AI-Augmented Software Engineering

1-Day Course @ Zühlke

Zühlke | 19.8.2026 | Zürich

Public

Our Coaches

1-Day Course @ Zühlke

Marco Romanutti

Lead Software Architect

marco.romanutti@zuehlke.com





Jonathan Ziller

Principal Software Architect

jonathan.ziller@zuehlke.com



Lukas Grigis

Lead Software Architect

lukas.grigis@zuehlke.com

Public

The machine knows…





Demo: Introducing GitHub Copilot and the Flight API



LAB: Vibe Coding

Context

We are going to build an application that displays flight delays.

You are given an initial **Java project** that starts successfully but does not yet expose the required REST endpoints.

Familiarize yourself with the repo using GitHub Copilot and implement the endpoints described on the right.

Your Task

Run **task lab/vibe-coding** and implement the following endpoints:

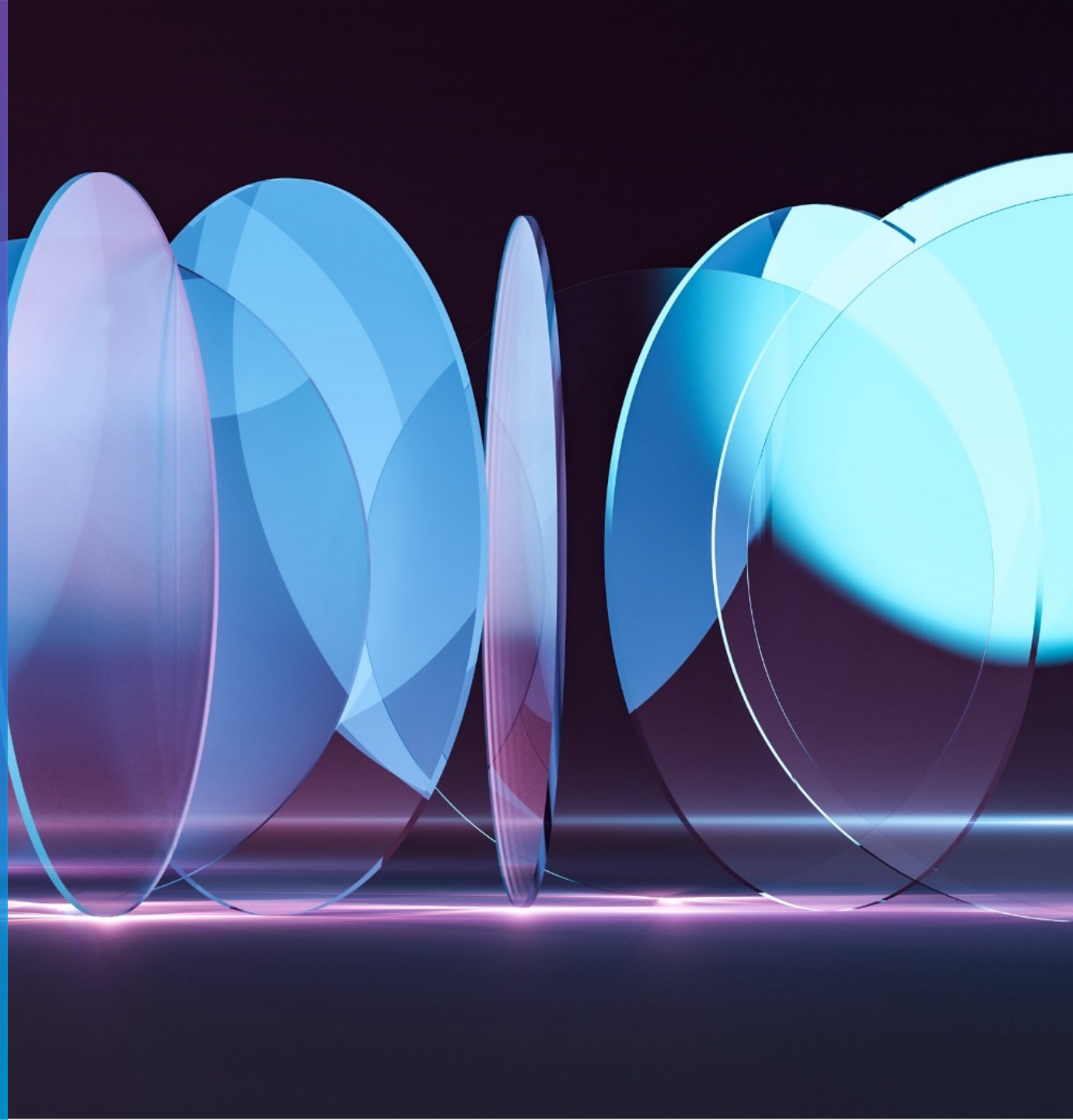
- GET /delays?airline_code=AA&limit=50

  → returns **200 OK** or **404 Not Found**

Tipps & Tricks

Try out the available working modes of your AI coding tool.

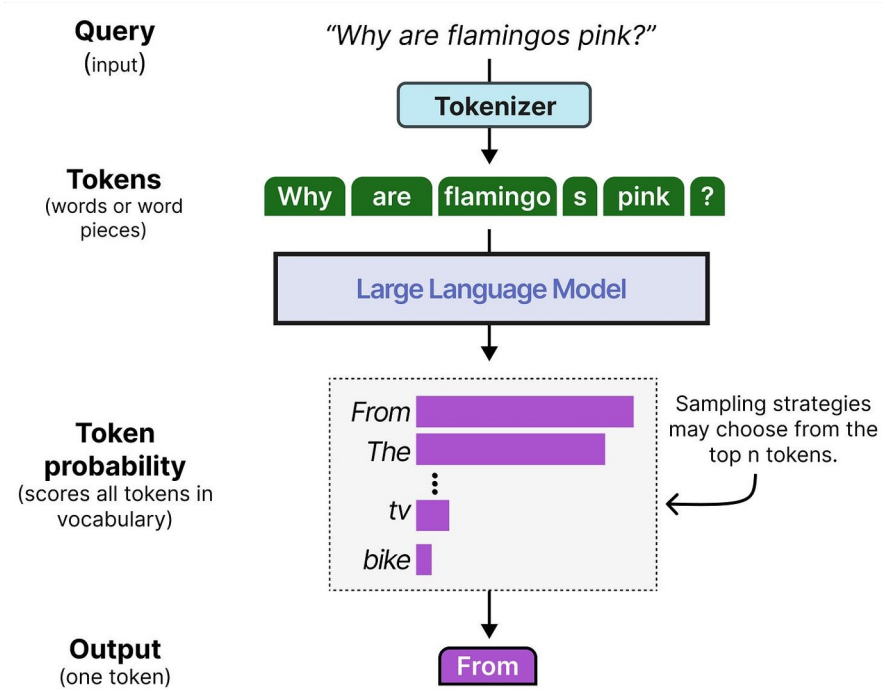
Theoretical Foundation



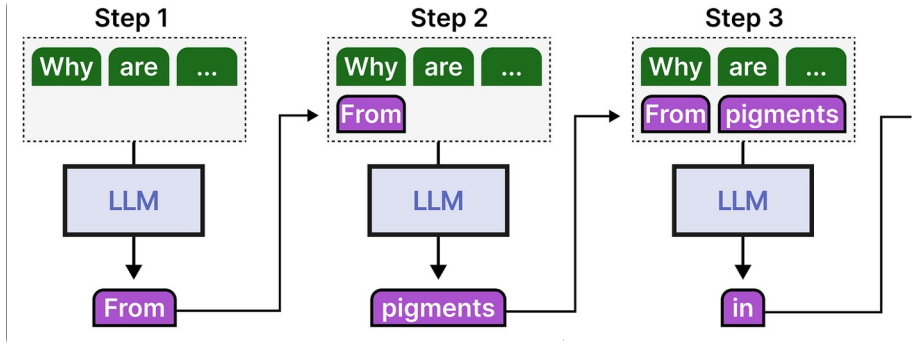
Public

Large Language Model (LLM) are pure word probability machines

A large language model (LLM) is a transformer-based neural network trained to **predict the next token in a text sequence**. For any given input text sequence, it computes a probability distribution over all possible next tokens, from which one token is sampled.



By repeating that step token by token, we can mimic conversations:



Source: An Illustrated Guide to AI Agents, Maarten Grootendorst, Jay Alammar (2026)

Public

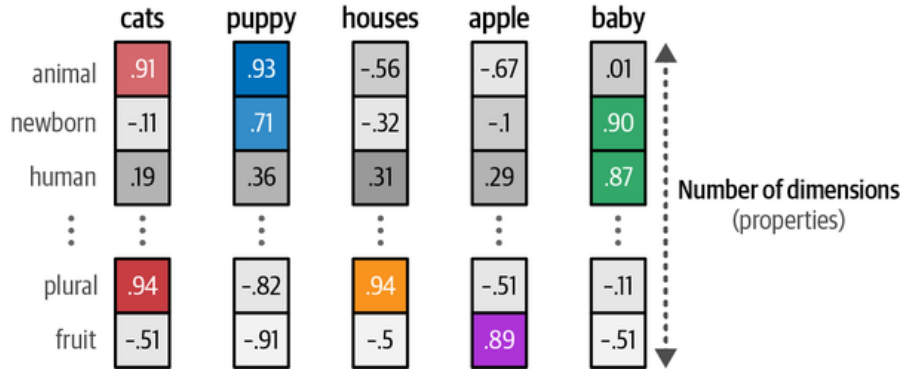
Embeddings are representations of meaning that lets models compare and retrieve related content

Embeddings are **vector representations of tokens**, sentences, or documents that encode their meaning and relationships in a continuous numerical space.

$$V = [v_1, v_2, v_3, v_4]$$

Each v_i is a number representing how much a vector matches a concept. Imagine the example for flamingo:



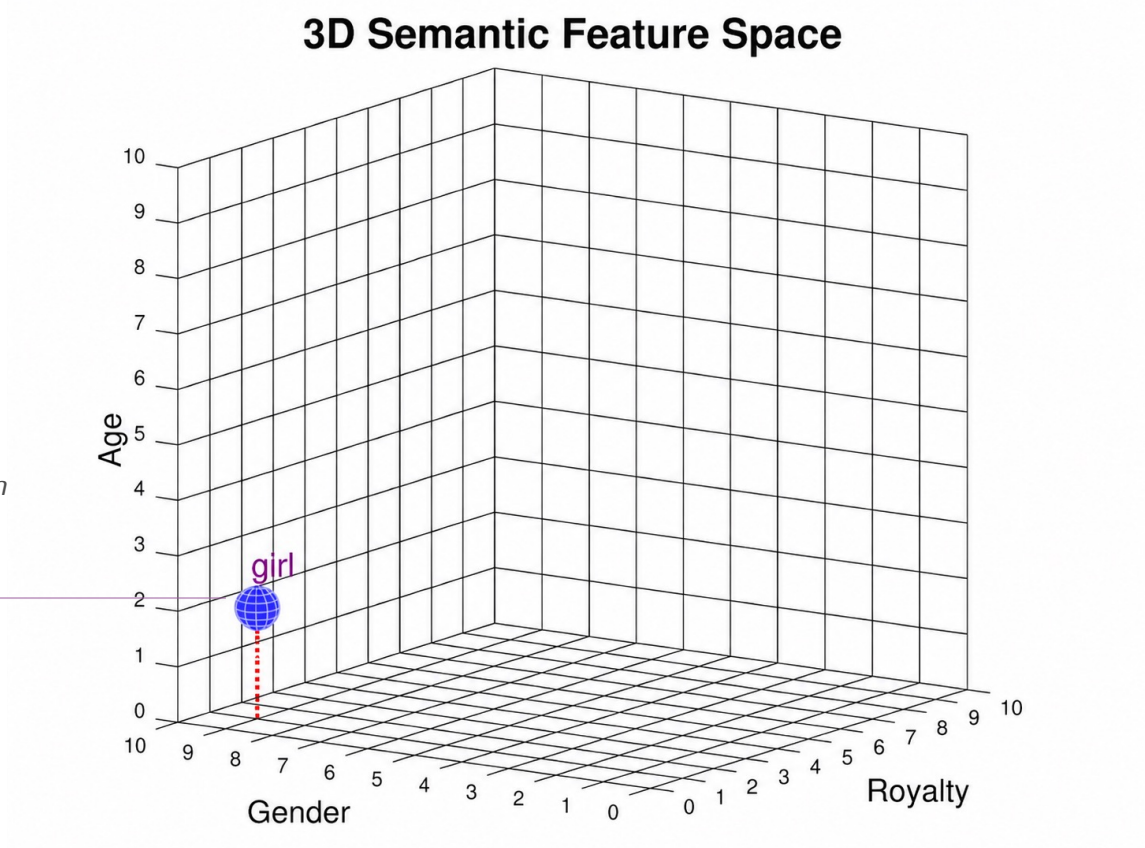


Source: Hands-On Large Language Models, Jay Alammar & Maarten Grootendorst, (2024)

Public

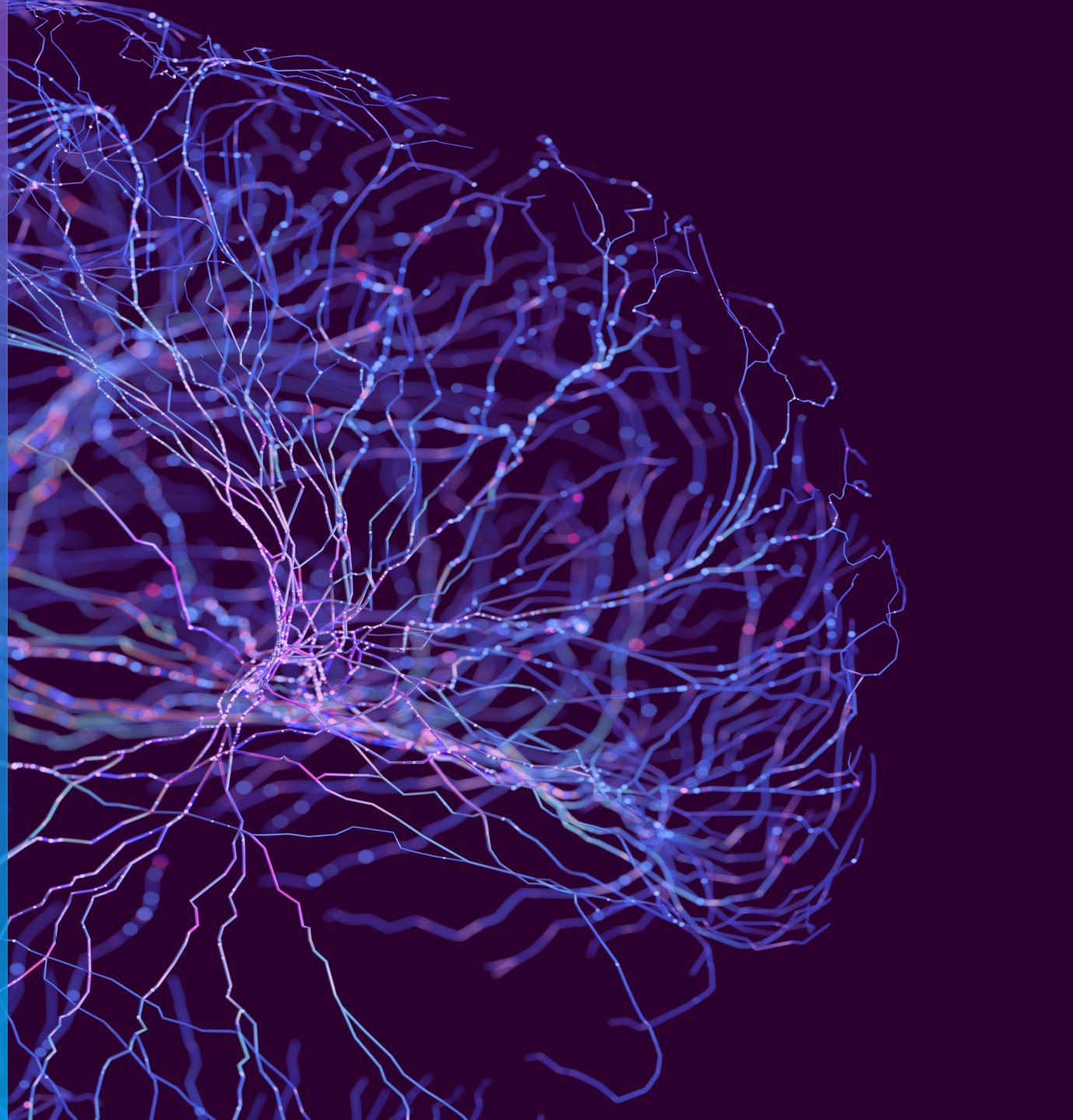
Using distance metrics, we can determine how close tokens are in meaning

When visualized in a simplified three-dimensional space, words reveal semantic relationships, *e.g. the distance between girl → woman and princess → queen is similar as the main thing that differentiates them is the royalty.*



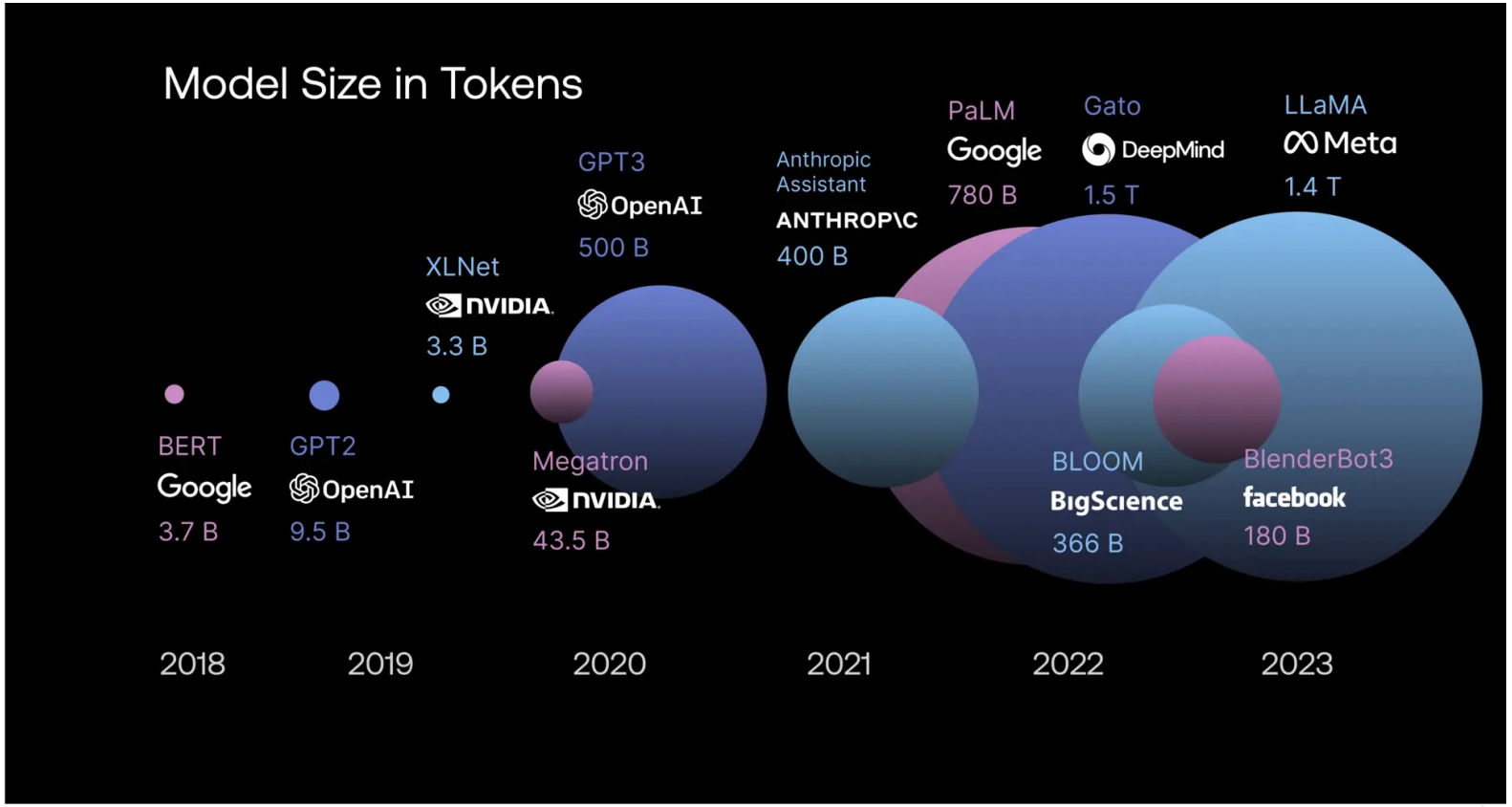
Source: https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html

Different Models



Public

Larger models and higher-dimensional representations can capture more complex language patterns, often leading to more precise and context-aware outputs



Over the past years, model sizes and representation dimensions have increased significantly.

Source: https://scale.com/guides/large-language-models

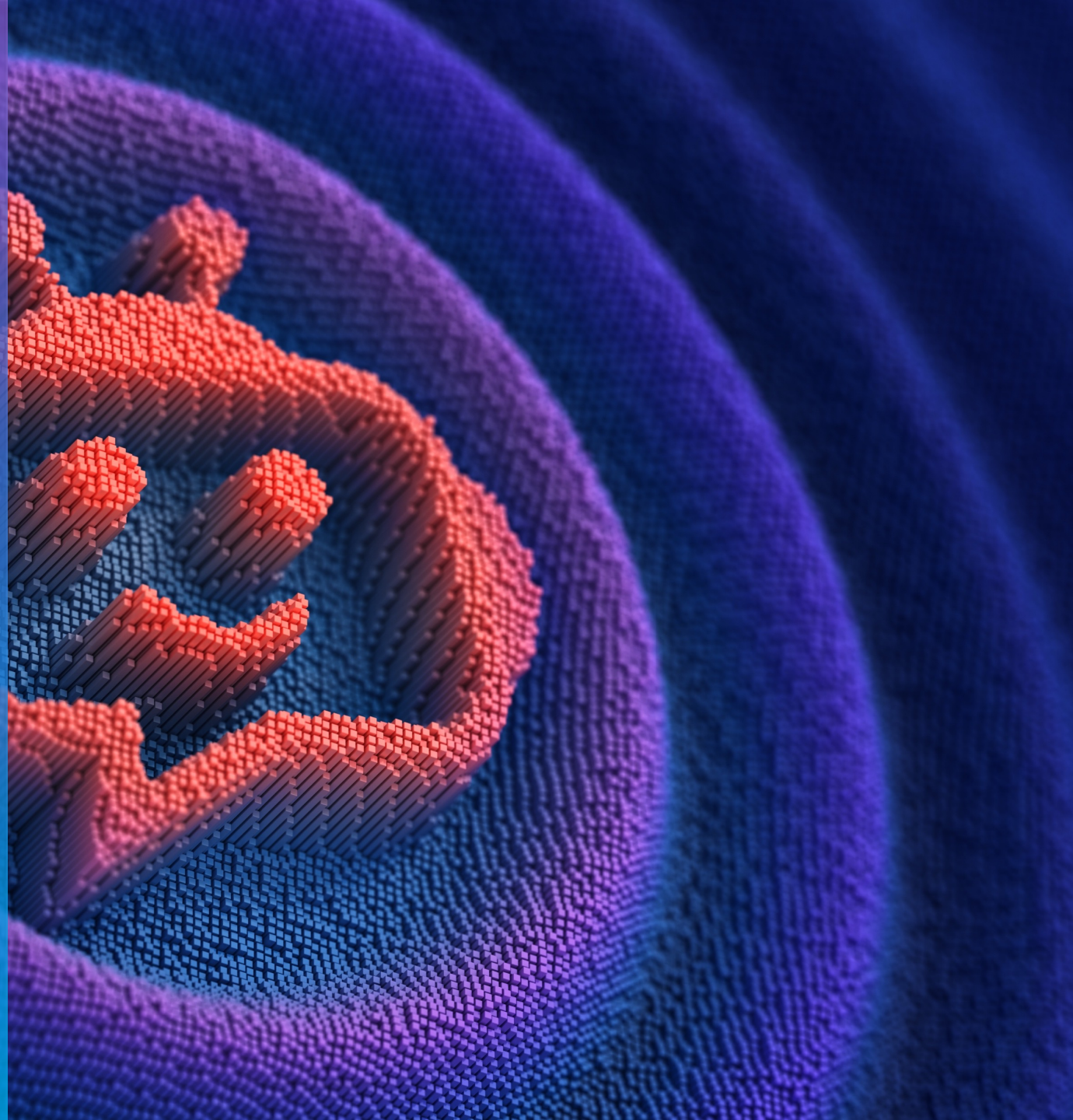


Discussion

Choose the right model for the task

Notes from discussion

Agent Harness





Demo

Interacting with an LLM directly

Public

Limitations of traditional LLMs

(1) No memory

When we continue the “conversation”, any given LLM will showcase one of its main disadvantages. It does not remember conversations!

What is 1 + 1?

The answer is 2

What did I just ask?

I have no knowledge about a previous question.

(2) Capabilities that go beyond text

There are many other tasks that LLMs often fail at, including basic math like multiplication and division.

What is 8 * 34.2 / 3.49?

Thinking…

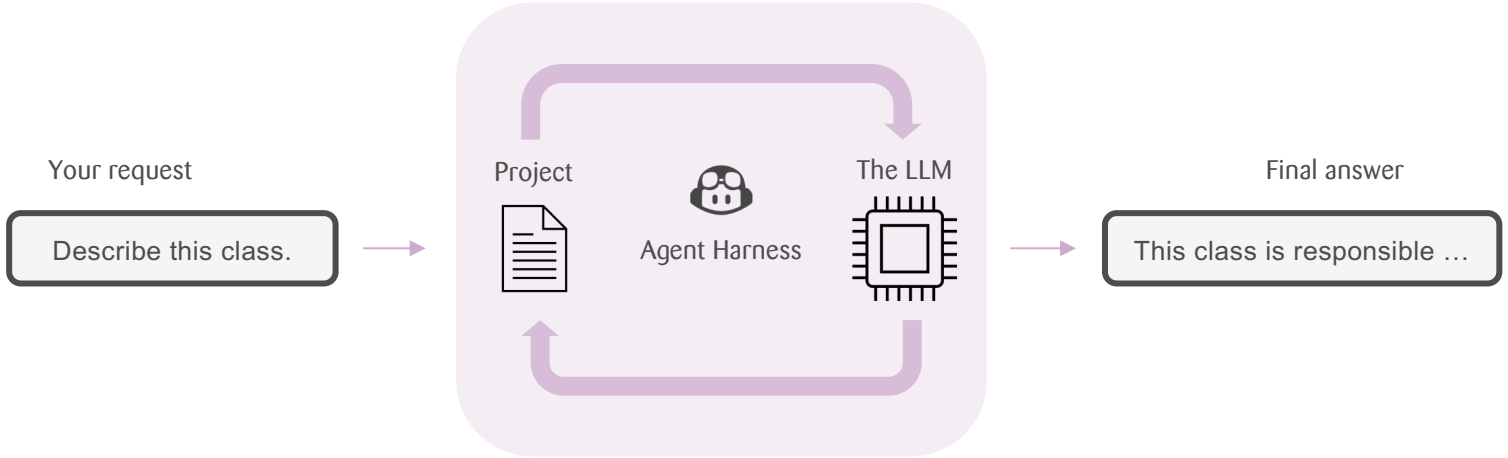


What is an agent harness?

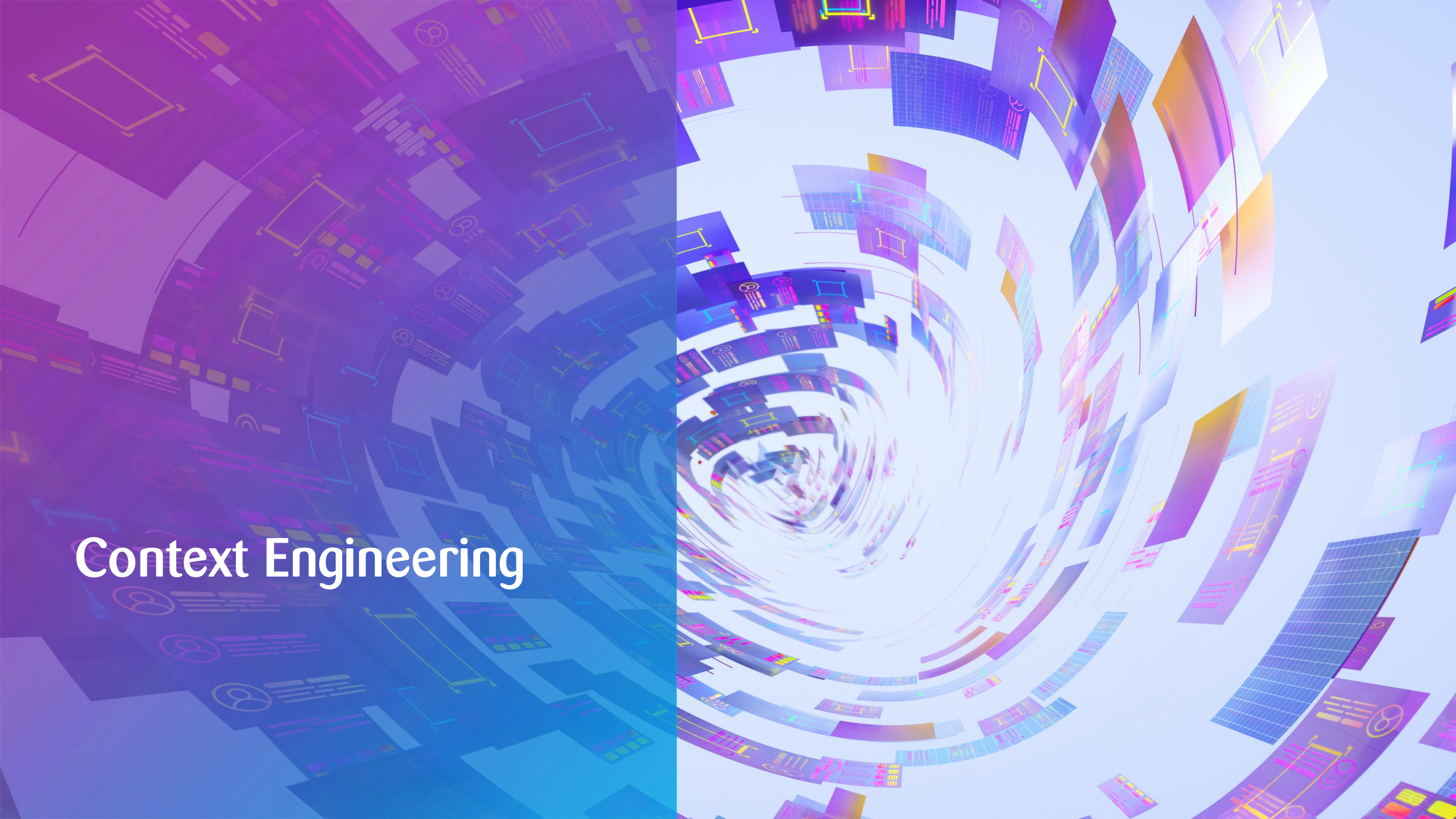
Definition

An agent harness is a program that calls an LLM in a loop and executes its decisions until the task is complete. The agent harness uses a reasoning loop that includes understanding the task, acting on it, and validating the result before continuing or finishing: Understand → Act → Validate.

Inline suggestions offer quick completions without running agents or tools, ideal for small edits. Inline chat provides a simple, file-specific chat for focused changes.

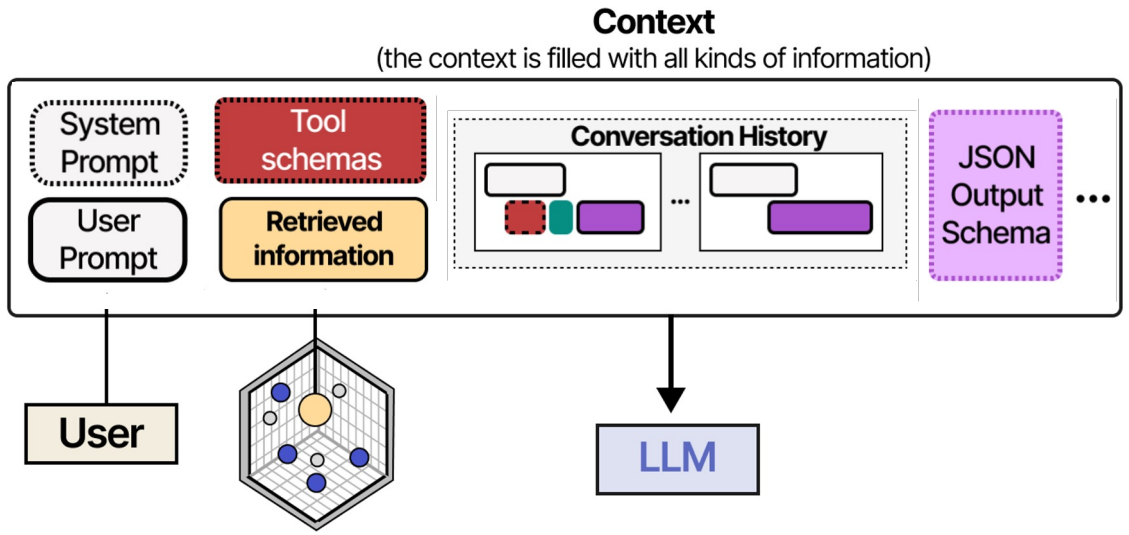



15’ Break


Context Engineering

Public

Augmenting the LLM with additional information enhances its capabilities and allows it to reason from project-specific context instead of generic knowledge



What belongs to a Context?

- **System prompt:** The core context and rules for the agent, which define how it should behave.
- **User prompt:** What you write into the chat window.
- **Tool schemas:** Description of core tools like reading and writing files built into agent. Can be extended and limited by you.
- **Retrieved information:** External information that is typically stored in a vector database and accessed through RAG-like techniques (semantic memory).
- **Conversation history:** Both the conversation between the user and assistant, but also the LLM's internal thoughts.

Source: An Illustrated Guide to AI Agents, Maarten Grootendorst, Jay Alammar (2026)



Demo

Debug an agentic session



Discussion

When should you start new sessions?
How do you manage your sessions?

Notes from discussion



How can content be preserved over sessions?

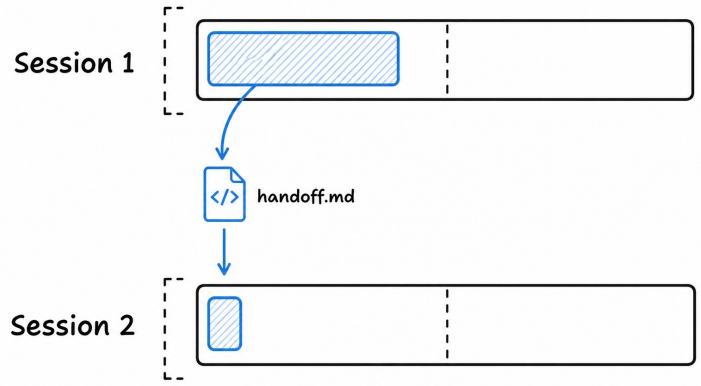
Handoffs

A handoff is a structured summary that transfers context from one session to the next.

It captures the goal, current state, key decisions, open questions, relevant artifacts, and next steps – often in a file like handoff.md.

The benefit is continuity: the next session can continue from the previous state instead of starting from scratch. This reduces context loss, avoids repeated explanations, keeps decisions consistent, and gives the next agent a clear starting point.

Link: Handoff Skill





Demo

Shaping the behavior using instructions

Public

Mechanisms that improve the likelihood of acceptable outputs from non-deterministic models

Mechanism	Guidance
Instructions	▪ Loaded at start of conversation (or path-specific when LLM tries to read files) ▪ Use as navigation aid and for a few rules that address repeated observed failures
Skills	Usually loaded by LLM on-demand, but also supports explicit invocation Minimal a SKILL.md file, often also bundled with references and scripts Can be used for pure knowledge, for procedural workflows, or for both
Prompts	Loaded when explicitly invoked by user through /my-prompt Use for repeated workflows A harness-agnostic alternative is to use explicit invokable skills
Tools	Invoked by the model or user to access external capabilities Perform actions such as API calls, code execution, or web search Return structured results for the model to use
Custom Agents	Selected by the user at the start of a conversation Can also be used as instructions for sub-agents Frequently used for "personas", but can contain any kind of context
Subagents	Used by model to offload certain work (either decided autonomously or prompted) Runs with a separate context window, and optionally with a different model Used to reduce the noise in the main conversation's context window

Public

Using progressive disclosure, we can reveal complex information gradually instead of all-at-once

- Load only the context the agent needs right now, with pointers to the rest
- Treat your AGENTS.md as index containing a sentence each, plus a pointer

One stuffed AGENTS.md

Style guide
…

Code Conventions
…

Testing Conventions
…

All of it, on every turn

AGENTS.md acting as an index

Style guide – Corporate Design and Component Styles

Code Conventions – Rules and Conventions for Coding

Testing Conventions – Rules and Conventions for Testing

Only what this task actually calls for

Progressive disclosure is a fundamental building block for providing knowledge to agents. To dig deeper into the topic, check out LLM Wikis and Google's Open Knowledge Format (OKF).



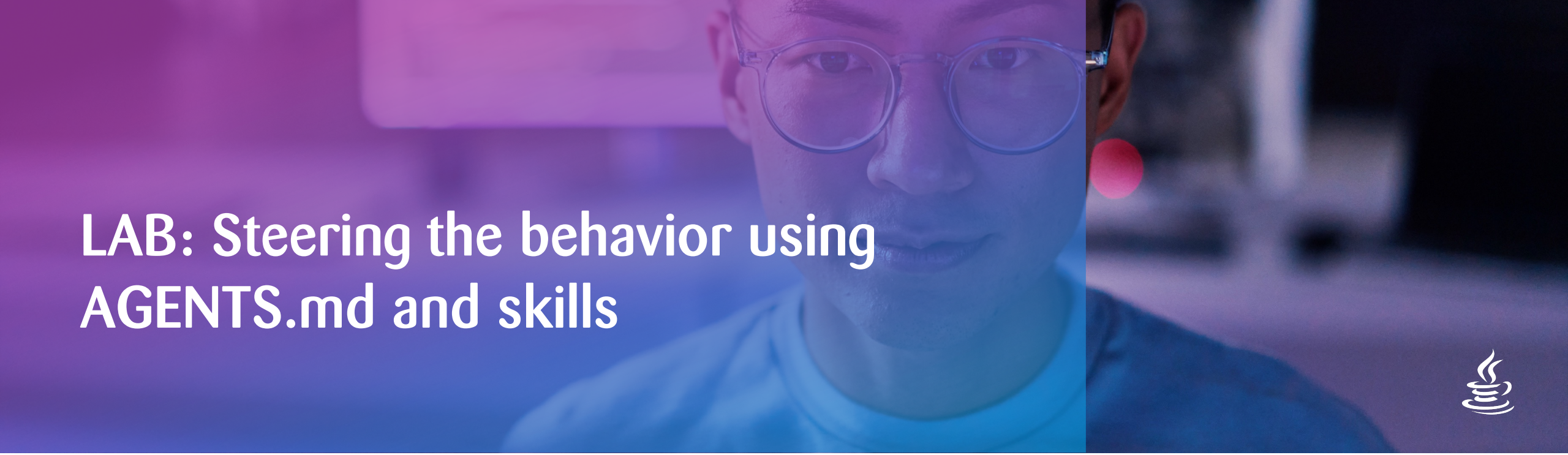
Demo

Extend capabilities and expertise with skills

Public

Mechanisms that improve the likelihood of acceptable outputs from non-deterministic models

Mechanism	Guidance
Instructions	▪ Loaded at start of conversation (or path-specific when LLM tries to read files) ▪ Use as navigation aid and for a few rules that address repeated observed failures
Skills	▪ Usually loaded by LLM on-demand, but also supports explicit invocation ▪ Minimal a SKILL.md file, often also bundled with references and scripts ▪ Can be used for pure knowledge, for procedural workflows, or for both
Prompts	Loaded when explicitly invoked by user through /my-prompt Use for repeated workflows A harness-agnostic alternative is to use explicit invokable skills
Tools	Invoked by the model or user to access external capabilities Perform actions such as API calls, code execution, or web search Return structured results for the model to use
Custom Agents	Selected by the user at the start of a conversation Can also be used as instructions for sub-agents Frequently used for "personas", but can contain any kind of context
Subagents	Used by model to offload certain work (either decided autonomously or prompted) Runs with a separate context window, and optionally with a different model Used to reduce the noise in the main conversation's context window



LAB: Steering the behavior using AGENTS.md and skills

Context

In this exercise, you will configure your project so GitHub Copilot produces more consistent and probabilistic outputs.

Your goal is to reduce ambiguity and improve output quality by changing how the agent behaves and providing guidance.

Your Task

Run **task lab/steering** and

- describe the role and behavior of agents in `AGENTS.md`,
- add the required frontmatter for the Java skill in `.agents/skills/java/SKILLS.md` and
- implement the same endpoint again:
  - GET /delays?airline_code=AA&limit=50

  → returns **200 OK** or **404 Not Found**

Tipps & Tricks

- The TODOs in the relevant files will help you get started.

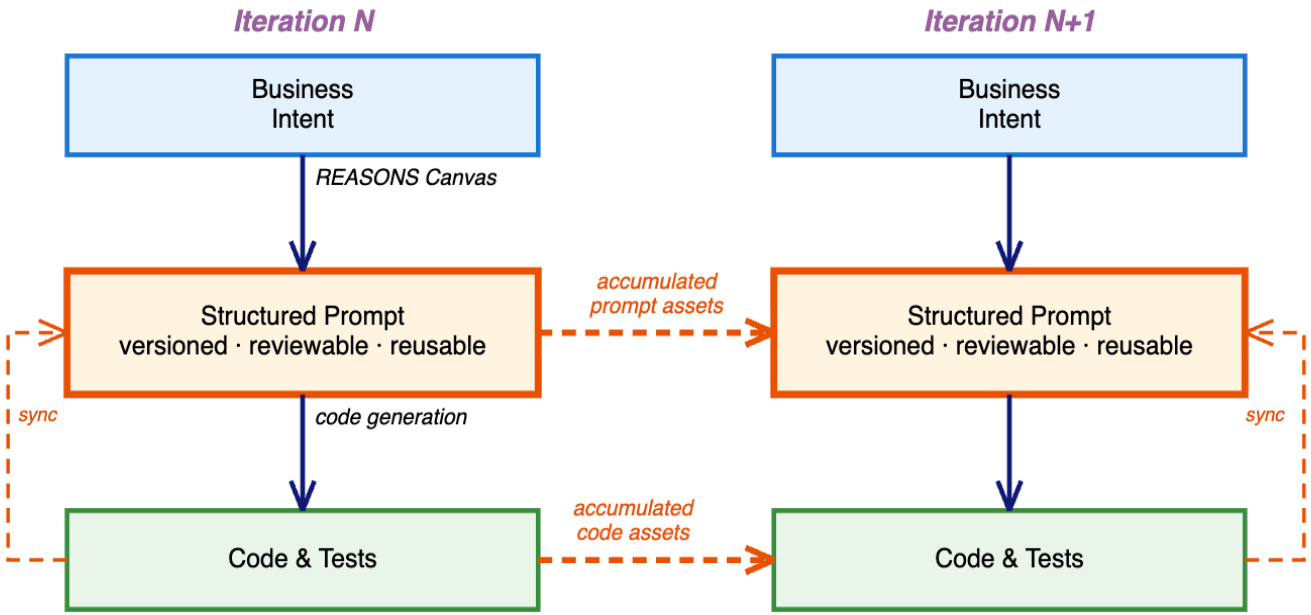

60’ Break

Reusable Prompts



Public

Instead of relying on ad hoc chats, structuring prompts turns them into assets that can be: version controlled, reviewed, reused, and improved over time



Source: https://martinfowler.com/articles/structured-prompt-driven



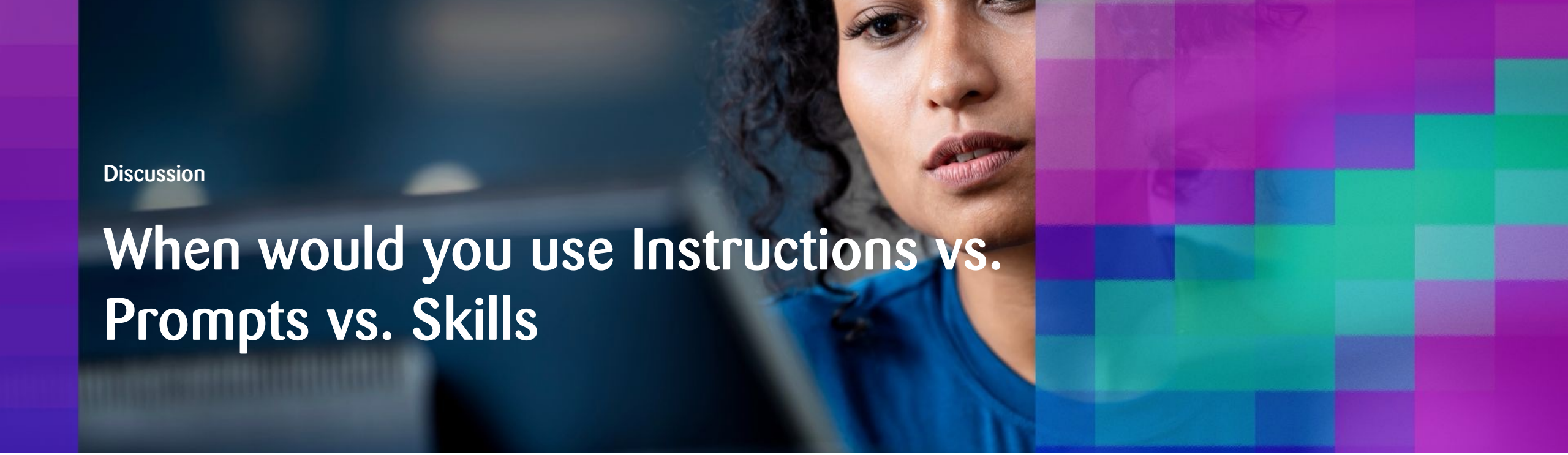
Demo

Create a reusable & shareable prompt

Public

Mechanisms that improve the likelihood of acceptable outputs from non-deterministic models

Mechanism	Guidance
Instructions	▪ Loaded at start of conversation (or path-specific when LLM tries to read files) ▪ Use as navigation aid and for a few rules that address repeated observed failures
Skills	▪ Usually loaded by LLM on-demand, but also supports explicit invocation ▪ Minimal a SKILL.md file, often also bundled with references and scripts ▪ Can be used for pure knowledge, for procedural workflows, or for both
Prompts	▪ Loaded when explicitly invoked by user through /my-prompt ▪ Use for repeated workflows ▪ A harness-agnostic alternative is to use explicit invokable skills
Tools	Invoked by the model or user to access external capabilities Perform actions such as API calls, code execution, or web search Return structured results for the model to use
Custom Agents	Selected by the user at the start of a conversation Can also be used as instructions for sub-agents Frequently used for "personas", but can contain any kind of context
Subagents	Used by model to offload certain work (either decided autonomously or prompted) Runs with a separate context window, and optionally with a different model Used to reduce the noise in the main conversation's context window



Discussion

When would you use Instructions vs. Prompts vs. Skills

Instructions

- Always-on behavioral rules that define how the agent should work.
- Use instructions for stable guidance such as coding standards, communication style, security rules, constraints or preferred workflows.

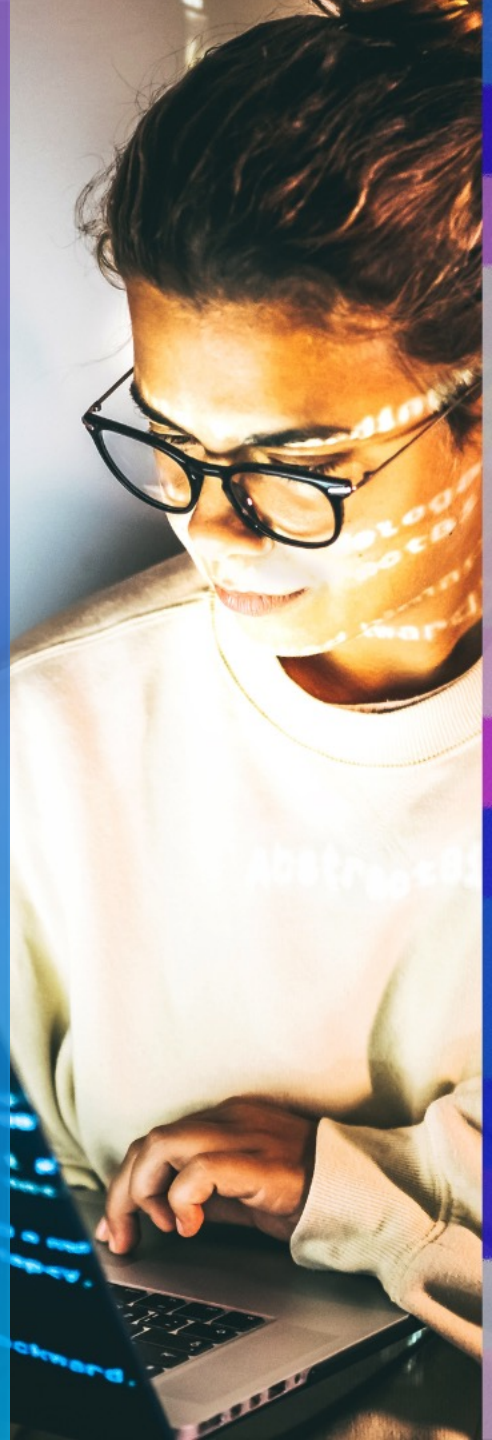
Prompts

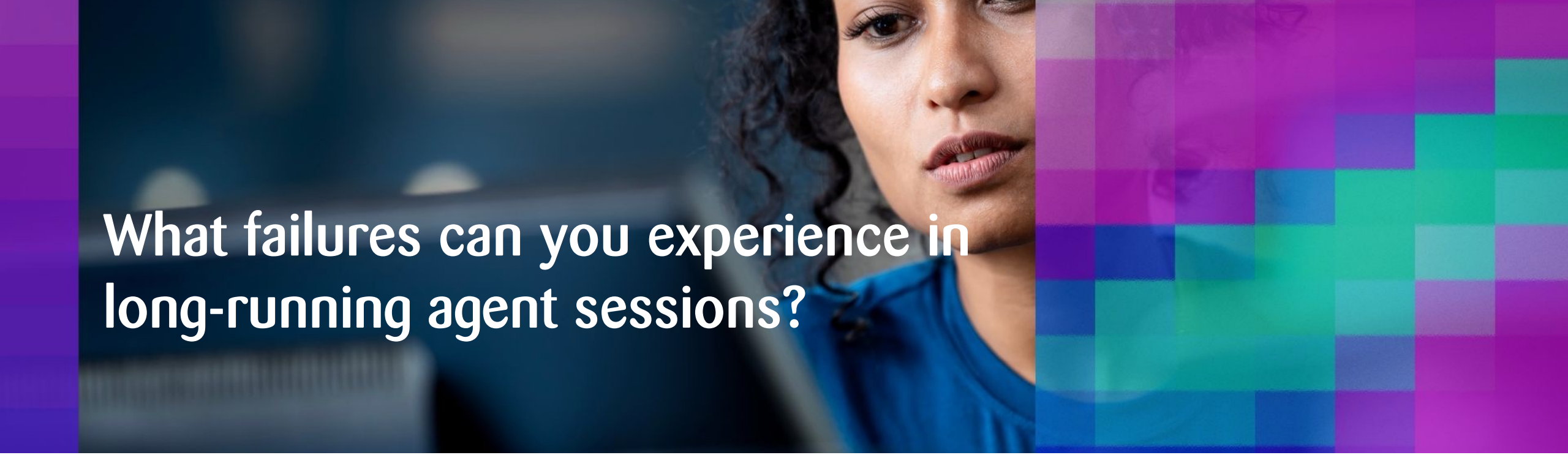
- Task-specific input that defines what needs to be done.
- By executing a prompt you push Copilot to do a specific task.
- Use structured prompts to package repeatable tasks, so they can be reused, shared with others, and improved over time.

Skills

- Reusable capabilities the agent can pull in when needed.
- Use skills for specialized workflows, tools, or domain knowledge that should stay available without being loaded every time.

Spec-Driven Development (SDD)





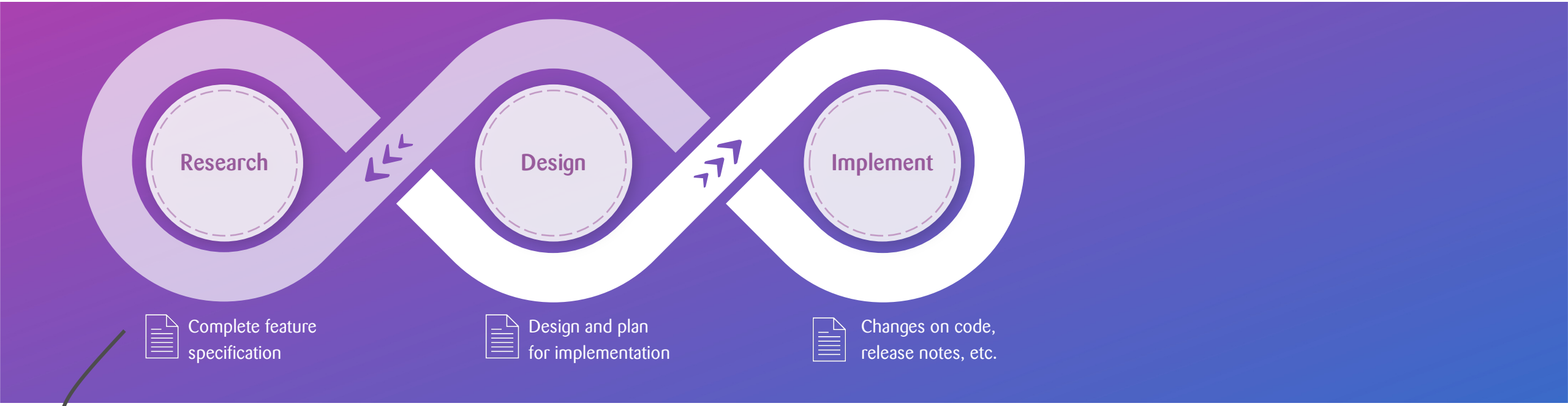
What failures can you experience in long-running agent sessions?

Common AI failures:

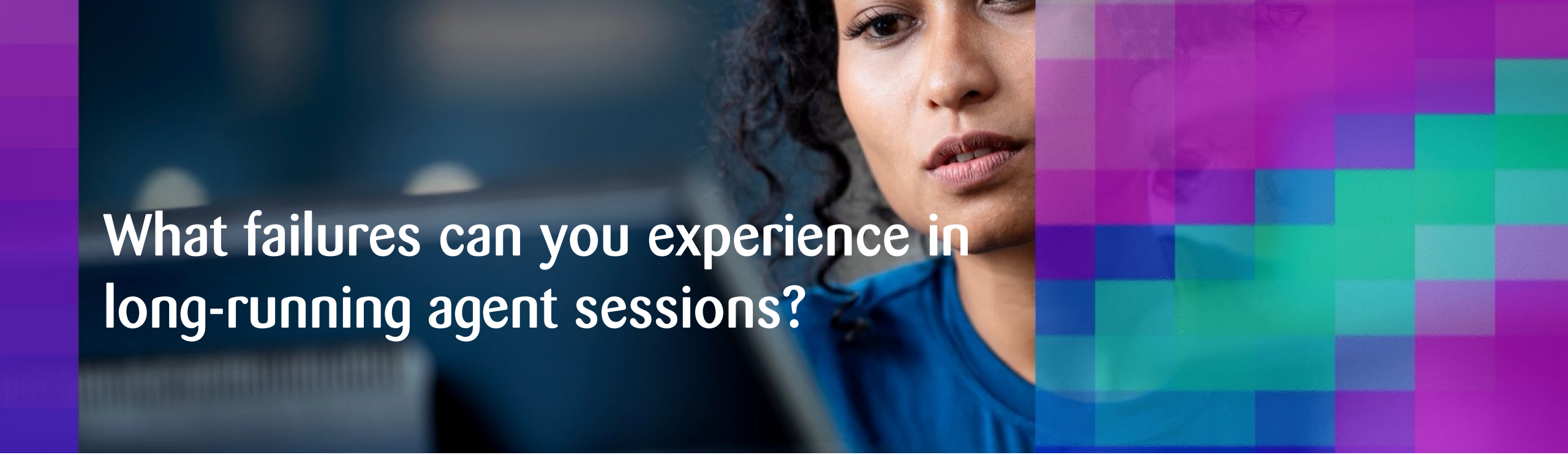
Failure	What goes wrong
Hallucination	Output sound right but is actually wrong
Scope creep	Agent drifts beyond the intended task
Wrong assumptions	Agent fills gaps with incorrect guesses
Unverified output	Bad results are accepted as is
Context overflow	Too much information degrades quality

Public

Splitting work phases enables early alignment, reducing surprise, and closing the gap between what you asked for and what you get



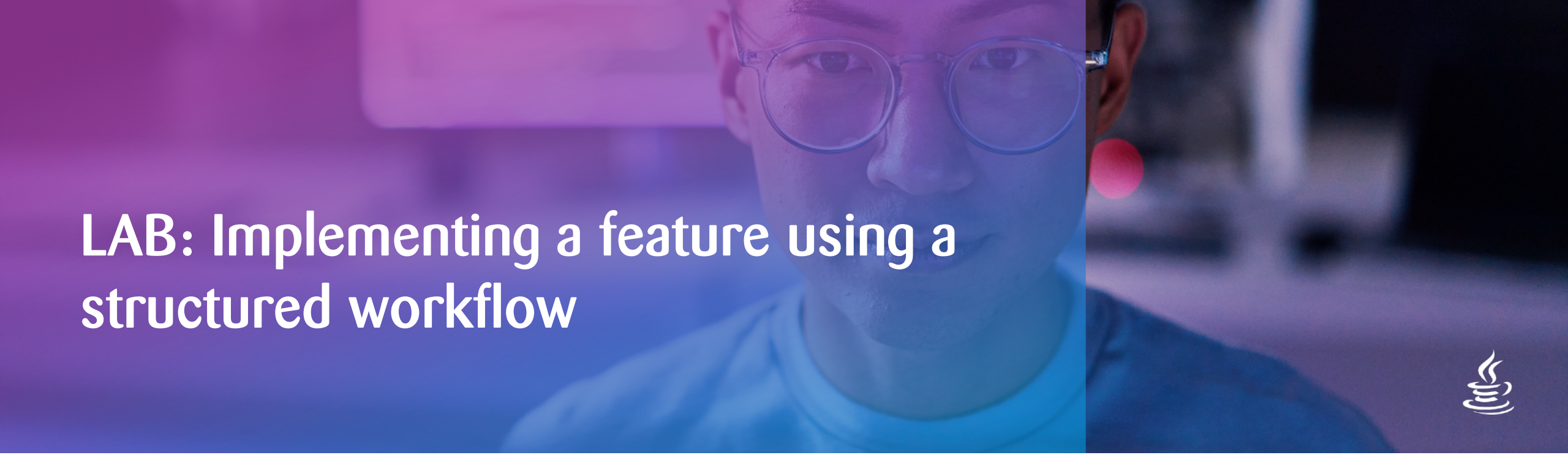
Keeping humans in the loop: AI can research, plan, and implement based on the spec, while humans remain responsible for setting direction, defining intent, making key trade-offs, reviewing outcomes, and approving the final result. The spec becomes the shared reference point between human judgment and AI execution.



What failures can you experience in long-running agent sessions?

Common AI failures:

Failure	What goes wrong	How a structured workflow helps
Hallucination	Output sound right but is actually wrong	Verification step catches error before they propagate
Scope creep	Agent drifts beyond the intended task	Research step locks down scope before work begins
Wrong assumptions	Agent fills gaps with incorrect guesses	Context is explicitly provided during refinement
Unverified output	Bad results are accepted as is	Checkpoints and feedback loops catch errors easily
Context overflow	Too much information degrades quality	Structured steps keep each phase focused and concise



LAB: Implementing a feature using a structured workflow

Context

We will apply the full structured workflow that uses the phases:

- research,
- design and
- implement

Your Task

Run **task lab/structured-workflow** and

- write the missing frontmatter for the skills
- Implement the same endpoint again using the research – design - implement workflow:
  - GET /delays/?airline_code=AA&limit=50

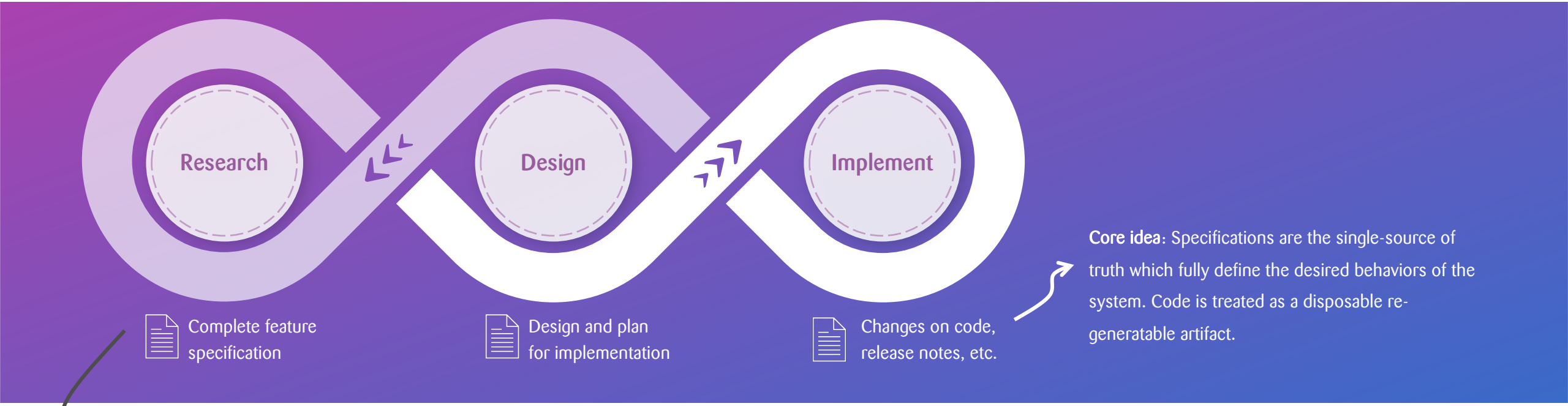
Tipps & Tricks

- The TODOs in the relevant files will help you get started.


15’ Break

Public

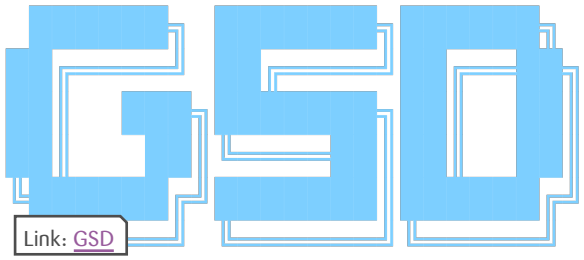
Spec-Driven Development (SDD) builds on structured workflows and derives implementation from explicit specifications



Keeping humans in the loop: AI can research, plan, and implement based on the spec, while humans remain responsible for setting direction, defining intent, making key trade-offs, reviewing outcomes, and approving the final result. The spec becomes the shared reference point between human judgment and AI execution.

Public

Many opinionated Spec-Driven Development frameworks, one shared principle: Specs before code



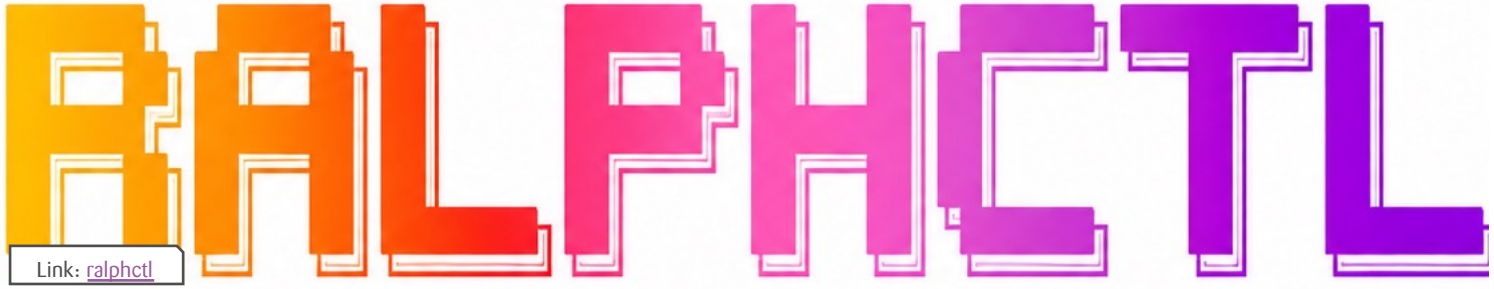
Link: GSD



Link: OPENSPEC



Link: Spec Kit



Link: ralphctl

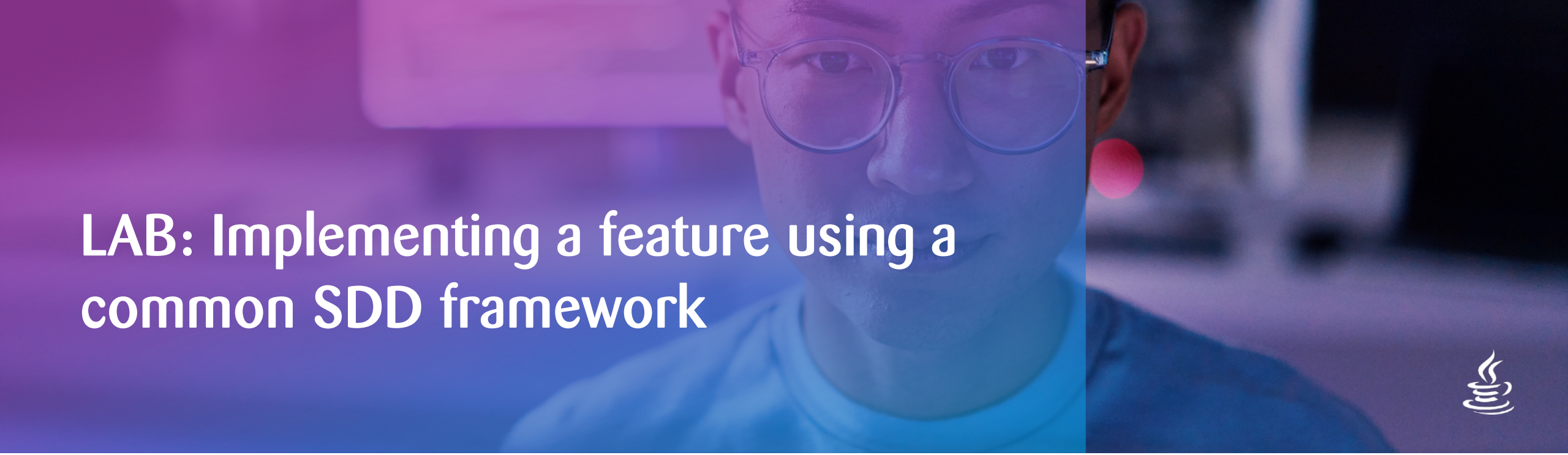


Link: Kiro



Demo

Show OpenSpecs workflow



LAB: Implementing a feature using a common SDD framework

Context

Instead of using our own research → plan → implement workflow make yourself familiar with one publicly available SDD framework and implement the same feature using their workflow. Popular frameworks:

- GSD
- OPENSPEC
- Spec Kit
- …

Your Task

Run **task lab/sdd** and

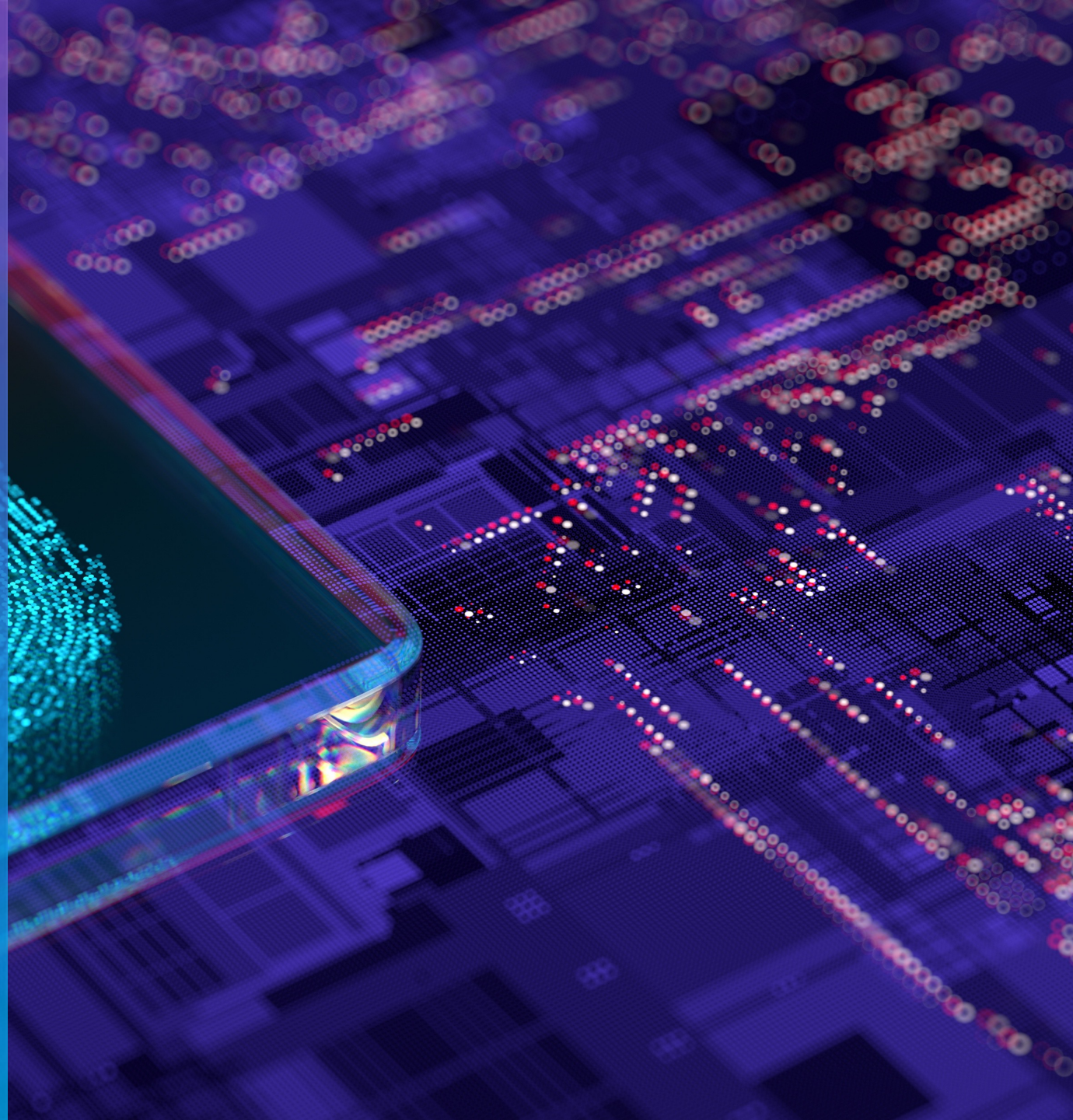
- make yourself familiar with a SDD framework of your choice to
- implement the same endpoint again using their workflow:
  - GET /delays/?airline_code=AA&limit=50

Tipps & Tricks

Files that are created depend on the framework. You can find links to different frameworks on the previous slide.


15’ Break

Permissions



Public

Three layers protect your machine: rules and modes decide whether an action runs, the sandbox limits what it can reach



Permission rules

Restrict actions before the agent even runs a command

Your own list of what the agent may do. Each tool or command is set to allow, ask or deny, and Claude Code checks the list before every tool call.

Example: allow npm test, ask before git push, deny reading .env files.



Permission modes

How autonomous you let the agent use commands

One setting for the whole session that decides how often the agent stops for your approval, from asking before most actions to asking for nothing.

Example: autopilot mode (“YOLO”) that skips every approval an runs from start to finish



Sandboxing Day 2

Reducing blast-radius of running commands

The operating system fences in shell commands and the processes they start, so they only reach folders and domains you approved.

Example: run your agent in a container that blocks network access except to approved hosts.



Demo

Permissions & permission modes

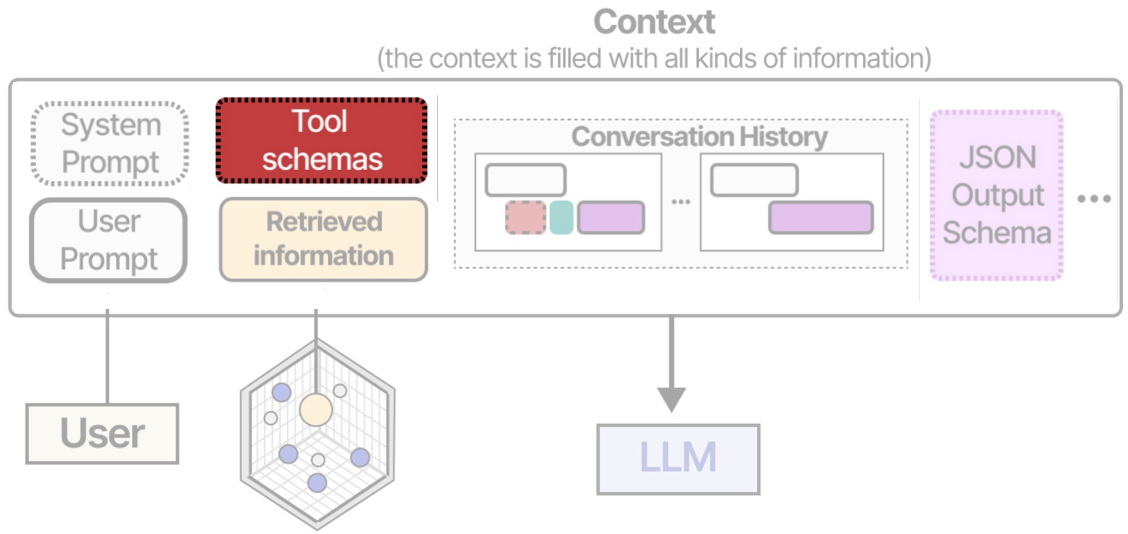


How can an agent interact with the environment?

Tools

Enable agents the power to interact with their environment: read/edit files, run terminal commands, and call external services. Without tools, a model can only talk; with tools, an agent can change your repo and verify results.

Tools can come from VS Code itself or via MCP servers (Model Context Protocol) to safely expose external systems and APIs as agent-invokable capabilities.





Demo

Ad-hoc tool selection and tools frontmatter

Public

Mechanisms that improve the likelihood of acceptable outputs from non-deterministic models

Mechanism	Guidance
Instructions	▪ Loaded at start of conversation (or path-specific when LLM tries to read files) ▪ Use as navigation aid and for a few rules that address repeated observed failures
Skills	▪ Usually loaded by LLM on-demand, but also supports explicit invocation ▪ Minimal a SKILL.md file, often also bundled with references and scripts ▪ Can be used for pure knowledge, for procedural workflows, or for both
Prompts	▪ Loaded when explicitly invoked by user through /my-prompt ▪ Use for repeated workflows ▪ A harness-agnostic alternative is to use explicit invokable skills
Tools	▪ Invoked by the model or user to access external capabilities ▪ Perform actions such as API calls, code execution, or web search ▪ Return structured results for the model to use
Custom Agents	Selected by the user at the start of a conversation Can also be used as instructions for sub-agents Frequently used for "personas", but can contain any kind of context
Subagents	Used by model to offload certain work (either decided autonomously or prompted) Runs with a separate context window, and optionally with a different model Used to reduce the noise in the main conversation's context window

Automated Feedback Loops





What is the difference between verification vs. validation?

Verification

Answers the question “Are we building it right?”

Verification checks whether the system has been implemented according to the agreed requirements, design, and quality standards. It focuses on technical correctness: does the code behave as expected, are tests passing, and is the implementation consistent with the specification? AI can be leveraged here to generate test cases, review code, detect inconsistencies, suggest edge cases, and speed up documentation.

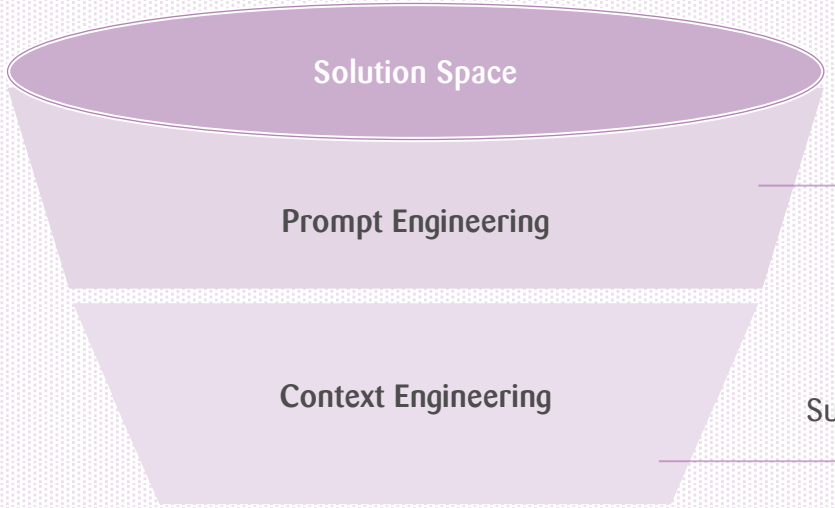
Validation

Answers the question “Are we building the right thing?”

Validation checks whether the system actually solves the real user or business problem in its intended context. It focuses on usefulness, fit, and real-world impact: does the solution meet user needs, support the process, and create value? AI can support validation by summarizing user feedback, simulating scenarios, and highlighting possible risks, but it cannot replace real users, domain experts, ethical judgment, or business accountability.

Public

The verification harness ensures the quality of LLM output by using deterministic tools



Structured Prompts, Zero-, One- and Few-shot Prompting

Instruction Files, Custom Agents, Subagents, Skills, Handoff, Tools, Evals

Increasing the likelihood of acceptable outputs from non-deterministic models



Deterministic scripts, Agent Hooks, Git Hooks, CI/CD

Ensuring quality of LLM output using deterministic tools



Demo

Pre-tool-usage hook

Public

Ensuring quality of LLM output using deterministic tools

Mechanism	Guidance
Hooks	▪ Run code at key lifecycle events ▪ Automate workflows, enforce policies or validate actions ▪ Deterministic, unlike instructions or prompts



LAB: Enforce quality through an agent hook

Context

As we know the usage of skills and instructions is non-deterministic. For our verification harness we want to rely on deterministic scripts using hooks.

Your Task

Run **task lab/hook** and

- update the hook in `verification-harness.json` to make sure tests are run whenever the agent stops.

Open a new chat and verify if the tests are being executed.

Tipps & Tricks

- The TODOs in the relevant files will help you get started.

Advanced Agentic Techniques





Demo

Create a custom agent

Public

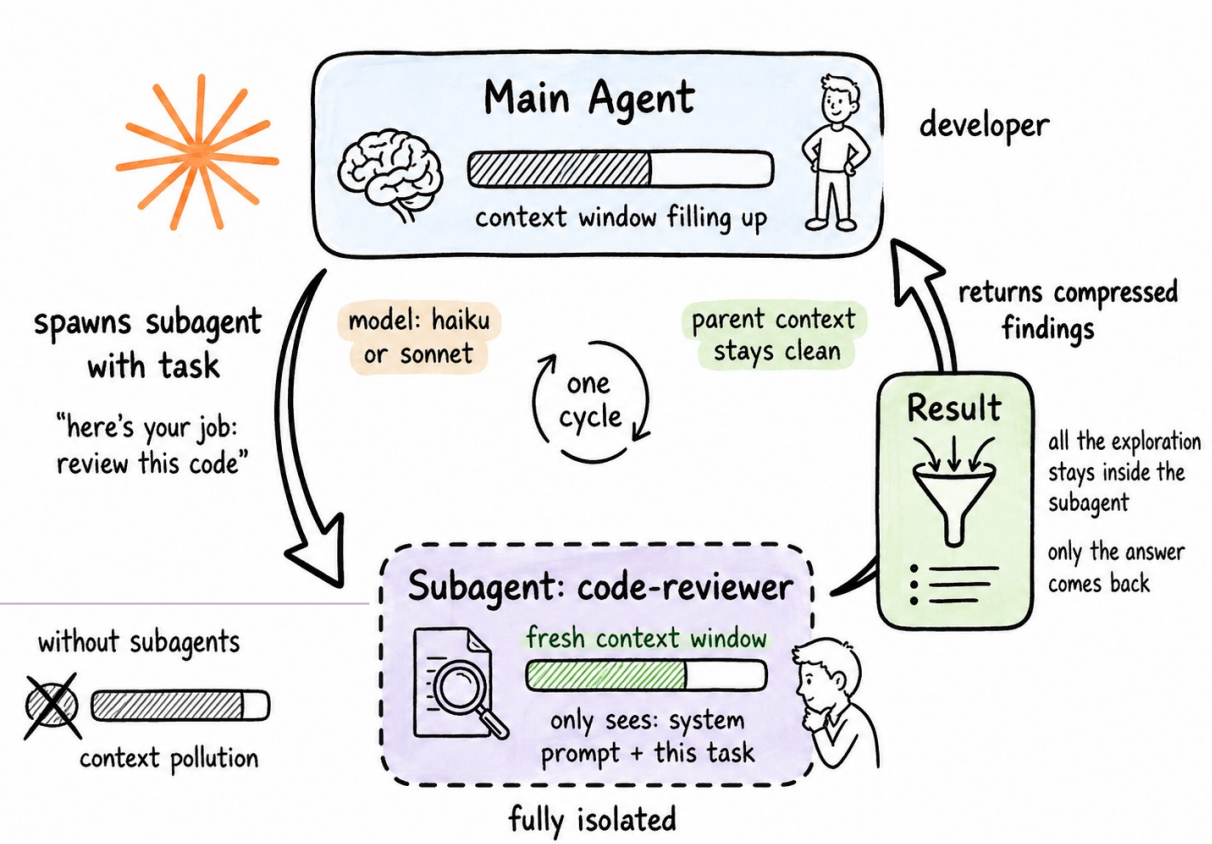
Mechanisms that improve the likelihood of acceptable outputs from non-deterministic models

Mechanism	Guidance
Instructions	▪ Loaded at start of conversation (or path-specific when LLM tries to read files) ▪ Use as navigation aid and for a few rules that address repeated observed failures
Skills	▪ Usually loaded by LLM on-demand, but also supports explicit invocation ▪ Minimal a SKILL.md file, often also bundled with references and scripts ▪ Can be used for pure knowledge, for procedural workflows, or for both
Prompts	▪ Loaded when explicitly invoked by user through /my-prompt ▪ Use for repeated workflows ▪ A harness-agnostic alternative is to use explicit invokable skills
Tools	▪ Invoked by the model or user to access external capabilities ▪ Perform actions such as API calls, code execution, or web search ▪ Return structured results for the model to use
Custom Agents	▪ Selected by the user at the start of a conversation ▪ Can also be used as instructions for sub-agents ▪ Frequently used for "personas", but can contain any kind of context
Subagents	▪ Used by model to offload certain work (either decided autonomously or prompted) ▪ Runs with a separate context window, and optionally with a different model ▪ Used to reduce the noise in the main conversation's context window

Public

Subagents handle isolated subtasks in separate context windows, keeping the main context clean and enabling less-biased judgment across different models

The **generator–validator pattern** uses one agent, often with one model, to produce candidate solutions, and a separate validator agent, often using a different model, to verify them. This improves reliability by separating creation from evaluation.



Source: Akshay

Public

Mechanisms that improve the likelihood of acceptable outputs from non-deterministic models

Mechanism	Guidance
Instructions	▪ Loaded at start of conversation (or path-specific when LLM tries to read files) ▪ Use as navigation aid and for a few rules that address repeated observed failures
Skills	▪ Usually loaded by LLM on-demand, but also supports explicit invocation ▪ Minimal a SKILL.md file, often also bundled with references and scripts ▪ Can be used for pure knowledge, for procedural workflows, or for both
Prompts	▪ Loaded when explicitly invoked by user through /my-prompt ▪ Use for repeated workflows ▪ A harness-agnostic alternative is to use explicit invokable skills
Tools	▪ Invoked by the model or user to access external capabilities ▪ Perform actions such as API calls, code execution, or web search ▪ Return structured results for the model to use
Custom Agents	▪ Selected by the user at the start of a conversation ▪ Can also be used as instructions for sub-agents ▪ Frequently used for "personas", but can contain any kind of context
Subagents	▪ Used by model to offload certain work (either decided autonomously or prompted) ▪ Runs with a separate context window, and optionally with a different model ▪ Used to reduce the noise in the main conversation's context window



Demo

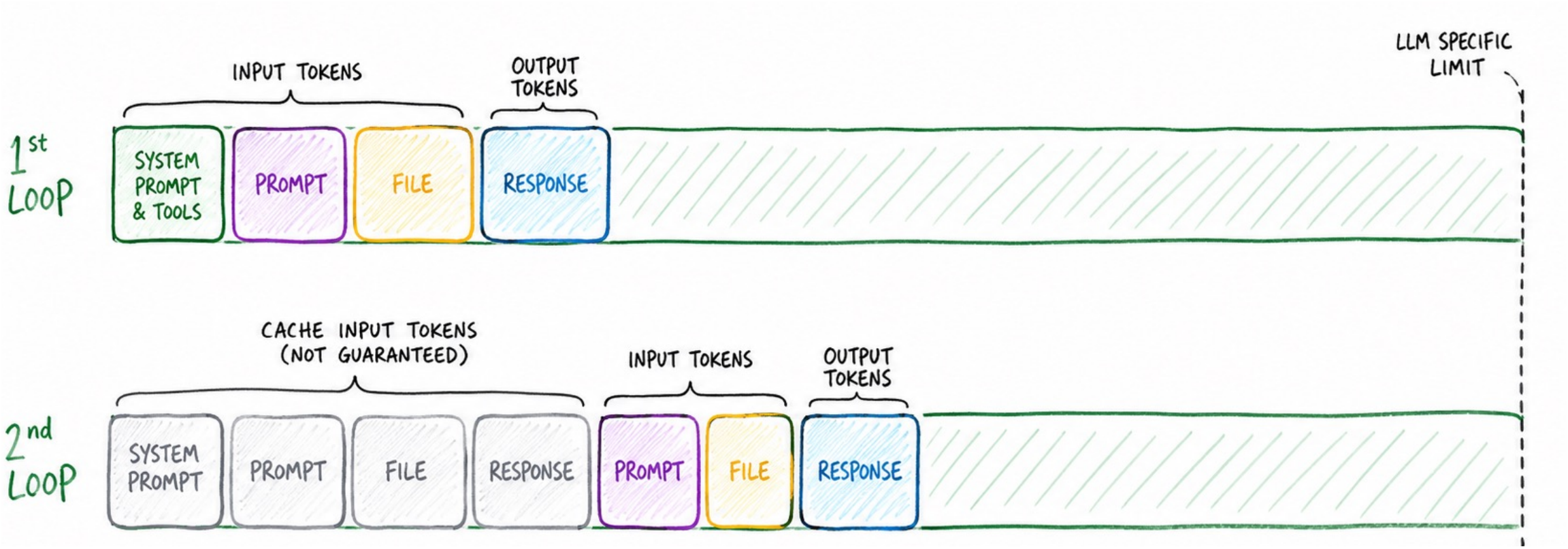
Spawn a subagent



Token Economics

Public

To own token allowance effectively, we have to understand the different types of tokens: input tokens, cached tokens, and output tokens



Input tokens - what you send to the model **Cached tokens** - context the model uses from previous interactions **Output tokens** - what the model generates in response.

Source: https://www.youtube.com/watch?v=0IOcPqubpMc



How can I get the most out of my Token allowance?

Provide more clear specification of what to do

If you provide a vague prompt, the model has to do a lot of work to figure out the necessary context and the target. This means it will perform a lot of file searches, read files that may be unrelated to the tasks, and may have to backtrack a lot. All of this costs a lot of tokens.

If instead you provide a clear task with hints on what input to use and how to verify the result, the agent will have to process much less data.

Use Research-Plan-Implement

By following a structured workflow, you significantly increase the chance that the agent will get a task right on the first attempt, preventing costly rework.

Use the right model for the right job

Not every task needs the most capable model. Learn when to use GPT-5 mini or GPT-5.2 instead of GPT-5.5 and when to use Claude Sonnet over Claude Opus.



Instruct the model to generate less output tokens

Output tokens are much more expensive than input tokens. By instructing the model to be more concise, for example, using a caveman skill, you can significantly reduce your token spend.

Use code for deterministic tasks, and only use LLMs where their strengths matter

Take advantage of deterministic tools like scripts, linters, tests to provide guardrails and offload tasks from the model.

And if you realize that you are using LLMs repeatedly for the same task, ask the LLM to build a script instead of doing the work directly and then tell it, via instructions, to use that script in the future.

Consider when to start a new conversation

Long conversations can benefit from caching, but they also keep growing: each new turn may require more previous context to be re-processed. For related tasks, staying in one conversation can make sense. For independent tasks, splitting them into separate conversations may reduce re-processed tokens.

As a rule of thumb: start a new conversation when tasks are mostly independent, or when you return after a longer break and the cache may have expired.

Avoid switching models mid-conversation

Changing the model can reset or reduce the benefit of cached tokens. If the task does not require a different model, stay with the same one to avoid paying again for context that was already processed.

Toggle tools intentionally

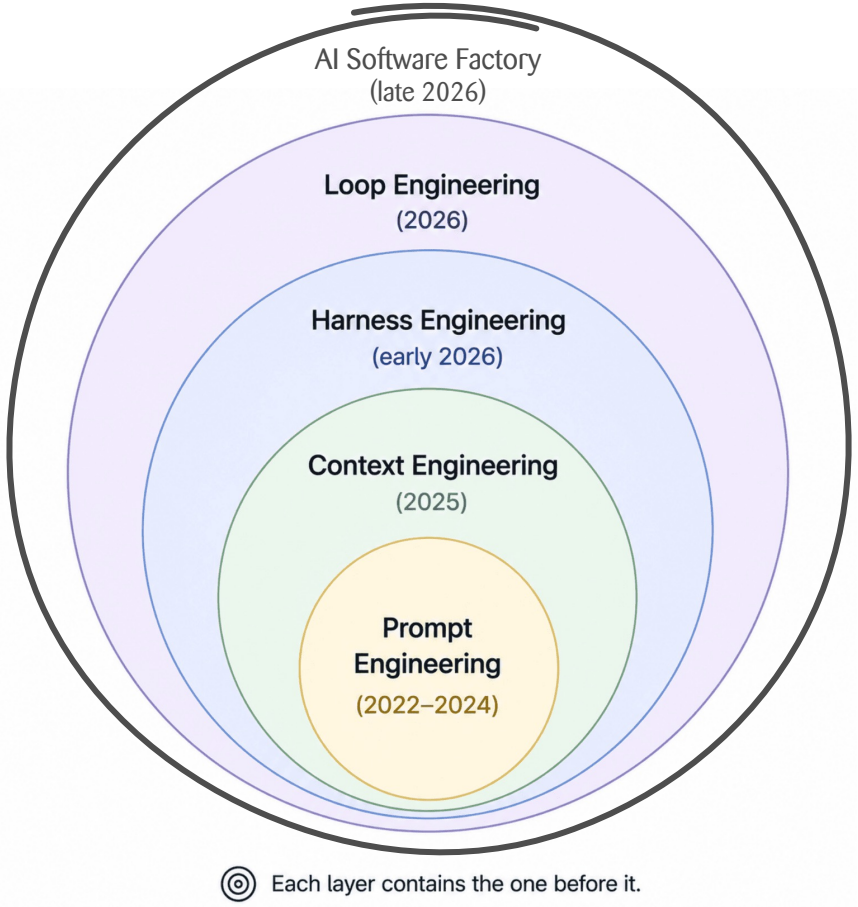
Enabled tools add context to the conversation and can increase token usage. Keep only the tools active that are needed for the current task, and turn off anything unrelated.

Progressive disclosure

Don't front-load everything into the context. Start with lightweight pointers – file names, headings, skill descriptions – and let the agent pull the full detail only when a task actually needs it. Context stays small, and the model keeps its attention on what matters.

Public

What "using AI" in software engineering means changes annually - teams need to find what fits them and remain willing to change it as they learn more



Source: https://machinelearningmastery.com/an-introduction-to-loop-engineering/

Public

It takes some time to develop a good intuition. Use the formats and communities within Zühlke to exchange experiences, share knowledge, and learn from each other



”How I use AI” Sessions

A peer-to-peer community series at Zühlke. Each session, a colleague walks through how they use AI in their day-to-day work, with concrete tools and live demos rather than slideware. Register for upcoming sessions in Workday.



Courses & Trainings

The AI Education Catalogue offers curated AI learning opportunities across formats, from live classrooms and online sessions to self-paced courses, videos, LinkedIn Learning, and mandatory training, tailored to audiences such as everyone, management, change managers, software engineering, cybersecurity, UX, sales, and AI product builders.



Community of Practice

The Cybernetic Delivery Community of Practice at Zühlke Switzerland helps teams adopt AI in software delivery through practical use cases, validated learnings, and reusable patterns across the SDLC. Join them via Link.


zühlke
empowering ideas