How AI will Transform Software Engineering Landscape?

Vibe Coding with AI

AI Standards & Best Practices for Software Engineering

By Varun Nagpal

Introduction to AI Standards and Best Practices - Part 1

- Industry Sneak Peak - AI Adoption Journey of Amazon, Microsoft, Meta/Facebook, etc.
- How AI will transform the Software Engineering landscape? (4 Pillars of Transformation)
- Areas of Impact and New Ways to Working in Software Engineering with AI
- What is Vibe Coding with AI?
- New AI Skills required to adapt to new ways of working in Software Engineering
- Prompt Engineering Best Practices for Effective Code Generation
- Prompt Components and Structure. Real world Prompt Examples.
- Introduction to Debugging Skills to work with AI
- Best Practices of Debugging Skills with AI
- Introduction to AI Hallucinations (What?, Why?, How?)
- How to validate AI's work and Checking for AI Hallucinations?
- Best Practices to reduce AI Hallucinations and Improve Code Quality
- Guard Rails and Points of Cautions when working with AI

Adoption of New AI Tools and Protocols - Part 2

- Adoption of New Tools and Protocols in AI Landscape
- What are AI Agents?
- AI Agents Architecture
- Practical Use of AI Agents (VSCode Agent Mode, etc.) for Software Engineering
- What is MCP?
- MCP Client Server Architecture
- Practical Use of MCP (Git Flow, Connect DB, Search Web, Fetch Latest Docs, etc.) for Software Engineering
- New Inputs Modes to AI Tools
- Practical Use of New Inputs Modes (Screenshots, Live Screen share, Voice Chat, etc.) for Software Engineering
- Guard Rails and Points of Cautions when working with New AI Tools and Protocols

Practical Demo and Real World Examples - Part 3

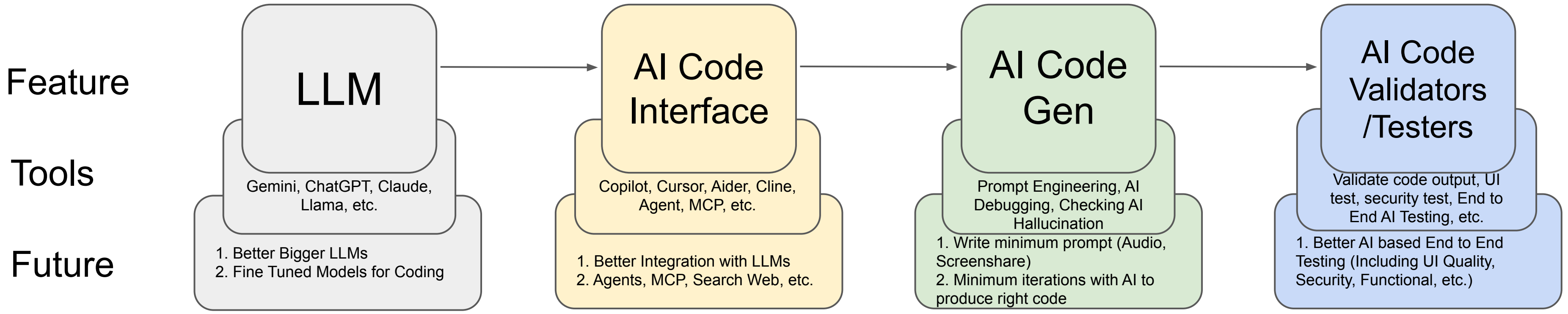
- Demo - Documentation Generation with AI (Any Existing Repository)
- Demo - Code Generation with AI (Java Microservice)
- Demo - Unit Test Generation with AI
- Demo - Swagger Specs Generation with AI
- Demo - Fix a Defect with AI
- Demo - Libraries/Frameworks version upgrade/migration with AI (Java 8 -> Java 21)
- Demo - Pull Request (PR) Generation, Review and Approval with AI

Industry Sneak Peak - AI Adoption Journey

- **Microsoft**
  - **Today 30% of code is written by AI**
    - Source: https://techcrunch.com/2025/04/29/microsoft-ceo-says-up-to-30-of-the-companys-code-was-written-by-ai/
- **Meta (Facebook)**
  - **50% by 2026**
    - Source: https://mashable.com/article/llamacon-mark-zuckerberg-ai-writes-meta-code
  - **Most of Code will be written by AI in next 12-18 months**
    - Source: https://www.engadget.com/ai/mark-zuckerberg-predicts-ai-will-write-most-of-metas-code-within-12-to-18-months-213851646.html
- **Amazon**
  - **4,500 developer-years of work done by AI**
    - Source: https://finance.yahoo.com/news/amazon-ceo-andy-jassy-says-213018283.html

How AI will transform the Software Engineering Landscape?

4 Pillars of Transformation


Feature
Tools
Future
LLM
Gemini, ChatGPT, Claude, Llama, etc.
1. Better Bigger LLMs
2. Fine Tuned Models for Coding
AI Code Interface
Copilot, Cursor, Aider, Cline, Agent, MCP, etc.
1. Better Integration with LLMs
2. Agents, MCP, Search Web, etc.
AI Code Gen
Prompt Engineering, AI Debugging, Checking AI Hallucination
1. Write minimum prompt (Audio, Screenshare)
2. Minimum iterations with AI to produce right code
AI Code Validators /Testers
Validate code output, UI test, security test, End to End AI Testing, etc.
1. Better AI based End to End Testing (Including UI Quality, Security, Functional, etc.)

Areas of Impact & New Ways to Working in Software Engineering with AI

Major Area	Old Manual Ways of Doing Things	Time	New Ways of Doing Things with AI	Time	Benefits, Improvements
Documentation (Documenting a feature, service, etc.)	1. Understand the feature, review the code. 2. Write/Review the documentation	Days	1. Write Accurate Prompt, with link to code or existing documentation & Generate Documentation with AI 2. Review/Validate AI work and Check for AI Hallucination	Hours	1. Superior Documentation Quality 2. Huge Productivity boost and Save Time
Code Generation (Implementing a feature)	1. Understand the requirements & existing code. 2. Write/Test/Review the code	Days	1. Write Accurate Prompt & Generate Code with AI 2. Review/Validate AI work and Check for AI Hallucination 3. Perform Debugging with AI, if AI generated incorrect code	Hours	1. Huge Productivity boost and Save Time 2. Superior Code Quality and Less Defects 3. Faster Time to Market
Unit Tests Generation (Writing a Unit Test)	1. Understand the requirements & existing code. 2. Write/Test/Review the code	Days	1. Write Accurate Prompt & Generate unit test with AI 2. Review/Validate AI work and Check for AI Hallucination	Hours	1. Close to 100% Code Coverage 2. Superior Unit Test Code Quality 3. Faster Time to Market
Swagger Specs Generation (Create Postman Collection)	1. Understand the requirements & existing code. 2. Write/Review the Swaggers Specs	Days	1. Write Accurate Prompt & Generate swagger specs with AI 2. Review/Validate AI work and Check for AI Hallucination	Hours	1. Huge Productivity boost and Save Time 2. Superior Swaggers Specs Quality and Less Defects 3. Faster Time to Market

Areas of Impact & New Ways to Working in Software Engineering with AI

Major Area	Old Manual Ways of Doing Things	Time	New Ways of Doing Things with AI	Time	Benefits, Improvements
Fix a defect	1. Understand the defect, review the code. 2. Write/Update/Test the code to fix the defect	Days	1. Write Accurate Prompt to fix the defect with AI 2. Review/Validate AI work and Check for AI Hallucination 3. Perform Debugging with AI, if AI generated incorrect code	Hours	1. Superior Code Quality and Less Defects 2. Huge Productivity boost and Save Time 3. Faster Time to Market
Code Refactoring	1. Understand the existing code and refactoring requirements 2. Rewrite/Review/Test the new refactored code	Days	1. Write Accurate Prompt & Refactor Code with AI 2. Review/Validate AI work and Check for AI Hallucination 3. Perform Debugging with AI, if AI generated incorrect code	Hours	1. Huge Productivity boost and Save Time 2. Superior Code Quality and Less Defects 3. Faster Time to Market
Upgrade Library or Framework Version	1. Understand the new version updates & existing code. 2. Update/Test/Review the code with new version	Days	1. Write Accurate Prompt to upgrade the version with AI 2. Review/Validate AI work and Check for AI Hallucination 3. Perform Debugging with AI, if AI generated incorrect code	Hours	1. Huge Productivity boost and Save Time 2. Faster Time to Market
Review & Approve Pull Request (PR)	1. Understand the requirements, existing code and best coding practices 2. Review the code against best practices and provide comments	Days	1. AI tools integration with workflow 2. Review/Validate AI work and Check for AI Hallucination	Hours	1. Superior Code Quality and Less Defects 2. Huge Productivity boost and Save Time 3. Faster Time to Market

New AI Skills required to adapt to new ways of working in Software Engineering

1. Good understanding of Prompt Engineering. How to create effective prompts? Good prompt components, structure, etc.
2. Special AI focused debugging skills required to work with AI issues
3. Validating AI's work and Checking for AI Hallucinations
4. Guard Rails and Points of Cautions when working with AI
5. Knowledge of new AI Tools and Protocols and how to use them properly. For example, VSCode Agent Mode, MCP (Git Flow,Connect DB, Search Web, Fetch Latest Docs, etc.)

AI Code Interface

Prompt Engineering

LLM

AI Workflow

AI Code Gen

AI Code Validators /Testers

AI Debugging

Validating AI Work

Checking for AI Hallucination

What is Prompt Engineering and Why it's important?

Prompt engineering is the process of carefully crafting input prompts to guide AI (Large Language Models) toward generating the most accurate, relevant, and useful outputs. *Think of it as the art of asking the right questions in the right way to get the best possible answers.*

Why Prompt Engineering is Important?

1. **Improved Accuracy, Efficiency and Saved Time:**
   Well-crafted prompts enhance both the accuracy and efficiency of your interactions with AI (Large Language Models). By providing clear instructions and context, you guide the AI to understand your intent and deliver the information you need quickly and precisely, minimizing unnecessary back-and-forth and saving you valuable time.
2. **Best Code Quality:**
   Crafting precise and detailed prompts with complete requirements help the AI to deliver the best code quality by using the best applicable coding standards and best practices.
3. **No or Minimal Defects:**
   By providing a accurate and detailed prompt helps AI to generate a defect free code. Incomplete or inaccurate prompts cause the AI to make false assumptions, which eventually result in defects in the code.

Prompt Engineering Best Practices

1. Be Specific and Provide Rich Context
   - Clearly articulate your goal. The more relevant details, background information, and specific instructions you provide, the better the LLM can understand and fulfill your request. Ambiguity leads to generic or off-target responses.
2. Define Desired Output Format & Style
   - Explicitly state the desired output structure (e.g., JSON, list, paragraph, table), tone (e.g., formal, casual, expert), length, and any stylistic requirements (e.g., "use camel case," "write as a pirate," "explain like I'm five").
   - Example: "Generate a Python function, not a full script, to sort a list of dictionaries by the 'lastName' key, returning the sorted list. Ensure the code is well-commented."
3. Decompose Complex Tasks into Simpler Sub-tasks
   - For multi-faceted requests, break them down into a sequence of simpler, more manageable sub-prompts. This allows the LLM to focus on each component, improving accuracy and making it easier to debug or refine specific steps.

Prompt Engineering Best Practices - Cont’

4. Leverage Few-Shot or One-Shot Prompting (Provide Examples)
    - Guide the LLM by providing concrete examples of the desired input-output behavior. A single example (one-shot) or a few examples (few-shot) can significantly improve performance for specific formats, styles, or complex reasoning patterns. This is especially effective for tasks requiring a nuanced understanding.
5. Clearly State Constraints and Guiding Assumptions
    - Explicitly list any constraints (e.g., "do not use external libraries," "avoid discussing X topic") or key assumptions the LLM should operate under (e.g., "assume the audience is non-technical," "assume the input data will always be valid"). This narrows the solution space and focuses the LLM.
6. Understand Your Model's Capabilities and Limitations
    - Be aware of the specific LLM you're using. Different models have varying strengths (e.g., coding, creative writing, reasoning), knowledge cut-off dates, token limits, and input modalities. Tailor your prompts and expectations accordingly. For instance, use a code-specialized model for programming tasks.

Prompt Engineering Best Practices - Cont’

7. Employ Chain-of-Thought (CoT) Prompting for Reasoning
   - For tasks requiring complex reasoning or multi-step problem-solving, instruct the LLM to "think step-by-step" or "explain its reasoning before giving the final answer." This encourages a more structured thought process, often leading to more accurate results and making its logic transparent.
8. Iterate and Refine Your Prompts
   - Prompt engineering is an iterative process. If the initial response isn't perfect, analyze what went wrong, adjust your prompt (e.g., add clarity, examples, constraints, or rephrase), and try again. Treat each attempt as an experiment to learn what works best.
9. Critically Evaluate and Validate AI Outputs
   - LLMs can generate plausible-sounding but incorrect, biased, or nonsensical information ("hallucinations"). Always critically review outputs for accuracy, relevance, and potential biases, especially for factual claims or critical applications. Cross-reference with reliable sources when necessary.

Prompt Engineering Best Practices - Cont’

10. Protect Sensitive Information: Avoid Sharing PII/Confidential Data
    - Never include Personally Identifiable Information (PII), sensitive business data, or confidential secrets in your prompts when using public or third-party LLMs, as this data may be used for training or could be exposed. Use anonymized, generalized, or synthetic data if you need to work with sensitive structures

Recommended Prompt Components, Structure and Examples

Components of a Good Prompt

1. Context (The Big Picture)
2. Current Task Specifications (Provide example or sample data, if possible)
3. Constraints, Assumptions or Expectations

Structure of a Good Prompt with an Example

Context: ....(Always start with the big picture, for example: In Java 17 with Springboot 3.5, Employee Microservice).....***Current Task Specifications:*** ......(Specific what, where and how to do? for example: For all the REST APIs present in the employee microservice, add API version string "v1" in the URI "/api/v1/employees/").....***Constraints/Assumptions/Expectations:***......(Mention the expectations, for example: Make sure this versioning doesn't affect any other part of the code)......

Why debugging skills with AI is required?

- When AI makes mistakes or can’t fix an defect or an compilation issue, then the debugging skills comes in handy. It can save a ton of time and frustration.
- AI models and tools are always evolving, they will make mistakes or hallucinate or may not know the answer, as they may not have seen that answer in their training data. In all these situations, AI debugging skills comes to the rescue.
- AI debugging skills differentiate an average developer from a highly efficient developer. As slowly everyone migrate from coding done by humans to coding done by AI, debugging skills will become critical part of this trade.

How to develop the debugging skills with AI?

- You develop the AI debugging skills by understanding and applying the best practices of debugging skills (Explained in next section).
- Our brain learns from repetition (deliberate practice). The more AI debugging skills you practice, the more proficient you become at it.

Best Practices of debugging skills with AI

1. **Use the same principles of Prompt Engineering for debugging skills**
   - Always start the debugging prompt with the Context (Project Background, Code, Technologies, etc.)
   - In the debugging prompt, provide the details of the error and the expected result.
2. **Provide the relevant error messages to the AI**
   - Let the tools read all the errors by itself, if this functionality is not supported by the AI tool, then copy and paste the error manually in the AI prompt.
   - In rare cases, if the error logs can't be copied directly into the AI tool, then take the screenshot of the error logs and attach it with the debugging prompt. Explicitly ask the AI to read the errors from the attached screenshot of the error logs.
3. **Fix One Error at a Time**
   - As a best practice, it is recommended to fix one error at a time. Otherwise, AI might get confuse and may not produce the best results.
   - If there are many different types of errors, then select the most important error first, then ask the AI to fix that one error at a time. Repeat the process with one error at a time, until all errors are fixed.

Best Practices of debugging skills with AI – Cont’

4. **Debugging is all about Experimentation, Iteration and Patience**
    - If an error is not fixed in one shot prompt, always try with different variations, like providing more context or providing extra details about the error.
    - If AI is not able to fix the errors by itself, then provide the AI with your suggestions, on how to fix the error or What you think, could be the solution.
    - Ask the AI to search the web, in order to find the solution to the error, if this functionality is not supported by the AI tool, then you manually research the possible solutions, and copy and paste the solutions from web into the AI prompt.
5. **Try With Different AI Model**
    - There is possibility that the AI model you are using may have not seen the error in its training data, hence it may not be able to fix the error. Either try using a latest AI model or try to use a different AI model from a different provider, which might have seen this error in its training data.
6. **Try The Old Way - GOOGLE IT**
    - If No AI Model works, then try the old Google approach. Copy and paste the error in google and try to find the solutions online. There is possibility, that the error you encountered is very new and no AI Model is trained on that type of error.

Validating AI’s work and Checking for AI Hallucinations

- What is AI hallucination?
- Why AI hallucinate?
- Examples of AI hallucinations in Code Generation
- Impact of language maturity on AI hallucination
- Impact of Task Complexity on AI hallucination
- How to validate AI’s work and check for hallucinations?
- How to reduce AI’s hallucinations and Improve Code Quality
- Summary of AI hallucination and its remediation

What is AI hallucination?

Formal or Technical Definitions

- AI hallucination refers to instances when an artificial intelligence system generates outputs such as text, images, data, or code, that are not grounded in reality or factual accuracy, despite appearing logical.
- A phenomenon where AI models produce false or misleading information due to limitations in their understanding or training data, despite presenting it as credible.

General Common Definitions

- Imagine asking a robot a question, and it gives you an answer that seems legit, but it just totally guessed, this is what we call hallucination.
- It’s like when an AI "makes stuff up" confidently, giving answers or creating things that sound or look right but aren’t true at all.
- It’s as if the AI is daydreaming or imagining things that don’t exist, blending creativity with a lack of reality checking.

Why AI hallucinate?

AI hallucinate because of the following 3 major reasons

1. **Insufficient or Biased Training Data:** If the data used to train the AI is limited, biased, or contains inaccuracies, the AI may learn incorrect patterns or make flawed assumptions.
2. **Overfitting:** The AI may become too specialized in the training data, making it unable to generalize to new situations or data.
3. **Lack of Real World Understanding:** AI models, especially language models, lack a true understanding of the world. They process information based on patterns and relationships in the data, but they don't have common sense or real world experience.

Examples of AI hallucinations in Code Generation

Factually Incorrect or Fabricated Code

- **Example**: AI might generate a function referencing a nonexistent library or API, like suggesting a magic_sort() method from a library that doesn't exist in reality.
- **Impact**: Developers might waste time debugging or searching for missing components that were fabricated.

Nonsensical or Illogical Code

- **Example**: The AI could produce code snippets that are syntactically correct but lack logical coherence, such as defining variables that are never used or including redundant loops.
- **Impact**: This can confuse developers and lead to bloated, inefficient, or meaningless code.

Plausible but Incorrect Code

- **Example**: AI might suggest incorrect syntax or semantics that seem plausible, such as using await in a non-asynchronous function in JavaScript.
- **Impact**: Bugs may arise, especially when the issue isn't immediately apparent, leading to downstream errors in the application.

Examples of AI hallucinations in Code Generation - Cont’

Flawed Pattern Recognition in Code Context

- **Example**: The AI might misinterpret the context of a problem, like generating bubble sort when the requirement was for selection sort, or mismatching a design pattern to a use case, especially when proper context is not provided in the prompt.
- **Impact**: Misalignment between the problem and solution results in non-functional or unsuitable code.

Limitations in Understanding Complex Logic

- **Example**: The AI may hallucinate dependencies or infer relationships between variables or classes that do not exist, e.g., assuming a User class automatically contains a getProfile() method.
- **Impact**: Developers could unknowingly rely on incorrect assumptions, leading to runtime errors and broken functionality.

As the code specific fine tuned AI models are further improved, the chances of AI making these mistakes will be significantly reduced. But Human validation and review of AI generated code will still remain essential.

Impact of language maturity on AI hallucination

- AI is prone to hallucination when presented with unfamiliar topics. In such cases, it may fabricate information, leading to inaccurate or untrue outputs. In the context of code generation, this applies to the maturity and training data (sample code) availability for a specific language or frameworks.
- When working with established and mature languages or frameworks (e.g., Java, Spring Boot, C#, .Net, Python, Flask, JavaScript, iOS, Android, etc.), the likelihood of AI hallucination or producing errors in code is significantly reduced, because of the abundant availability of training data (sample code) on the internet.
- Conversely, when using newly released languages, frameworks, or technologies with limited training data (sample code) on the internet, the probability of AI hallucination and inaccurate outputs is high.
- Even when AI avoids hallucination or explicit errors with new technologies, the generated code may still be of low quality.

Impact of Task Complexity on AI hallucination

- AI hallucinations can vary depending on the type of task you are attempting to solve, particularly in the domain of programming. For instance, the AI excels at handling common and well-defined tasks, such as CRUD (Create, Read, Update, Delete) operations on resources, sorting algorithms, search functionalities, REST APIs, and microservices, html pages, etc. These tasks are rooted in widely used patterns and practices, and the AI has encountered extensive examples of them during its training.

- However, when faced with more complex and specialized challenges, such as designing a novel encryption algorithm or tackling tasks outside its training data, the AI is more likely to hallucinate or struggle to provide accurate solutions. This limitation arises because the AI lacks exposure to these niche or innovative problems, leading to uncertainty or incorrect responses.

How to validate AI’s work and check for hallucinations? - 1

1. Exhaustive Manual Testing

Manual testing remains one of the most reliable methods for validating AI generated code. By systematically executing and verifying every aspect of the software, you can identify bugs, logic errors, or performance issues introduced by the AI.

Key Steps in Manual Testing:

- **Systematic Analysis:** Thoroughly assess the software’s functionality against the requirements to ensure it performs as expected.
- **Edge Case Testing:** Test for boundary conditions and scenarios that may not have been anticipated during development.
- **Iterative Validation:** Revisit testing after every iteration or modification to catch newly introduced issues.

This foundational approach is invaluable for identifying hallucinations or defects and is indispensable for critical applications.

How to validate AI’s work and check for hallucinations? - 2

2. Incorporate Unit Testing

Unit testing is essential for verifying individual components or functions in isolation. Properly designed unit tests can help catch logical errors or flaws in core functionality introduced by AI.

Tips for Effective Unit Testing:

- **Separate Code and Test Generation:** Avoid generating the main code and its unit tests within a single AI prompt. This prevents the risk of the same logical errors being mirrored in both outputs.
- **Diverse AI Sources:** Generate the code and its tests using different AI models or providers. The chance of two models hallucinating in identical ways is significantly reduced, enhancing reliability.
- **Automated Execution:** Use tools to automate unit tests, ensuring faster and more consistent results.

How to validate AI’s work and check for hallucinations? - 3

3. Automate Validation with Integration, API, and End-to-End Tests

Automation testing goes beyond individual functions to validate the software’s overall integration and performance.

Types of Automation Tests:

- **Integration Tests:** Verify how different modules interact and ensure seamless functionality across components.
- **API Contract Tests:** Ensure that APIs adhere to their specifications, maintaining stability and reliability in communication.
- **End-to-End Tests:** Simulate real-world user scenarios to validate the entire system from input to output.

Investing in automation tools and frameworks can significantly improve testing efficiency and coverage, especially for complex systems.

How to validate AI’s work and check for hallucinations? - 4

4. AI Assisted Code Review

Leveraging AI for code review can be a quick and effective way to detect errors in AI-generated work.

Best Practices for AI Assisted Code Reviews:

- **Use a Different AI Model or Provider**: Employ a different AI system to review the original AI's output. This reduces the likelihood of shared errors or blind spots.
- **Request Specific Insights:** Ask the reviewing AI to highlight defects, potential bugs, and areas for improvement, ensuring a comprehensive assessment.
- **Iterative Feedback:** Use the AI’s suggestions to refine and re-validate the code iteratively.

While this method is efficient, it should ideally complement manual review for critical applications.

How to validate AI’s work and check for hallucinations? - 5

5. Human Code Review

Despite advancements in AI, human expertise remains unparalleled for code validation. A human reviewer can assess the code against best practices, design patterns, and the specific requirements of the language or framework.

Steps for Effective Human Code Reviews:

- **Standardized Checklists:** Use checklists to ensure consistent evaluation across various code aspects such as readability, maintainability, and performance.
- **Pair Programming:** Collaborate with peers for more thorough reviews and brainstorming.
- **Continuous Learning:** Reviewers should stay updated with the latest industry trends and best practices.

Though time intensive, human review is critical for high stakes projects where precision is paramount.

How to validate AI’s work and check for hallucinations?

Choosing the Right Approach to Validate AI’s work and check for hallucinations

The optimal method(s) for validating AI-generated work depend on the project’s complexity, available resources, and criticality. Ideally, a combination of these approaches is employed:

- **Manual Testing, Unit Testing** and **Human Code Reviews** are indispensable for comprehensive validation.
- **Automated Testing** and **AI-Assisted Reviews** can complement manual methods, providing scalability and speed.
- For mission-critical software, investing in multiple layers of validation is essential to ensure robustness and reliability.

By adopting a tailored blend of these strategies, you can minimize risks and ensure the highest quality in AI generated work.

How to reduce AI’s hallucinations and Improve Code Quality

Ask AI Explicitly to Follow Specific Coding Best Practices

- When crafting your prompt, explicitly instruct the AI to adhere to best practices relevant to the specific language or framework you are using.
- If the AI occasionally fails to follow these practices, you can include specific examples of important coding standards directly in the prompt for reinforcement.

Provide Organization adopted or Industry wide adopted Coding Standards

- If your organization has established coding standards, include those standards in the prompt and explicitly ask the AI to follow them.
- In this course, we’ve shared a industry coding practices. You can use this as a reference to incorporate into your prompts for consistency and quality.

Use High-Quality Sample Code for New Languages or Frameworks

- When working with relatively new programming languages or frameworks (with limited training data available), include a high quality sample code snippet in the prompt.
- Ask the AI to follow the patterns and coding best practices demonstrated in the provided sample.

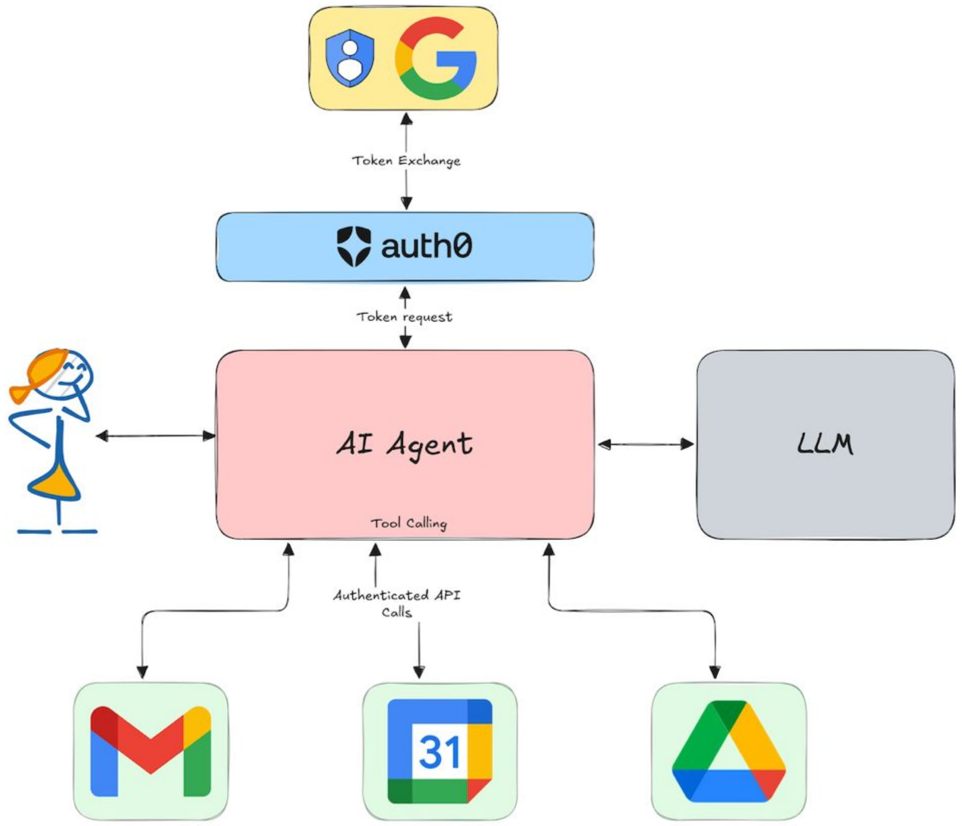
Summary of AI hallucination and its remediation

- **In summary,** for the most common day to day coding tasks, AI can be used reliably for code generation with mature languages and frameworks. However, when working with **newer languages** or **complex novel tasks**, while AI can still be helpful, but be aware of the potential for low quality code or hidden errors.
- Validating AI's work becomes a critical step of the code generation process using AI. By adopting a tailored blend of these 5 testings and validation strategies, you can minimize risks and ensure the highest quality in AI generated code.
  - Exhaustive Manual Testing
  - Incorporate Unit Testing
  - Automation Testing
  - AI Assisted Code Review
  - Human Code Review

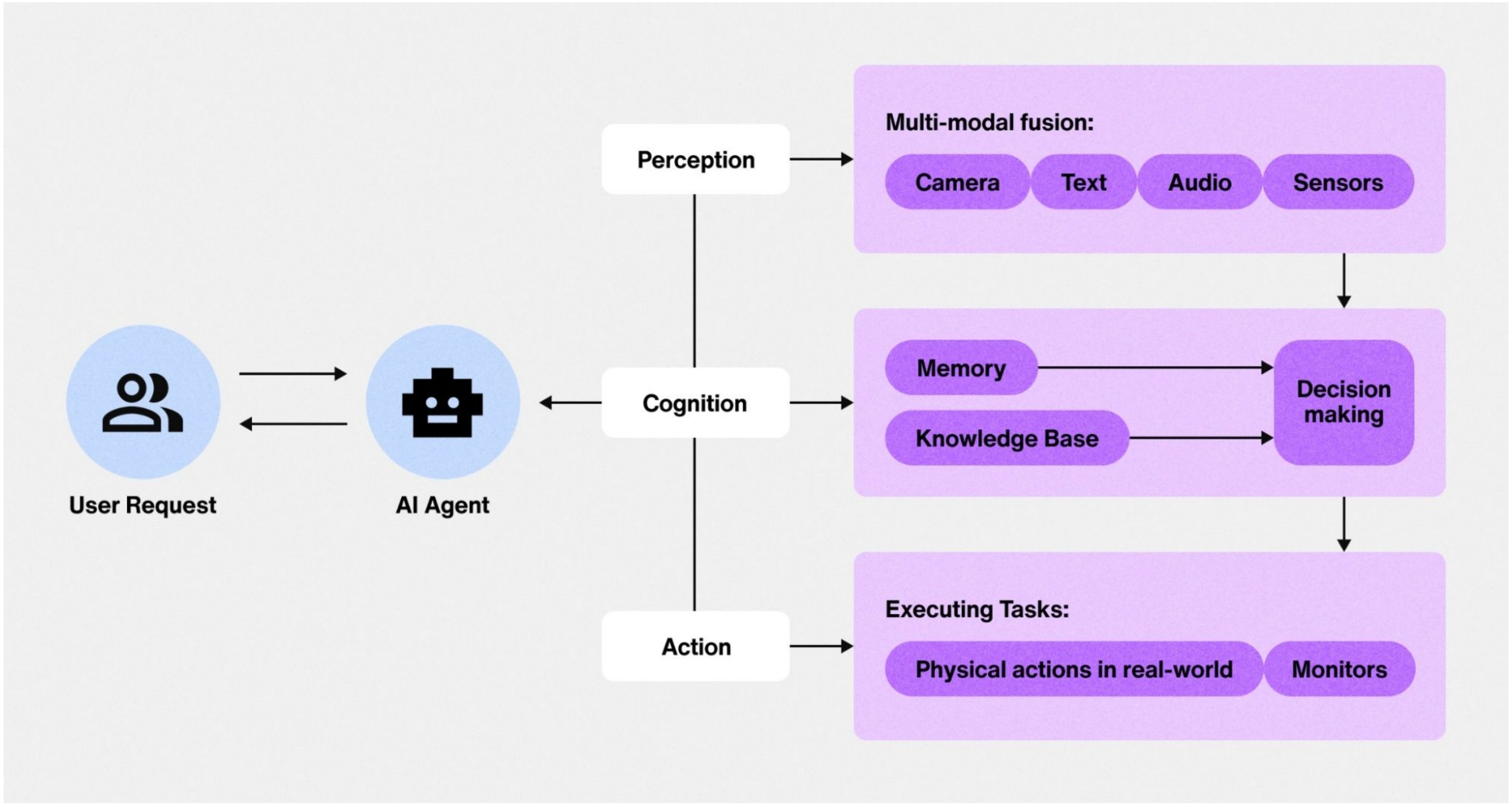
Guard Rails and Points of Caution when working with AI

1. **Commit Your Code Before Each Iteration with AI**
   a. Git provides a safety net when using AI for code modifications. If unwanted changes occur, you can easily revert to your original code.
2. **Be Prepared to Handle AI Disruption**
   a. AI tools can be disrupted by API limits, server issues, or other unexpected problems. To protect your work, save regularly and back up your prompts.
3. **Expect Multiple Iterations and Constant Feedback Cycles with AI**
   a. AI models are constantly learning. To achieve the best results, provide feedback on the initial output, guiding the AI with refinements until it meets your expectations. One-shot or few-shot prompting can be helpful techniques.
4. **Provide as much details as possible to AI**
   a. To avoid inaccuracies and wasted effort, provide the AI with detailed instructions in your initial prompt. This minimizes false assumptions and helps the AI deliver the desired functionality quickly.
5. **AI might introduce defects in your code.**
   a. Never blindly trust AI generated code. Always validate its output. Even seemingly minor errors, like the incorrect date format and off-by-one error we encountered in our Employee Microservice, demonstrate the importance of thorough testing. It is recommended to add unit tests, automated tests and manual tests to your code.
6. **Avoid sharing PII or Sensitive or Confidential Data with AI**
   a. Avoid sharing organization’s or customer’s real data (PII, Sensitive or Confidential information), instead, try to use anonymized data. As this data may further be used to train the future models.

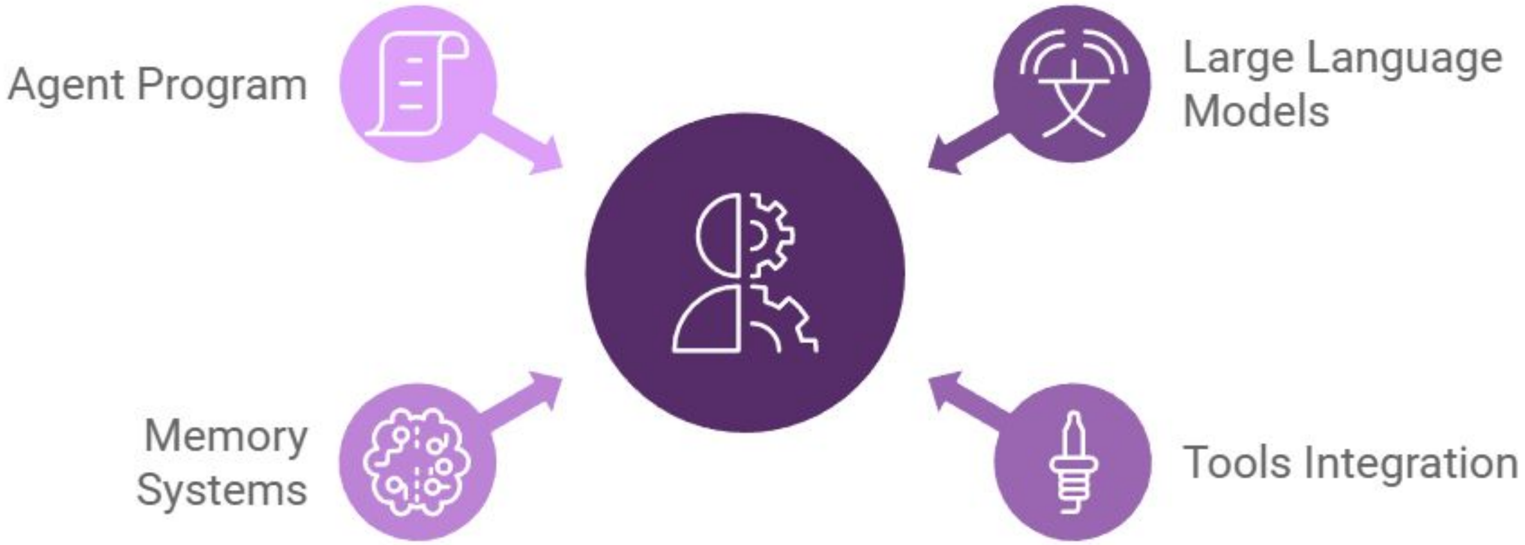
What are AI Agents?



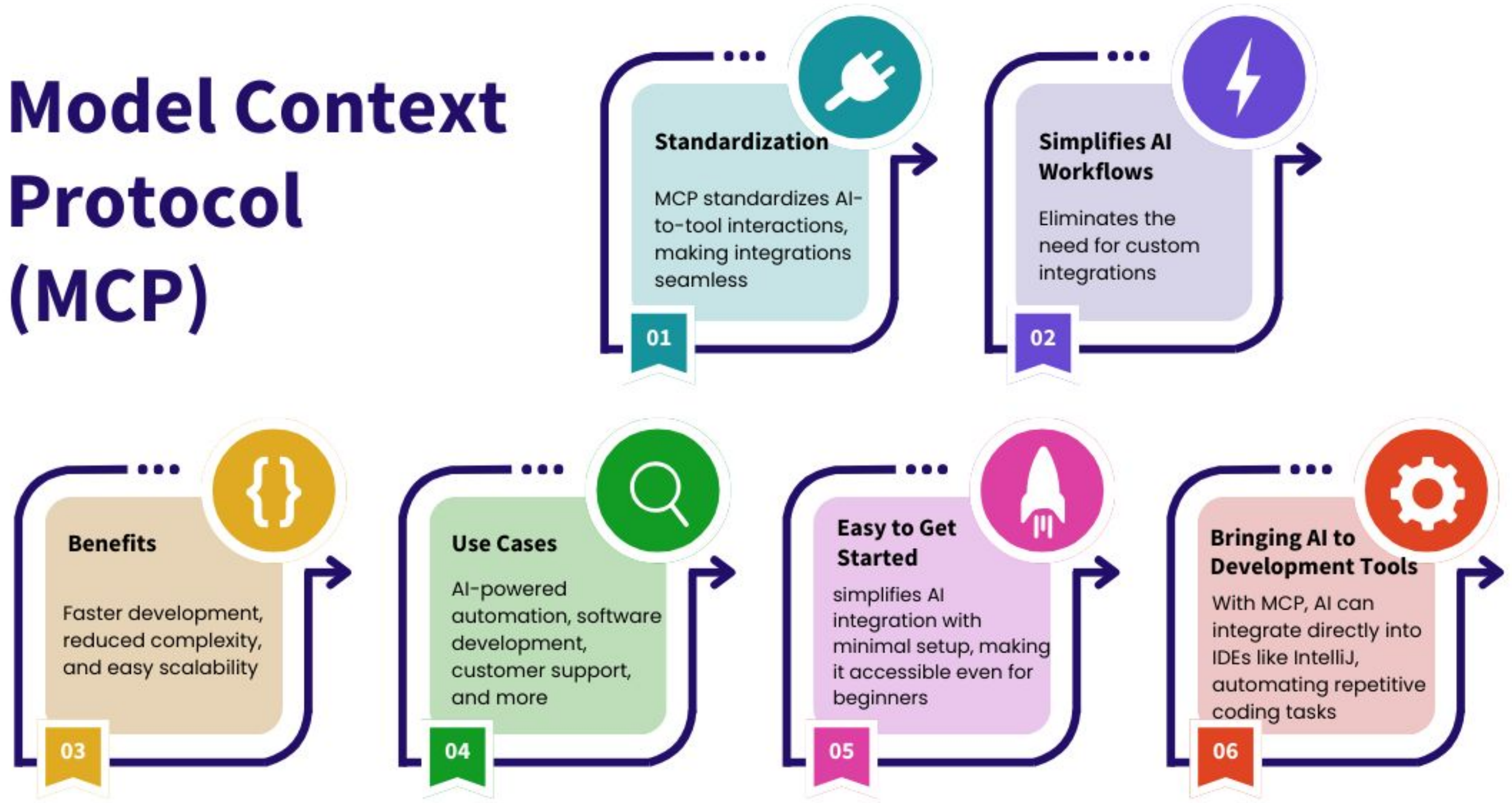
What are AI Agents?



Components of AI Agent Architecture



What is MCP?



Model Context Protocol (MCP)

01 Standardization

MCP standardizes AI-to-tool interactions, making integrations seamless

02 Simplifies AI Workflows

Eliminates the need for custom integrations

03 Benefits

Faster development, reduced complexity, and easy scalability

04 Use Cases

AI-powered automation, software development, customer support, and more

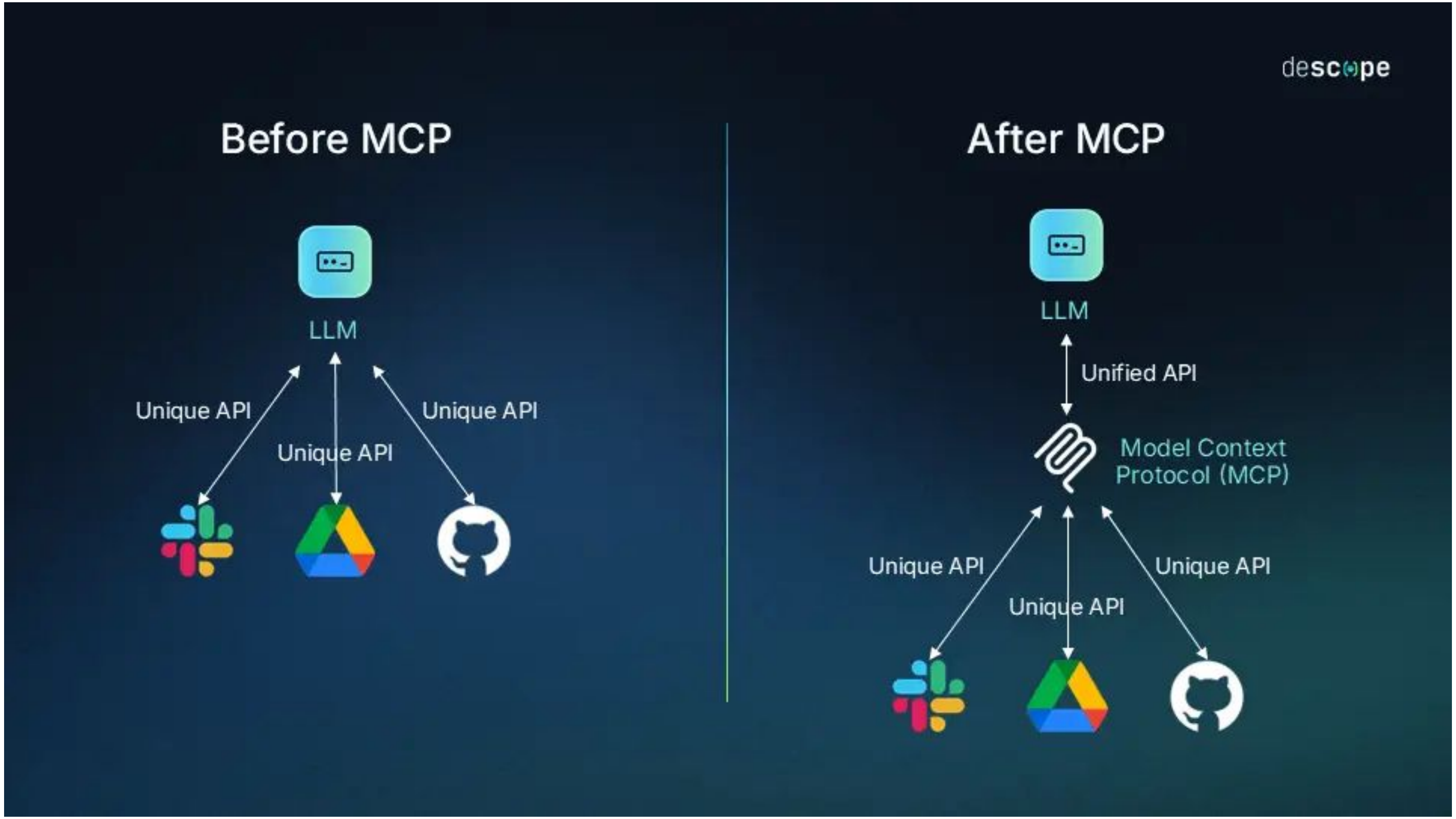
05 Easy to Get Started

simplifies AI integration with minimal setup, making it accessible even for beginners

06 Bringing AI to Development Tools

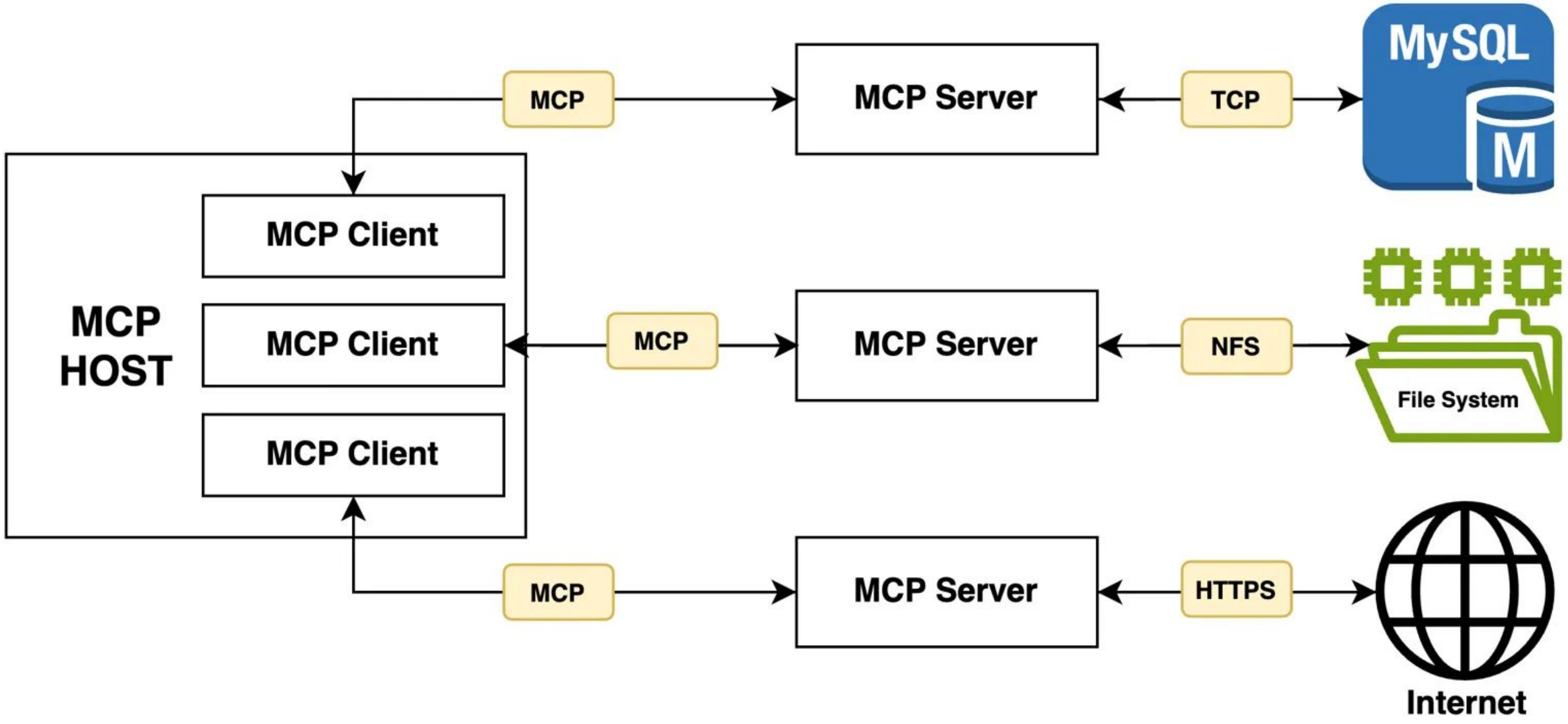
With MCP, AI can integrate directly into IDEs like IntelliJ, automating repetitive coding tasks

What is MCP?

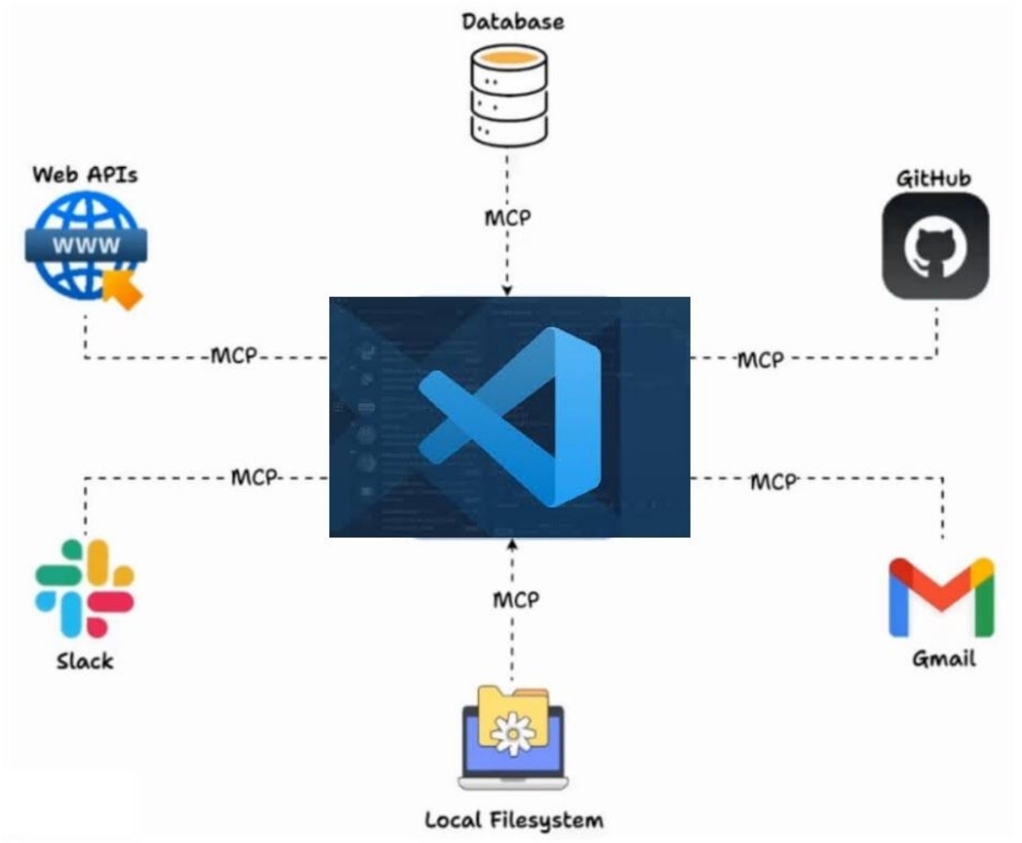


descope
Before MCP
LLM
Unique API
Unique API
Unique API
After MCP
LLM
Unified API
Model Context Protocol (MCP)
Unique API
Unique API
Unique API

MCP Client Server Architecture



VSCode AI Agent Mode + MCP + Tools = Vibe Coding



This is Vibe Coding!

- **Human/User** will provide the Prompt
- **LLM + AI Agent** will write the code
- **LLM + AI Agent** will execute & test the code
- **LLM + AI Agent** will read the errors from console and will fix the code
- **LLM + AI Agent** will the search the web for code defect solutions
- **LLM + AI Agent** will create test data & insert in the database
- **LLM + AI Agent** will create the PR and will generate PR description
- **LLM + AI Agent** will notify the user on Slack/Email that the work is done.

Guard Rails and Points of Caution when working with AI Agents

1. **Create Sub- Feature Branch for AI Agent**
   a. Git provides a safety net when using AI for code modifications. If unwanted changes occur, you can easily revert to your original code.
2. **Only Connect to Local Machine Database (Don't Connect AI Agent to Dev/UAT Environments Database)**
   a. AI tools can be disrupted by API limits, server issues, or other unexpected problems. To protect your work, save regularly and back up your prompts.
3. **Use the Fetch Tools (Search Web) - Check what is allowed by RBC security policy**
   a. AI models are constantly learning. To achieve the best results, provide feedback on the initial output, guiding the AI with refinements until it meets your expectations. One-shot or few-shot prompting can be helpful techniques.
4. **Create test data only on Local Database**
   a. To avoid inaccuracies and wasted effort, provide the AI with detailed instructions in your initial prompt. This minimizes false assumptions and helps the AI deliver the desired functionality quickly.
5. **Provide PR Generation Template for AI Agent.**
   a. Never blindly trust AI generated code. Always validate its output. Even seemingly minor errors, like the incorrect date format and off-by-one error we encountered in our Employee Microservice, demonstrate the importance of thorough testing. It is recommended to add unit tests, automated tests and manual tests to your code.
6. **Don't Install Any Unapproved Tools in VSCode**
   a. Avoid sharing organization's or customer's real data (PII, Sensitive or Confidential information), instead, try to use anonymized data. As this data may further be used to train the future models.

Demo

Documentation Generation with AI

Demo - Documentation Generation with AI

- Understand any repository and create all documentation (sequence diagram, component diagrams, class diagram, architecture diagram)
  - Option 1 (Manual Prompt to generate docs)
  - Option 2 (Use a open source lib to generate docs)
    - https://github.com/The-Pocket/PocketFlow-Tutorial-Codebase-Knowledge

Demo Time

Demo

Code Generation with AI
(End to End Flow for Code Generation of Employee Microservice)

Implementation Plan of Employee Microservice with AI

1. Initial Tools Setup
   - Setup IDE, AI tools and API Keys, Project (via start.spring.io), Database (In memory - H2), API Client (Web version of Postman for testing APIs)
2. Review Employee Microservice requirements and Divide the implementation into small independent steps
3. Code Generation with AI for each step/feature of Employee Microservice
4. Review Implementation Summary of Employee Microservice with AI
5. Review Benefits and Limitations of AI Code Generation for Employee Microservice
6. Next Steps - Assignment for Employee Microservice with AI

Initial Tools Setup

1. Set up IDE, AI tools and API Keys
   a. IDE (Integrated Development Environment)
      i. VSCode
   b. AI Tools
      i. Cline Extension (For End to End Code Generation)
         1. Setup Paid version via Anthropic APIs (Better Performance)
         2. Setup Free version via Google Gemini APIs (Ok Performance)
      ii. Supermaven Extension (For Intelligent Code Autocompletion)
         1. Setup Free Version
   c. Java Extensions (JDK and Extension Pack for Java)
2. Setup Springboot Project with Maven from scratch
   a. Create empty project structure via start.spring.io with 4 dependencies in pom.xml (Spring Web, Spring Data JPA, H2 Database, Lombok)
3. Setup In Memory Database using H2 (Ask AI to Setup H2 Configuration)
   a. We will use H2 in memory for ease of usage (no installation required)
4. Setup API Client for testing APIs locally
   a. We will use Postman (free web version)

Review Employee Microservice requirements and Divide the implementation into small independent steps

We will develop a Employee Microservice from scratch using AI, which will perform all CRUD (Create, Read, Update, Delete) operations on the employee data using the REST APIs. It will have an exclusive access to “employees” table in H2 database.

We will divide our implementation of Employee Microservice in small independent 6 steps

- **Step 1** - Create an Employee Model with AI
- **Step 2** - Create a POST API to create an employee in the database with AI
- **Step 3** - Create a GET API to fetch all the employees from the database with AI
- **Step 4** - Create a GET API to fetch a single employee with id from the database with AI
- **Step 5** - Create a PATCH API to edit a single employee details in the database with AI
- **Step 6** - Create a DELETE API to delete an employee from the database with AI
- **Step X** - Any New Feature or Defect will be fixed or developed in a separate step

AI Pro Tip - Divide and Conquer- In Small Independent Steps

To get the best results from AI when generating code for a microservice, break down complex requirements into smaller, testable steps instead of providing all the information at once!

Step 1 - Create an Employee Model (JPA Entity Class Mapped to “employees” Table) with the following data fields

1. ID [Type:INT, Autogenerated, Column Name: id]
2. Name [Type:String, Column Name: name]
3. Date Of Joining [Type:Date, Column Name: date_of_joining]
4. Status [Type:ENUM, Possible Values (Active, Not Active), Column Name: status]
5. Department [Type:ENUM, Possible Values (HR, IT, Finance, Sales, Marketing), Column Name: department]
6. Salary [Type:DOUBLE, Column Name: salary]
7. ManagerID [Type:INT, Column Name: manager_id]

AI Pro Tips

1. The More Details You Can Provide for a small step/task, The Better Results You Will Get
2. Provide AI with all the data types including auto generated ids, Enum possible values, Table Name and Column Names mappings, etc.

Step 1 - Recommended Prompt to Create an Employee Model (JPA Entity Class) with following data fields

Context: I am creating Employee Microservice from scratch in Java 17 using Springboot 3.2.11 and with Maven with the following 4 dependencies inside pom.xml (Spring Web, JPA, Lombok, H2). ***Current Task:*** Using the dependencies and specifications provided in the context, Create an Employee Model (JPA Entity Class) which will map to Table Name “employees” with the following employee data fields and specifications: 1. ID [Type:INT, Autogenerated, Column Name: id], 2. Name [Type:String, Column Name: name], 3. Date Of Joining [Type:Date, Column Name: date_of_joining], 4. Status [Type:ENUM, Possible Values (Active, Not Active), Column Name: status], 5. Department [Type:ENUM, Possible Values (HR, IT, Finance, Sales, Marketing), Column Name: department], 6. Salary [Type:DOUBLE, Column Name: salary], 7. ManagerID [Type:INT, Column Name: manager_id]. *Create all necessary classes, packages, annotations, enums, getters, setters, constructors. Also, setup the default configurations H2 database in application.properties file*.

Prompt Engineering Skills

1. First provide the context (big picture), like language, framework, dependencies, architecture type, attach relevant code (if possible), etc.
2. Second provide the current task specific details (field name, type, enum values, table column mappings, etc.)

Step 2 - Develop a POST REST API to create employee object with following data fields in the JSON request

1. Name [String, Mandatory]
2. Date Of Joining [Date, Mandatory]
3. Status [ENUM, Possible Values (Active, Not Active), Throw Error if Outside this value, Mandatory]
4. Department [ENUM, Possible Values (HR, IT, Finance, Sales, Marketing)Throw Error if Outside this value, Mandatory]
5. Salary [DOUBLE, Optional]
6. ManagerID [INT, Optional]

Note: Also add data validation for each data fields

AI Pro Tips

1. If you already have defined JSON Structure, then provide that in the context and Ask AI to generate the POST API using this JSON structure
2. Ask AI to use 3 Layered Architecture (Controller, Service and Repository)

Step 2 - Recommended Prompt to Create a POST REST API with following data fields

Context: I have an existing Employee Microservice in Java 17 using Springboot 3.2.11 and with Maven with the following 4 dependencies inside pom.xml (Spring Web, JPA, Lombok, H2) with an employee model already present. ***Current Task:*** Using the dependencies and specifications provided in the context, Create a POST REST API using 3 layered architecture (Controller, Service and Repository) which will save an employee data in the database with the following data fields coming from JSON request: 1. Name [Mandatory], 2. Date Of Joining [Mandatory], 3. Status [Mandatory, Type: ENUM, Possible Values (Active, Not Active)], 4. Department [Mandatory, Type: ENUM, Possible Values (HR, IT, Finance, Sales, Marketing)], 5. Salary [Optional], 6. ManagerID [Optional]. *Also add all data validation for each data fields as required and as mentioned in specifications and throw a meaningful error back, when any data validation fails. Use camel Case for naming the JSON Request data fields.*

Prompt Engineering Skills

1. First provide the context (big picture), like language, framework, dependencies, etc. Ask AI to use 3 Layered Architecture (Controller, Service and Repository)
2. Second provide the current task specific details, if you have predefined JSON Structure, then provide in details.

Step 3 - Create a GET REST API to fetch all employee details with AI using the below data fields

1. ID
2. Name
3. Date Of Joining
4. Status
5. Department
6. Salary [If doesn’t exist, don’t send this value in the response]
7. ManagerID [If doesn’t exist, don’t send this value in the response]

AI Pro Tips

1. Give AI specific requirements (if data field doesn’t exist, then don’t sent it in response)
2. Regularly commit your code, especially before allowing AI to write your code, so if that, if the AI make a mistake, you can always go back to the previous committed code and start again.

Step 3 - Recommended Prompt to Create a GET REST API with following data fields

Context: Please review source code of my existing Employee Microservice in Java using Springboot (attached source code) ***Current Task:*** Using the dependencies and specifications in the existing source code, Create a GET REST API on the path "/api/employees", which will fetch all the available employees data from the database with the following data fields: 1. ID, 2. Name, 3. Date Of Joining, 4. Status, 5. Department, 6. Salary [If doesn't exist, don't send this value in the response], 7. ManagerID [If doesn't exist, don't send this value in the response]. *Send all the employee data in a list and if no employee exists, then send an empty list. Use camel Case for naming the JSON Response data fields.*

Prompt Engineering Skills

1. Instead of writing full context in plain text, you can also send (attach) the relevant source code.
2. For new project or when source code is not present or cannot be sent due to privacy or legal reasons then provide the full context in plain text.
3. For accurate results it's best to provide only the relevant section of the source code.

Step 4 - Create a GET REST API to fetch a single employee details by {id} with AI using the below data fields

1. ID
2. Name
3. Date Of Joining
4. Status
5. Department
6. Salary [If doesn't exist, don't send this value in the response]
7. ManagerID [If doesn't exist, don't send this value in the response]

AI Pro Tips

1. Regularly commit your code, especially before allowing AI to write your code, so if that, if the AI make a mistake, you can always go back to the previous committed code and start again. (Super Important)

Step 4 - Recommended Prompt to Create a GET REST API by {id} with following data fields

Context: Please review source code of my existing Employee Microservice in Java using Springboot (attached source code) ***Current Task:*** Using the dependencies and specifications in the existing source code, Create a GET REST API on the path “/api/employees/{id}”, which will fetch a single employee data from the database for the provided {id} in the request. In the response, return following data fields: 1. ID, 2. Name, 3. Date Of Joining, 4. Status, 5. Department, 6. Salary [If doesn’t exist, don’t send this value in the response], 7. ManagerID [If doesn’t exist, don’t send this value in the response]. *If the employee data doesn’t exist, then return 404 Not Found Error. Use camel Case for naming the JSON Response data fields.*

Prompt Engineering Skills

1. When possibility of confusion, like with two similar GET APIs (get all employees & get employee by id), then add clarity by providing specific details like “path:/api/employees/{id}” to AI

Step 5 - Develop a PATCH REST API to edit any of the below data fields of the employee object

1. Name [Optional]
2. Date Of Joining [Optional]
3. Status [Optional]
4. Department [Optional]
5. Salary [Optional]
6. ManagerID [Optional]

Note: Also add data validation for each data fields as necessary

AI Pro Tips

1. Ask AI explicitly to use Patch API instead of using PUT API, as Patch would only update one or more data fields as per the need, while PUT would override the entire employee object in the database.

Step 5 - Recommended Prompt to edit any data field of the employee object via PATCH REST API

Context: Please review source code of my existing Employee Microservice in Java using Springboot (attached source code) ***Current Task:*** Using the dependencies and specifications in the existing source code, Create a PATCH REST API on the path "/api/employees/{id}", which will update any of the provided data fields in the json request for existing employee object in the database. *Except the "id", all the other existing employee data fields can be modified by the PATCH API. Do a proper data validations and error handling, like accept only allowed enum values. Throw error when employee doesn't exist, etc.*

Prompt Engineering Skills

1. For the existing project, when data fields already exist, we don't need to provide the all the data fields in the prompt, let AI read it from existing code.
2. It is a good practice to explicitly mention all the important data validations and error handling in the prompt to the AI.

Step 6 - Develop a DELETE REST API to delete an employee object

Requirements: Develop a DELETE REST API on the path “/api/employees/{id}”, also perform standard error handling, like when no employee exist for the provided id, then throw relevant error.

Recommended Prompt to create a DELETE REST API with AI

Context: Please review source code of my existing Employee Microservice in Java using Springboot (attached source code) ***Current Task:*** Using the dependencies and specifications in the existing source code, Create a DELETE REST API on the path “/api/employees/{id}”, which will do a HARD Delete of the employee object from the database. *Do the proper data validations and error handling, like throw relevant error when employee doesn’t exist, etc.*

Prompt Engineering Skills

1. It is good practice to explicitly mention HARD delete or SOFT delete, so that AI will not confuse and code the required implementation.
2. In Hard Delete, the record is actually deleted from the database, while in Soft delete the record is flagged as deleted by changing one of the data field (like status, etc.) value to “deleted”, “inactive”, etc.

Add 2 New Features to Employee Microservice

First Simple Feature: Add API Version String (“v1”) in the path for all APIs, example “/api/v1/employees/”

Recommended Prompt to add a new feature (version string)

Context: Please review source code of my existing Employee Microservice in Java using Springboot (attached source code) ***Current Task:*** For all the APIs present in the employee microservice, add API version string “v1” in the URI “/api/v1/employees/”. *Make sure this versioning doesn’t affect any other part of the code.*

AI Pro Tips

1. As a best practice, always version your APIs, either via URI or Headers or QueryString.
2. Generally the standard way is to use version string “v1” in the URI

Add 2 New Features to Employee Microservice

Second Medium Feature: Enhance the "Get All Employee API" to allow response sorting by salary.

- Implement ascending/descending sort order based on employee salary.
- Use a query string parameter to specify the sort order ("sort=asc" or "sort=desc").
- Default to no sorting if no query string is provided.

Recommended Prompt to add a second new feature (sorting)

Context: Please review source code of my existing Employee Microservice in Java using Springboot (attached source code) ***Current Task:*** Using the dependencies and specifications in the existing source code, enhance the "Get All Employee API" to allow response sorting by salary. Implement ascending/descending sort using a query string parameter to specify the sort order ("sort=asc" or "sort=desc"). *Default to no sorting if no query string is provided. Do a proper data validations and error handling, like accept only allowed sorting options.*

AI Pro Tips

1. As a best practice, always ask AI to perform data validation and error handling and provide default values to the API

Fix the Date of Joining Defects in Employee Microservice

- **Date of Joining Format Inconsistency:** When submitting employee data via the POST API, the Date of Joining field is accepted in "yyyy-mm-dd" format, but, the API response returns this date in the format "yyyy-mm-ddT00:00:00.000+00:00", creating inconsistency.
- **Date of Joining Data Error:** The Date of Joining saved in the database is one day earlier than the date submitted through the POST API. This discrepancy results in incorrect date retrieval through Get All Employees and Get Employee By ID APIs.

Recommended Prompt to Fix the Date of Joining Defect

Context: In the existing Employee Microservice using Java using Springboot (review source code if required) ***Current Task:*** Fix the **Date of Joining Format Inconsistency.** When submitting employee data via the POST API, the Date of Joining field is accepted in "yyyy-mm-dd" format, but, the API response returns this date in the format "yyyy-mm-ddT00:00:00.000+00:00", creating inconsistency. Also fix the **Date of Joining Data Error.** The Date of Joining saved in the database is one day earlier than the date submitted through the POST API. This discrepancy results in incorrect date retrieval through Get All Employees and Get Employee By ID APIs.

Prompt Engineering Skills

1. Provide enough context for API to fix the defect, like which API or section of Code? What's Wrong (Defect Description)? What's Expected (Acceptance Criteria)?
2. Two related defects (e.g. defects related to Date of Joining) can be combined together to save time and increase AI efficiency.

Fix the Bad Request Format Defects in Employee Microservice

In the POST API request data validation, instead of giving 400 (Bad Request) for all the data validation errors, explain the error in details for each data field. For example, what is wrong with “dateOfJoining” format, what are the allowed enums (possible values) for “status” and “department” data fields. What is the correct expected type and format of “salary” and “managerId” data fields.

Recommended Prompt to Fix the Date of Joining Defect

Context: In the existing Employee Microservice using Java using Springboot (review source code if required) ***Current Task:*** In the POST API request data validation, instead of giving 400 (Bad Request) for all the data validation errors, explain the error in details for each data field. For example, what is wrong with “dateOfJoining” format, what are the allowed enums (possible values) for “status” and “department” data fields. What is the correct expected type and format of “salary” and “managerId” data fields.

Prompt Engineering Skills

1. Provide enough context (data fields level details, API name, HTTP status code, etc.) to the AI and also provide the expectation (Acceptance Criteria)!
2. Sometimes AI might not give a complete working solution in first iteration, we have to iterate multiple time with constant feedback to AI, until AI give a satisfactory working solution.