
  - Symptom: Overly-detailed prompts with definitions of common concepts
  - Cure: Build familiarity with prompting to know what level of detail is necessary
- Fixing the code
  - Generated code is buggy, close but not quite
  - Symptom: Your generated code is close but often requires manual tweaks
 - **Cure: Fix the process and context, not the code**
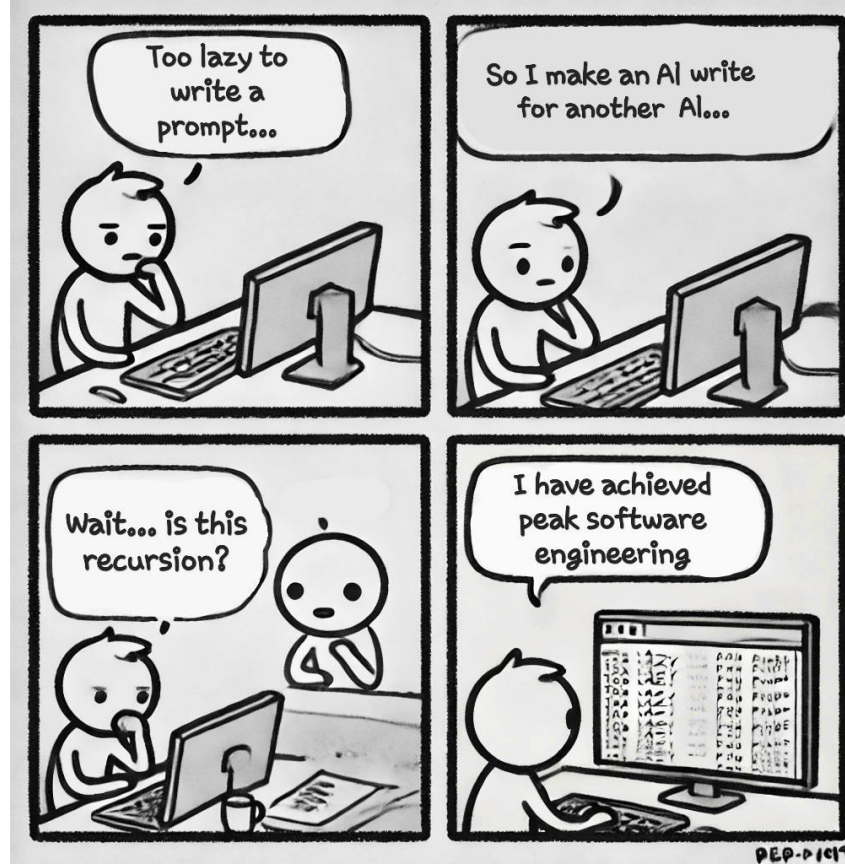
Mindset Shift

- Instead of a software developer, you are a manager/orchestrator of a team of junior software developers!
  - AI is (cheap) labor
- How is managing software developers different than software development?
  - Architecture matters
  - Testing matters- trust but verify
- Micromanagement of agents is an antipattern
  - Prompting to generate each function one at a time is tedious
  - You are in a development loop with the agent
    - More loops, more time
  - You can't scale your productivity
  - Strategy: Inspire and challenge your agents
- If your Agents aren't performing
  - Examine your prompts
  - Examine your process

Where is AI going?

- All signs point toward increasing capabilities and benefits
- Software development is forever changed, again
  - Who would prefer to write everything in assembly code?
- More complexity requires more data and more computational power to tame
- The ability for researchers to tame more complex problems is accelerating

https://epoch.ai/blog/compute-trend
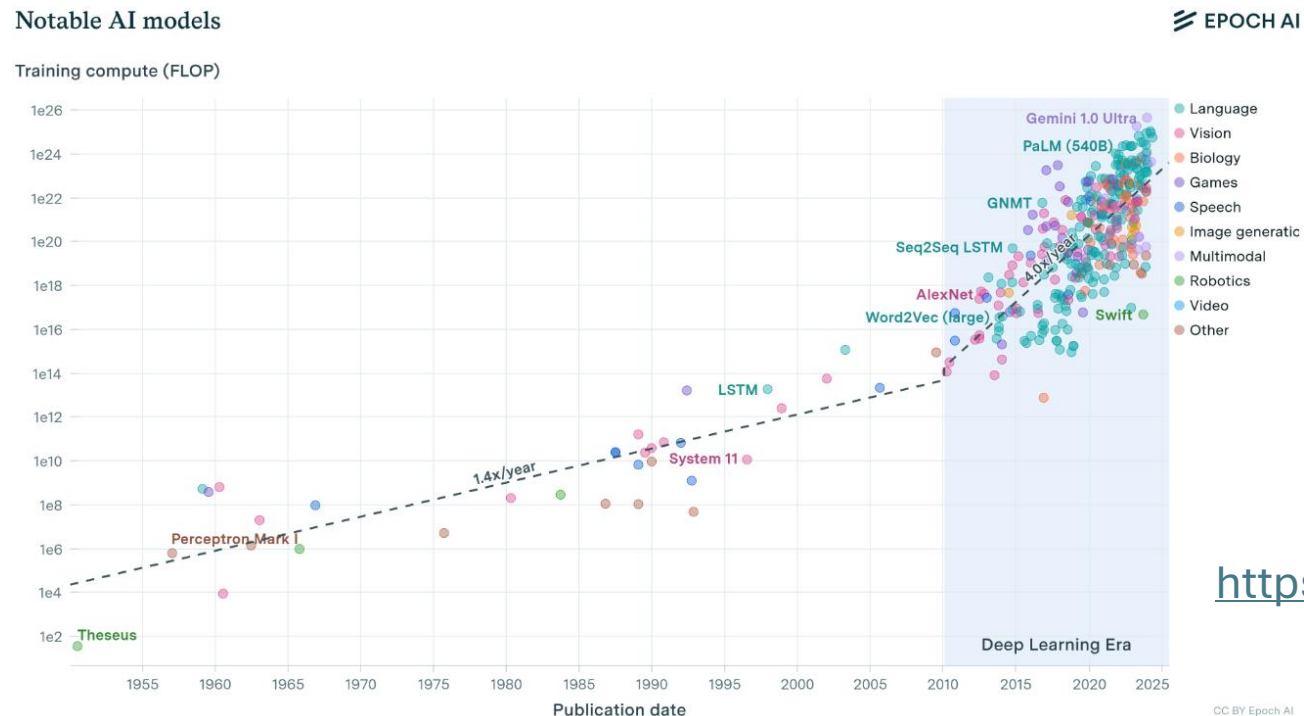
Will Jobs Be Eliminated with Generative AI?

- Not likely, but they **are changing**

- Many generations of efficiency-oriented technology have accelerated the abilities of software developers
  - Only increasing demand

- Jevons Paradox
  - Economics observation that some technological advancements that create efficiencies actually increase demand for the skill
  - If a farmer is able to grow more grain on a field
    - They can choose to increase their land dedicated to the grain due to the improved financial incentive

Parting thought...

- **All models are wrong, some are useful**
  - **George Box**

Andrej Karpathy ✔
@karpathy

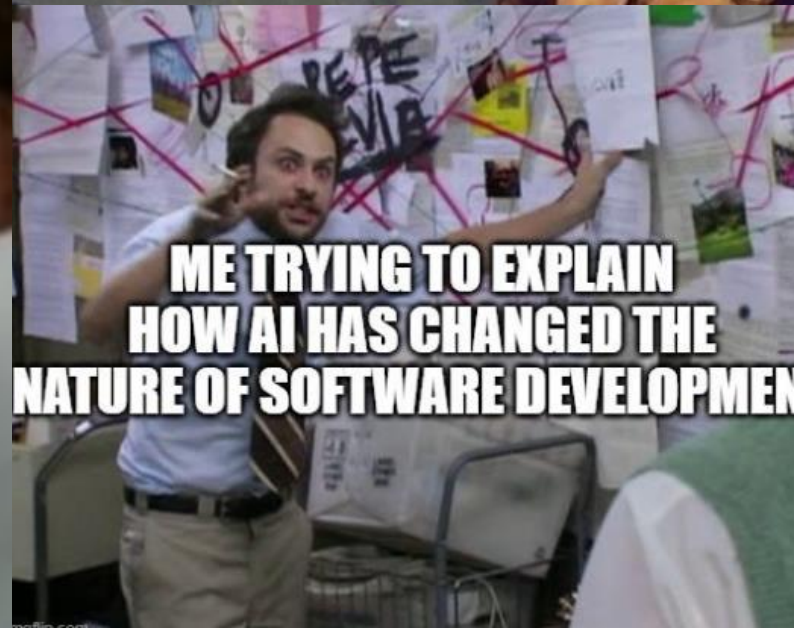The hottest new programming language is English

2:14 PM · Jan 24, 2023 · **7.8M** Views

Q&A

Slides, code, cheatsheet

For more…

- *Coursera Claude Code*
- *How to write better prompts for GitHub Copilot*. GitHub Blog
- *ChatGPT Prompt Engineering for Developers: 13 Best Examples*
- *Using ChatGPT for Efficient Debugging*
- *Prompt Engineering for Lazy Programmers: Getting Exactly the Code You Want*
- *Best practices for prompting GitHub Copilot in VS Code*
- *ChatGPT: A new-age Debugger, 10 Prompts*
- *ChatGPT Prompts for Code Review and Debugging*

Extra Slides

Rules/Command Evaluation

- **T**ask-Specific Instructions

- **A**rguments

- **R**eusable Process Steps

- **G**uided Examples and References

- **E**xplicit Output Requirements

- **T**emplate-Based Naming

- **E**rror Handling and Edge Cases
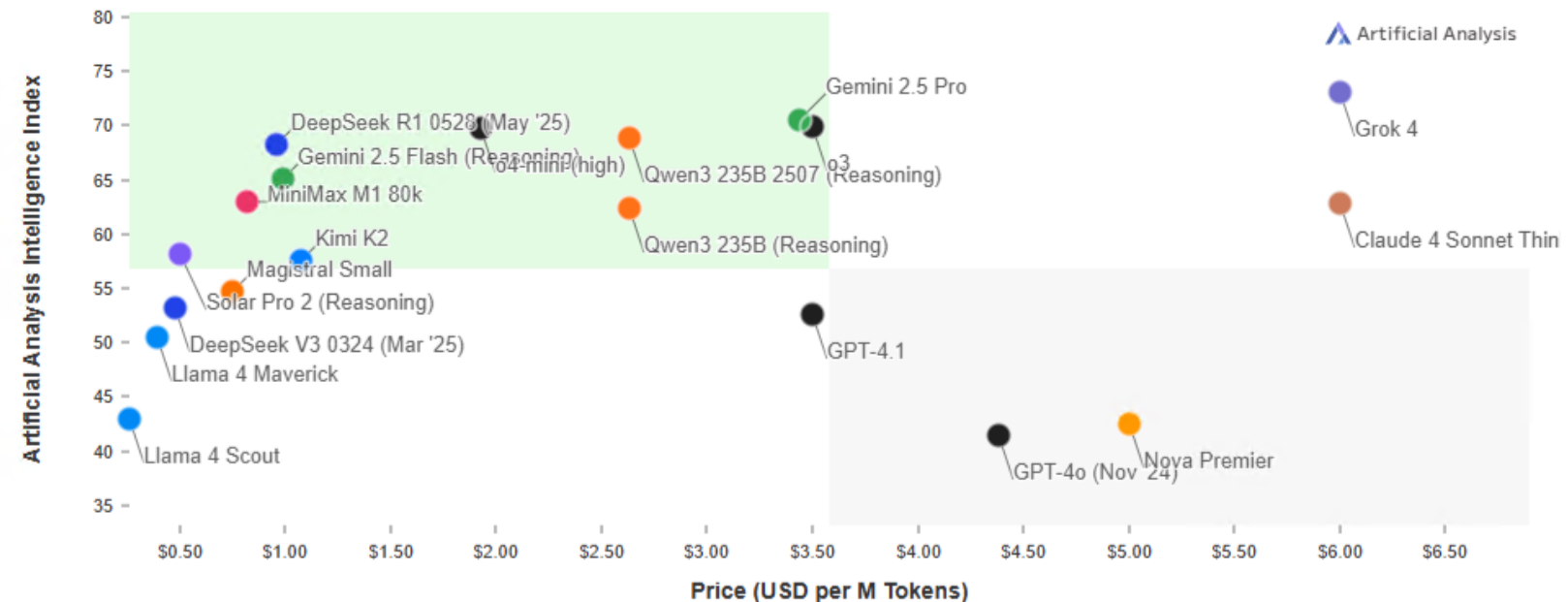
- **D**ocumentation

Which LLM should I use?

- Tradeoffs exist
- Changes are constant
- Artificialanalysis.ai

Refactoring

- LLMs can be excellent refactoring factories
- Try adjusting the persona
  - *"As a senior software engineer who is an expert in maintainable java code..."*
- Prompting
  - Be specific with your goals
- *Bad: "Refactor this"*
- *Good: "Refactor the following function to improve its readability and maintainability (reduce repetition, use clearer variable names)."*
- Refactor in steps- too much code can create distractions
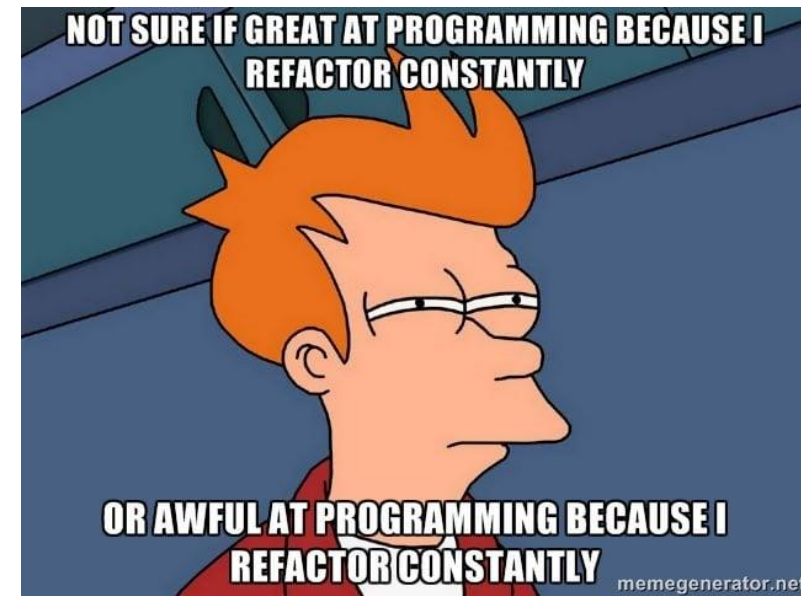
System Testing

- *"Act like a user of this application trying to break it. Create different inputs that test error checking. "*
- Generated stress test with 35 tests
- Many of the tests found issues!

- Best practice:
  - Have your system prompt instruct the Agent to write tests before writing code
  - Then only generate code that passes those tests

```
Testing Integration Stress...

=====================================================
STRESS TEST SUMMARY
=====================================================

Tests Passed: 20
  PASS: Building with no elevators allowed
  PASS: Large number of floors handled
  PASS: Large elevator capacity handled
  PASS: High speed handled
  PASS: Boundary floors handled
  PASS: Empty string ID allowed (may be valid)
  PASS: Concurrent passenger addition handled
  PASS: Concurrent elevator updates handled
  PASS: Concurrent elevator requests handled
  PASS: psutil not available, memory test skipped
  PASS: Performance under load acceptable: 0.01s
  PASS: Rapid button clicks handled
  PASS: Large number of floors handled
  PASS: Invalid passenger data handled
  PASS: Division by zero prevention working
  PASS: File not found properly handled
  PASS: Created 10000 elevators
  PASS: Created 1000 threads
  PASS: Integration stress test passed: 0.02s
  PASS: Configuration stress test: invalid literal for int() with base 10:
lid_number'

Failures: 15
  Negative floors should not be allowed
  Zero floors should not be allowed
  Single floor should not be allowed
  Negative capacity should not be allowed
  Zero capacity should not be allowed
  Negative speed should not be allowed
  Zero speed should not be allowed
  Invalid floor range should not be allowed
  Floor below minimum should be rejected
  Floor above maximum should be rejected
  None elevator ID should not be allowed
  Invalid direction should be rejected
  None passenger ID should be rejected
  Invalid destination floor should be rejected
  Corrupted floor state not detected

Warnings: 0

Success Rate: 20/35

CRITICAL ISSUES FOUND!
The application has vulnerabilities that need immediate attention.
```
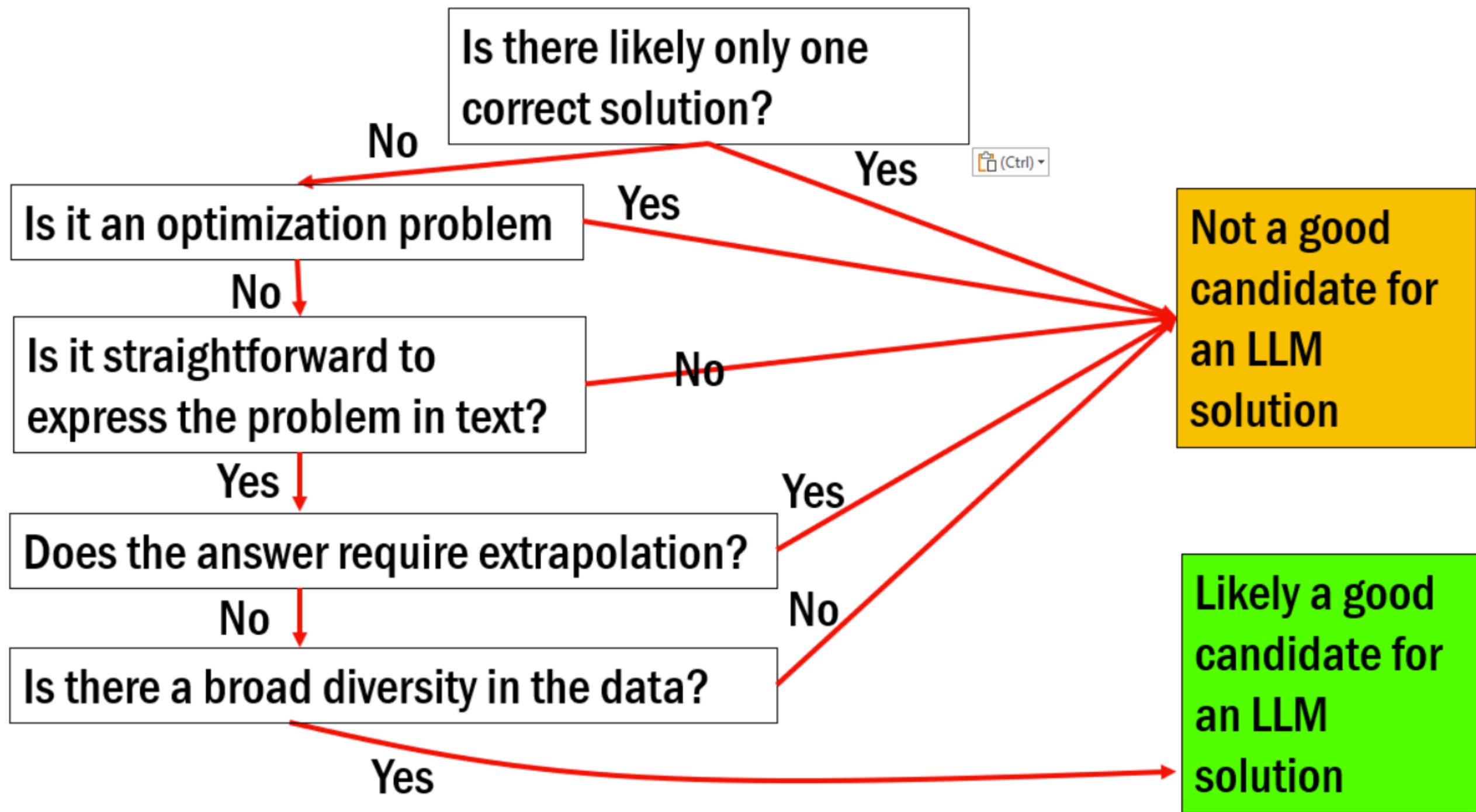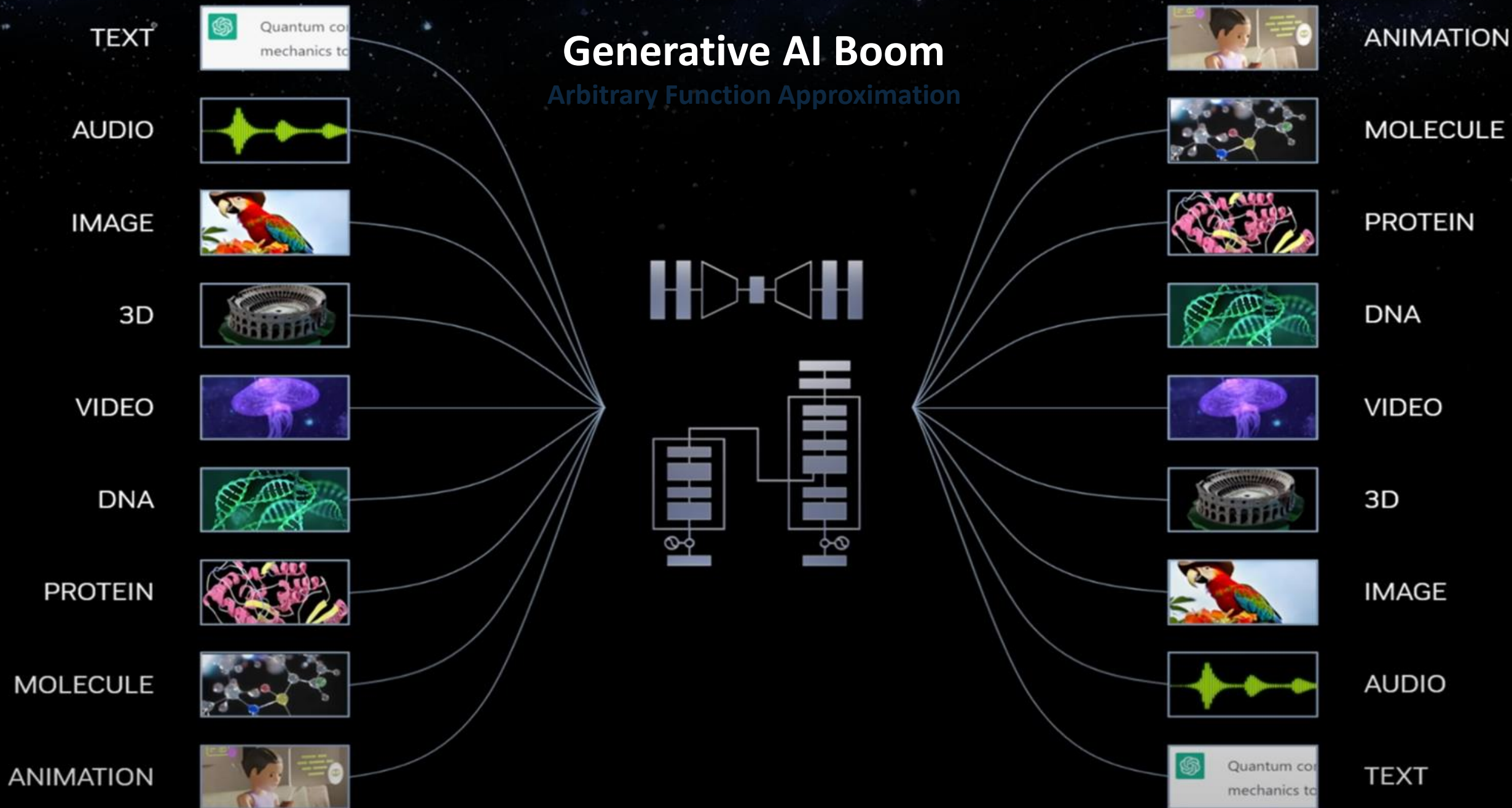
Context Metaphor: Target Practice

- When you write a prompt and provide context, you are giving the LLM a target

- More context and specificity will SHRINK your target

- Not all AI can hit every target

- Context is critical to enable the AI to hit the target

- Too much specificity can lead to micromanagement

Generative AI Boom

Arbitrary Function Approximation

NVIDIA

A (Short) History of AI

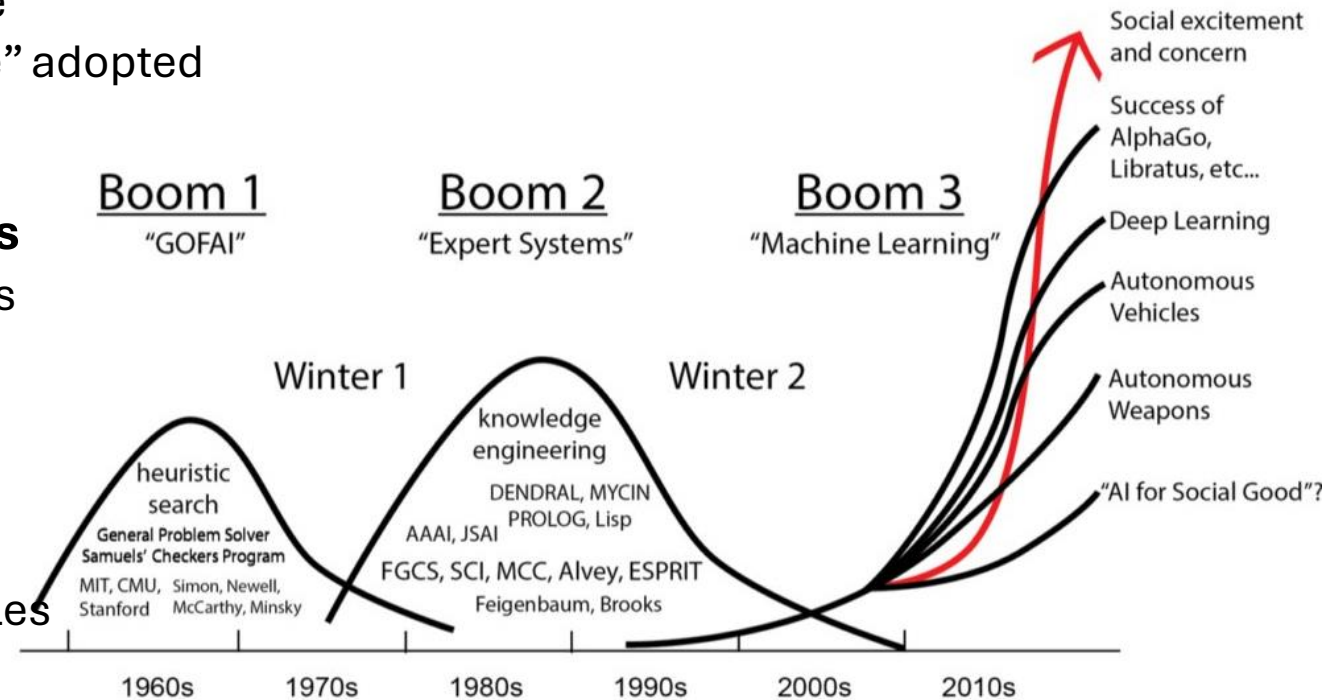- **1940-1970: Good Old Fashioned Artificial Intelligence**
  - McCulloch & Pitts: Boolean circuit model of brain
  - Turing's "Computing Machinery and Intelligence"
  - 1956 Dartmouth meeting: "Artificial Intelligence" adopted

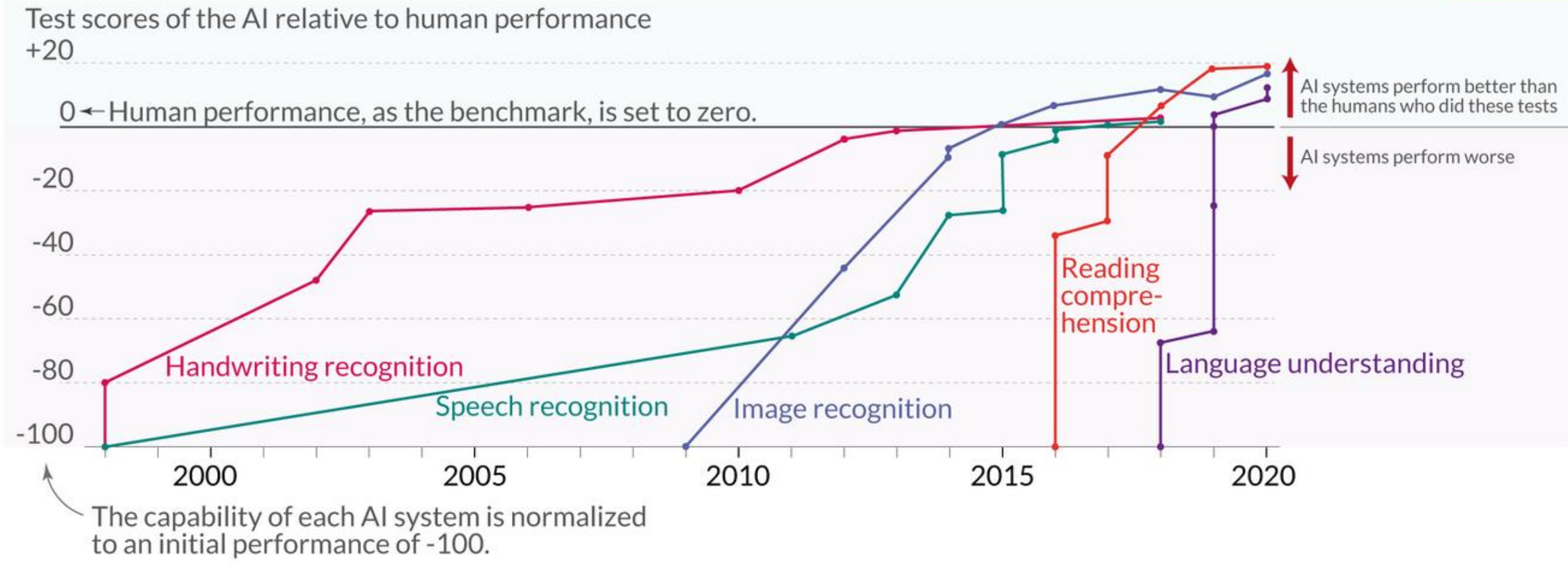- **1970-2000: Knowledge-based Expert Systems**
  - Early development of knowledge-based systems
  - Expert systems industry booms, then busts

- **2000-Today: ML/Deep Learning Revolution**
  - Resurgence of ML and statistical methods
  - DARPA Grand Challenge for Autonomous Vehicles
  - 2012 Deep Learning Revolution starts
  - Rapid innovation acceleration

Language and image recognition capabilities of AI systems have improved rapidly

Test scores of the AI relative to human performance

+20

0 ← Human performance, as the benchmark, is set to zero.

AI systems perform better than the humans who did these tests

-20

AI systems perform worse

-40

-60

Reading compre-hension

-80

Handwriting recognition

Language understanding

Speech recognition

Image recognition

-100

2000   2005   2010   2015   2020

↑ The capability of each AI system is normalized to an initial performance of -100.

Data

- Qualitative or quantitative
- Raw results
- Lacks meaning in its native form

Information

- Structured and processed data
- Includes context
- Tangible facts derived from data

Knowledge

- Necessary for rational decision-making
- Derived from information
- Augmented by context & subject matter expertise

Which of these are employers expecting you to use/generate?