



Cloud Computing 2026
Lisbon, Portugal

Orchestrating a brighter world



Keynote

Software engineering in the era of AI/LLMs

Kunal Rao
Researcher

NEC Laboratories America, Inc.

Email: kunal@nec-labs.com

April 22, 2026

Brief bio

- ◆ Work as Researcher at NEC Laboratories America, Inc. in Princeton, NJ
- ◆ **Current research:** Generative AI-based applications and platform, Edge and Cloud computing, leveraging AI/ML models to solve systems problems, gaining utility from AI/ML models for real-world applications
- ◆ **Past research:** High Performance Computing, GPGPU and Xeon Phi computing, Graph analytics, Video Analytics (Computer Vision, Applied Machine Learning)
- ◆ 34 granted patents (US and worldwide) and several are published and pending, 35 published papers



Context (Human Need)

- ◆ As societies grew larger and more complex, **gathering and processing information** became a major challenge. Massive amounts of information started appearing in many domains:
 - Census data
 - Banking transactions
 - Medical records
 - Tax records
 - Inventory and logistics
- ◆ Scalable methods were required



Generated using ChatGPT

How do we record, store, and process information at scale?

Mechanizing Data Processing

- ◆ **Problem** (Late 19th century):
 - 1880 U.S. Census took **8 years** to process
 - Population growth made manual processing unsustainable
- ◆ **Solution:**
 - Punch card tabulation machine was invented by Herman Hollerith
 - 1890 count was tallied in only **6 months** using Hollerith's machines
 - Hollerith's company later became **IBM**

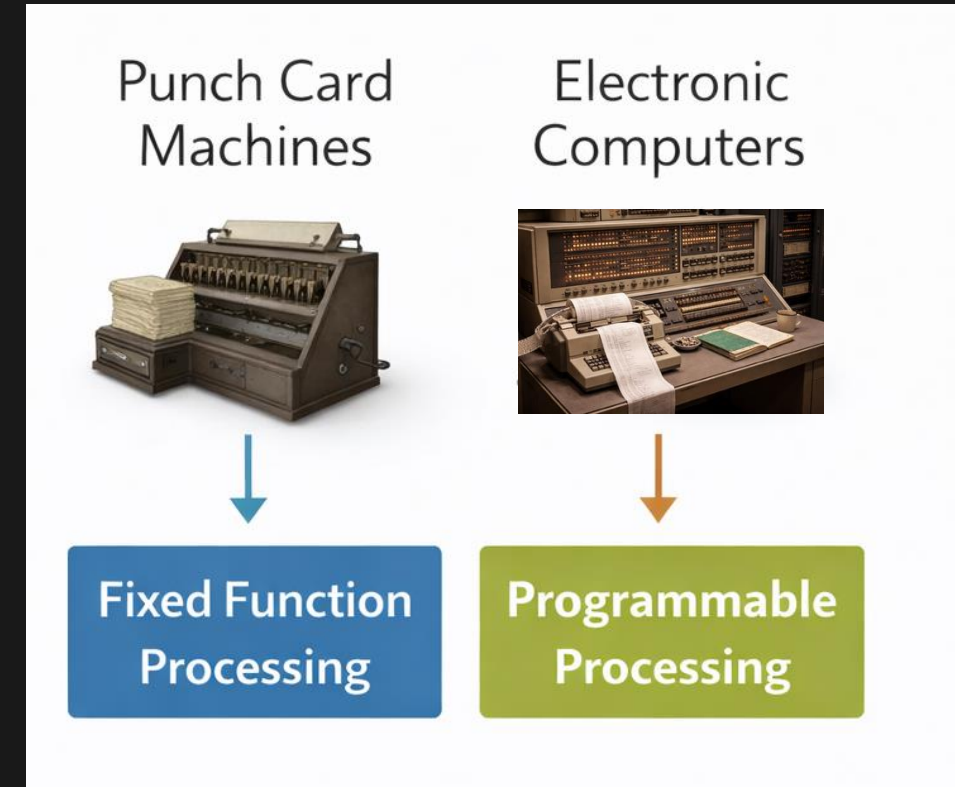


Generated using ChatGPT

Society's data challenge led to the first mechanized information processing systems.

Rise of application building (and programming)

- ◆ Punch card machines were still **specialized data processors**
 - Designed to perform specific tabulation tasks
 - Used extensively for census and large-scale data counting
- ◆ In mid-20th century, a new idea emerged: What if we built **machines that could perform many different tasks** ?
 - General-purpose electronic computers were invented
 - Could solve different problems using the same machine
- ◆ However, this introduced a new requirement:
 - We must tell the computer exactly what task to perform
- ◆ This gave **rise to programming and application building**



Generated using ChatGPT

General-purpose computers require programming to perform tasks.

Assembly programming

- ◆ Early programs were written very close to the machine. Programmers had to specify:
 - Memory locations
 - Registers
 - Arithmetic operations
 - Conditional jumps
- ◆ **Applications enabled:** Custom, hardware-specific programs (early tabulation, fine-grained control logic)



Generated using ChatGPT

Early programming required specifying detailed machine instructions.

Early High-level languages (FORTRAN and COBOL)

- ◆ **Need:** Programming directly in assembly was too slow and error-prone for growing scientific and business applications
- ◆ **Problem:** Different domains (scientific vs business) required very different ways of expressing computation
- ◆ **Solution:** FORTRAN (scientific computing): mathematical expressions and numerical computation; COBOL (business computing): data processing with human-readable syntax
- ◆ **Result:** Programs could be written closer to human reasoning within specific domains, significantly increasing productivity
- ◆ **Applications enabled:** Scientific simulations (FORTRAN), large-scale business data processing (COBOL)



Generated using ChatGPT

Application development began to align with how humans think about problems.

Programming in C

- ◆ **Need:** Build general-purpose systems (operating systems, compilers, infrastructure) across different machines
- ◆ **Problem:** Domain-specific languages (FORTRAN/COBOL) were not suitable for system-level programming
- ◆ **Solution:** C provided a portable, efficient abstraction close to the machine
- ◆ **Result:** Enabled development of system software that could run across platforms
- ◆ **Applications enabled:** Operating systems (Unix), compilers, foundational infrastructure for all applications

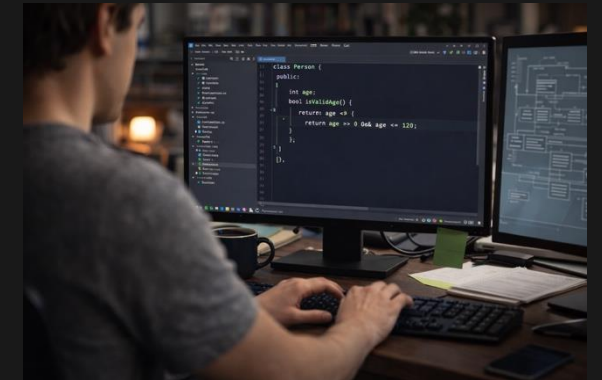


Generated using ChatGPT

Application development expanded from building domain solutions to building the systems that power all applications.

Programming in C++

- ◆ **Need:** Software systems became large and multi-component
- ◆ **Problem:** Code reuse, organization, and maintainability broke down
- ◆ **Solution:** Object-oriented programming (C++) was invented
- ◆ **Result:** Encapsulation and modular design enabled teams to build large systems
- ◆ **Applications enabled:** Large-scale software systems (enterprise software, financial systems)

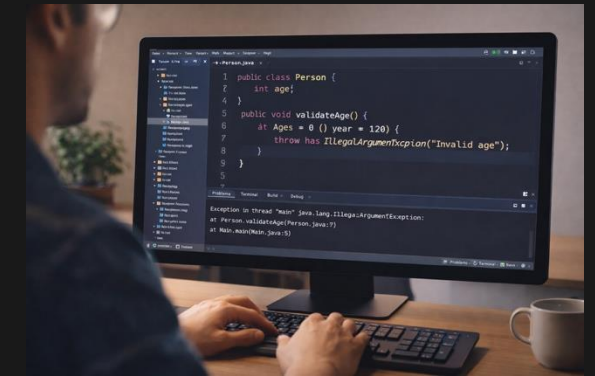


Generated using ChatGPT

Object-oriented programming enabled modular and reusable software design.

Programming in Java

- ◆ **Need:** Software systems needed to run across different machines and environments at scale
- ◆ **Problem:** C/C++ programs were tightly coupled to hardware and operating systems, making portability and deployment difficult
- ◆ **Solution:** Java introduced the Java Virtual Machine (JVM) and “write once, run anywhere” model
- ◆ **Result:** Developers could build portable, large-scale distributed systems with standardized runtime environments
- ◆ **Applications enabled:** Distributed systems, user facing applications, control kinds of systems, web backends, enterprise applications at scale

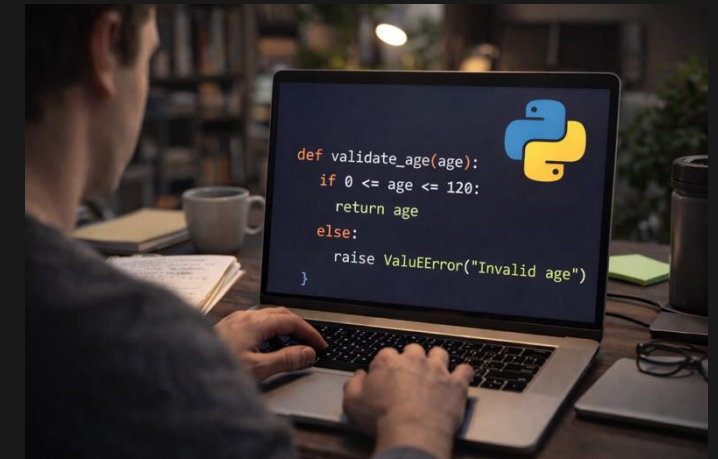


Generated using ChatGPT

Java didn't just abstract code, it abstracted the *execution environment itself*.

Programming in Python

- ◆ **Need:** Faster development cycles and broader developer accessibility
- ◆ **Problem:** Too much boilerplate and cognitive overhead
- ◆ **Solution:** Python simplified syntax and included rich libraries
- ◆ **Result:** Same logic, fewer lines led to higher developer productivity
- ◆ **Applications enabled:** Rapid application development, data science, scripting, early AI/ML workflows

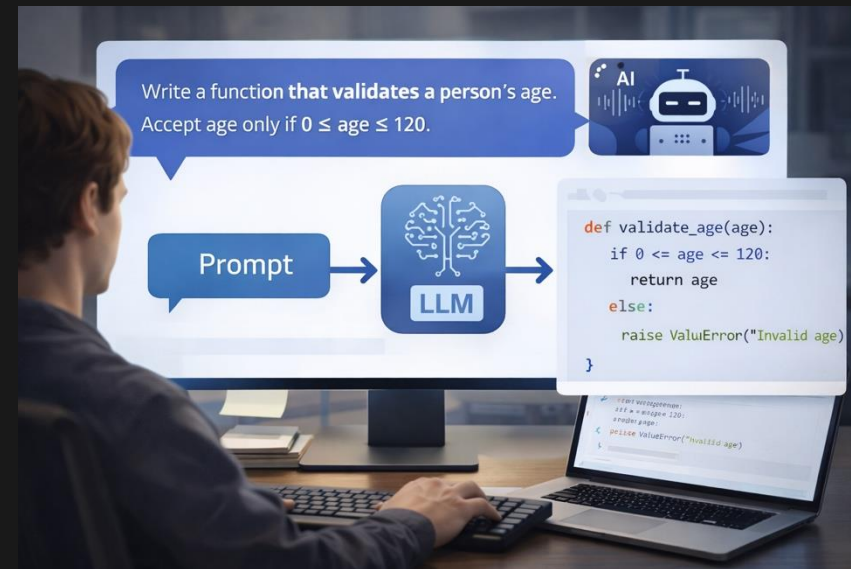


Generated using ChatGPT

Higher-level languages reduce the amount of code required to express the same logic.

Programming with Natural Language (LLMs)

- ◆ **Need:** Even writing code itself is becoming the bottleneck and everyone wants their own custom application
- ◆ **Problem:** Humans must still translate intent to code manually
- ◆ **Solution:** LLMs allow direct expression of intent in natural language
- ◆ **Result:** Machines generate implementations thus humans focus on *what*, not *how*
- ◆ **Applications enabled:** Personalized custom applications, AI-native applications



Generated using ChatGPT

Developers can now express intent, and machines generate the code.

Evolution of application development

Paradigm	Human Specification	Machine Work	α (Amplification)	What changed across decades	Application Impact
Assembly	Very high	Low	~1x	None (direct control)	First programmable-behavior
FORTRAN/ COBOL	High	Moderate	~5x	Domain abstraction	First large-scale applications
C	High	Moderate	~10x	System abstraction	Platforms & infrastructure
C++	Medium	Slightly higher	~50x	Modularity	Large software systems
Java	Medium-low	High	~100x	Portability	Distributed applications
Python	Low	Very high	~500x	Productivity	Rapid application development
LLMs	Very low	Massive	~1000x+	Intent to code	AI-native applications

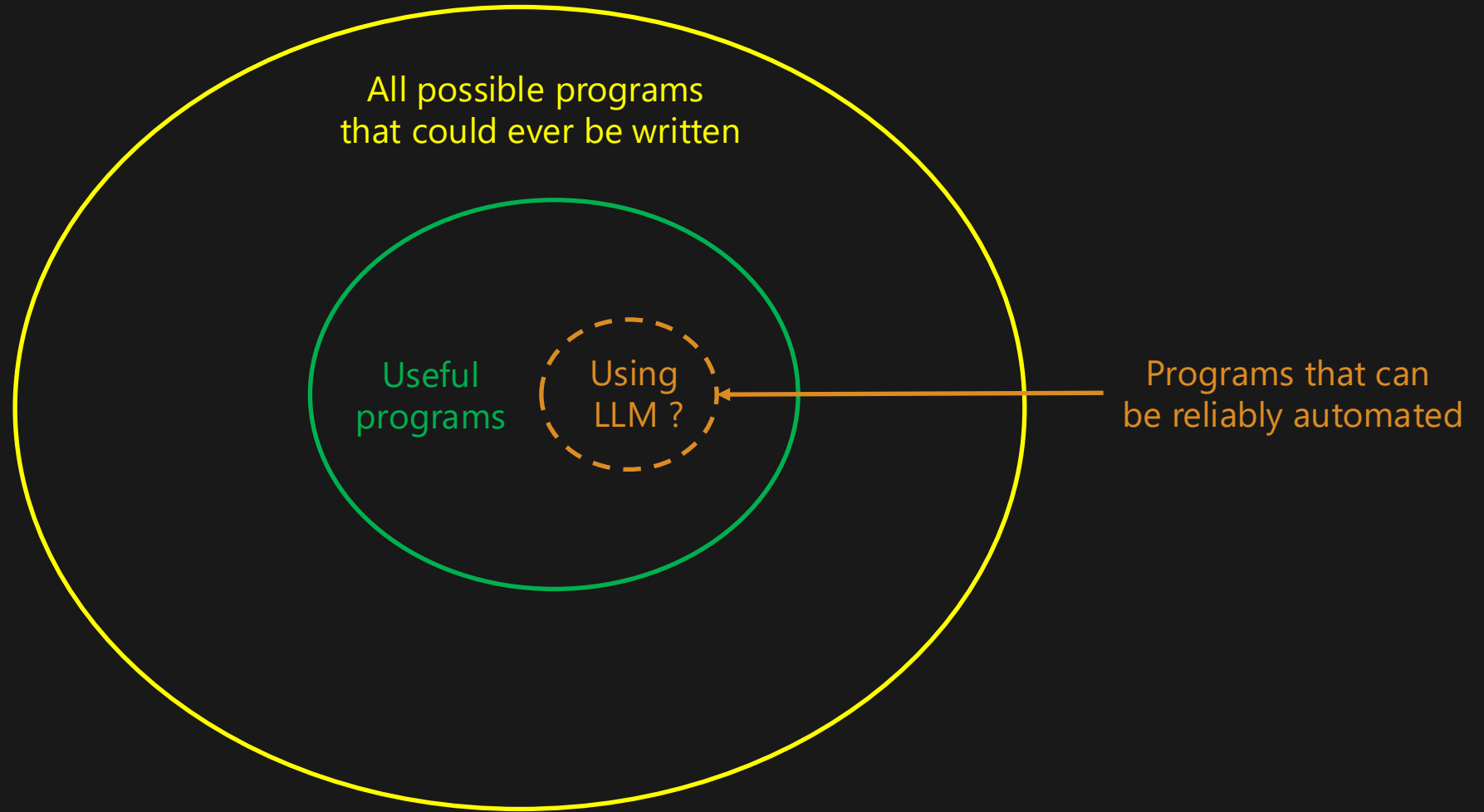
$$\alpha = \frac{\text{Machine Work}}{\text{Human Specification}}$$

Machine work: Low-level machine instructions executed

Human specification: Lines of code/length of prompt

Each new paradigm removes a constraint, enabling higher abstraction and greater amplification.

Program space



Only a subset of useful programs can be automatically generated by machines.

LLM Output (Without verification)

Prompt:

A car's odometer reads 15951. It is a **palindrome**. What is the next number that will be shown on the odometer ?

Answer (1/2)

A palindrome reads the same forward and backward.

The current reading is:

15951

For a 5-digit palindrome of the form:

abcba

the next palindrome is found by increasing the middle appropriately.

Step 1: Identify the structure

15951 corresponds to:

- a = 1
- b = 5
- c = 9

So it is of the form **1 5 9 5 1**.

Answer (2/2)

Step 2: Find the next palindrome

To get the next larger palindrome, increase the middle digit(s).

Since the middle digit is **9**, we cannot increase it without carrying over.

So we:

- Reset the middle digit to 0
- Increase the left side (15 → 16)

Thus the next palindrome becomes:

1 6 0 6 1

Final Answer:

16061

LLMs still struggle to reliably understand and respond to user query !

LLM Output (With verification)

Verification:

A car's odometer reads 15951. It is a **palindrome**. Is 16061 the next number that will be shown on the odometer ?

Thought for a second

Yes.

From 15951 onward, the numbers go 15952, 15953, and so on. The next palindrome must read the same backward, and the first one after 15951 is **16061**.

So the next palindrome shown is **16061**.

📄 👍 🗨️ ⬆️ 🔄 ⋮

A car's odometer reads 15951. Is 16061 the next number that will be shown on the odometer ?

Thought for a second

No. The next number shown on the odometer is **15952**.

16061 is the next **palindrome**, not the next number.

📄 👍 🗨️ ⬆️ 🔄 ⋮

No mention of palindrome

LLMs will make mistakes, but *verification* can save the day !

Current State of the Art

- ◆ We now have LLMs which are **capable of generating** code of very high quality
- ◆ Today, these LLMs are directly **embedded within our IDE** e.g. Cursor, Visual Studio, etc.
- ◆ We just have to prompt in NL and it **spits out the code** in no time
- ◆ However, even though they are getting extremely good at it, we have to be **extremely careful** and cannot totally trust the code that is being generated
- ◆ The code is only as good as your NL prompt (**garbage in, garbage out**)
- ◆ **Vibe coding** is becoming popular, but is extremely dangerous for production apps
- ◆ A more methodical **Agentic Engineering** is what the industry is pushing
 - <https://addyosmani.com/blog/agentic-engineering/>
- ◆ Many **agent frameworks** are appearing and this area is under active development

Things are moving super fast, and there is a lot of hype and fear-mongering!

Good News: Human Expertise Still Remains Essential

	HUMAN EXPERTISE
Specification difficulty Experts must articulate the problem	✓
Search complexity Humans guide the solution space	✓
Verification difficulty Humans judge correctness and acceptance	✓
Novelty & architecture New ideas and system design	✓

Automation thrives on patterns; human expertise is needed for design, reasoning and verification.

XPF: Agentic AI System for Business Workflow Automation

Published in the Proceedings of the 34th International Symposium on High-Performance Parallel and Distributed Computing (HPDC 2025)

XPF: Agentic AI System for Business Workflow Automation



Agent construction

Natural language-based agent construction; developers can use XPF to define a workflow for the specific business processes and supply necessary tools for the agent to accomplish certain goal

Generated using ChatGPT



Agent Deployment

Deployment of agent on a distributed cluster, scaling and resource management is automatically handled under-the-hood by XPF

Generated using ChatGPT

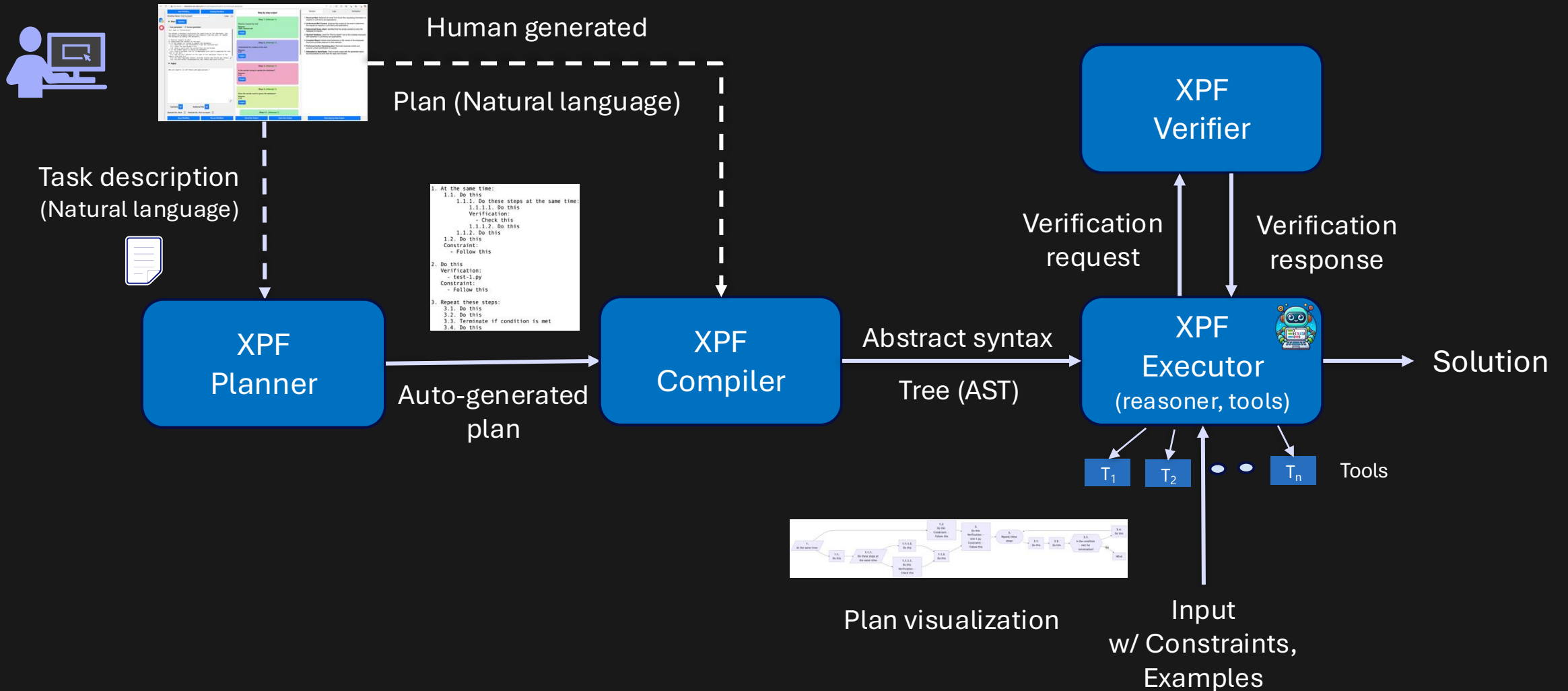


Monitoring and self-adaptation

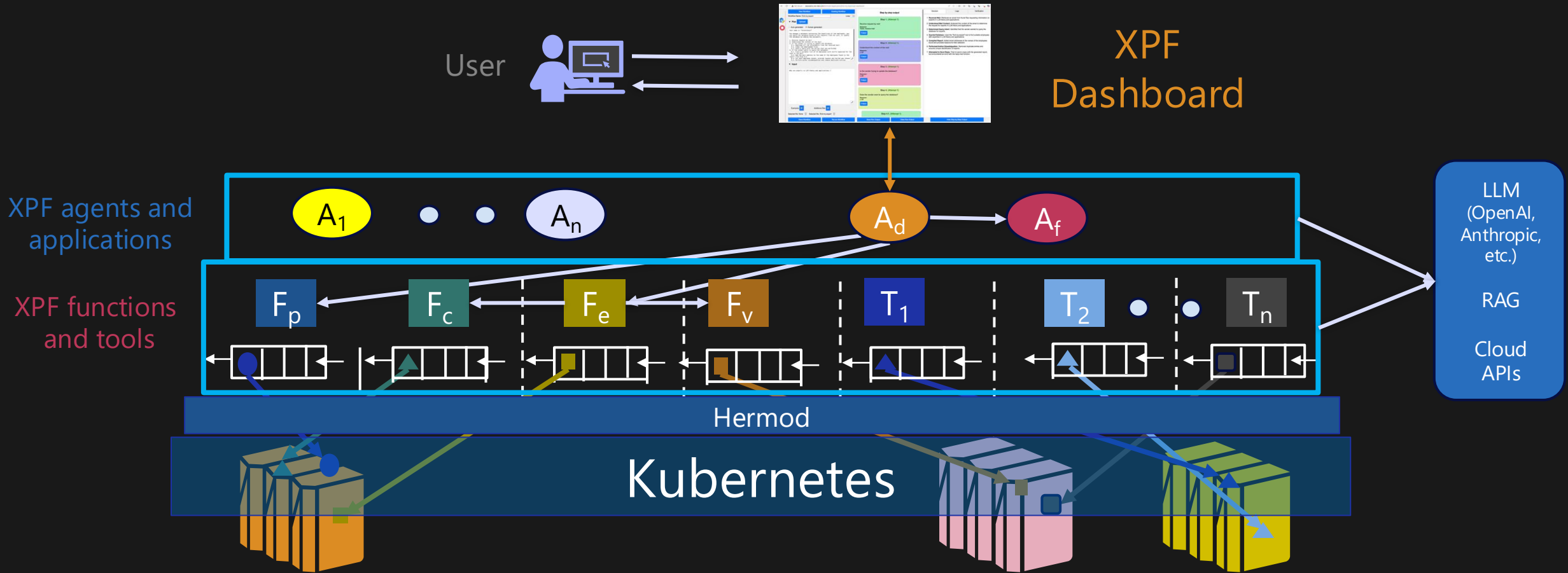
Complete telemetry and logging showing the performance of the agent; agents self-adapt and dynamically start using newly added tools to accomplish any particular task

Generated using ChatGPT

XPF: High Level System Architecture



XPF: Low level System Implementation



A_d XPF Dashboard A_f File manager A_1 • A_n Agents

F_p xpf-planner F_c xpf-compiler F_e xpf-executor F_v xpf-verifier T_1 • T_n Tools

XPF Components

- ◆ **XPF Planner** is used to automatically generate a plan for workflow
 - ◆ **Task description** is used by the planner while generating the plan
 - ◆ Plan can also be human-generated
 - ◆ **Note: XPF uses LLM while automatically generating a plan**
- ◆ **XPF compiler** compiles the provided plan into a workflow, which is represented as a graph
 - Supports parallelism, looping
 - **Note: XPF uses LLM while compiling the workflow**

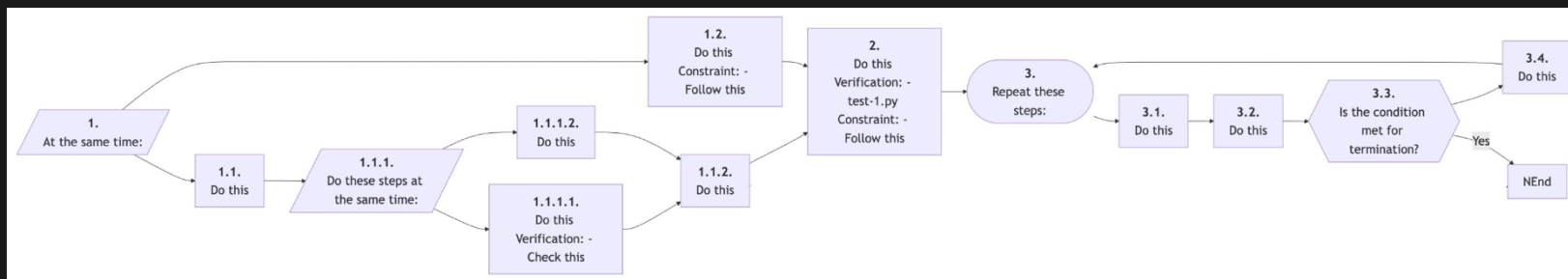
- ◆ **XPF executor** executes the workflow while maintaining parallelism and repetition of steps until termination, if any
 - Necessary **tools** (user provided and pre-set system tools) are automatically used during execution
 - **Note: XPF uses LLM during execution**
- ◆ **XPF Verifier** is used by executor to verify and check if the output of the step is acceptable
 - ◆ Supports natural language-based verification instructions and programs
 - ◆ **Note: XPF uses LLM for verification instructions**

Plan in XPF

Plan Constructs and Structure (parallelism, looping)

1. At the same time:
 - 1.1. Do this
 - 1.1.1. Do these steps at the same time:
 - 1.1.1.1. Do this
Verification:
 - Check this
 - 1.1.1.2. Do this
 - 1.1.2. Do this
 - 1.2. Do this
Constraint:
 - Follow this
2. Do this
Verification:
 - test-1.py
Constraint:
 - Follow this
3. Repeat these steps:
 - 3.1. Do this
 - 3.2. Do this
 - 3.3. Terminate if condition is met
 - 3.4. Do this

Plan compilation (visualization)



Agents on XPF

- ◆ **Bike Safety:** The goal of this agent is to detect bike riders within 1.5 meters of a vehicle and not wearing a helmet.
- ◆ **Marine Security:** The goal of this agent is to detect unauthorized motorized vehicle in marine environment and send them an audio alert, when detected.
- ◆ **Poster:** The goal of this agent is to automatically generate poster (2 powerpoint slides) in a very specific format for a given topic.
- ◆ **Whitepaper:** The goal of this agent is to generate a whitepaper for a given topic and then summarize it for an executive in 250 words.
- ◆ **Graph:** The goal of this agent is to generate a graph in JSON format describing the configuration for an enterprise network. The nodes and edges of this graph need to follow certain constraints.

Demo

Bike Safety Agent

Task description

Check if the image has a bicycle rider with helmet and report distance of each rider from other vehicles in the image. Generate a 25 words report.

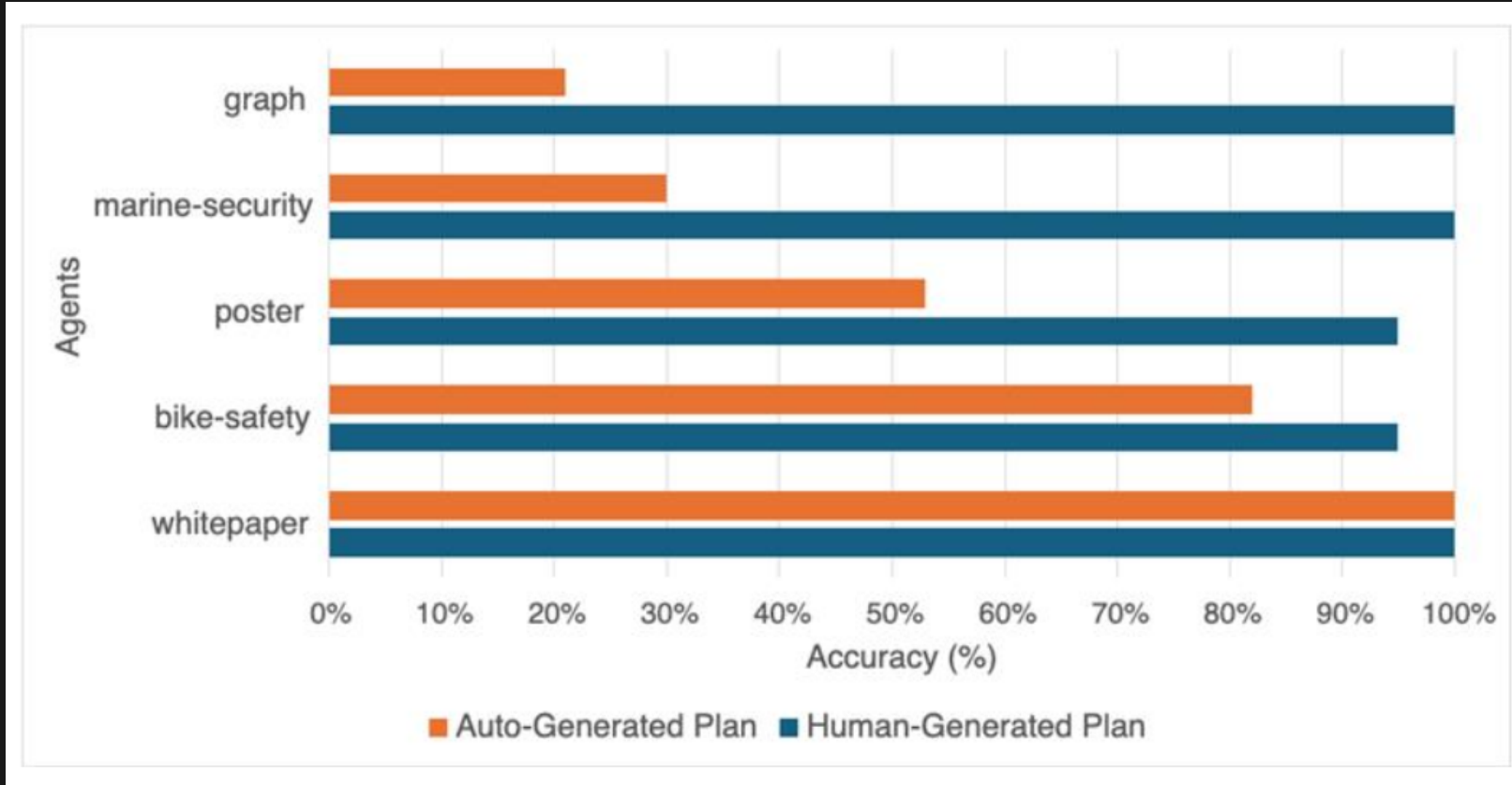
Auto-generated plan

1. Use the glip tool to detect and locate bicycle riders in the image.
2. Use the moondream2-vqa tool to check if each detected bicycle rider is wearing a helmet.
3. Use the glip tool to detect and locate other vehicles in the image.
4. Use the depth-anything-distance tool to calculate the distance between each bicycle rider and other vehicles.
5. Compile the findings into a 25-word report.

Human-generated plan

1. Find the vehicles in the image.
 2. Find a description (kind, make, type, color) of each vehicle found.
 3. Discard the vehicles that are bicycles.
 4. Find the bicycles in the image.
 5. For each bicycle find its rider; the rider of a bicycle is the closest person to the bicycle.
 6. Check each bicycle rider if they are wearing a helmet.
 7. Find the real-world distance in meters between each vehicle found in step 3 and each bicycle.
 8. Generate a table that includes: bicycle location, vehicle location, if the bicycle rider is wearing the helmet, the description of the vehicle and the distance between vehicle and bicycle.
 9. Generate a report which is less than 25 words.
- Verification:
- 25_words_count_verifier.py

Results (1/2)



*We choose 2 different inputs for each agent and execute using "auto-generated" and "human-generated" plans (10 times each). Fig. shows the average accuracy (across 10 runs for each input).

Results (2/2)



*Fig. shows a box plot indicating the similarity using Rouge-L score for each input for each agent. We note that in all cases, the reliability with "human-generated" plan is higher than "auto-generated" plan.

Conclusion

- ◆ We have come a long way from **build simple applications** using assembly programming to **complex enterprise applications** using high-level programming languages and now we are transitioning into a **completely new method of application development** (using Natural Language Instructions)
- ◆ The dearth of programmers has constantly been met with invention of higher-level abstractions and ease of programming to the point where today, **everyone can be a programmer**
- ◆ We introduced XPF, an **agentic AI system** with various components including **planner, compiler, executor, and verifier** to help users develop, test, and deploy agents to automate business workflows across diverse applications.
- ◆ The future of application engineering is super exciting where humans interact (in **Natural language**) and **collaborate with AI** to build systems !

The future of application engineering is specifying intent and collaborating with AI to build systems.

 **Orchestrating** a brighter world

NEC

NEC Laboratories America