Advanced Software Engineering – TDT4242

AI for Software Engineering

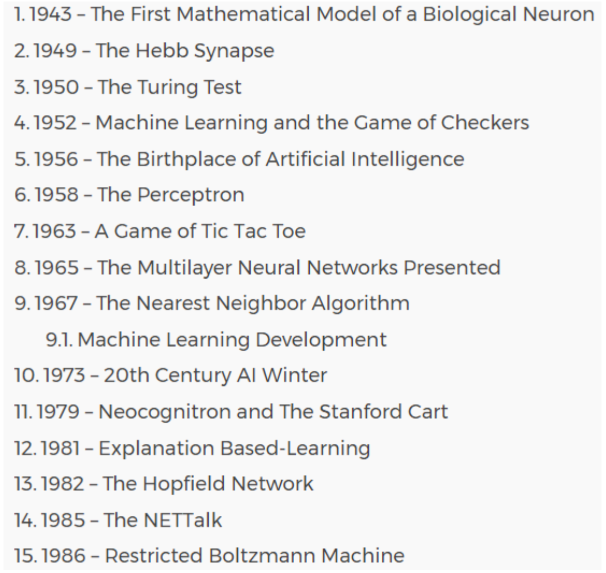
Anh Nguyen-Duc & Jingyue Li

Jan 2026

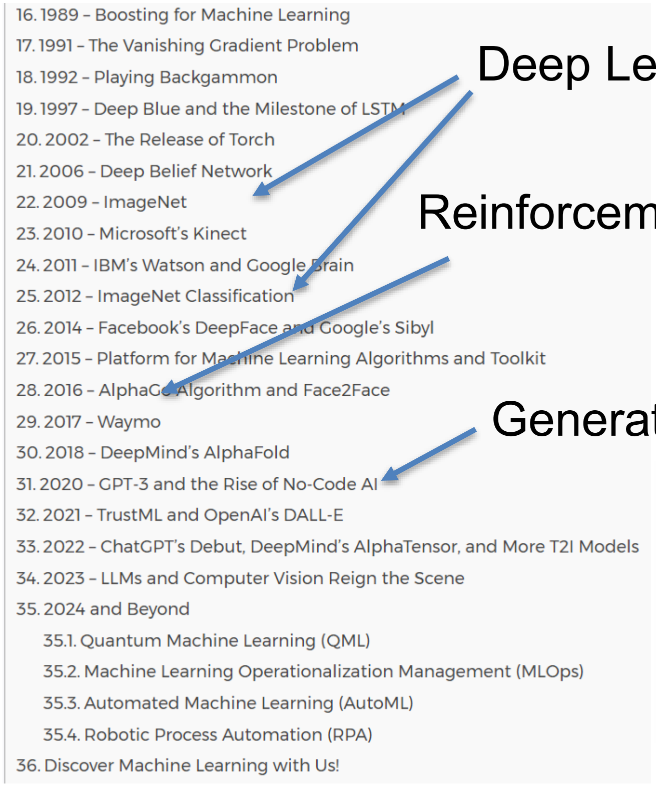
Agenda

- ➢ Classical AI for software engineering
- Generative AI for software engineering
- Augmented, compound, and agentic AI for software engineering

The history of ML*



1. 1943 – The First Mathematical Model of a Biological Neuron
2. 1949 – The Hebb Synapse
3. 1950 – The Turing Test
4. 1952 – Machine Learning and the Game of Checkers
5. 1956 – The Birthplace of Artificial Intelligence
6. 1958 – The Perceptron
7. 1963 – A Game of Tic Tac Toe
8. 1965 – The Multilayer Neural Networks Presented
9. 1967 – The Nearest Neighbor Algorithm
    9.1. Machine Learning Development
10. 1973 – 20th Century AI Winter
11. 1979 – Neocognitron and The Stanford Cart
12. 1981 – Explanation Based-Learning
13. 1982 – The Hopfield Network
14. 1985 – The NETTalk
15. 1986 – Restricted Boltzmann Machine



16. 1989 – Boosting for Machine Learning
17. 1991 – The Vanishing Gradient Problem
18. 1992 – Playing Backgammon
19. 1997 – Deep Blue and the Milestone of LSTM
20. 2002 – The Release of Torch
21. 2006 – Deep Belief Network
22. 2009 – ImageNet
23. 2010 – Microsoft's Kinect
24. 2011 – IBM's Watson and Google Brain
25. 2012 – ImageNet Classification
26. 2014 – Facebook's DeepFace and Google's Sibyl
27. 2015 – Platform for Machine Learning Algorithms and Toolkit
28. 2016 – AlphaGo Algorithm and Face2Face
29. 2017 – Waymo
30. 2018 – DeepMind's AlphaFold
31. 2020 – GPT-3 and the Rise of No-Code AI
32. 2021 – TrustML and OpenAI's DALL-E
33. 2022 – ChatGPT's Debut, DeepMind's AlphaTensor, and More T2I Models
34. 2023 – LLMs and Computer Vision Reign the Scene
35. 2024 and Beyond
    35.1. Quantum Machine Learning (QML)
    35.2. Machine Learning Operationalization Management (MLOps)
    35.3. Automated Machine Learning (AutoML)
    35.4. Robotic Process Automation (RPA)
36. Discover Machine Learning with Us!

Deep Learning (DL)

Reinforcement learning

Generative AI

RAG,
Agentic
AI …

* https://www.startechup.com/blog/machine-learning-history/

Typical applications of AI in Software Engineering

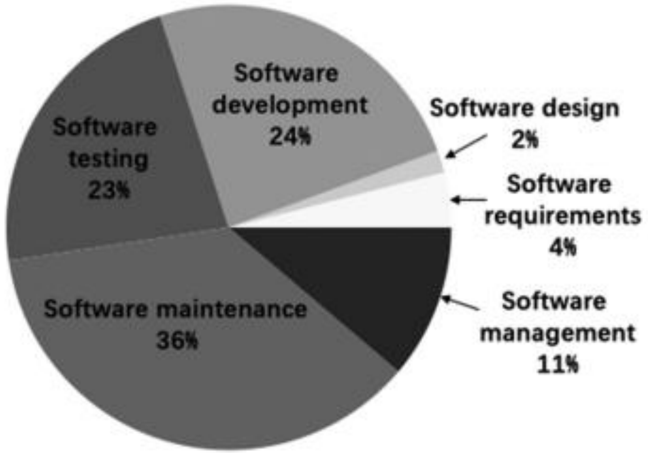


Fig. 2. The distribution of DL techniques in Different SE activities.

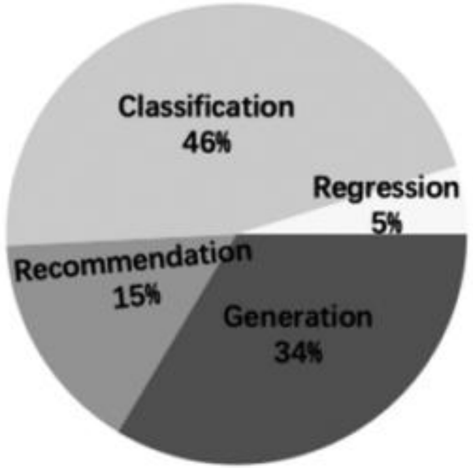


Fig. 3. The classification of primary studies.

*

* Yang et al. "A Survey on Deep Learning for Software Engineering." ACM Comput. Surv. 54, 10s, Article 206 (January 2022).

Supervised learning: Classification

Labels

Features

- Given $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Learn a function f(x) to predict y given x
  - y is categorical == classification

```
f(LOC, cohesion, cyclomatic complexity)
         ↓
Long Method ?  Yes / No
```

In Software Engineering:

Learn the labels of patterns of software artifacts

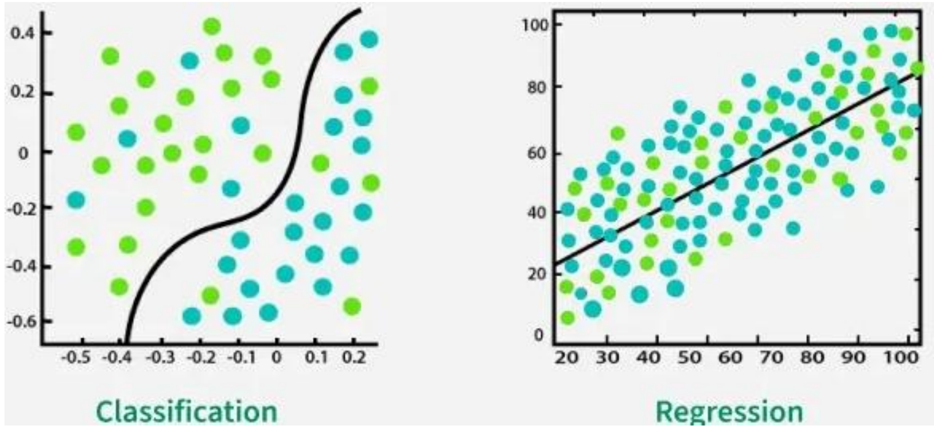
Features:
- Lines of code
- Cohension of method
- Class cohesion
- McCabe's cyclomatic
- Class Cohesion (LCOM)
- Number of parameters

Label:
- Is a long method code smell
- Is not a long method code smell

* Liu et al. "Deep Learning Based Code Smell Detection," in IEEE Transactions on Software Engineering, 2021.

Supervised learning: Regression

- y is a numerical value == regression
- regression model maps features to numerical value
- Unlike classification (“yes/no”, “smell/no smell”), regression answers:
  - How much?
  - How long?
  - How complex?
  - How risky?



SE Problem	Regression Output
Effort estimation	Person-hours
Defect prediction	Expected defect count
Maintainability	Maintainability index
Performance	Response time (ms)
Technical debt	Estimated remediation cost

Unsupervised learning: Clustering

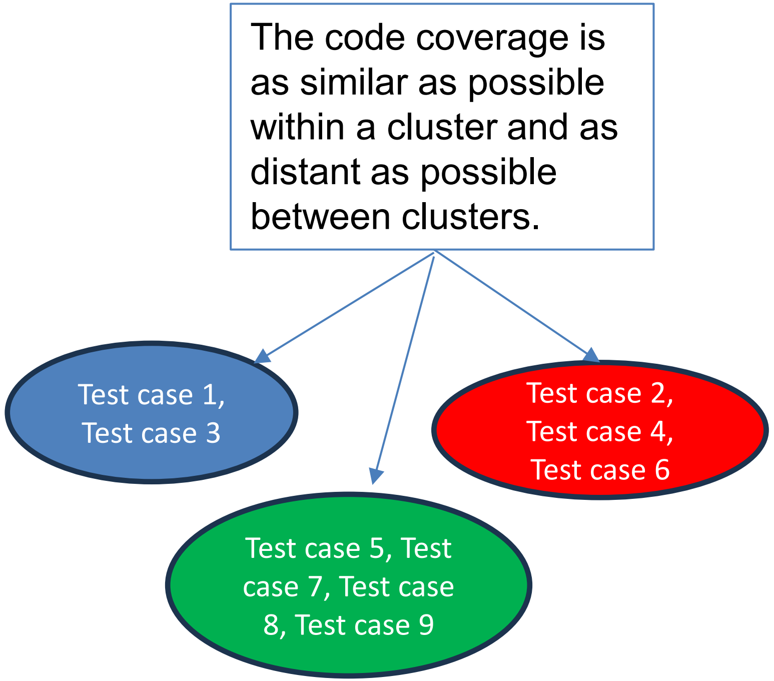
Features

- Given x_1, x_2, ..., x_n (without labels)
- Output hidden structure behind the x's
  - E.g., clustering

Using clustering to improve regression test case prioritization*

Features:

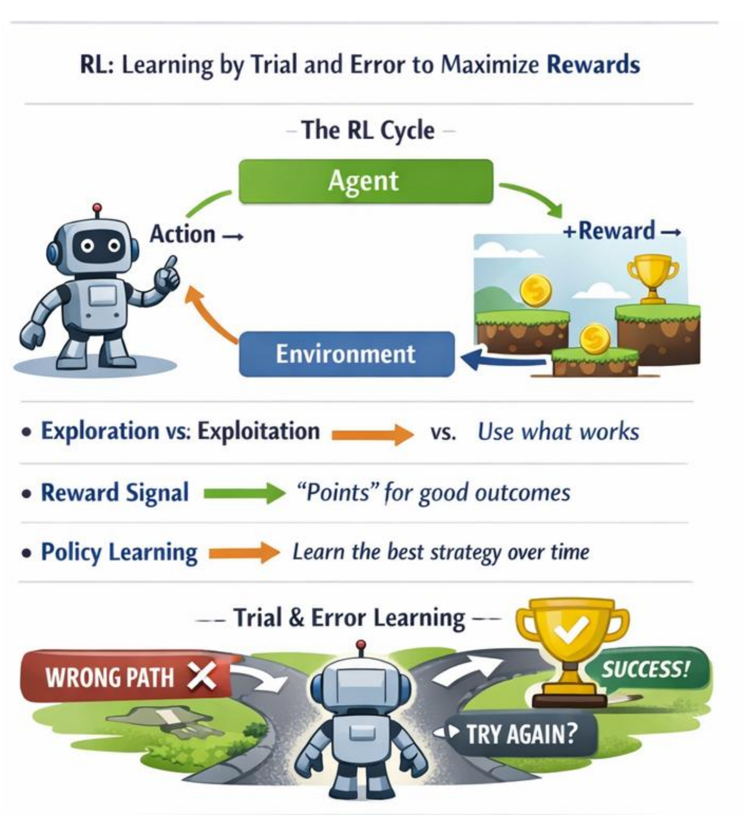
- Code coverage of regression test cases

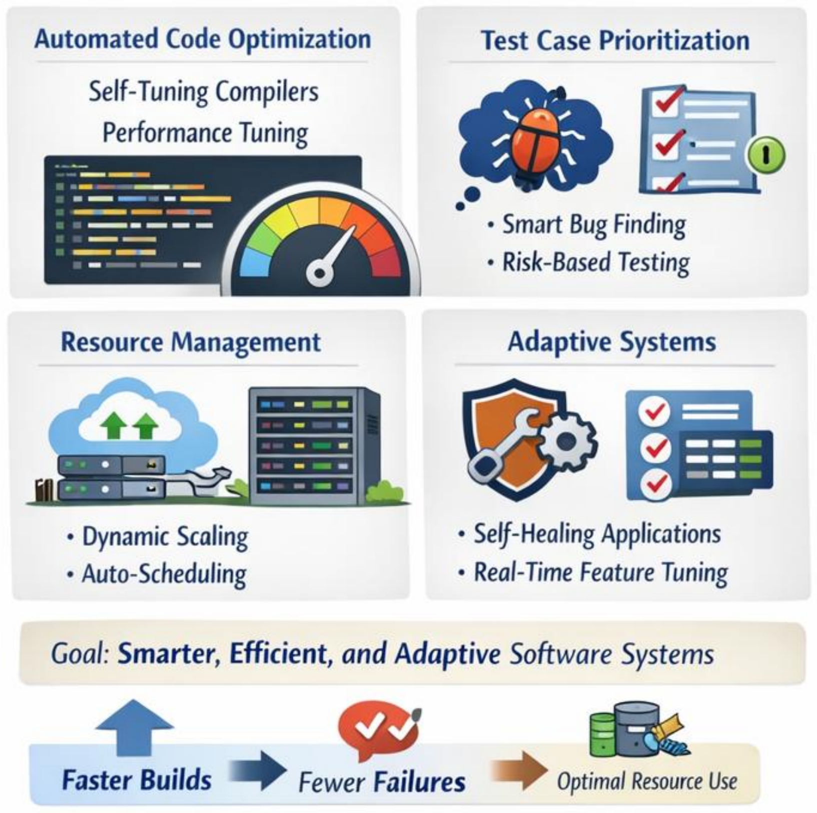

The code coverage is as similar as possible within a cluster and as distant as possible between clusters.

Test case 1, Test case 3
Test case 2, Test case 4, Test case 6
Test case 5, Test case 7, Test case 8, Test case 9

* Carlson et al. "A clustering approach to improving test case prioritization: An industrial case study," 27th IEEE International Conference on Software Maintenance (ICSM), Williamsburg, VA, USA, 2011.

Reinforcement learning (RL)





Building a AI model for specific SE task?

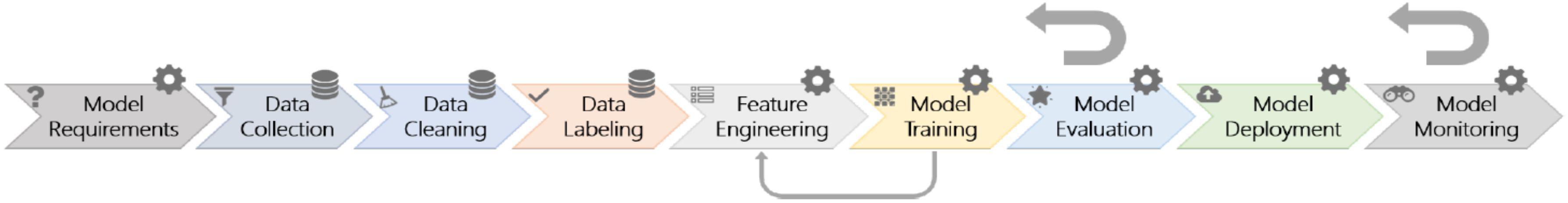


Fig. 1. The nine stages of the machine learning workflow. Some stages are data-oriented (e.g., collection, cleaning, and labeling) and others are model-oriented (e.g., model requirements, feature engineering, training, evaluation, deployment, and monitoring). There are many feedback loops in the workflow. The larger feedback arrows denote that model evaluation and monitoring may loop back to any of the previous stages. The smaller feedback arrow illustrates that model training may loop back to feature engineering (e.g., in representation learning).

- In SE application, not only «just train a model»
- A socio-technical workflow involving data, developers, decisions and tools

* S. Amershi et al., "Software Engineering for Machine Learning: A Case Study," ICSE 2019.

Data collection & cleaning

- Common SE data sources:
  - Version Control System
  - Issues trackers (Jira, Github issues)
  - CI/ CD logs
  - Static analysis tools
  - Runtime logs
  - Code reponsiory

- Key challenges with SE data
  - Noisy
  - Incomplete
  - Highly contextual
  - Imbalanced (few bugs, many clean files)
- Typical SE cleaning tasks:
  - Remove duplicates commits or issues
  - Handle missing values in metrics
  - Normalize time stamps
  - Filter bot-generated data in logs

Feature engineering

- Represent software artifact numerically
- Select, transform raw data into constructing variables
- Classical features:
    - Code metrics (LoC, cyclomatic)
    - Change metrics (churn, no. of commits)
    - Developer activity metrics
    - Dependency metrics

Model training

- Learn pattern from data
  - Define an objective
  - Adjust model parameters to improve the objective
  - Use data and feedback to guide learning

- Supervised learning
  - Regression: mean squared error
    - Fed data to model
    - Compared predicted y value with the true y value
  - Classification: cross entropy
    - Update model parameters

- Unsupervised learning
  - Fed data into model
  - Model produce internal reprsentation
  - Model tries to reconstruct the input
  - Minimise the reconstruction error
  - Parameter is updated

Model evaluation

- Asses whether models are useful
- Standard ML metrics:
  - Accuracy, Percision, Recall, F1
  - AUC curve
- Ranking metrics (used in search, recommendation, retrieval, RAG):
  - Precision@k, Recall@k
  - Mean Average Precision (MAP)
- Regression metrics:
  - Mean Absolute Error (MAE)
  - Root Mean Squared Error (RMSE)

Out of the **top k results** the system returned, how many are actually relevant?
How well relevant items are ranked early

Average size of prediction errors
Average error with heavy penalty on big mistakes

Exercise 1 (10 minutes)

- You are tasked with designing a machine learning system that can automatically detect **potential bugs in source code** before the software is deployed. The goal is to help developers identify error-prone code segments such as null pointer risks, incorrect condition logic, resource leaks, or misuse of APIs. The system should analyze program elements (e.g., functions, classes, or code snippets) and predict whether they are likely to contain a bug.
- Q1. Which type of machine learning would be most suitable for automatic bug detection (supervised, unsupervised, or reinforcement learning)?
- Q2. What kind of data should be collected to train the model, and what features or code representations?
- Q3. Which metrics would you use to evaluate if the model works?

Possible issues of ML

Issue	What it Means	Why It’s a Problem in Practice
Data Drift	The input data distribution changes over time	Model was trained on old world, deployed in new world → accuracy silently drops
Concept Drift	The relationship between input and output changes	The definition of the problem itself shifts (not just the data)
Training–Serving Skew	Data in production ≠ data used in training	Pipeline mismatch, feature engineering inconsistency
Non-Determinism	Same code ≠ same model every run	Harder debugging, testing, reproducibility
Lack of Explainability	Hard to trace why a prediction happened	Violates SE principles of traceability & accountability
Hidden Technical Debt	Data dependencies replace code dependencies	Changes outside the codebase break the system

Agenda

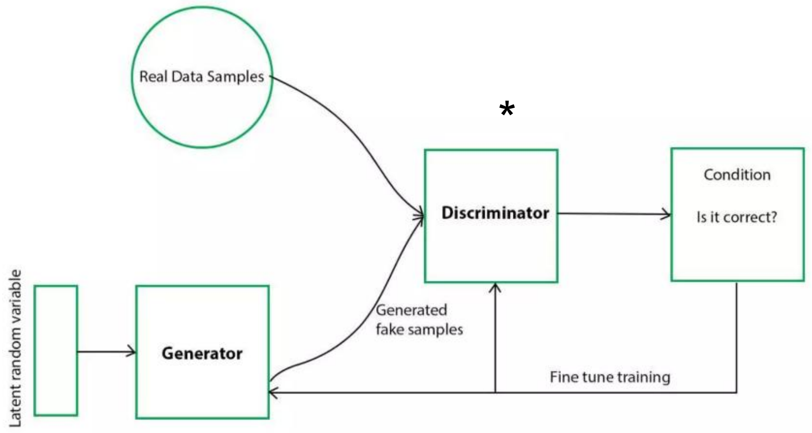
- Classical AI for software engineering
- ➢ Generative AI for software engineering
- Augmented, compound, and agentic AI for software engineering

Generative AI: Image generator

Generative adversarial network (GAN)

- A Generator and Discriminator
- Generator generates fake samples of data and tries to fool the Discriminator.
- Discriminator tries to distinguish between the real and the fake samples.
- They compete with each other in the training phase.
- The steps are repeated until the generated samples are not much distinguishable from the real samples (can fool the Discriminator).

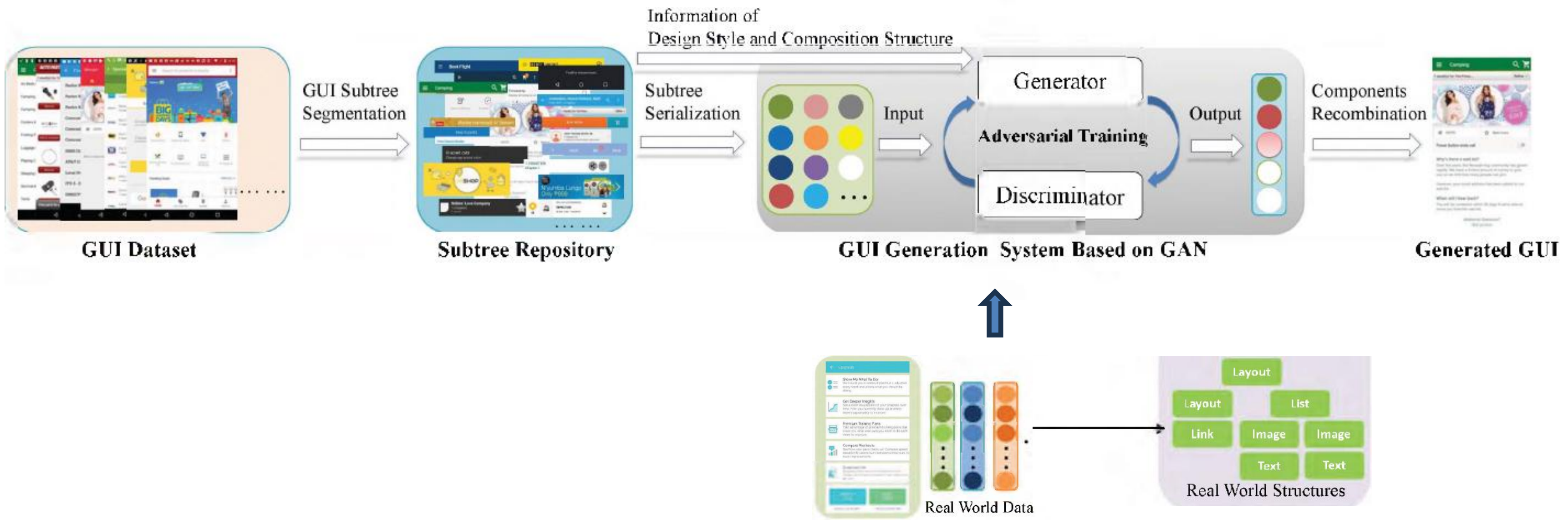
*



* https://www.slideshare.net/PrakharRastogi23/generative-adversarial-network-gan

GAN for software engineering

Using GAN to generate brand new GUI designs for designers' inspiration.*

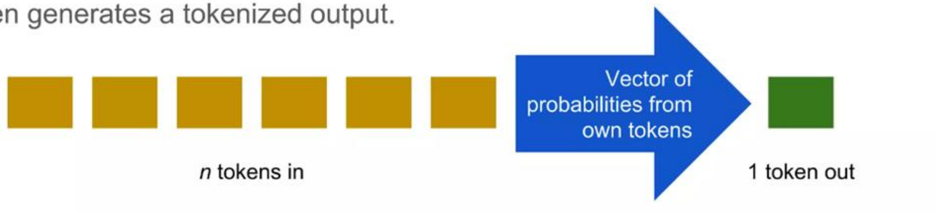


* Zhao et al. "GUIGAN: Learning to Generate GUI Designs Using Generative Adversarial Networks," IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021

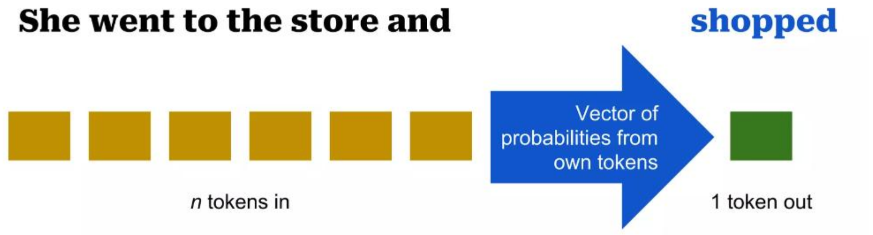
Generative AI: Large Language Models

The prompts are converted into tokens *(= “chunks” of words, punctuation marks, pixels, etc.)*, then the system analyzes what is likely to come next, based on the tokens in its own dataset (as many as 32,000 in GPT-4!). *

It then generates a tokenized output.

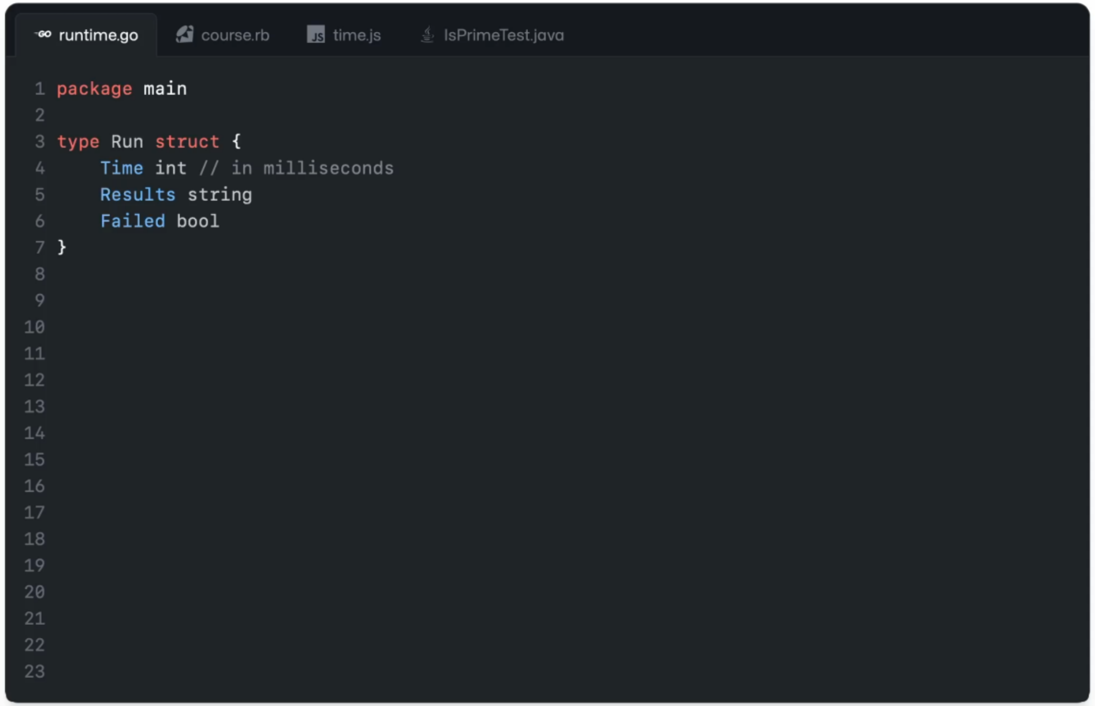


With each output, it keeps re-analyzing the probabilities to decide next tokens.


She went to the store and
shopped
Vector of probabilities from own tokens
n tokens in
1 token out

* https://www.slideshare.net/CoriFaklaris/an-introduction-to-generative-ai

Code completion using Co-pilot



```go
package main

type Run struct {
    Time int // in milliseconds
    Results string
    Failed bool
}
```

Prompt Engineering

- Designing, formatting, and optimizing conversational prompts to better guide the discourse with AI
- Including AI chatbots, AI customer service agents, voice-first applications, and other AI interaction interfaces.

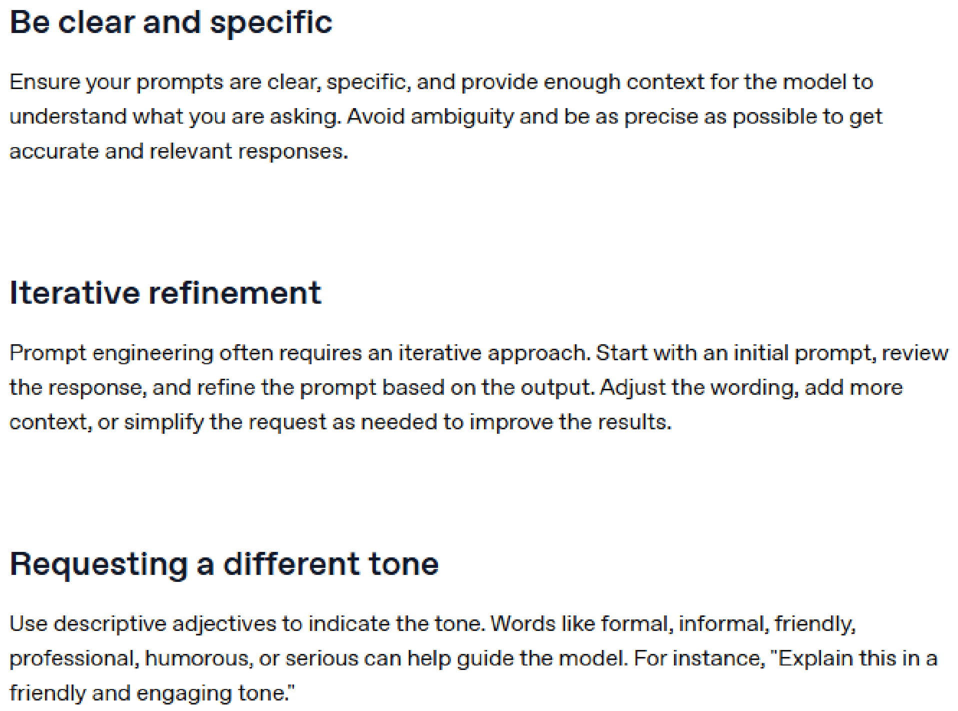
For CS professionals

- Enable rapid prototyping
- Faster idea-to-implementation
- “improve” productivity
- Prompt as a collaborative artifact

For Non-CS professionals

- Programming via natural language (vibe coding, etc)
- Democratize power of programming
- Shift focus to problem-solving
- leveraging computational resources

Prompt Engineering – ChatGPT best practices



Be clear and specific

Ensure your prompts are clear, specific, and provide enough context for the model to understand what you are asking. Avoid ambiguity and be as precise as possible to get accurate and relevant responses.

Iterative refinement

Prompt engineering often requires an iterative approach. Start with an initial prompt, review the response, and refine the prompt based on the output. Adjust the wording, add more context, or simplify the request as needed to improve the results.

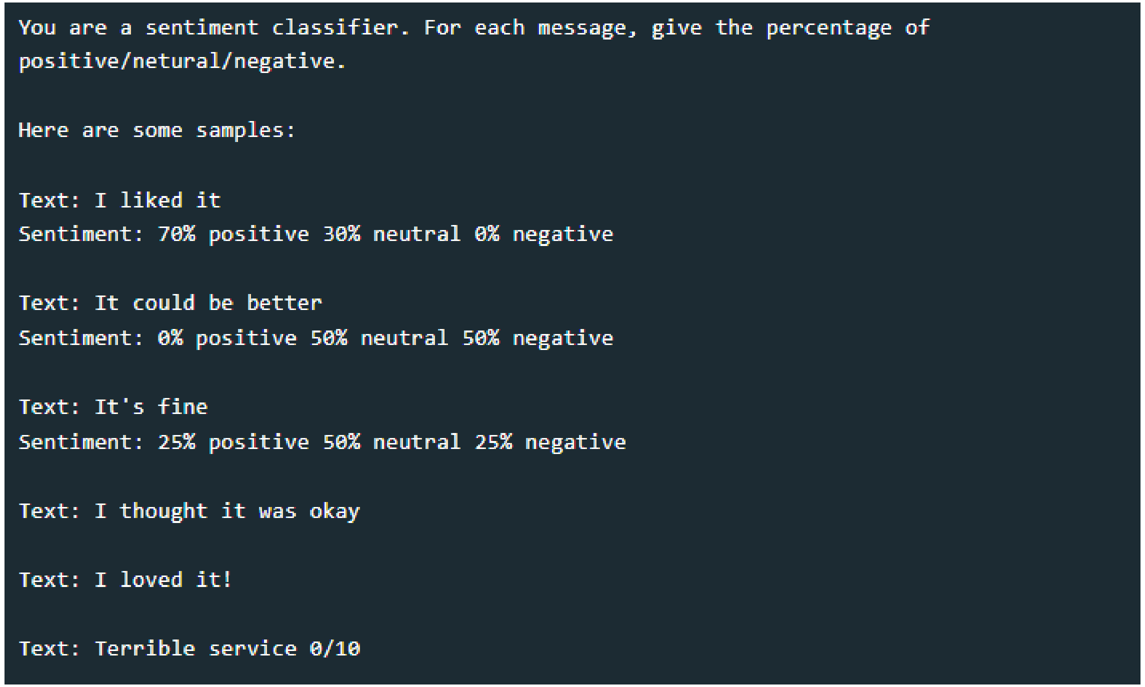
Requesting a different tone

Use descriptive adjectives to indicate the tone. Words like formal, informal, friendly, professional, humorous, or serious can help guide the model. For instance, "Explain this in a friendly and engaging tone."

ChatGPT

https://help.openai.com/en/articles/10032626-prompt-engineering-best-practices-for-chatgpt

Prompt Engineering – Fewshot learning



```
You are a sentiment classifier. For each message, give the percentage of
positive/netural/negative.

Here are some samples:

Text: I liked it
Sentiment: 70% positive 30% neutral 0% negative

Text: It could be better
Sentiment: 0% positive 50% neutral 50% negative

Text: It's fine
Sentiment: 25% positive 50% neutral 25% negative

Text: I thought it was okay

Text: I loved it!

Text: Terrible service 0/10
```

https://www.llama.com/docs/how-to-guides/prompting/

Prompt Engineering – Role-based prompts

```
You are a senior full-stack web developer.
Help me build a web app step by step using React + Node.js.

Rules:
- Explain briefly, then give code
- One file at a time
- Ask before moving to the next step
- Use clean and secure coding practices

App idea: [describe]

What should we build first?
```

https://www.llama.com/docs/how-to-guides/prompting/

Prompt Engineering – Role-based prompts

You are a senior full-stack software engineer with strong experience in:

- Frontend: HTML, CSS, JavaScript, React
- Backend: Node.js, Express
- Databases: MongoDB / PostgreSQL
- Software engineering practices: clean code, modular design, REST APIs, security best practices

Your job is to help me design and implement a web application step by step like a professional developer working in a real team.

Guidelines:
1. Always explain your reasoning briefly before writing code.
2. Produce only ONE file of code at a time.
3. Clearly state the file name at the top (e.g., `/server/index.js`).
4. If information is missing, ask for clarification before coding.
5. Do not skip steps in the development process.

We are building:
[A short description of the app, e.g., "a task management web app where users can register, log in, and manage personal tasks"]

Start by:
- Proposing the system architecture
- Listing the required files and folders
- Then ask me which file to generate first.

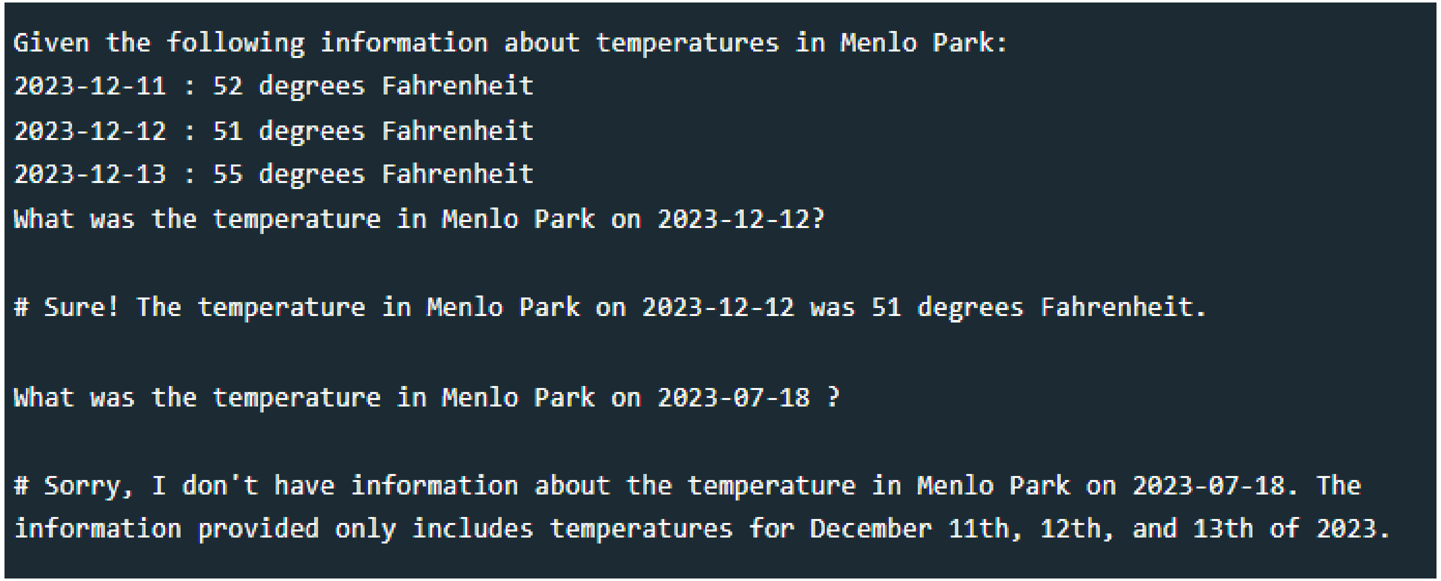
Prompt Engineering – Role-based prompts

```
You are a senior software engineer acting as a professional code reviewer in an experienced development team. Your expertise includes:
- Clean code principles
- Readability and maintainability
- Code Smell for web app development
- Testing and edge cases
Your job is NOT to rewrite the code immediately, but to review it like in a real pull request.
When reviewing code, follow this structure:
1. **Summary**
  - What the code does
  - Overall quality impression
2. **Strengths**
  - Good practices used
  - Clear design decisions
3. **Issues Found**
  Categorize by severity:
  - 🔴 Critical (bugs, security risks, logic errors)
  - 🟠 Important (performance, maintainability, edge cases)
  - 🟡 Minor (style, naming, readability)
4. **Specific Suggestions**
  - Point to exact lines or parts
  - Explain why it's an issue
  - Suggest improvements, but do not rewrite everything
5. **Missing Considerations**
  - Error handling
  - Input validation
  - Testing
  - Scalability
Guidelines:
- Be constructive, not harsh.
- Explain reasoning clearly for learning purposes.
- Assume the developer is competent but can improve.

Here is the code to review:
[PASTE CODE HERE]
```

Prompt Engineering – Chain of thoughts

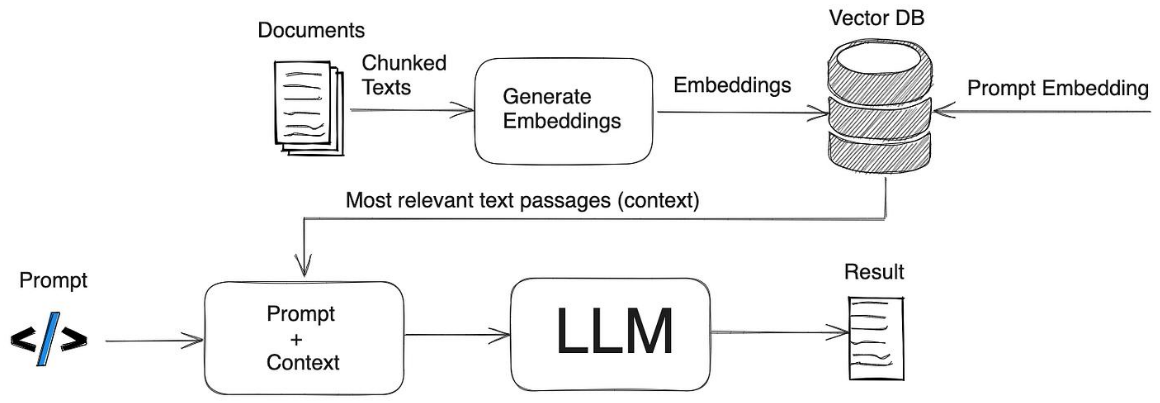
```
You are a senior software engineer helping debug an issue.
Analyze the problem step by step using the following
reasoning process:
1. **Understand the Goal**
  - What is the program supposed to do?
2. **Describe Current Behavior**
  - What is actually happening?
  - What error messages or incorrect outputs appear?
3. **Locate the Problem Area**
  - Which part of the code is most likely responsible?
  - Explain why.
4. **List Possible Causes**
  - Provide multiple hypotheses ranked by likelihood.
5. **Test Each Hypothesis Mentally**
  - For each possible cause, explain how it would lead to the
observed issue.
6. **Identify the Most Probable Root Cause**
7. **Propose a Fix**
  - Show only the corrected part of the code
  - Brief explanation of why it works
8. **Suggest Preventive Measures**
  - How could this bug be avoided in the future?
Here is the code and error:
[PASTE CODE + ERROR MESSAGE]
```

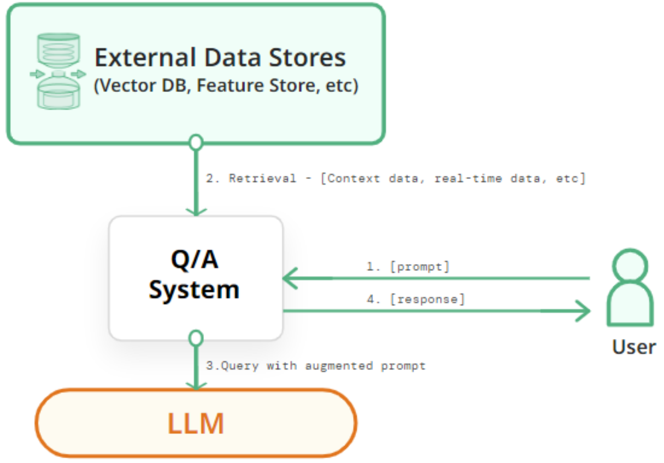
Prompt Engineering – RAG



```
Given the following information about temperatures in Menlo Park:
2023-12-11 : 52 degrees Fahrenheit
2023-12-12 : 51 degrees Fahrenheit
2023-12-13 : 55 degrees Fahrenheit
What was the temperature in Menlo Park on 2023-12-12?

# Sure! The temperature in Menlo Park on 2023-12-12 was 51 degrees Fahrenheit.

What was the temperature in Menlo Park on 2023-07-18 ?

# Sorry, I don't have information about the temperature in Menlo Park on 2023-07-18. The
information provided only includes temperatures for December 11th, 12th, and 13th of 2023.
```

Prompt Engineering – RAG




External Data Stores
(Vector DB, Feature Store, etc)
2. Retrieval - [Context data, real-time data, etc]
Q/A System
1. [prompt]
4. [response]
User
3.Query with augmented prompt
LLM

https://learnmycourse.medium.com/retrieval-augmented-generation-rag-process-using-an-llm-339430ff0a05

Nguyen-Duc, A., Manh, C. V., Tran, B. A., Ngo, V. P., Chi, L. L., & Nguyen, A. Q. (2025). *An Empirical Study of Multi-Agent RAG for Real-World University Admissions Counseling* (arXiv:2507.11272).

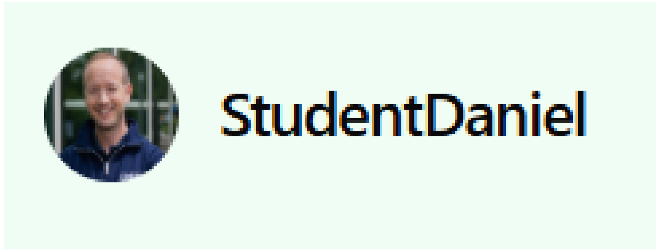
Prompt Engineering for SE tasks

Requirements Elicitation	Requirements Simulator Specification Disambiguation Change Request Simulation
System Design and Simulation	API Generator API Simulator Few-shot Example Generator Domain-Specific Language (DSL) Creation Architectural Possibilities
Code Quality	Code Clustering Intermediate Abstraction Principled Code Hidden Assumptions
Refactoring	Pseudo-code Refactoring Data-guided Refactoring

Schmidt, D. C., Spencer-Smith, J., Fu, Q., & White, J. (2024). Towards a Catalog of Prompt Patterns to Enhance the Discipline of Prompt Engineering. *Ada Lett.*, *43*(2), 43–51. https://doi.org/10.1145/3672359.3672364

Prompt Engineering - Requirements Simulator

- explore the requirements of a software-reliant system interactively to determine if certain functionality is captured properly
- provide additional details regarding the initial requirements



Schmidt, D. C., Spencer-Smith, J., Fu, Q., & White, J. (2024). Towards a Catalog of Prompt Patterns to Enhance the Discipline of Prompt Engineering. *Ada Lett.*, *43*(2), 43–51. https://doi.org/10.1145/3672359.3672364

Prompt Engineering - Specification Disambiguation

- LLM to review specifications provided to a developer or development team by non-technical or semi-technical personnel.

"The following will represent system requirements. Point out any areas that could be construed as ambiguous or lead to unintended outcomes. Provide ways in which the language can be more precise."

1) Create a user interface that allows file uploads via web browser.
2) Keep the files secure.
3) Make sure the files are movie file types.
4) Files can be no larger than 100 MB

The system requirement to "Keep the files secure" could be made more precise by specifying which security measures should be implemented to ensure the files are kept secure. For example: "Implement access control and encryption measures to ensure that uploaded files can only be accessed by authorized users and are protected against unauthorized access or tampering."

Prompt Engineering - Architectural Possibilities

- Generates several different architectures for developers to consider, with little effort on the part of developers

> *”I am developing a python web application using FastApi that allows users to publish interesting ChatGPT prompts, similar to twitter. Describe three possible architectures for this system. Describe the architecture with respect to modules and the functionality that each module contains.”*

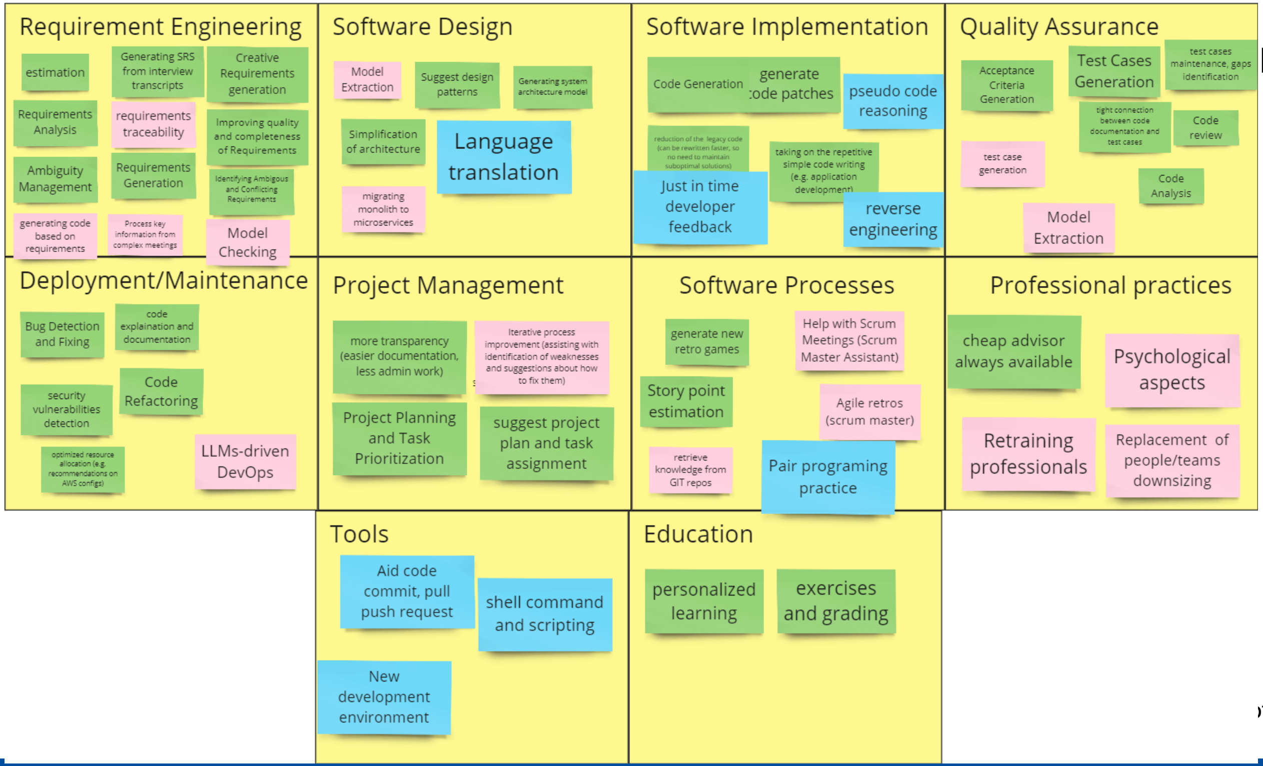
White, J., Hays, S., Fu, Q., Spencer-Smith, J., & Schmidt, D. C. (2023). *ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design*

Exercise 2 (10 minutes)

- Applying prompt patterns to create a prompt that asks an AI to help design a mobile app for reminding students about class assignments.
- Your prompt should guide the app development from strategic and design level. The AI should not generate any code, but instead provide an organized overview of how the app should work and what would be needed to build it.

GenAI for SE – a roadmap (towards 2030+)

Step 1: Move the notes below to the SE areas you think can gain benefits from GenAI tools



Requirement Engineering
- estimation
- Generating SRS from interview transcripts
- Creative Requirements generation
- Requirements Analysis
- requirements traceability
- Improving quality and completeness of Requirements
- Ambiguity Management
- Requirements Generation
- Identifying Ambigous and Conflicting Requirements
- generating code based on requirements
- Process key information from complex meetings
- Model Checking

Software Design
- Model Extraction
- Suggest design patterns
- Generating system architecture model
- Simplification of architecture
- Language translation
- migrating monolith to microservices

Software Implementation
- Code Generation
- generate code patches
- pseudo code reasoning
- reduction of the legacy code (can be rewritten faster, so no need to maintain suboptimal solutions)
- Just in time developer feedback
- taking on the repetitive simple code writing (e.g. application development)
- reverse engineering

Quality Assurance
- Acceptance Criteria Generation
- Test Cases Generation
- test cases maintenance, gaps identification
- tight connection between code documentation and test cases
- Code review
- test case generation
- Code Analysis
- Model Extraction

Deployment/Maintenance
- Bug Detection and Fixing
- code explaination and documentation
- security vulnerabilities detection
- Code Refactoring
- optimized resource allocation (e.g. recommendations on AWS configs)
- LLMs-driven DevOps

Project Management
- more transparency (easier documentation, less admin work)
- Iterative process improvement (assisting with identification of weaknesses and suggestions about how to fix them)
- Project Planning and Task Prioritization
- suggest project plan and task assignment

Software Processes
- generate new retro games
- Help with Scrum Meetings (Scrum Master Assistant)
- Story point estimation
- Agile retros (scrum master)
- retrieve knowledge from GIT repos
- Pair programing practice

Professional practices
- cheap advisor always available
- Psychological aspects
- Retraining professionals
- Replacement of people/teams downsizing

Tools
- Aid code commit, pull push request
- shell command and scripting
- New development environment

Education
- personalized learning
- exercises and grading

r

f ICSE

Nguyen-Duc, A., Cabrero-Daniel, B., Przybylek, A., Arora, C., Khanna, D., Herda, T., Rafiq, U., Melegati, J., Guerra, E., Kemell, K.-K., Saari, M., Zhang, Z., Le, H., Quan, T., & Abrahamsson, P. (u.å.). Generative Artificial Intelligence for Software Engineering—A Research Agenda. *Software: Practice and Experience*, *n/a*(n/a). https://doi.org/10.1002/spe.70005

GenAI for Requirement Engineering

- GenAI support requirements elicitation?
  - iRE agent: simulated stakeholders
- GenAI effectively generate requirements specifications from high-level user inputs?
- Validation against constraints and regulations
- iRE Framework for collecting requirement

Nguyen-Duc, A., Cabrero-Daniel, B., Przybylek, A., Arora, C., Khanna, D., Herda, T., Rafiq, U., Melegati, J., Guerra, E., Kemell, K.-K., Saari, M., Zhang, Z., Le, H., Quan, T., & Abrahamsson, P. (u.å.). Generative Artificial Intelligence for Software Engineering—A Research Agenda. *Software: Practice and Experience*, *n/a*(n/a). https://doi.org/10.1002/spe.70005

Exercise 3 – iRE reflection

- https://www.menti.com/alervphc6b61

GenAI for Software Design

- Potential areas for GenAI application – not dive yet into code generation
- How can GenAI assist in identifying and selecting appropriate design and architectural patterns based on requirements and constraints?
- How can generative AI be utilized to automate the designand implementation of user interfaces?
- How can GenAI optimize the trade-offs between differentquality attributes, such as performance, scalability, security, and maintainability?

Nguyen-Duc, A., Cabrero-Daniel, B., Przybylek, A., Arora, C., Khanna, D., Herda, T., Rafiq, U., Melegati, J., Guerra, E., Kemell, K.-K., Saari, M., Zhang, Z., Le, H., Quan, T., & Abrahamsson, P. (u.å.). Generative Artificial Intelligence for Software Engineering—A Research Agenda. *Software: Practice and Experience*, *n/a*(n/a). https://doi.org/10.1002/spe.70005

Common issues of GenAI for SE

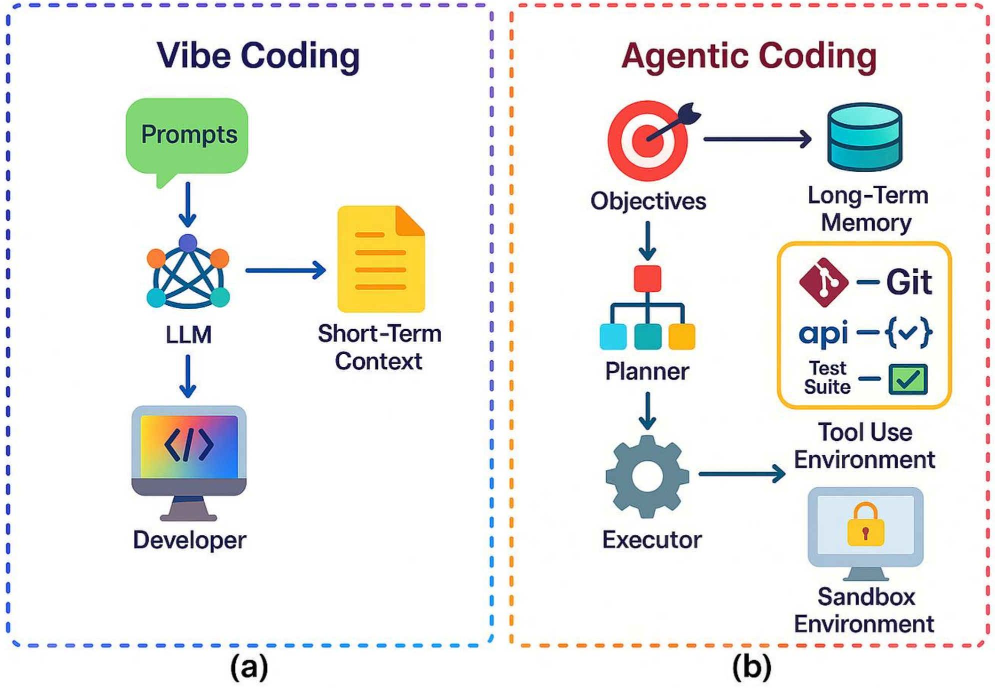
1. Hallucination
2. Prompt Sensitivity
3. Nondeterministic
4. Explainability
5. Context Limitations
6. Generating insecure code
7. High Resource Consumption
8. Data Bias & Fairness Issues
9. Overreliance on AI output
10. Reduced Skill Development
11. Lack of industrial level evaluation

Agenda

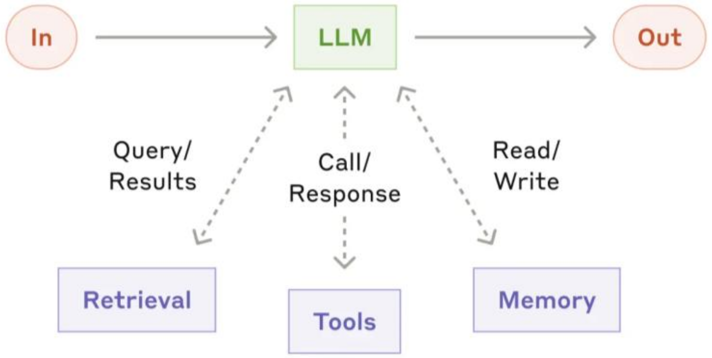
- Classical AI for software engineering
- Generative AI for software engineering
- ➢ Agentic AI for software engineering

Augemented, Compounding and Agentic AI

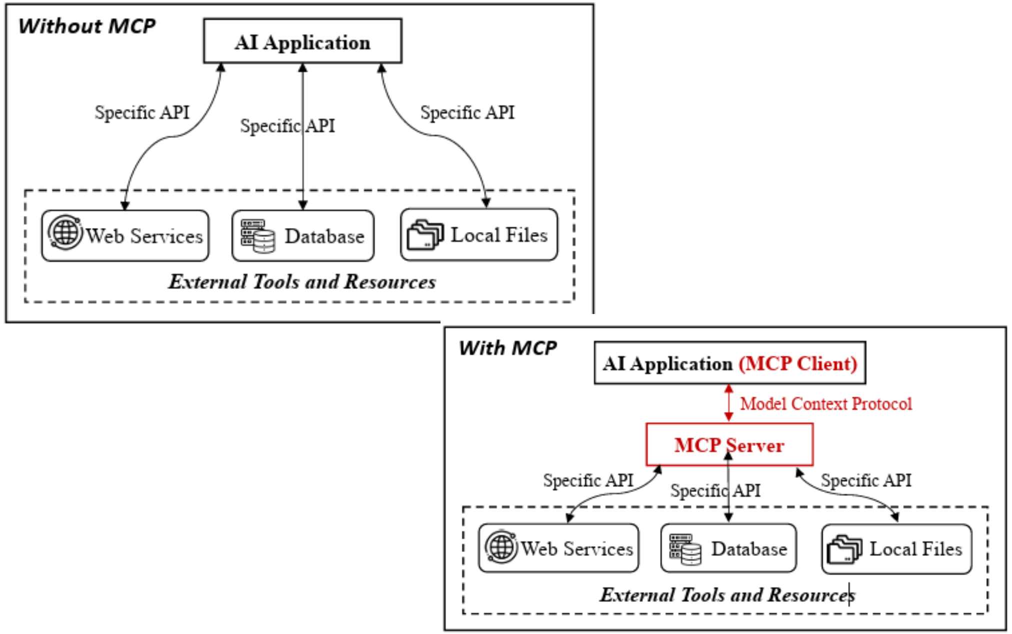
- LLM Agent are evolving fast
- Many agentic libraries, packages, and products
- Agentic vs Vibe coding
- Open agent standards such as MCP for tool use
- Many proposed evaluations: AgentBench, WebArena, AgentDojo, …



Agentic patterns - Augmented LLM



The augmented LLM



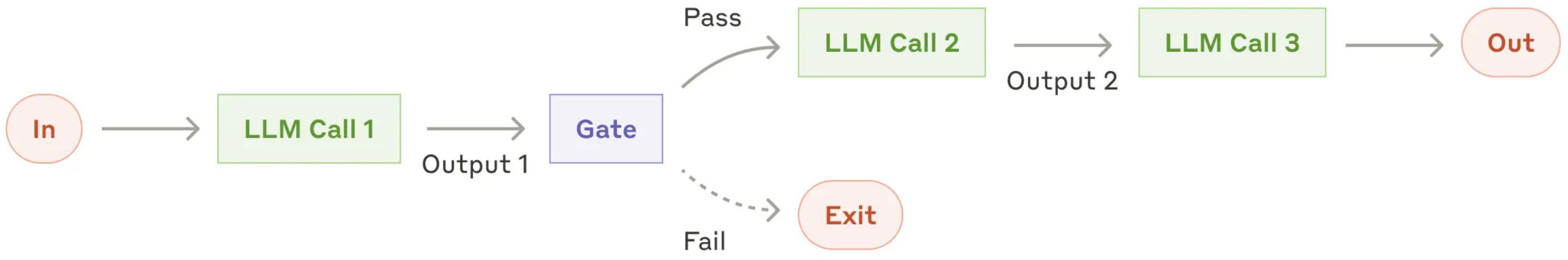
MCP (Model Context Protocol)

https://www.anthropic.com/engineering/building-effective-agents

Hou, X., Zhao, Y., Wang, S., & Wang, H. (2025). *Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions* (arXiv:2503.23278). arXiv. https://doi.org/10.48550/arXiv.2503.23278

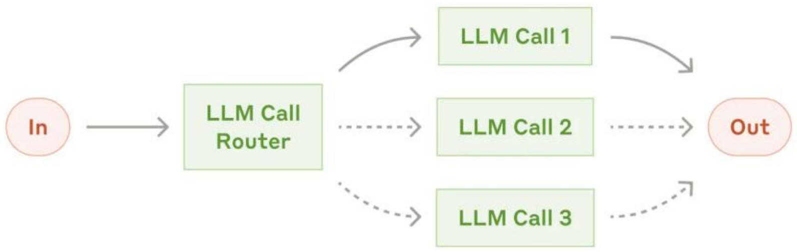
Compounding AI- Prompt chaining

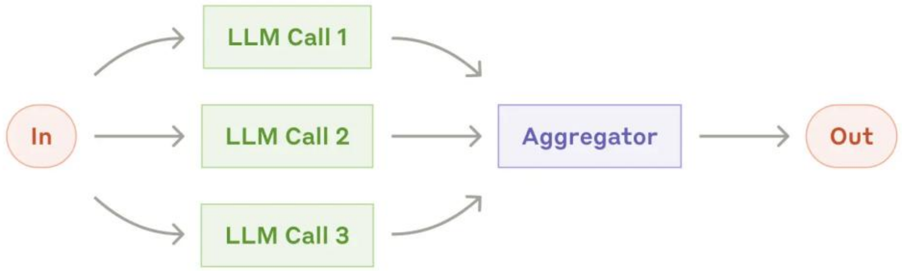
- Combining multiple components, not a single model



https://www.anthropic.com/engineering/building-effective-agents

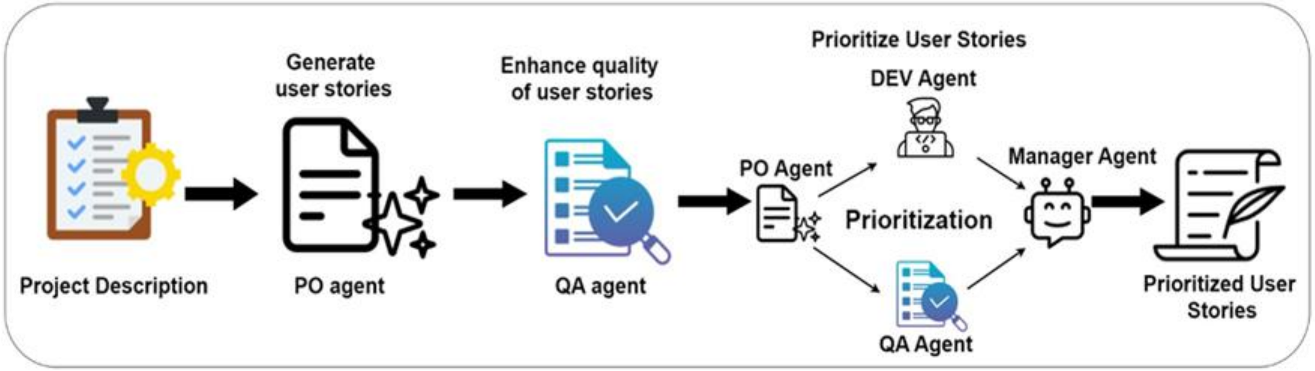
Agentic patterns - Routing





https://www.anthropic.com/engineering/building-effective-agents

Examples of Compound AI for SE



PO: Product owner, use story generation, communication, and prioritization
QA: Quality assurance: quality assurance
DEV: Sugget user story prioritization

Malik Abdul Sami, et al. AI based Multiagent Approach for Requirements Elicitation and Analysis, https://arxiv.org/abs/2409.00038, 2024

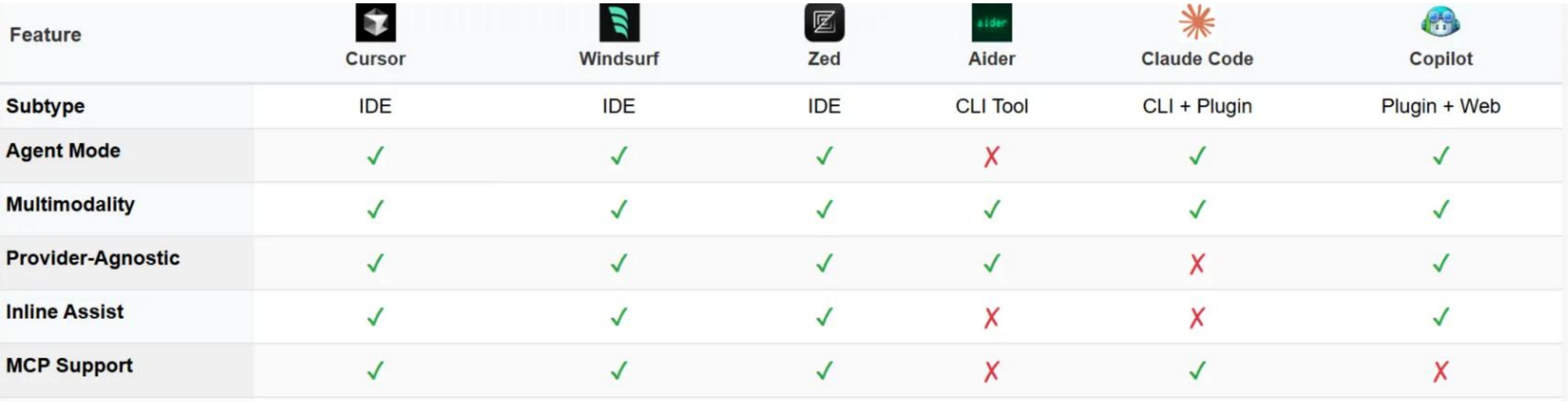
Agentic AI

- Different from compound AI
  - Not all components have to be LLMs
  - There should be an orchestrator
  - The control or information flows do not have to be pre-defined (i.e., the orchestration is more autonomous)
- Towards autonomy: AI behaves like an **agent** that can **plan, decide, and act** toward goals.

https://www.youtube.com/watch?v=O0GNrvO7wD0&t=142s

Agentic AI

- Agentic AI tools in 2025…



Feature	Cursor	Windsurf	Zed	Aider	Claude Code	Copilot
Subtype	IDE	IDE	IDE	CLI Tool	CLI + Plugin	Plugin + Web
Agent Mode	✓	✓	✓	✗	✓	✓
Multimodality	✓	✓	✓	✓	✓	✓
Provider-Agnostic	✓	✓	✓	✓	✗	✓
Inline Assist	✓	✓	✓	✗	✗	✓
MCP Support	✓	✓	✓	✗	✓	✗

https://www.pragmaticcoders.com/resources/ai-developer-tools

Summary

- ML can facilitate software engineering practices
- Classical ML is application/domain specific
- Generative AI leads to new applications
- LLM can also be combined or integrated with other tools