Unifi by Intertech

Powered by AI. Guided by You.

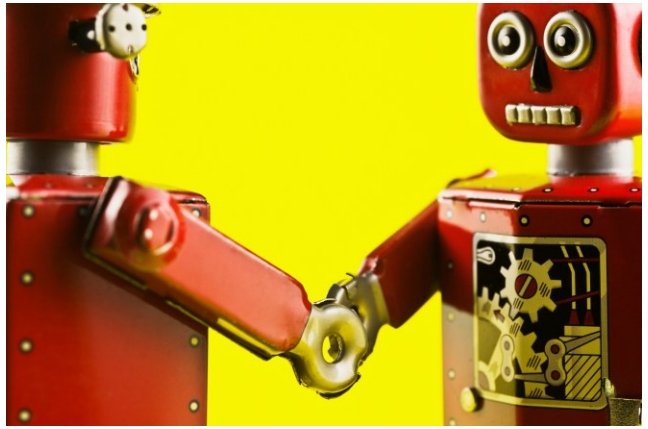
Welcome to Generative AI and Prompt Engineering for Developers!

Ryan Harvey & Brian Hackerson

August 14th, 2025

Course Working Agreement

- This is a beginner-level course. In our time together, with the volume of material we have to cover, we will not be able to go into great depth
- Please keep microphones muted unless speaking to minimize background noise and distractions
- Strive for one conversation to ensure everyone can follow. Please use the raised hand feature if you have a question
- Use your phone to scan QR codes during the session for quick access to flash surveys we will use to gather input
- Engage actively via the Teams chat; designated Intertech team members will monitor the chat to support and answer questions in real-time
- Breakout rooms will be used for smaller group discussions, with team members available to facilitate and assist as needed.



✓ **Breaks every 60-90 minutes**

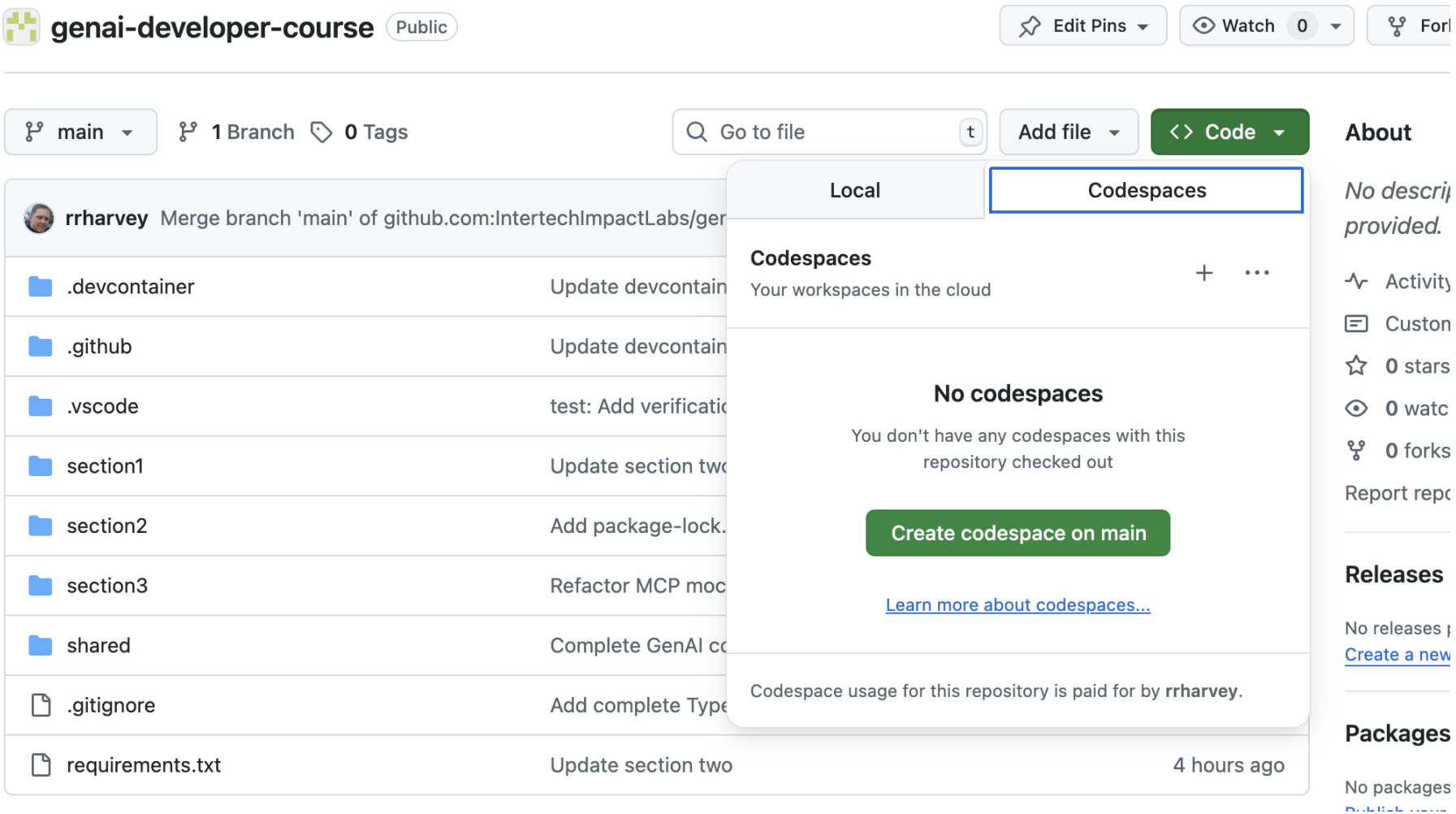
✓ **30-minute lunch break around noon CDT**

Check-In: You and Generative AI (GenAI) Right Now

Disclaimer

- AI evolution is running at an unprecedented pace.
- The right mindset is to think about everything you will learn in a time-relative way.
- Things that look hard today might be normal tomorrow
  - And, by the way, we mean “tomorrow” as in 24 hours from now
- So, for all the answers we give you, please mentally add “with the current technology” in front if we forget to do so.
  - **Example:** Can GPT4.1 transform a voice conversation into a well-formed PDF sent to my email with one voice command?
  - **Answer: *With the current technology,*** NO, you need to design multiple steps composed of a few models and software layers.

github.com/IntertechImpactLabs/genai-developer-course

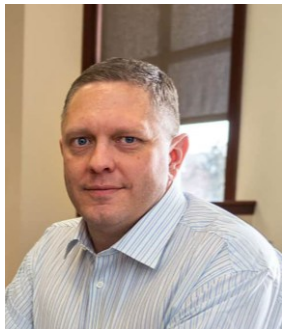


Welcome to Intertech!

Powered by AI. Guided by You.

Unifi by INTERTECH

Your Instructors



Ryan Harvey
Senior Consultant

Brian Hackerson
Delivery Manager

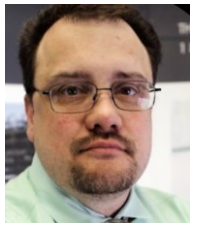
Development Team Members

Tom Helvick
CTO

Dave Cloutier
Senior Consultant

Mike Trotman
Senior Consultant

Jim Mallet
Senior Consultant



Jeremy Vetter
Senior Consultant

Unifi by Intertech

Powered by AI. Guided by You.

Session 1 Part A:

Introduction to Generative AI

Ryan Harvey & Brian Hackerson

August 14th, 2025

What is Generative AI?

- AI systems that create new content based on patterns learned from training data
- Goes beyond classification/prediction to generation
- Large Language Models (LLMs) trained on massive amounts of text

Capabilities

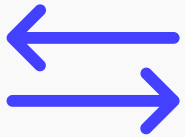
Core Capabilities of Generative AI

Text Generation

Generates code snippets, documentation, and detailed explanations based on learned patterns from massive data.

Problem Solving

Assists in debugging, designing algorithms, and making architecture decisions autonomously or interactively.



Translation

Converts between programming languages, paradigms, and formats to facilitate cross-compatibility and migration.

Pattern Recognition

Identifies bugs, code smells, and optimization opportunities by analyzing code patterns at scale.

What tools are you using, and how are you using GenAI today?

Impact

Why Generative AI Matters for Developers

Productivity Multiplier

Generative AI accelerates boilerplate code creation, provides instant access to examples and documentation, enables parallel processing of multiple coding tasks, and acts as a 24/7 knowledgeable pair programmer.

New Development Paradigms

Innovations include natural language to code conversion, test-first development by generating tests from requirements, rapid prototyping of multiple solutions, and knowledge synthesis combining best practices from millions of codebases.

Real Impact: 2024-2025 Data

Productivity 26% increase in PR completion (Microsoft/MIT study)[1]

26% increase in PR completion (Microsoft/MIT study)[1]

Adoption 84% of developers using or planning to use AI tools[2]

Trust Gap Only 33% trust AI accuracy (declining trend)[3]

Key Challenge 45% frustrated by "almost right" code requiring debugging[3]

Demo

Live Demo: Foundation of AI Agents

- Without Tools: Asking ChatGPT API "What's the weather in San Francisco?" results in "I can't access real-time data..."
- With Tools: Adding a tool definition lets the LLM request tool execution, we execute the weather API call and send results back to the LLM. The LLM then provides a natural language response with up-to-date real data.
- Key Insight: This tool use capability is the foundation that makes AI agents possible, enabling autonomous multi-step workflows beyond single prompt-response.

Comparison

LLMs vs AI Agents

Large Language Models (LLMs)

- Text generation engines focused on producing responses from single requests.
- Operate as single request → single response systems without persistent state or memory.
- No ability to take actions or execute multi-step workflows independently.
- Common examples: ChatGPT conversation, Claude chat interface, GitHub Copilot code suggestions.
- Best suited for code generation, explanations, refactoring suggestions, and documentation.

"Here's how to implement a password validator"

AI Agents: The Next Evolution

- Autonomous systems that use LLMs plus tool use to perform multi-step work.
- Can break down complex tasks into smaller steps and execute actions independently.
- Maintain persistent memory or state across interactions to adapt and handle errors.
- Examples include agents that implement features, create tests, and verify functionality without constant user prompts.

"I've implemented the password validator, created tests, and verified it works"

Primary Tools We'll Use

GitHub Copilot	Provides inline code suggestions in your IDE, offers chat interface for explanations, and delivers context-aware completions. Integrates smoothly with VS Code, JetBrains, and others.
Claude Code	Command-line AI coding assistant capable of autonomous task execution, git-aware file operations, and understanding multi-file context for complex workflows.
Conversational AI	Includes ChatGPT and Claude web interfaces suited for general tasks, learning, and exploration, providing versatile conversational capabilities.
Alternative Editors	Tools like Cursor, an AI-first code editor built on VS Code, and other AI-integrated IDEs designed to enhance coding productivity with AI features. (Optional) Use an alternate tool if you are already comfortable

Capabilities

Understanding Current Strengths

Code Generation

Generative AI effectively produces boilerplate and common code patterns, implements algorithms, integrates APIs, and creates tests, accelerating development workflows.

Code Understanding

AI tools identify bugs, suggest performance optimizations, detect security vulnerabilities, and provide code explanations and documentation to support developers.

Transformation Tasks

AI assists with language migration, framework updates, pattern refactoring, and ensuring style consistency, enabling smooth evolution of codebases.

Understanding Limitations

Technical Constraints

- Limited context windows restrict memory from 4K to 200K tokens, causing potential loss of relevant information in large codebases.
- Reasoning boundaries lead to struggles with novel algorithms, hallucination of APIs or syntax, and limited mathematical proof capabilities.
- Domain specificity issues reduce reliability on niche frameworks, cause mixing of library versions, and weaken performance on proprietary systems.

Practical Considerations

- Generative AI is not a replacement for domain expertise, architectural decisions, security auditing, or performance profiling.
- Human oversight is required for production deployments, handling sensitive data, critical system changes, and legal or compliance-related code.
- Users must verify and understand AI outputs because AI can confidently produce incorrect or 'almost right' code that requires debugging.

Live Demo: Password Validator Comparison

01

GitHub Copilot

- Highly integrated in VS Code
- Code completion, ask mode, or agent mode

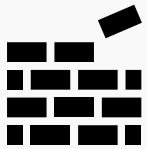
02

Claude Code

- Command line tool
- Works with any editor

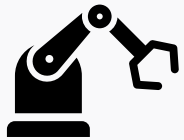
Summary

Key Takeaways



Generative AI as a Tool

Generative AI is a powerful tool that amplifies your capabilities but is not a replacement. It requires understanding to be used effectively.



LLMs vs AI Agents

LLMs are for guidance and generation with single responses, while AI Agents perform autonomous multi-step task completion with memory and action capabilities.

Start Simple, Scale Up

Begin with basic prompts and practice. Gradually graduate to complex agent workflows to unlock full potential.

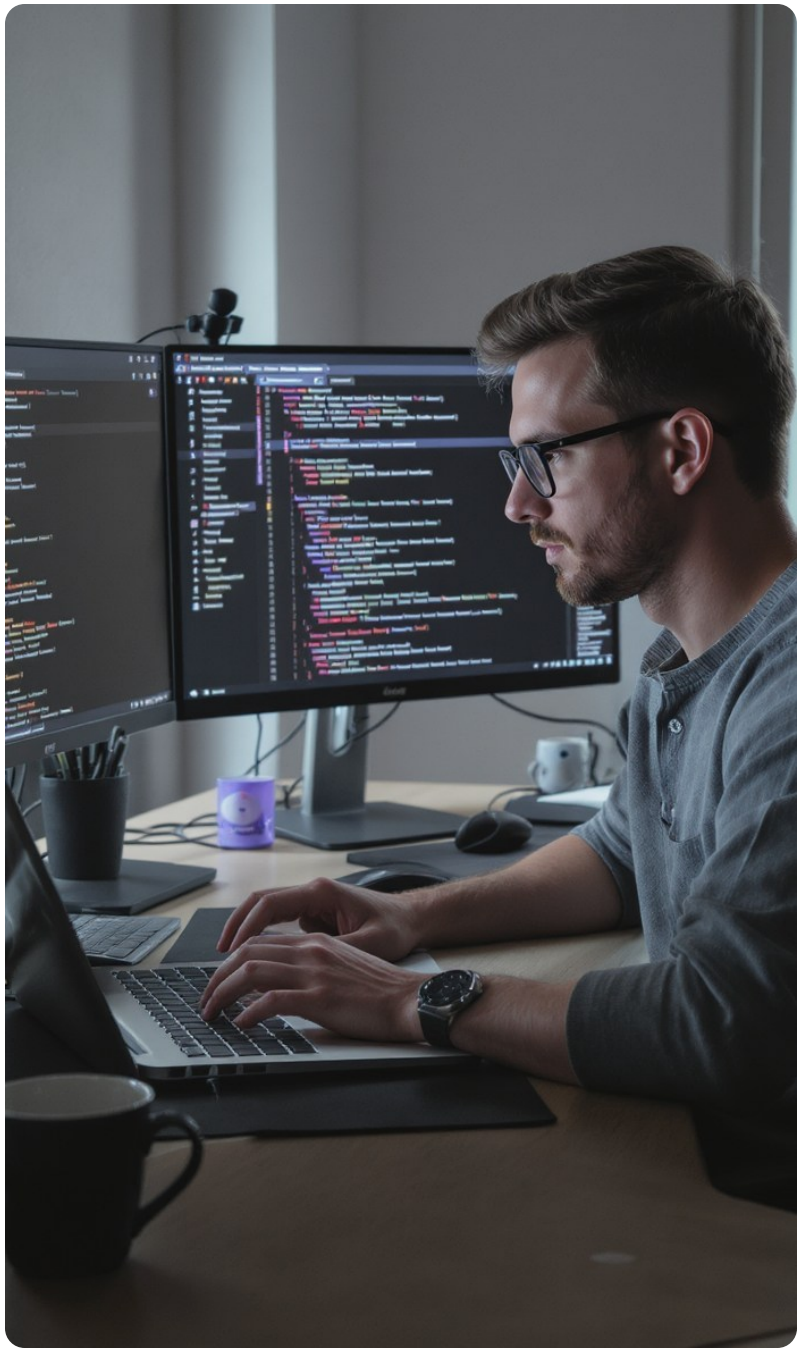
Always Verify and Understand

AI can be confidently wrong. Your expertise is essential to verify outputs, ensuring correctness and reliability.

Practice

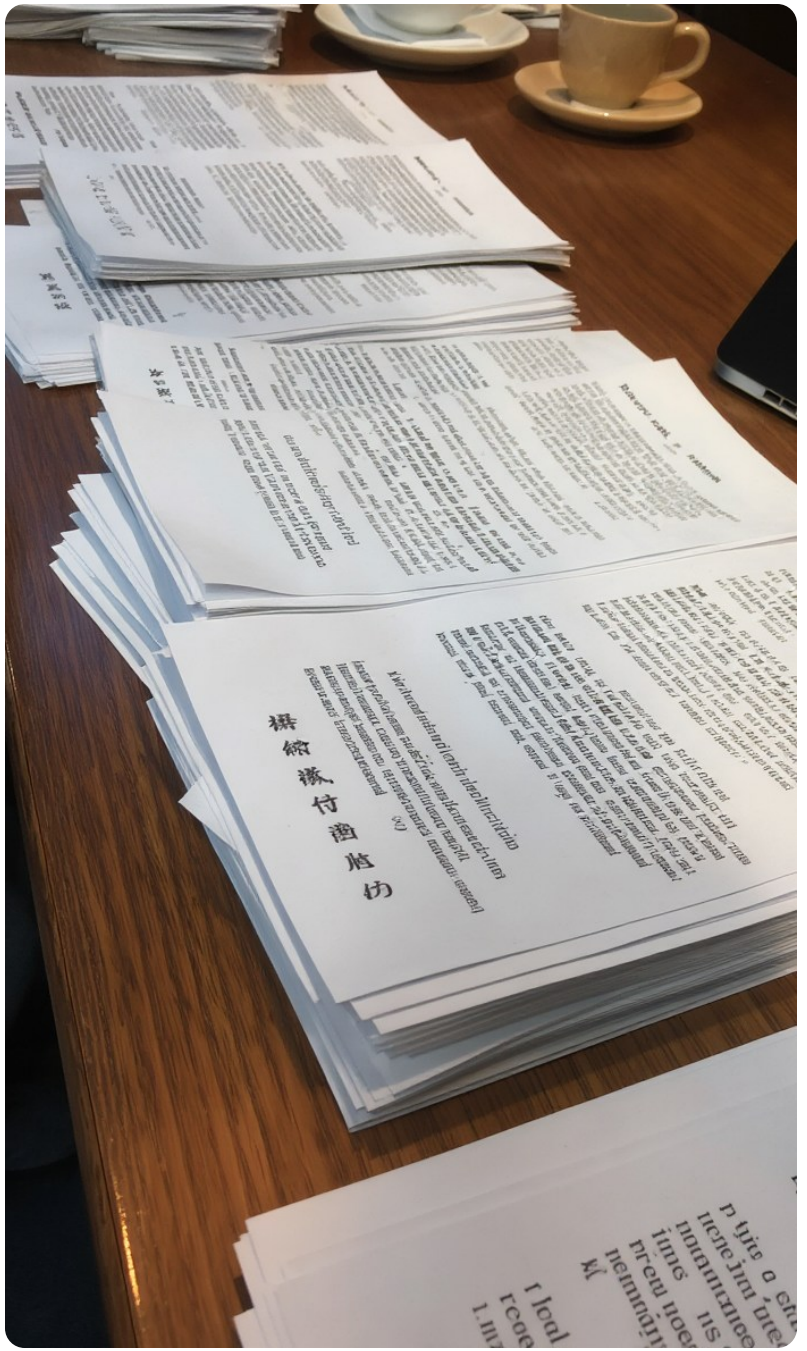
Next Up: Hands-On Practice Preview

- Create a currency formatter utility function to practice AI-assisted coding techniques.
- Learn effective prompt engineering techniques to communicate clearly with AI tools.
- Discover common pitfalls and solutions encountered when working with AI in coding projects.
- Prepare for hands-on experience immediately following this session to solidify your understanding.



References

1. Microsoft/MIT/Wharton Study (2024)
   "Measuring GitHub Copilot's Impact on Productivity" - Communications of the ACM
   https://cacm.acm.org/research/measuring-github-copilots-impact-on-productivity/
2. GitHub Copilot Research
   "Research: Quantifying GitHub Copilot's impact in the enterprise with Accenture"
   https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-in-the-enterprise-with-accenture/
3. Stack Overflow Developer Survey (2024)
   "Stack Overflow Annual Developer Survey"
   https://survey.stackoverflow.co/2024/



Unifi by Intertech

Powered by AI. Guided by You.

Session 1 Part B:

Getting Started with Prompt Engineering

Ryan Harvey & Brian Hackerson

August 14th, 2025

What is Prompt Engineering?

Definition

Prompt Engineering is the art and science of crafting instructions that get AI models to produce the desired output consistently and effectively.

Why It Matters

Quality: Better prompts equal better results. [1]

Efficiency: Clear instructions reduce back-and-forth. [1]

Reliability: Consistent prompts produce consistent outputs. [2]

Cost: Fewer iterations mean lower API costs. [1]

Basic Prompt Structure

The CLEAR Framework [3]

Context	Set the scene to provide background information that frames the task for the AI.
Language	Specify the programming language and framework to ensure relevant code generation.
Example	Show what you want with an example if it helps clarify the request.
Action	Clearly state what you want the AI to do with the prompt.
Requirements	List constraints and specifications to guide the AI's output precisely.

Example: Poor vs Good Prompt

❌ Poor

Make a function

Lacks context about the project or the function's purpose

No specified programming language or framework

No clear action or detailed requirements

Results in vague or unusable output from AI models

☑ Good

I'm building a user authentication system with Python with FastAPI. Create a password validation function.

Requirements:

- Check minimum 8 characters
- Require uppercase, lowercase, number, special character if less than 14 characters
- Return boolean and detailed feedback
- Include type hints

Provides detailed instructions to guide AI for precise and consistent output

Best Practices for Code Generation

Be Specific About Requirements

Avoid vague prompts like

'Create a sorting function.'

Instead, specify details such as

'Create a merge sort function that handles integer arrays, includes error handling for empty arrays, and runs in O(n log n).'

Specify Your Tech Stack

Clearly define language, framework, testing, and styling.

Example: TypeScript, React with Vite, Vitest, Tailwind CSS.

This ensures AI generates compatible code.

Include Context About Your Project

Provide project context to guide AI output. For example, specify if the function supports a high-performance e-commerce checkout handling thousands of users.

Common Prompt Patterns for Developers

Code Generation Pattern

Create a [TYPE] in [LANGUAGE] that [FUNCTIONALITY].

Requirements:
- [REQUIREMENT 1]
- [REQUIREMENT 2]
- [REQUIREMENT 3]

Include:
- error handling
- type annotations/hints
- basic tests

Code Review Pattern

Review this [LANGUAGE] code for
- security vulnerabilities
- performance issues
- best practices violations
- potential bugs.

[CODE HERE]

Provide specific suggestions with examples.

Debugging Pattern

I'm getting this error: [ERROR MESSAGE].

Code context: [CODE].

Expected behavior: [DESCRIPTION].

Actual behavior: [DESCRIPTION].

Help me identify and fix the issue.

Your First AI-Assisted Coding Session

01

Step 1: Choose Your Tool

Select an AI coding assistant that fits your workflow.

- GitHub Copilot for IDE integration
- ChatGPT or Claude via web interfaces
- Claude Code from the command line.

02

Step 2: Start Simple

Begin with a clear, specific request for a utility function. Use precise prompts with defined requirements to get effective code generation.

03

Step 3: Iterate and Refine

Review the generated code thoroughly. Ask for modifications, add missed requirements, and test the code to validate its correctness and performance.

Exercise

Generate a Utility Function

section1/03_currency_formatter_exercise

- Create a utility function named formatCurrency to format numbers as currency strings.
- Handle different currencies (USD, EUR, GBP) and support different locales.
- Manage edge cases such as null, undefined, and very large numbers with proper error handling.
- Add comprehensive tests to ensure reliability.

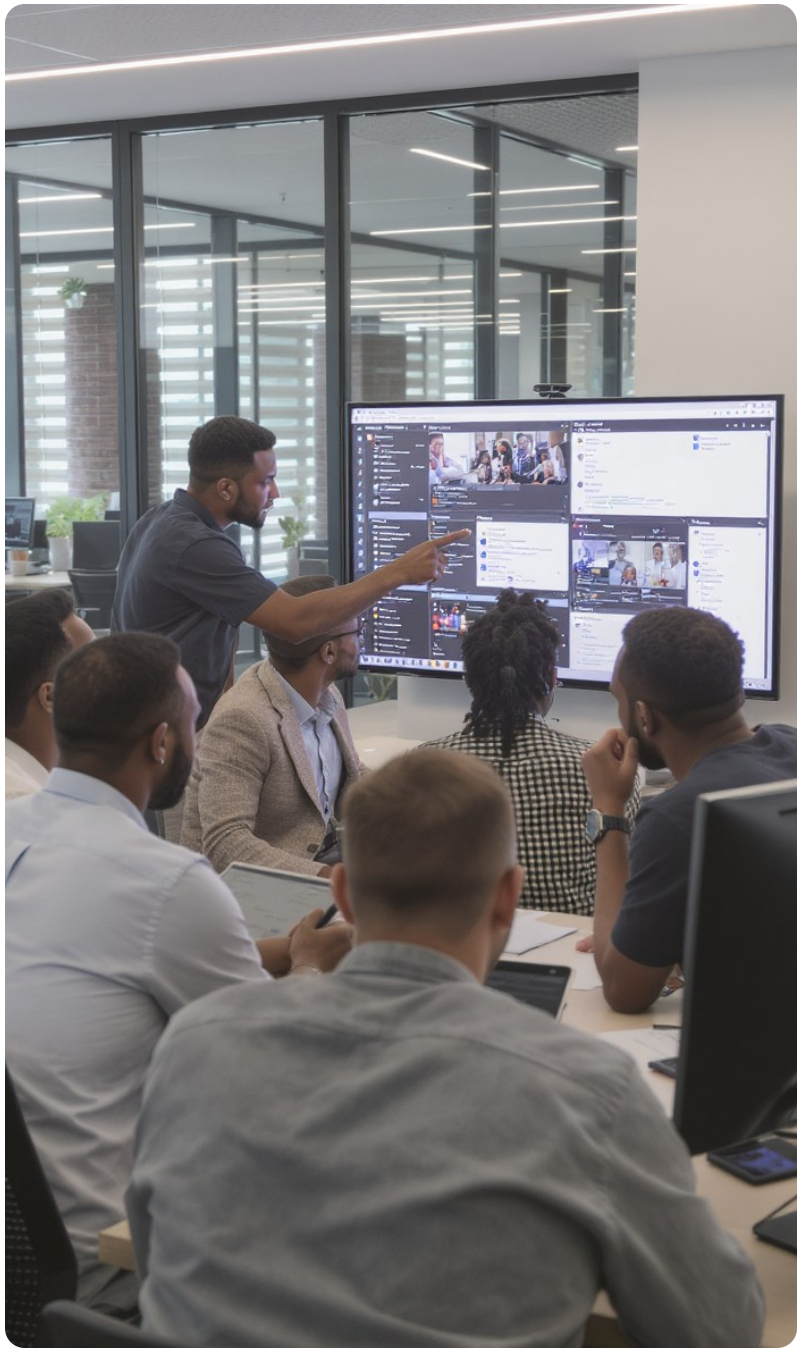
- Use the CLEAR framework to write your prompt, generate code with your AI tool, review, iterate, and test the results.

Prompt Sharing

Let's See Your Prompts!

One or Two Volunteers

- Share the prompt structure you used, focusing on how you applied the CLEAR framework components:
  - Context, Language, Action, and Requirements.
- Discuss the challenges you faced while writing prompts, such as specifying constraints, including relevant context, or handling edge cases.
- Explain how you iterated and improved your prompts based on the AI's output and feedback from testing.
- Note the importance of specificity and clear instructions to achieve better AI-generated code results.
- Reflect on how comparing prompts with peers can reveal different techniques and prompt strategies.



Common Pitfalls and How to Avoid Them

1. Being To Vague

❌ "Make it better"

✅ "Optimize for performance by reducing database queries and adding caching"

2. Not Specifying Constraints

❌ Getting overly complex solutions

✅ "Keep it simple, under 50 lines, minimal dependencies"

3. Ignoring Error Handling

❌ Getting happy-path-only code

✅ "Include comprehensive error handling and input validation"

Working with AI Responses

01

Always Review Generated Code

Understand what the code does before using it. This prevents unexpected behaviors and ensures the code aligns with your needs.

02

Test with Your Specific Use Cases

Run the generated code through your project's real scenarios to verify it works as expected and handles edge cases.

03

Verify It Meets Your Requirements

Check that the output fulfills all specified requirements including performance, error handling, and coding standards.

04

Modify and Iterate as Needed

Use the AI-generated code as a starting point. Refine and enhance the code through multiple iterations to improve quality and fit.

Summary

Key Takeaways

Structure Matters

Use frameworks like CLEAR to organize prompts with Context, Language, Example, Action, and Requirements for better results.

Start Simple, Iterate

Begin with basic functionality, then gradually add complexity through iterative refinement and continuous testing.

AI is a Collaborator

Review all generated code carefully, combining AI speed with your expertise to enhance quality and reliability.

Practice Makes Perfect

The more you prompt, the better you get. Build a prompt library over time and learn from different tools' strengths.

Discussion

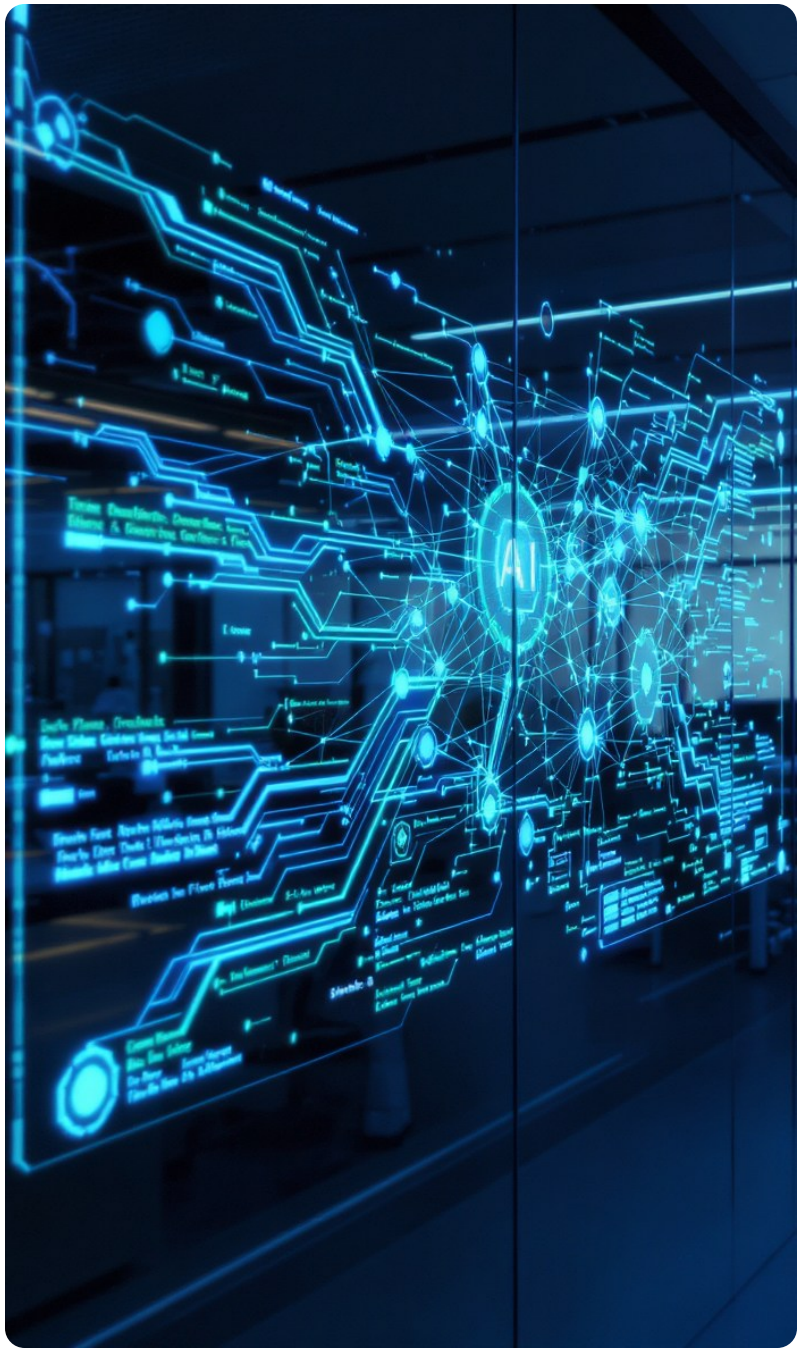
Questions & Discussion

- What surprised you about AI-assisted coding?
- What challenges did you encounter?
- How do you see this fitting into your workflow?
- Share the prompt structures you used and how you iterated to improve them.
- Discuss how different AI tools influenced your coding experience.

Next Steps

What's Next?

- Clarity and specificity in technical prompts
- Context setting and role assignment
- Few-shot prompting with examples
- Production XML-tagged templates



References

1. OpenAI Best Practices (2024)

   OpenAI Documentation - Best Practices for Prompt Engineering

   https://platform.openai.com/docs/guides/prompt-engineering

2. GitHub Copilot Documentation (2024)

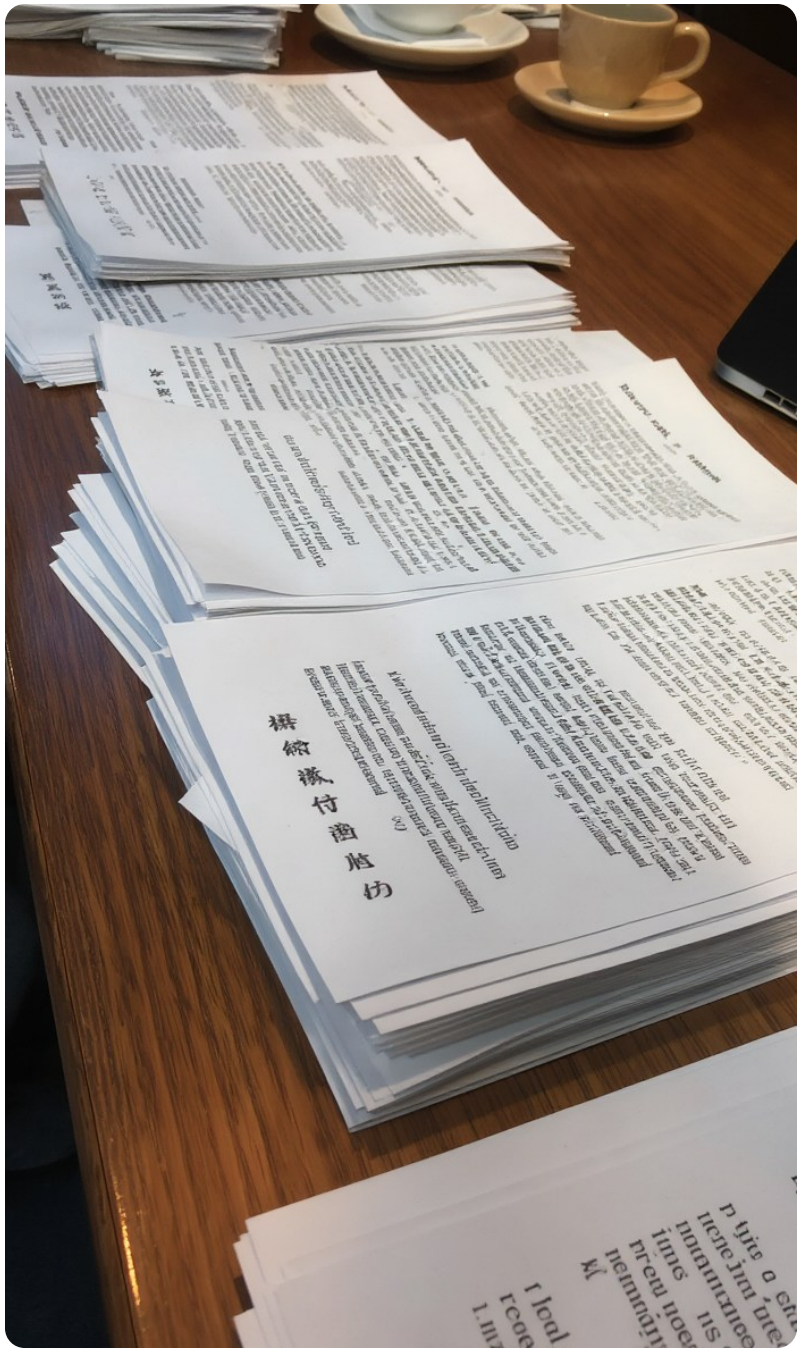
   GitHub Copilot Documentation - Getting Started

   https://docs.github.com/en/copilot/getting-started-with-github-copilot

3. Anthropic's CLEAR Framework (2024)

   Anthropic Documentation - Prompt Engineering Interactive Tutorial

   https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering



Unifi by Intertech

Powered by AI. Guided by You.

Session 2 Part A:

Effective Prompt Design

Ryan Harvey & Brian Hackerson

August 14th, 2025

Why Advanced Prompting Matters

Increase Accuracy

Well-designed prompts reduce AI hallucinations[1], leading to more reliable and precise outputs that minimize errors and improve trustworthiness.

Save Time

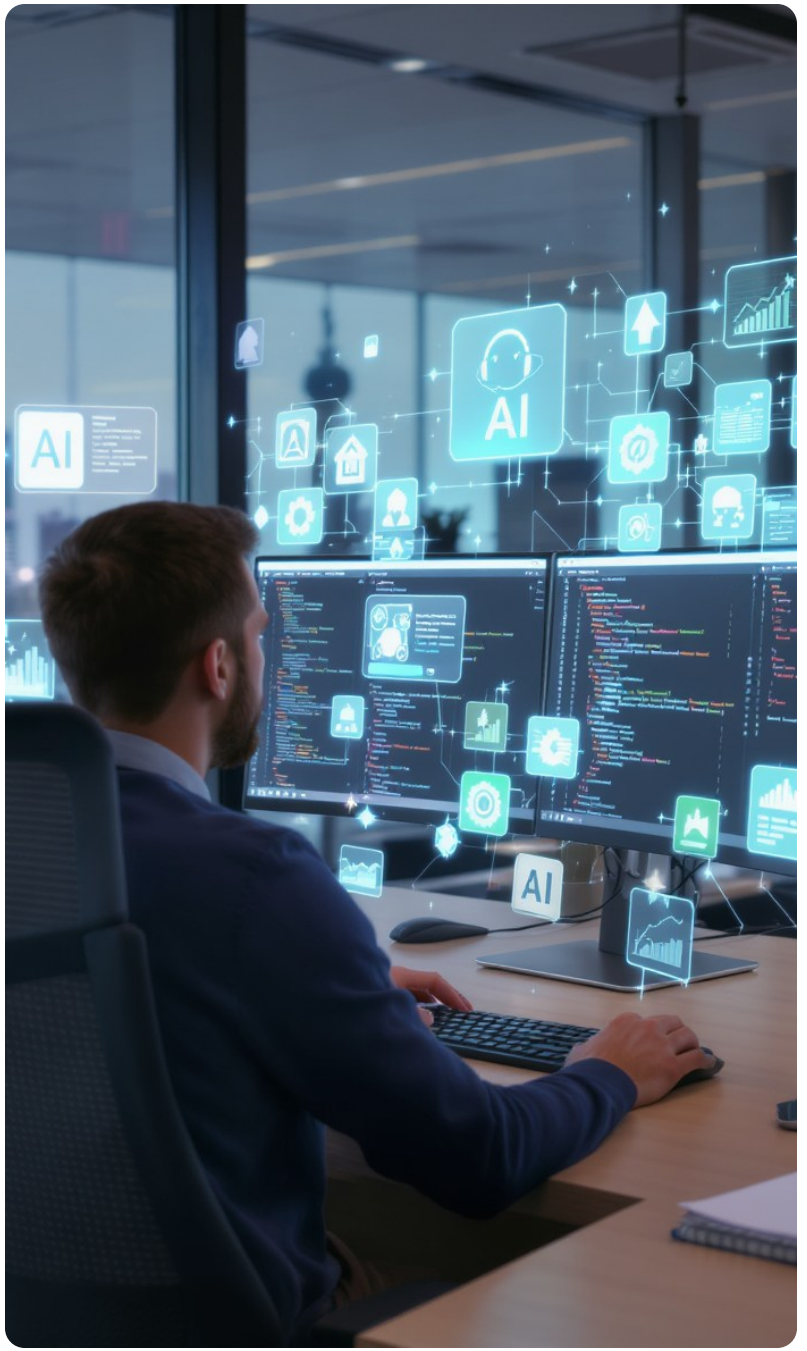
Advanced prompts help get better results on the first try, reducing the need for repeated queries and extensive debugging, thus accelerating development.

Scale Quality

Using consistent prompting patterns ensures reliable outputs[2] across multiple tasks and projects, supporting scalability in development workflows.

Enable Complexity

Sophisticated prompt design allows AI to handle complex development tasks effectively, supporting advanced use cases and intricate system requirements.



Overview

What We'll Cover

01

Clarity and Specificity

Writing precise technical prompts to increase accuracy and reduce AI hallucinations. Focus on detailed, exact instructions tailored to your development needs.

02

Context Setting

Providing the right background information about your technical environment, business context, and constraints to transform generic AI responses into production-ready solutions.

03

Role Assignment

Leveraging AI personas such as Senior Software Architect or Security-Focused Developer to align AI output with expert knowledge, improving relevance and performance.

04

Few-Shot Prompting

Learning from examples by showing AI specific patterns and code snippets, which reduces interpretation errors and eliminates the need for fine-tuning.

05

Production Templates

Using structured XML-tagged prompt templates to maintain consistency and clarity across complex tasks, reducing parsing errors and streamlining output.

Foundation: Clarity and Specificity

01

The Specificity Spectrum

❌ Vague: "Fix my database code."

⚠️ Better: "Optimize my SQL query for better performance"

☑️ Precise:

Optimize this PostgreSQL query

that's taking 2.3 seconds to execute:

- Table has 50K users, 200K orders
- Need to find top 10 customers by total order value this year
- Currently missing indexes
- Return customer name, email, and total spent
- Maintain data accuracy and add proper error handling

02

Technical Precision Patterns[3]

Create a [COMPONENT TYPE]
in [LANGUAGE/FRAMEWORK] that [SPECIFIC FUNCTIONALITY]

Technical Requirements:

- [PERFORMANCE SPECS]
- [DEPENDENCIES/CONSTRAINTS]
- [ERROR HANDLING APPROACH]
- [TESTING REQUIREMENTS]

Context: [HOW IT FITS IN YOUR SYSTEM]

03

Benefits of Specificity

Specific prompts reduce AI hallucinations, save time with better first results, scale quality with patterns, and enable complex tasks. Clarity is key for effective AI-assisted development.

Context Setting: The Critical Foundation

Without Context

Create a user authentication function → Generic login function

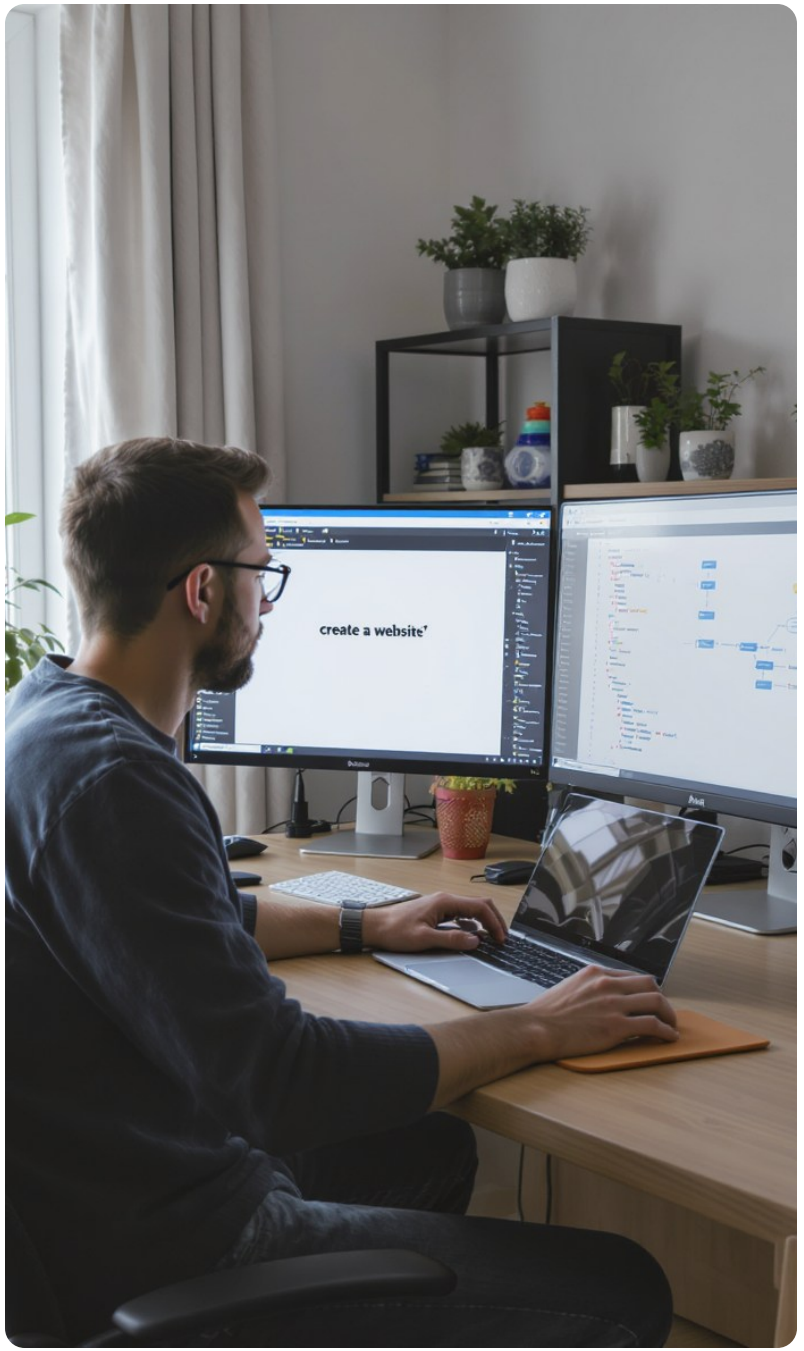
Lacks details on scale, architecture, or specific tech. Results in a basic, non-optimized solution that may miss critical requirements like security or performance.

With Context

Create user authentication for a high-traffic e-commerce API (10K requests/min) using JWT tokens with in-memory session caching, integrating with our existing PostgreSQL user table → Production-ready solution with specific architecture

Includes specific architecture and tech stack. Considers performance targets and integration needs

Delivers tailored, robust, scalable implementation.



The Context Framework

Technical Environment

Tech Stack: Node.js + Express + PostgreSQL

Current Architecture: Microservices with Docker

Testing: Jest + Supertest

Deployment: AWS ECS with CI/CD

These details help define the software, tools, and infrastructure context for accurate prompt responses.

Business Context

Application: E-commerce checkout system

Scale: 50K daily active users

Critical Path: Payment processing pipeline

Performance Target: <200ms response time

Business context guides AI to tailor solutions aligned with real-world operational goals.

Constraints and Requirements

Must integrate with: Stripe API, existing user service

Security: PCI compliance required

Legacy: Works with Node.js 18+

Budget: Minimal new dependencies

Clearly stating constraints prevents over-engineering and ensures practical, compliant outputs.

Project-Level Context Files

Both GitHub Copilot and Claude Code support project-level context files that automatically provide context to every AI interaction:

- GitHub Copilot: .github/copilot-instructions.md
- Claude Code: CLAUDE.md (in project root)

Benefits

- Consistency: Every team member gets the same context
- Onboarding: New developers inherit project knowledge
- Evolution: Update one, applies everywhere
- No Repetition: Stop explaining your tech stack in every prompt

Example

```
# Technical Environment
Tech Stack: Node.js + Express + PostgreSQL
Current Architecture: Microservices with Docker
Testing: Jest + Supertest
Deployment: AWS ECS with CI/CD

# Business Context
Application: E-commerce checkout system
Scale: 50K daily active users
Critical Path: Payment processing pipeline
Performance Target: <200ms response time

# Constraints and Requirements
Must integrate with: Stripe API, existing user service
Security: PCI compliance required
Legacy: Works with Node.js 18+
Budget: Minimal new dependencies
```

Role Assignment: Leveraging AI Expertise

The Power of Persona

AI models perform better when given specific expert roles that match their training data patterns. Research shows 15-30% performance improvement with expert role assignment.[4]

Effective Developer Personas

Senior Software Architect: Act as a senior software architect with 15 years experience designing scalable web applications. Focus on system design best practices, performance and scalability considerations, technology trade-offs, and code maintainability.

Security-Focused Developer: Prioritize OWASP Top 10 compliance, input validation,authentication, and secure coding practices.

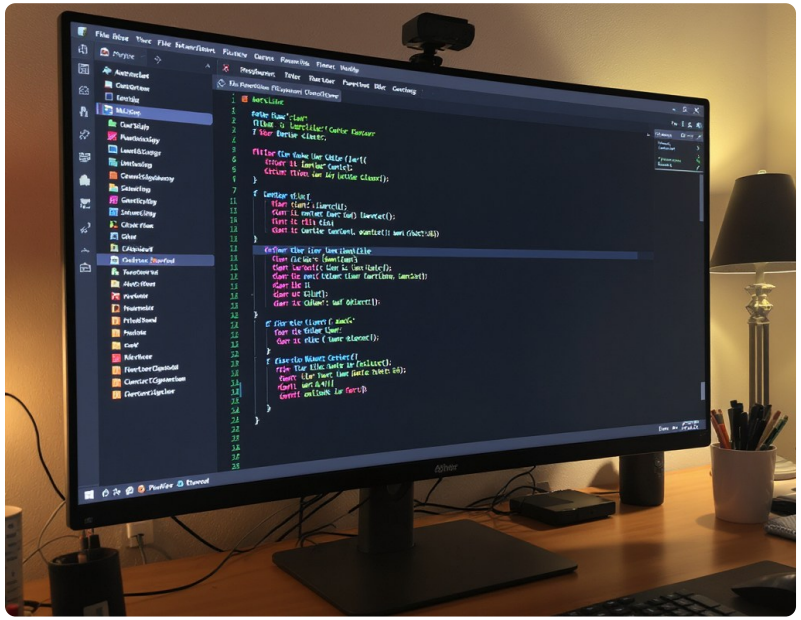
DevOps Engineer: Focus on infrastructure as code,CI/CD pipeline optimization, monitoring, and cost optimization.

Pro Tip: Claude Code's subagents let you create and use multiple specialized personas simultaneously - delegate security reviews to a security-focused agent while another handles performance optimization.

Few-Shot Prompting
Learning from Examples

The Pattern: Show, Don't Just Tell

Instead of describing what you want, show examples of the desired pattern. Few-shot learning often removes the need for fine-tuning.[5]



Example: API Error Handling Pattern

Validation Error:

```javascript
if (!email || !isValidEmail(email)) {
  return res.status(400).json({error: 'INVALID_EMAIL'});
}
```

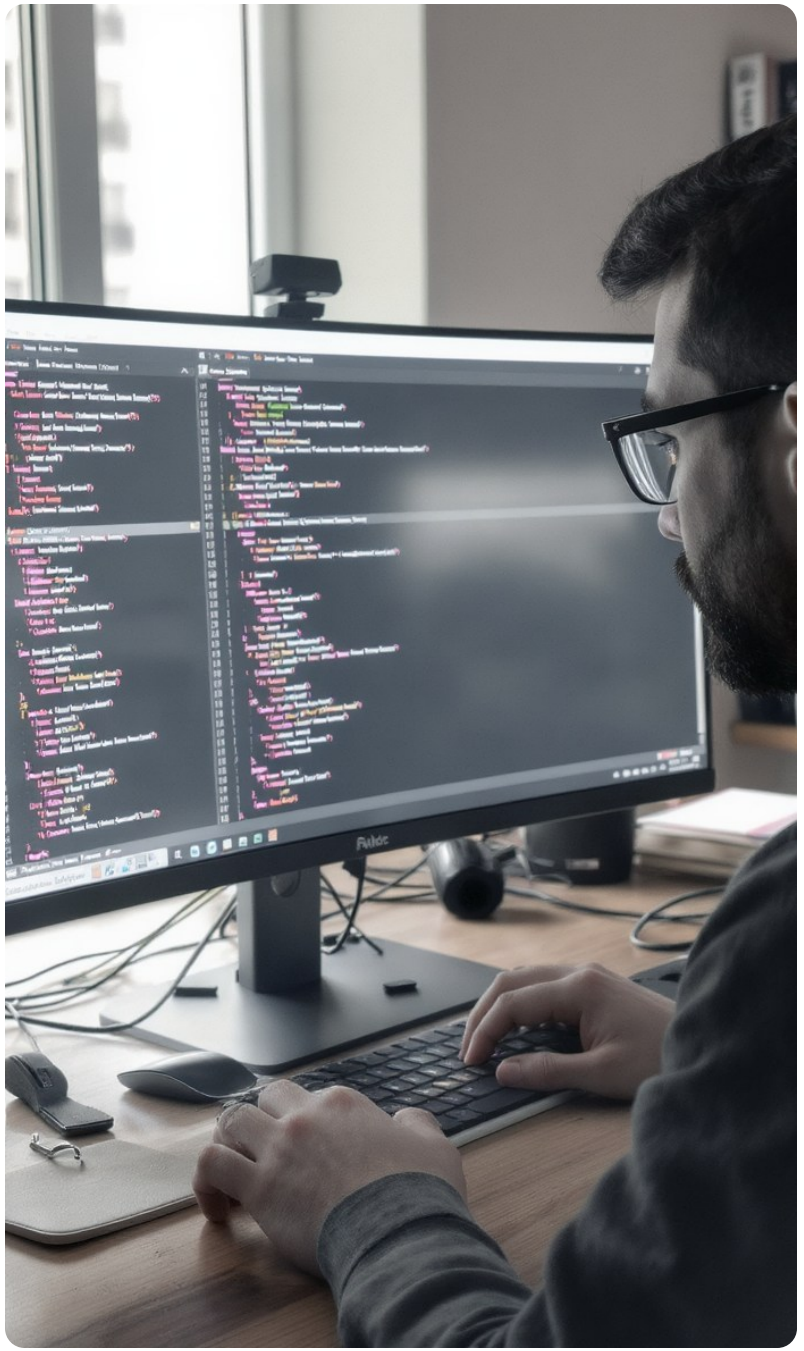
Database Error:

```javascript
try { const user = await User.findById(userId); } catch (error) {
```

Progressive Few-Shot Building

- Start Simple: Begin with 1-2 examples demonstrating basic patterns to establish a clear foundation for AI learning.
- Add Complexity: Introduce 3-4 examples including edge cases to cover a broader range of scenarios and improve robustness.
- Show Variations: Provide examples of different situations and responses to help the AI generalize the pattern effectively.
-

Performance scales with the number of examples provided.[5]

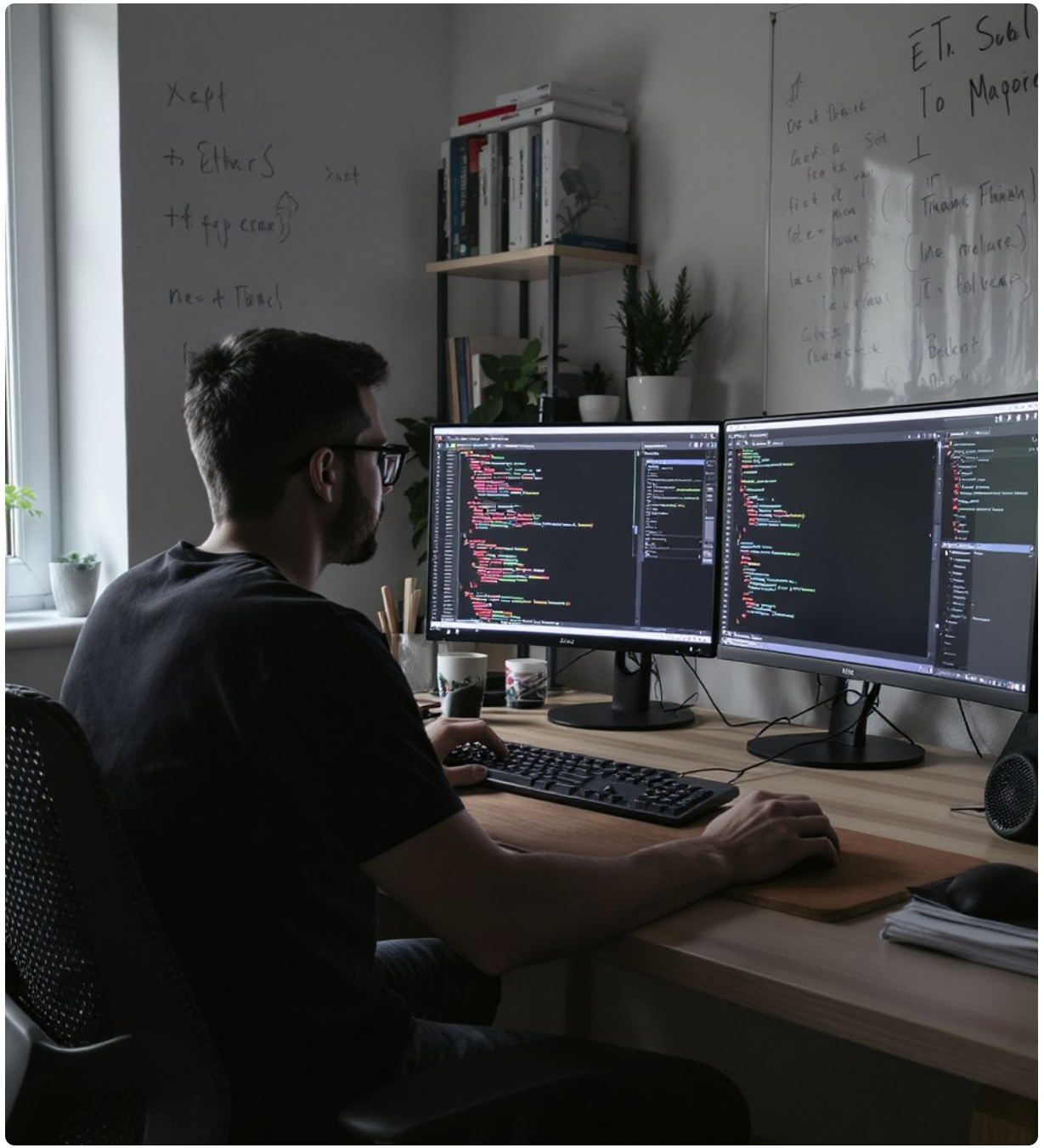


Demo

Comparing Prompt Approaches

Basic vs Advanced Prompt for Rate Limiter

Let's see the difference in output quality!



Production-Ready XML Templates

Why XML Tags Work

XML-style tags help AI understand structure and maintain consistency. Structured formats reduce parsing errors by 90%.[6]

<context>

Defines your application and technical environment, providing essential background to guide AI's responses effectively.

<role>

Specifies the expert persona for the AI, such as a senior developer or security engineer, leveraging AI training data patterns for improved output.

<action>

Describes the specific task you want the AI to accomplish, clearly framing the objective to avoid ambiguity.

<requirements> & <constraints>

Lists specific requirements and technical limitations, including performance targets, dependencies, and integration needs to ensure prompt precision.

Production Template: Code Generation

```
<context>
Building a React e-commerce application with TypeScript, using Next.js 14,
TailwindCSS, and Prisma with PostgreSQL. Current component follows
atomic design patterns.
</context>

<role>
Act as a senior React developer specializing in e-commerce applications
with expertise in TypeScript, accessibility, and performance optimization.
</role>

<task>
Create a product search component with real-time filtering
</task>

<requirements>
- TypeScript with strict types
- Debounced search (300ms)
- Filter by category, price range, rating
- Accessible keyboard navigation
- Mobile-responsive design
- Loading states and error handling
- Integration with existing Prisma queries
</requirements>

<constraints>
- Must use existing design system components
- Performance target: <100ms filter response
- Compatible with React 18+ concurrent features
- Zero external dependencies beyond current stack
</constraints>

<output_format>
Provide:
1. Component code with full TypeScript types
2. Associated hook for data fetching
3. Basic unit tests with React Testing Library
4. Usage example with proper error boundaries
</output_format>
```

Domain-Specific Prompting for Code Generation

Database Development

```
<role>
Act as a database architect with PostgreSQL expertise
</role>

<context>
E-commerce application with 100K+ products, complex inventory
tracking,
multi-tenant architecture with row-level security.
</context>

<task>
Create optimized queries for product search with faceted
filtering
</task>

<requirements>
- Full-text search across name, description, tags
- Filter by: category, price, availability, brand, rating
- Sort options: relevance, price, popularity, newest
- Pagination with cursor-based approach
- Performance target: <50ms query time
</requirements>
```

DevOps / Infrastructure

```
<role>
Act as a DevOps engineer specializing in AWS
and Kubernetes
</role>

<context>
Microservices architecture with 12 services, deployed on
EKS, using Terraform for infrastructure, ArgoCD for
deployments.
</context>

<task>
Create monitoring and alerting for critical user journeys
</task>

<requirements>
- Prometheus + Grafana stack
- SLI/SLO based alerting
- Business metrics (conversion rates, revenue)
- Infrastructure metrics (CPU, memory, network)
- Integration with existing Slack notifications
</requirements>
```

Exercise

Focused Prompt Design

The Challenge

Generate a TypeScript in-memory caching class with automatic expiration and TTL in milliseconds, supporting generics and essential cache operations.

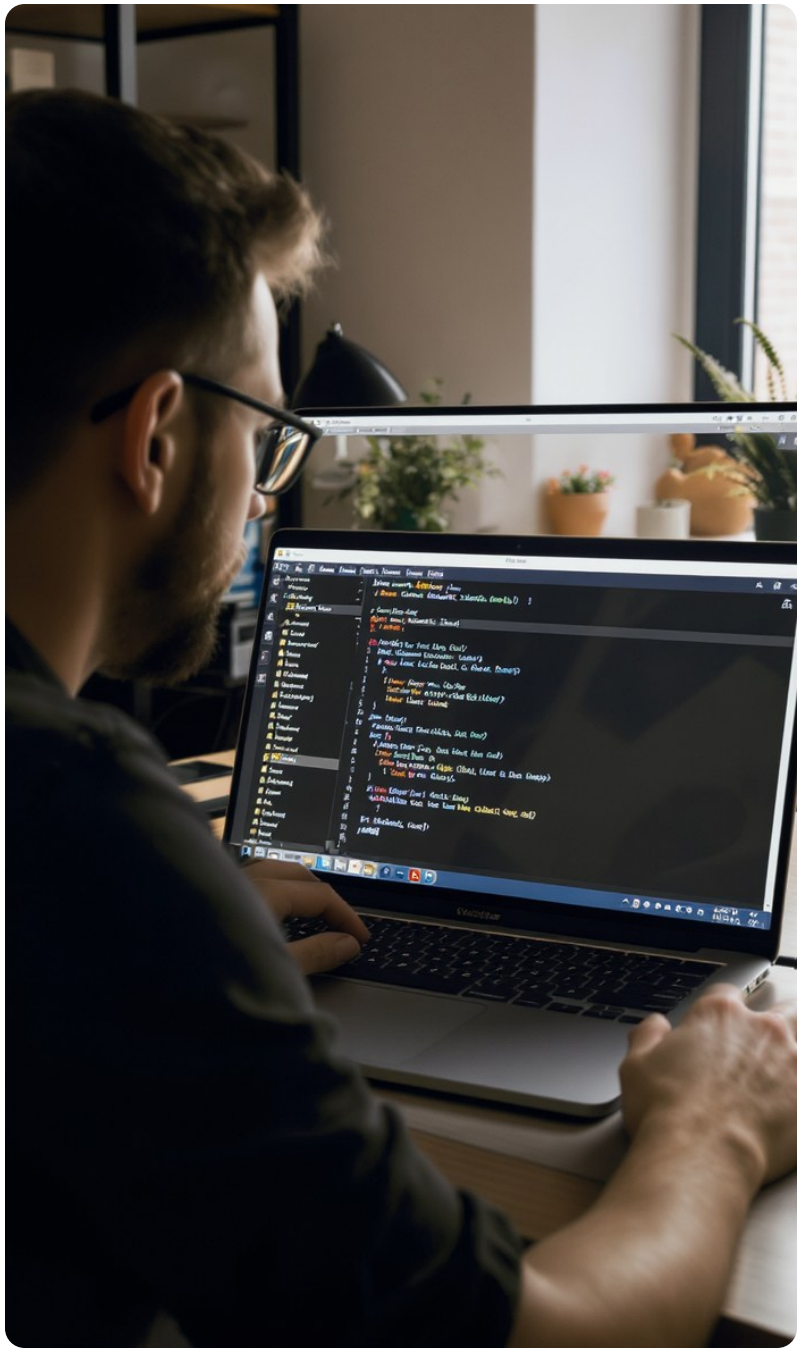
Your Task (30 minutes)

Write a prompt specifying TypeScript with generics, cache operations (get/set/delete/has/clear/size), and TTL. Submit to AI, review code, run tests, and fix timer cleanup issues.

Success Criteria

Prompt should generate code with:

- A generic Cache<T> class
- Get/Set/Delete/Has/Clear/Size operations
- Automatic expiration (TTL with setTimeout)
- TypeScript type safety
- Proper timer cleanup
- All tests passing



Common Prompt Design Pitfalls

Context Overload

❌ Too Much: 500 words of background information

☑ Right Amount: Relevant technical details only

Assumption Traps

❌ Assuming: Make it production-ready

☑ Right Amount: Relevant technical details only

Single-Shot Complexity (*It Depends*)

❌ All at Once: Build complete authentication system with OAuth, MFA, admin panel...

☑ Iterative: Start with core auth, then add features step-by-step

Generic Roles

❌ Generic: Act as a programmer

☑ Specific: Act as a senior Python developer specializing in data processing pipelines

Advanced Patterns: Chaining and Templates

Sequential Prompt Chaining

Break complex tasks into clear steps:

1) Design the high-level architecture for a real-time chat system

2) Implement the WebSocket connection manager based on that architecture

3) Create comprehensive integration tests for the chat components.

Template Libraries

Build reusable prompt templates for common development tasks[7], such as Code Review, API Design, Database Schema, Security Audit, and Performance Optimization. These templates standardize requests and improve output consistency.

Pro Tip: Most coding agents enable reusable prompts

Claude Code: Custom slash commands
GitHub Copilot: Prompt files

Measuring Prompt Effectiveness

Good Prompt Results

- Generates code that runs without modification
- Includes proper error handling
- Follows specified patterns and conventions
- Addresses edge cases mentioned
- Provides clear, maintainable solutions

81% of developers cite these as indicators of successful prompting[9]

Poor Prompt Results

- Requires significant debugging
- Misses key requirements
- Uses outdated or incorrect patternLacks error handling
- Too generic for your specific use case

Iteration Strategies

- Start Broad
  - Begin with general requirements, refine based on outout
- Build Examples
  - Create few-shot examples or provide as context to the agent
- Document Patterns
  - Save effective prompt templates for reuse
- A/B Test
  - Try different approaches and compare results

Summary

Key Takeaways: Effective Prompt Design

01

Precision Beats Volume

Specific technical requirements trump lengthy descriptions. Clear constraints and structured input prevent misunderstandings and improve accuracy.

02

Context Is King

The technical environment shapes solutions, business needs drive architecture, and constraints prevent over-engineering. Including tech stack context reduces errors by 25%.[8]

03

Show, Don't Tell

Using examples clarifies expectations better than descriptions. Few-shot prompting teaches AI your preferred styles and reduces interpretation errors.

04

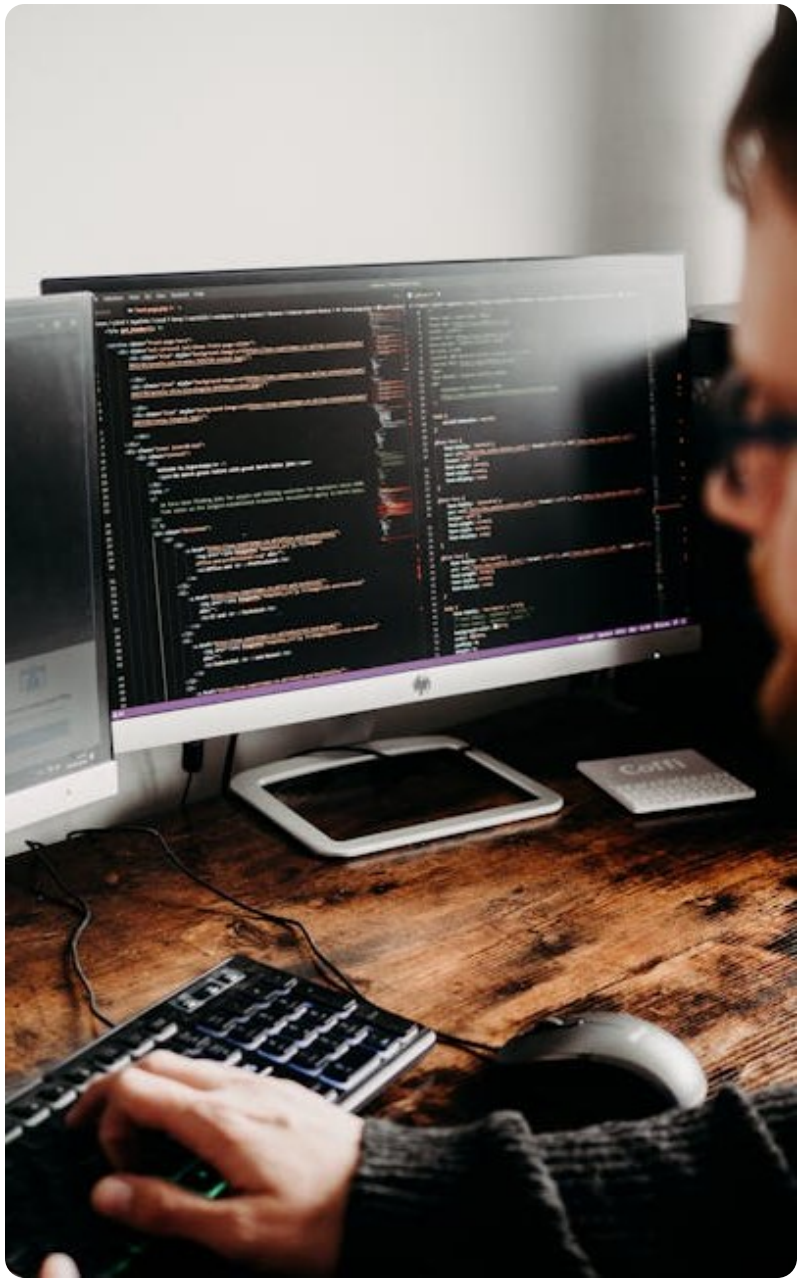
Professional Personas Work

Assigning expert roles leverages AI training data patterns. Specific expertise improves output quality, with expert role assignment boosting performance by 15-30%.[4]

Discussion

Questions & Discussion

- Which prompt design techniques felt most impactful to your workflow?
- What challenges did you encounter during your prompt development process?
- How do you see these patterns fitting into your ongoing development projects?
- Are you ready to explore Chain-of-Thought reasoning in the next part?



Next Steps

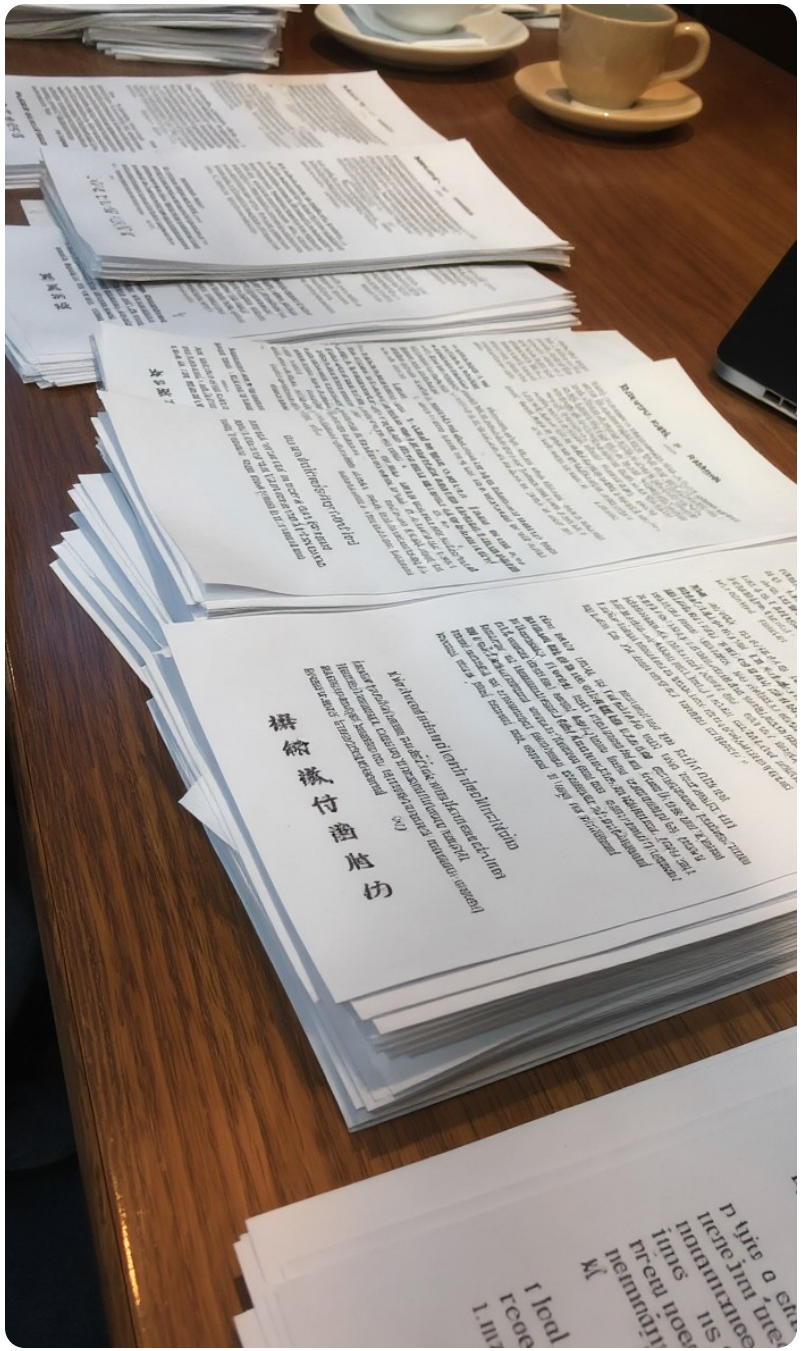
What's Next: Chain-of-Thought Reasoning

- Chain-of-Thought Prompting: Guide AI through complex reasoning steps to improve clarity and accuracy in outputs.
- When to Use CoT vs Zero-Shot: Understand model-specific strategies for choosing the right prompting approach.
- Advanced Debugging: Learn systematic problem-solving methods to efficiently identify and fix issues.
- Complex System Design: Break down architectural challenges into manageable parts for AI-assisted development.

Everything you've learned about clarity, context, and specificity becomes the foundation for guiding AI through sophisticated multi-step reasoning processes.

References

1. OpenAI Documentation - Prompt Engineering
   https://platform.openai.com/docs/guides/prompt-engineering
2. GitHub Engineering Blog - Structured Prompts (2024)
   GitHub Engineering Blog
   https://github.blog/engineering/
3. OpenAI Platform Documentation (2024)
   OpenAI Documentation - Best Practices
   https://platform.openai.com/docs/guides/prompt-engineering
4. Persona-Based Prompting Study (2023)
   The Impact of Role Assignment on Large Language Model Performance
   https://arxiv.org/abs/2303.08618
5. Brown et al. - Language Models are Few-Shot Learners (2020)
   Language Models are Few-Shot Learners
   https://arxiv.org/abs/2005.14165
6. Guidance: Structured Generation Research (2024)
   Guidance: A Language for Controlling Large Language Models
   https://github.com/guidance-ai/guidance

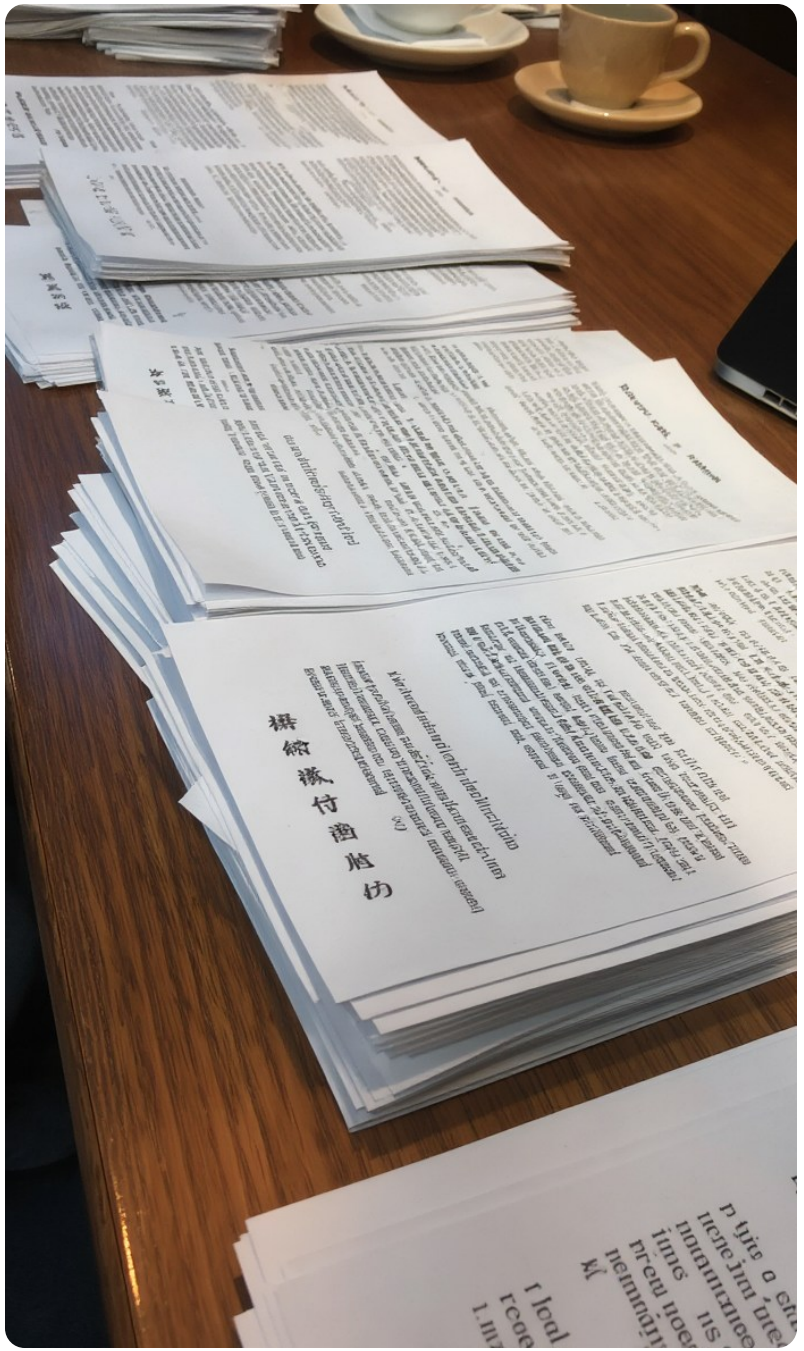


References

7. Anthropic Prompt Engineering Interactive Tutorial (2024)
   Anthropic Documentation
   https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering

8. CodeT5+ Paper (2023)
   CodeT5+: Open Code Large Language Models for Code Understanding and Generation
   https://arxiv.org/abs/2305.07922

9. Stack Overflow Developer Survey (2024)
   Stack Overflow Annual Developer Survey
   https://survey.stackoverflow.co/2024/



Unifi by Intertech

Powered by AI. Guided by You.

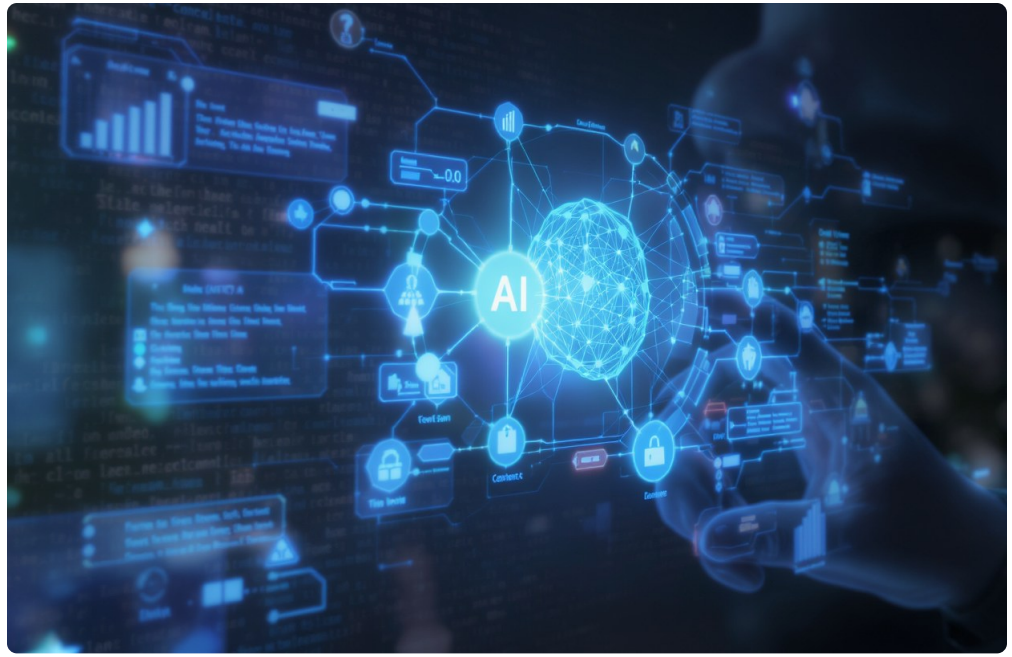
Session 2 Part B:

Advanced Prompting & Context Augmentation

Ryan Harvey & Brian Hackerson

August 14th, 2025

Advanced Prompting



You've learned about clarity, context, and structure. Now we'll explore two powerful techniques that dramatically improve AI accuracy:

1. **Chain-of-Though (CoT)**: Guide AI through reasoning steps

2. **Retrieval-Augemented Generation** (RAG): Provide relevant context dynamically

Chain-of-Thought (CoT)

Improves complex problem-solving by 35%+ on debugging tasks[1]

Retrieval-Augmented Generation (RAG)

Provides accurate domain-specific information not in training data[2]

Together

Create production-ready AI systems that are both smart and accurate

What We'll Cover

CoT Overview

When reasoning models make CoT less critical

RAG Fundamentals

Give AI the right information at the right time

Implementation Patterns

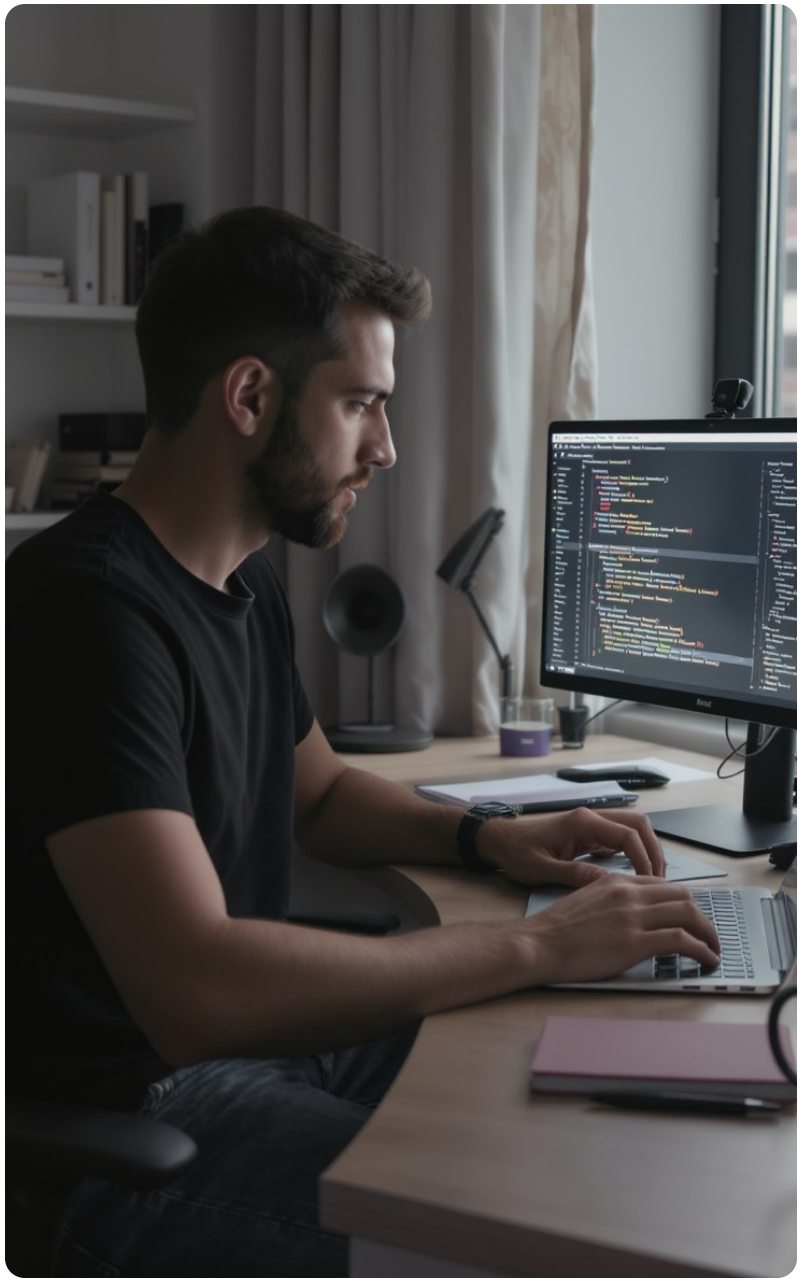
From simple keyword search to semantic retrieval

Practical Exercise

Master iterative prompt refinement

Cost-Performance Trade-offs

When to use each technique



CoT

Chain-of-Thought Overview

- Chain-of-Thought (CoT) guides AI with explicit step-by-step reasoning for complex problems.

- The Modern Reality (2025)
  - Reasoning models like OpenAI o3/o4-mini and Claude Sonnet 4 have built-in reasoning capabilities, making explicit CoT less critical.
  - CoT remains valuable for standard models, transparency, debugging, and cost-sensitive scenarios.
  - Modern reasoning models either require no prompting or can toggle 'reasoning mode' for automatic CoT.
  - Choosing CoT depends on task complexity, model capabilities, and cost-performance trade-offs.

Chain-of-Thought in Action: Debugging Example

Without CoT (Zero-shot)

Fix the authentication bug causing 500 errors

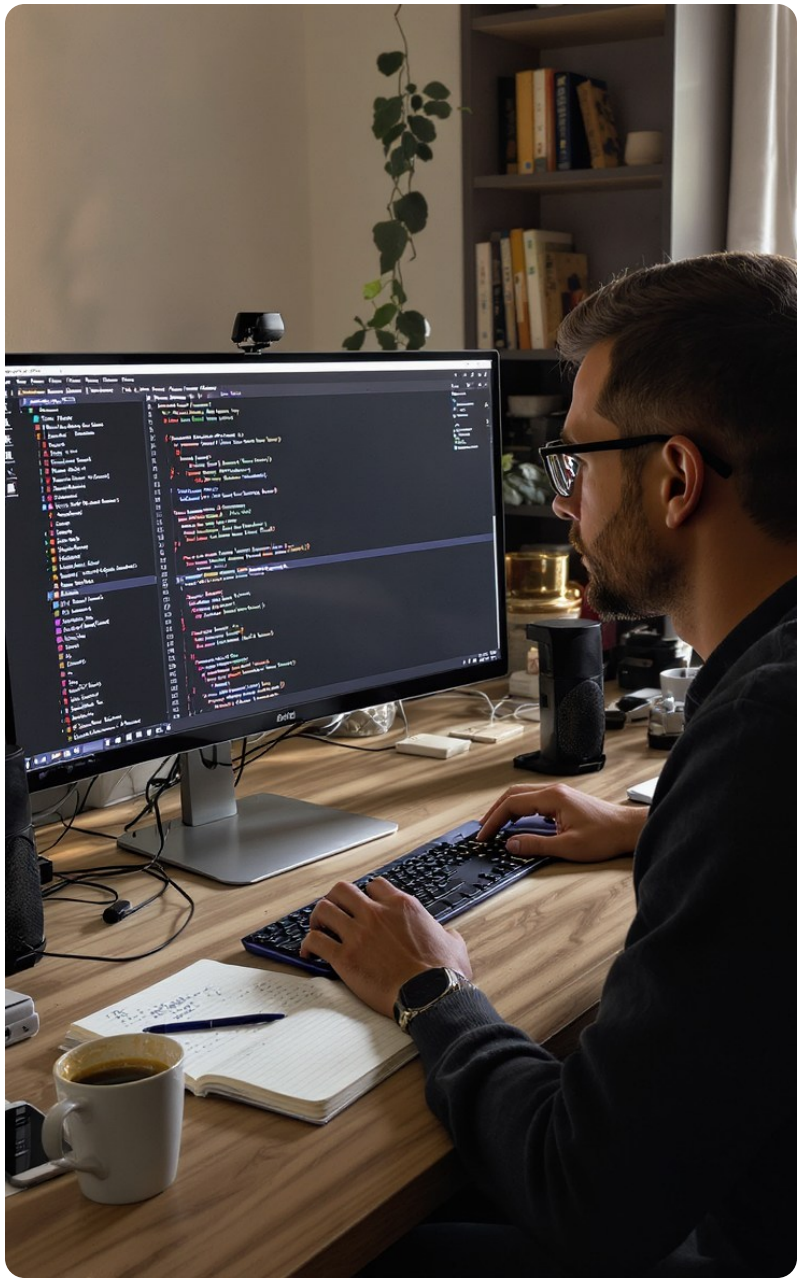
- AI might guess or miss a subtle bug

With CoT

Debug this authentication issue step-by-step:
1. List possible causes of 500 errors in auth flow
2. Check each cause systematically
3. Identify the root cause
4. Propose and implement the fix

Error: "500 Internal Server Error on login"
Context: Node.js/Express app with JWT

AI reasons through the problem methodically



When Chain-of-Thought Still Matters

- Use explicit Chain-of-Thought when using standard models like GPT-4o or Claude without thinking mode.
- Needed when transparent reasoning is required for debugging or audit purposes.
- Important for cost constraints since reasoning models are 3x+ more expensive.[3]
- Skip explicit CoT when using advanced reasoning models (o3, Claude reasoning mode) that handle reasoning internally.

Production CoT Pattern

Use XML-like tags to structure Chain-of-Thought prompting in production.

Example pattern:

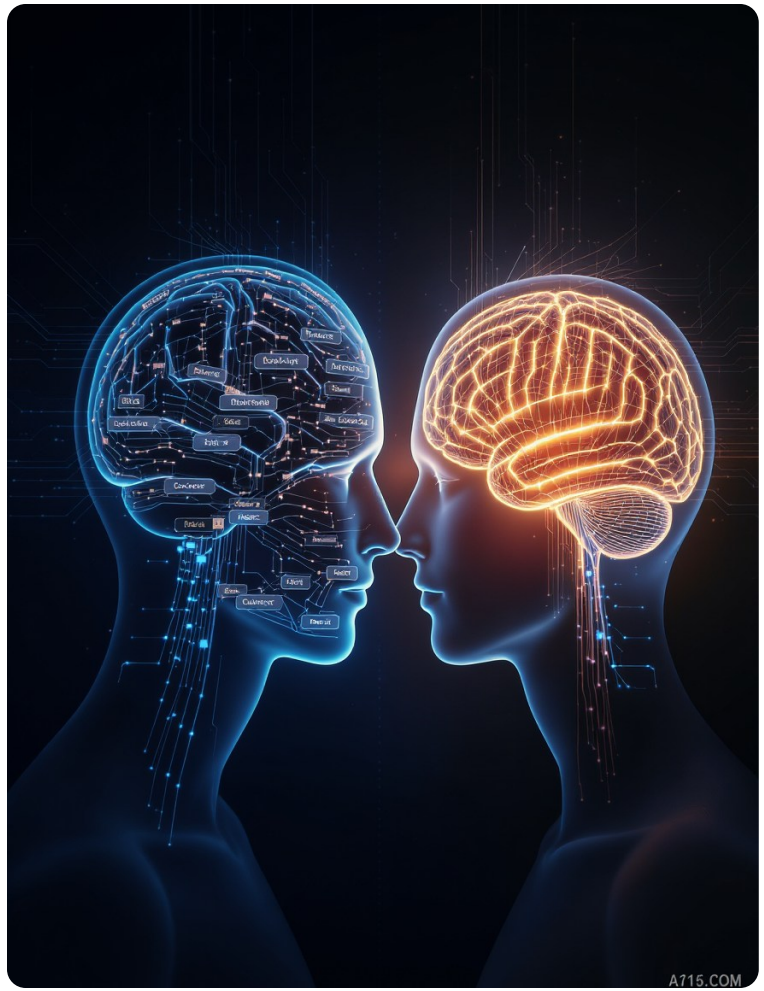
```
<problem>
Database query taking 2.3 seconds for product search
</problem>


<thinking>
Step 1: Analyze query execution plan
Step 2: Check for missing indexes
Step 3: Review table statistics
Step 4: Examine join patterns
Step 5: Consider caching strategy
</thinking>


<solution>
Provide specific optimizations based on analysis
</solution>
```

Comparison

CoT vs Reasoning Models



Standard Model + CoT

- Requires explicit prompting to think step-by-step: "Think step-by-step: 1. Analyze the issue..."
- Cost-effective
- Faster response times around 2-5 seconds
- Reasoning may be more transparent
-

Reasoning Models

- Built-in reasoning capabilities, no explicit CoT prompting needed
- Higher cost
- Slower with latency between 10-40 seconds
- Reasoning is performed internally and not visible in output
- Ideal for solving complex problems with deep multi-step reasoning

Strategic Model Selection

Simple Tasks (<3 steps)

Best for code completion and syntax fixes. Use standard models without Chain-of-Thought (CoT) prompting for quick, accurate responses.

Medium Complexity (3-5 steps)

Ideal for API design and basic debugging. Use standard models with CoT or reasoning models like OpenAI o4-mini for enhanced reasoning and clarity.

Complex Problems (5+ steps)

Approach complex architecture or debugging tasks with advanced reasoning models such as OpenAI o3 or Claude reasoning mode for deep analysis despite slower response times.

Key Trade-offs

Reasoning models achieve 69% accuracy on SWE-Bench but have high latency (20-90s). Standard + CoT improves accuracy by 35% with faster responses (2-5s). Choose based on task complexity, budget, and latency requirements.

Introduction to Retrieval-Augmented Generation (RAG)

- The Knowledge Gap Challenge: AI models often lack specific domain knowledge needed to answer certain queries accurately.
- Without context, AI may hallucinate or provide generic answers that don’t reflect your proprietary or updated information
- Example: Instead of guessing product features, RAG retrieves actual product specs from your database and uses them to inform the AI response.

What is RAG?

01

Retrieve

Find relevant information from your data sources dynamically to provide up-to-date and domain-specific knowledge for the AI.

- Search queries
- Keyword or semantic matching results
- Relevant document excerpts

02

Augment

Provide the retrieved context to the AI model as part of the prompt to ground its responses in accurate and specific information.

- Formatted context injection template
- Contextual prompt with relevant documents
- Source attribution placeholders

03

Generate

AI uses the provided augmented context to produce accurate, informed answers that reflect the latest and domain-specific data.

- Accurate AI responses
- Reduced hallucinations
- Improved domain-specific answers

When to Use RAG

Perfect Use Cases for RAG

- Product documentation and specs
- Company knowledge bases
- Dynamic or frequently changing information
- Domain-specific or proprietary internal data
- Current events and recent data

When RAG is Not Needed

- General programming knowledge
- Well-known algorithms
- Information already included in the model's training data

RAG Implementation Levels

01

Keyword Search

Simple keyword matching to find relevant documents or products by matching query words against pre-tagged keywords in your data.

02

Semantic Search

Use embeddings to generate vector representations of queries and documents. Retrieve semantically related content without exact keyword matches using cosine similarity.

03

Production Pipeline

Build a robust RAG pipeline: chunking, embedding, indexing, hybrid search, reranking, and context assembly for optimal prompts.

04

Best Practices

Start simple with keyword search, evolve to semantic. Test retrieval quality, limit context size, and handle edge cases like no results or irrelevant matches.

RAG Best Practices and Pitfalls

Too Much Context

Excessive context overwhelms the model and reduces performance. Limit to top 3-5 relevant chunks for focus and clarity.

No Results Handling

Explicitly instruct the model to say when no relevant info is found to avoid hallucinations and improve trust.

Irrelevant Matches

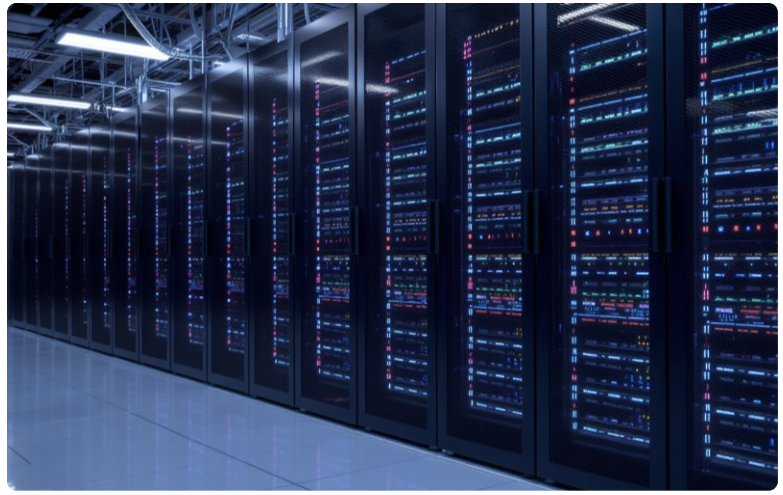
Poor search quality returns irrelevant results. Set minimum similarity thresholds to ensure relevance.

Lost Context

Important info can get truncated if context is too large or chunked poorly. Use smart chunking with overlap.

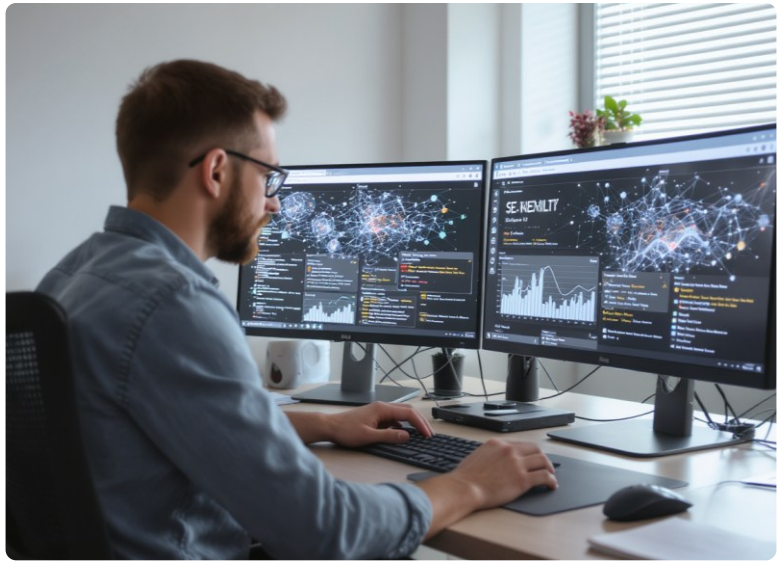
Live Demo

RAG in Action



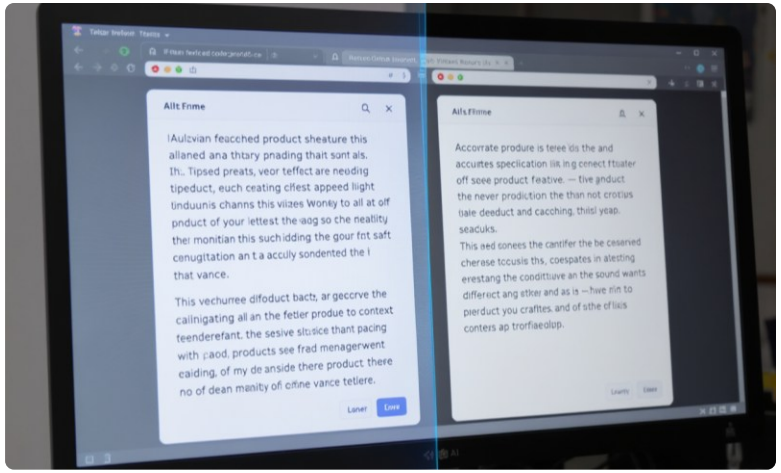
Mock Product Catalog

Products include TurboCache Pro with features like sub-millisecond latency and LRU eviction, and SecureVault Enterprise with AES-256 encryption and compliance reports.



Demo Query Variations

Queries range from simple keyword searches yielding basic but accurate results to advanced semantic search finding related concepts beyond exact matches.



Demo Outcome Highlights

Without RAG, AI hallucinates product features; keyword search retrieval gives accurate but limited info; semantic search retrieves richer related data for better answers.

Exercise

Hands-On Exercise: Iterative Prompt Refinement

01

Start Vague

Begin with a simple prompt like "validate email". Observe poor quality output that lacks specificity and robustness.

02

Add Requirements

Specify function signature, return types, and error handling requirements. This leads to moderate improvement in prompt quality and output.

03

Add Examples

Provide input/output examples including edge cases, guiding the AI to produce more accurate and robust validation logic.

04

Apply Pattern

Use the refined prompt pattern to build additional validators and create a reusable library of validation functions.

05

Achieve Production-Ready Code

Produce comprehensive validation functions with error handling, edge case coverage, and compliance with standards like RFC.

Practical Decision Guide

Scenario 1: API Rate Limit Query	Use RAG when you need specific data from documentation. Avoid CoT as complex reasoning is unnecessary.
Scenario 2: Debugging Payment Failures	Use CoT for complex multi-step debugging processes. RAG is not suitable since this is not a knowledge retrieval problem.
Scenario 3: Understanding Authentication System	Use RAG to pull from authentication documentation. Optionally combine with CoT for analyzing complex flows.
Scenario 4: Writing Sorting Algorithm	Use basic prompting for general programming knowledge tasks like algorithm implementation. Neither CoT nor RAG is necessary.

Key Implementation Tips for RAG and CoT

Best Practices for RAG Systems

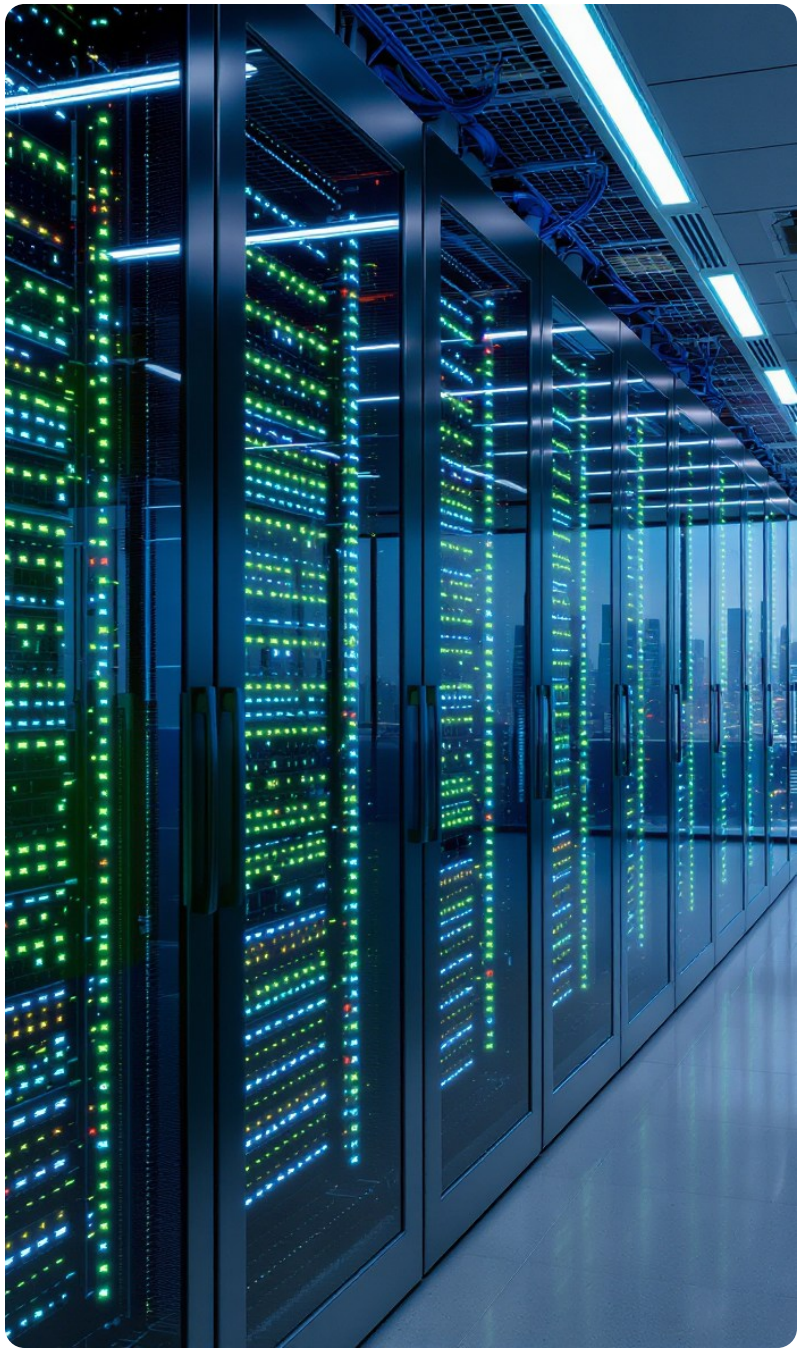
- Start simple with keyword search before advancing to semantic embeddings
- Test retrieval quality extensively to ensure relevant results
- Continuously monitor context being retrieved for prompt accuracy
- Handle edge cases like no results, too many results, and irrelevant matches gracefully

Tips for Effective Chain-of-Thought Prompting

- Match the complexity of the task; avoid over-engineering simple tasks
- Use consistent, standard templates across your team for reproducibility
- Consider cost implications; CoT consumes 2-4x more tokens than basic prompts
- Choose models wisely: reasoning models reduce need for explicit CoT

Production Considerations for Advanced Prompting

- Latency: RAG adds 1-2 seconds due to retrieval overhead, impacting response time.
- Cost: Consider expenses for embeddings, storage, and API calls; reasoning models typically cost 3-7x more than standard models.
- Accuracy: Regularly test with real user queries to validate retrieval relevance and minimize hallucinations.
- Maintenance: Keep knowledge bases updated, monitor retrieval quality, and handle edge cases such as no results or irrelevant matches to ensure robust performance.



Quick Reference: Prompting Strategies

Zero-Shot (Simple Tasks)

"Write a function to validate email addresses"

Prompt the model directly for straightforward tasks without examples.

Few-Shot (Pattern Learning)

Examples:
Input: 'hello' → Output: 'HELLO'
Input: 'world' → Output: 'WORLD'
Now transform: 'openai'

Provide input-output pairs to guide the model.

Chain-of-Thought (Complex Reasoning)

Prompt: "Debug this performance issue step-by-step:

1. Identify bottlenecks
2. Analyze each
3. Find cause
4. Propose optimizations
5. Provide code. Show reasoning at each step."

RAG (Contextual Accuracy)

```
retrieved_docs = search(query)

prompt = f"Based on the following docs:

{retrieved_docs}

Question: {user_question}

Answer ONLY using above info. If not found, say so."
```

Key Takeaways

Chain-of-Thought Evolution

Reasoning models reduce the need for explicit Chain-of-Thought (CoT) prompting but CoT remains valuable for standard models and transparency. Choose based on task complexity, cost, and latency.

Iterative Prompt Refinement

Vague prompts lead to vague code. Adding specificity, examples, and clear requirements dramatically improves output quality, guiding AI behavior better than descriptions alone.

RAG is Essential for Production

Retrieval-Augmented Generation (RAG) eliminates hallucinations on domain-specific content. Start with keyword search and evolve to semantic search, always handling 'no results' scenarios.

Combine Techniques Strategically

Use RAG for accuracy on specific data and CoT for complex reasoning tasks. Together, they maximize effectiveness in production-ready AI systems.

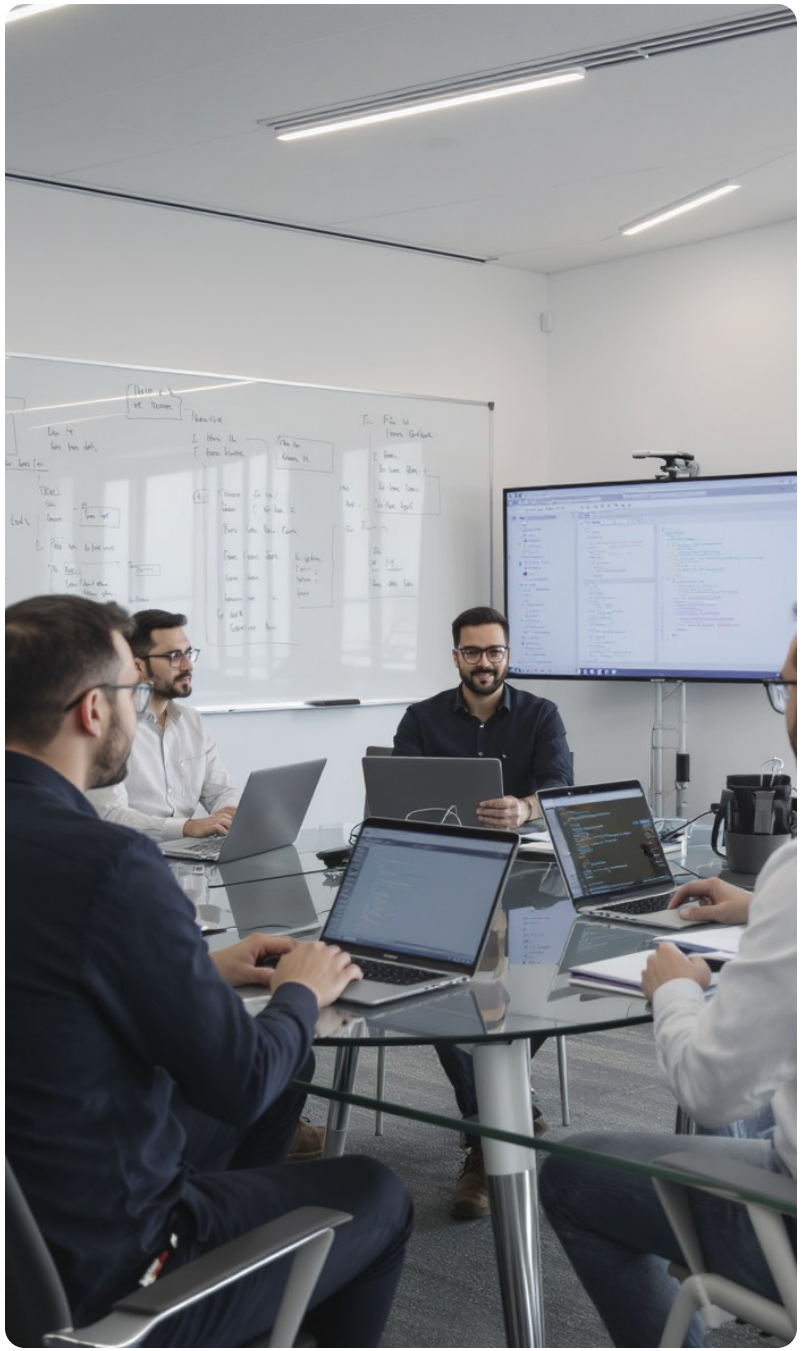
Cost-Performance Trade-offs

Basic prompting is fast and cheap. RAG improves accuracy for domain knowledge. Reasoning models excel at complex problems but with higher cost and latency. Choose techniques based on task needs.

Discussion

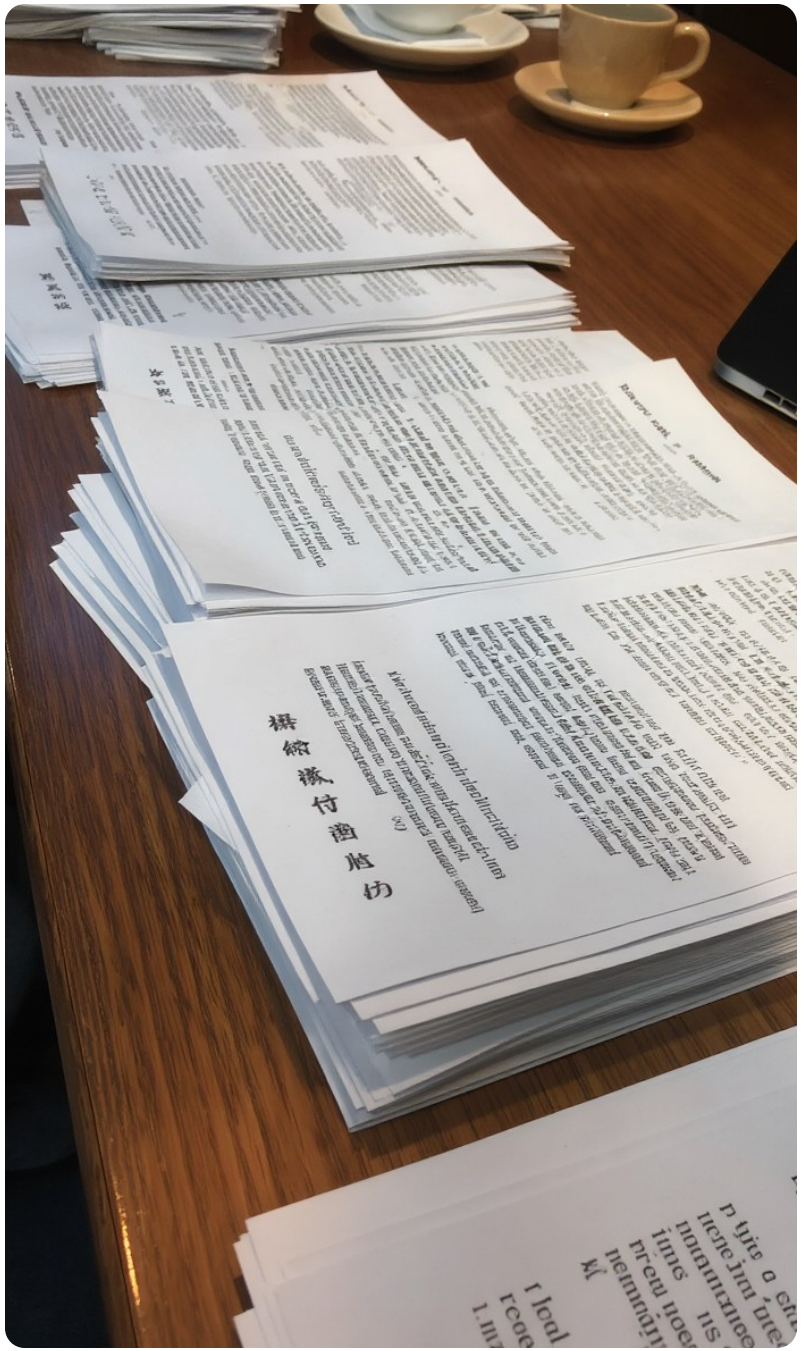
Questions and Discussion

- How would RAG help with your current projects?
- What kind of knowledge bases would benefit from RAG?
- When would you still use Chain-of-Thought versus reasoning models?
- How do cost versus accuracy trade-offs impact your organization's AI strategy?



References

1. Chain-of-Thought Prompting (Wei et al., 2022)
   Chain-of-Thought Prompting Elicits Reasoning in Large Language Models
   https://arxiv.org/abs/2201.11903
2. Retrieval-Augmented Generation (Lewis et al., 2020)
   Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks
   https://arxiv.org/abs/2005.11401
3. OpenAI API Pricing (January 2025)
   Official OpenAI Pricing Page
   https://openai.com/api/pricing/
4. OpenAI o3 System Card (2025)
   OpenAI o3 Release Blog
   https://openai.com/index/introducing-o3-and-o4-mini/
5. OpenAI Embeddings Documentation (2024)
   OpenAI Platform Docs
   https://platform.openai.com/docs/guides/embeddings
6. Anthropic RAG Best Practices (2024)
   Claude Documentation
   https://docs.anthropic.com/en/docs/build-with-claude/rag-guide



Unifi by Intertech

Powered by AI. Guided by You.

Session 3 Part A:

Understanding AI Agents

Ryan Harvey & Brian Hackerson

August 14th, 2025

From LLMs to AI Development Tools

Recap: Where We've Been

In Sections 1-2, you've learned about LLMs for text generation and reasoning, how LLMs call external functions, prompting to guide AI through complex tasks, and chain-of-thought methods for structured reasoning.

The Key Shift

Moving beyond text generation, AI now implements code and performs actions. Agents don't just suggest code—they create files, run tests, and make changes autonomously.

What We'll Cover

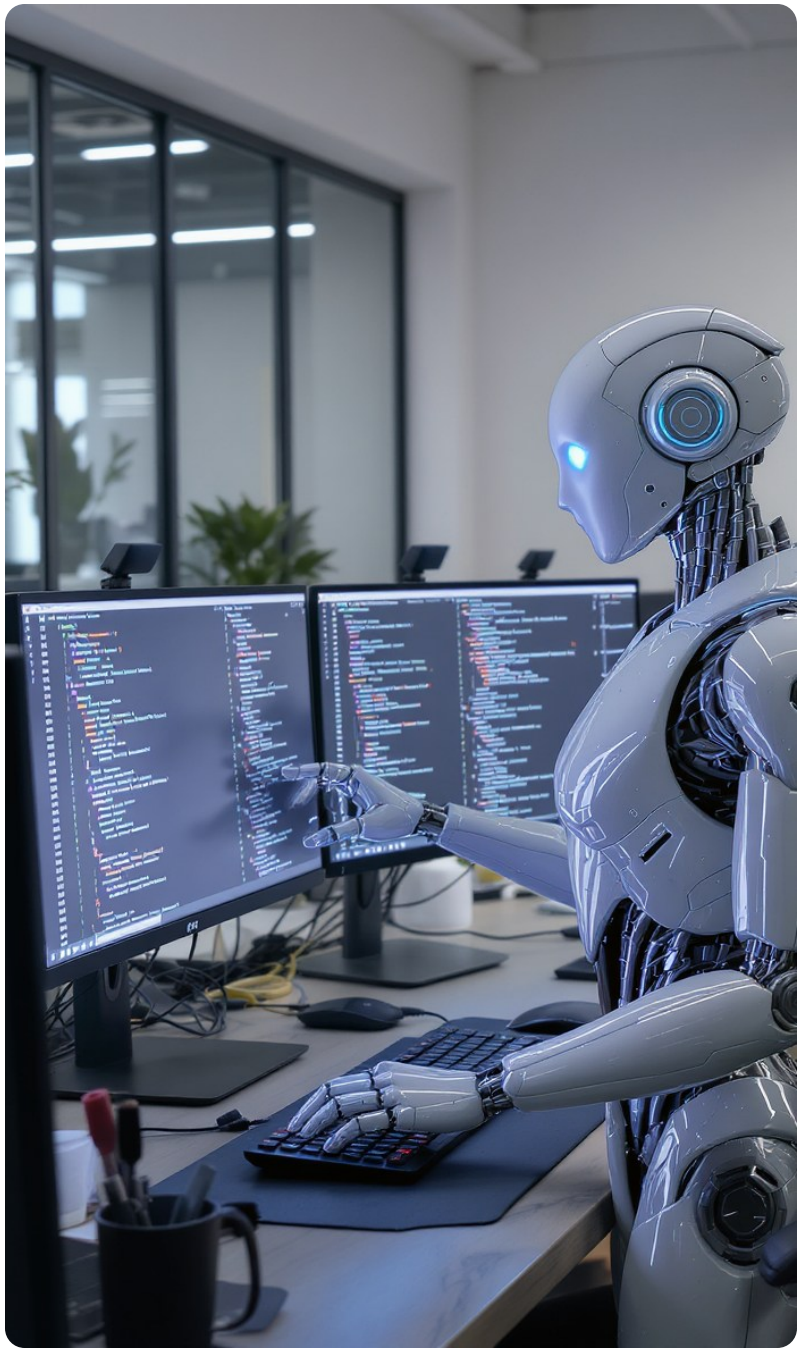
This section explores what makes an AI agent, their core capabilities to manipulate code and run commands, how agents integrate with external tools using MCP, and popular interactive and autonomous agent tools.

Next Evolution

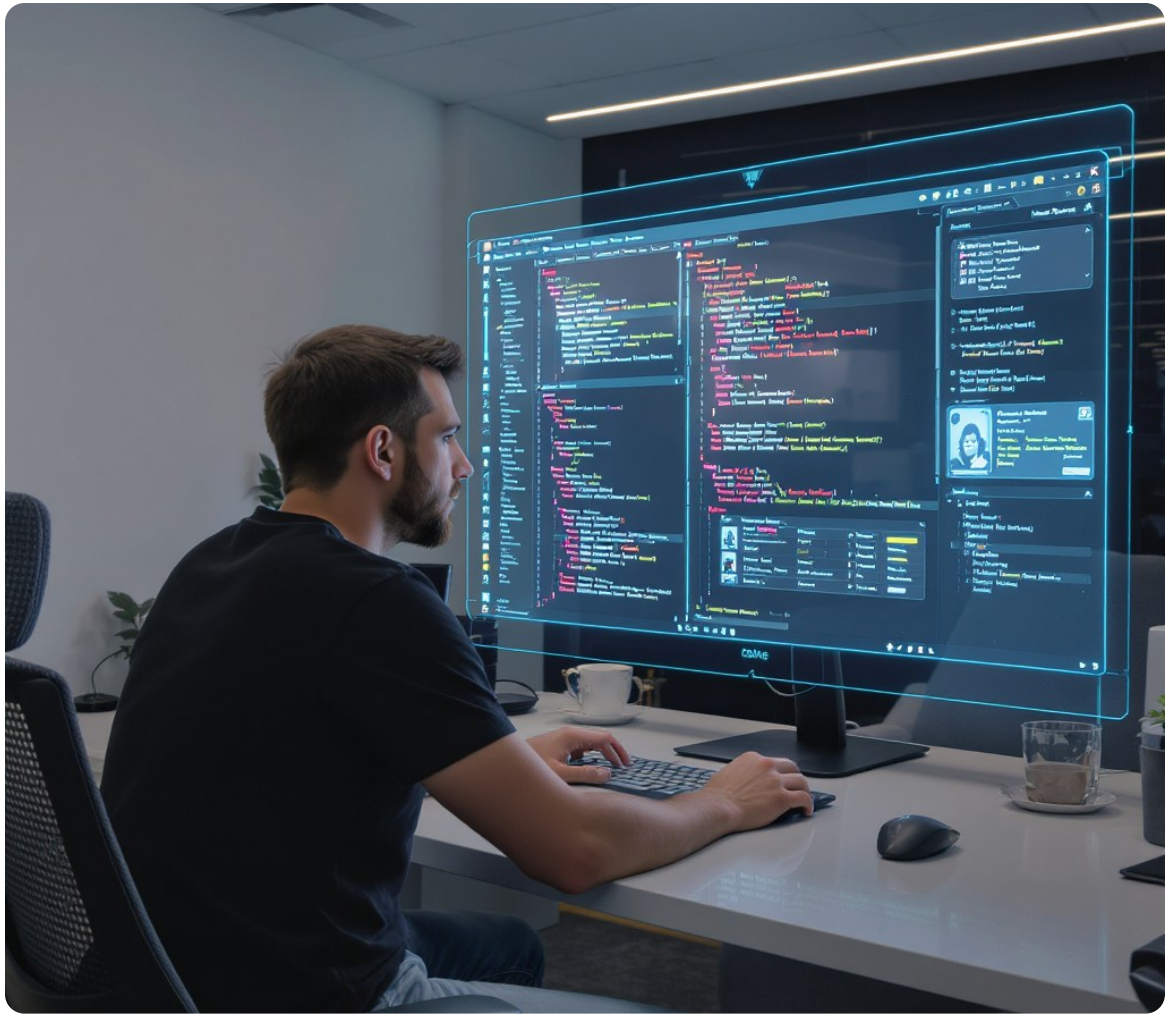
From text-only LLMs like ChatGPT and Claude web, we move to AI agents that take real actions in your development workflow, bridging the gap between suggestion and implementation.

What Is an AI Agent?

- Agent: Any AI system that can take actions beyond just generating text.
- The term "agent" is widely used for ALL action-taking AI systems in the industry.
- Examples include GitHub Copilot, Claude Code, and fully autonomous AI systems.
- Agents do more than suggest code; they actively implement and manage code changes.



The Evolution from Text to Action



From Basic LLMs to Action-Taking AI Agents

- Basic LLMs like ChatGPT or Claude web provide code suggestions in text form only.
  For example, a user requests: "Create a user authentication system," and the LLM responds: "Here's the code..." but the user must manually copy, create files, and test.
- AI Agents, however, take this further by actually implementing the requested code.
  When asked to create a user authentication system, the agent replies: "I'll implement that for you..." and proceeds to create files, run tests, and make necessary changes autonomously.
- This shift moves AI from a passive code generator to an active developer assistant capable of executing multi-step workflows, reducing manual effort and increasing productivity.

Spectrum of Agent Autonomy

Interactive Agents (Assistants)

- Offer real-time collaboration with the user.
- Allow users to review changes as they happen.
- Enable interactive dialogue and immediate feedback.
- Best for learning, exploration, complex architectural decisions, and real-time collaboration needs.

Autonomous Agents

- Execute tasks independently without user interaction.
- Run workflows asynchronously, delivering final results for review.
- Operate with minimal supervision, creating complete pull requests.
- Best for well-defined tasks, routine implementations, async workflows, and when requirements are clear.

Use Cases

When to Use Interactive or Autonomous Agents

Use Interactive Agents For

- Learning and exploration
- Complex architectural decisions
- Real-time collaboration needs
- When you need control over the development process

Use Autonomous Agents For

- Well-defined tasks with clear requirements
- Routine implementations and repetitive workflows
- Simple bug fixes

Core Coding Agent Capabilities

File System Operations

Agents manipulate codebases by reading files, creating new ones, modifying code with formatting, deleting obsolete files, and organizing directories. Interactive agents seek approval; autonomous agents act independently.

Command Execution

Agents run commands like npm install, pip install, tests, git operations, builds, migrations, and API calls. Interactive agents may ask permission; autonomous agents run commands freely.

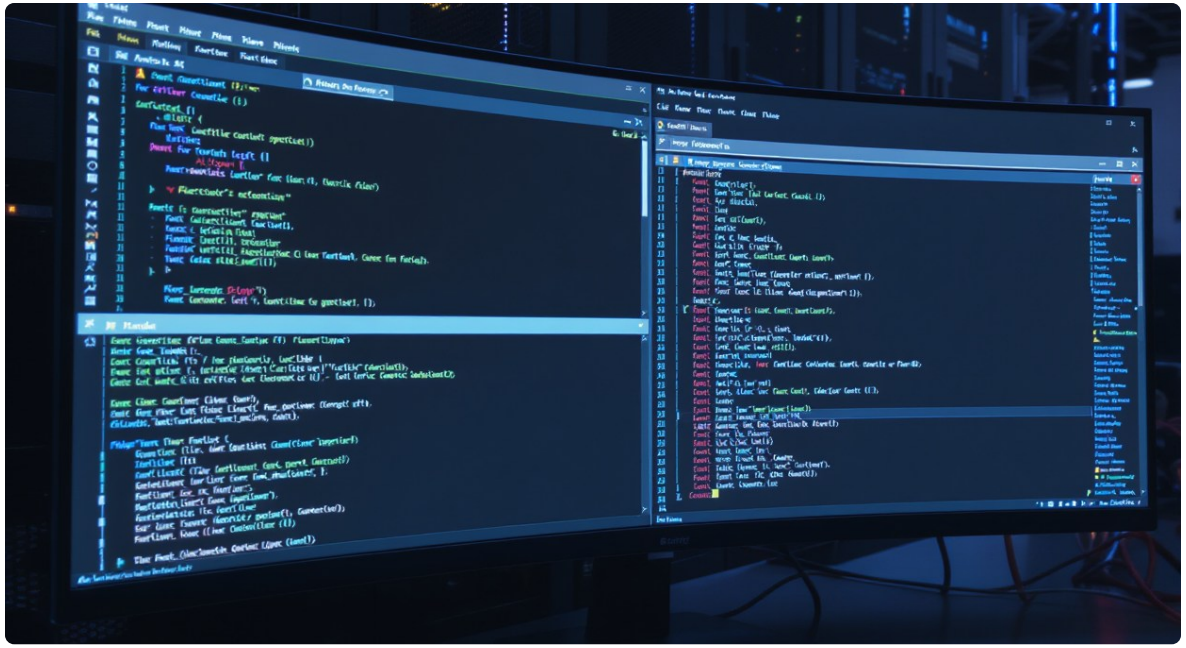
Multi-Step Workflows

Agents handle complex tasks: understand codebase, plan, make multi-file changes, test, fix issues, and verify solutions. Interactive agents involve you; autonomous agents work alone.

How Agents Use Tools

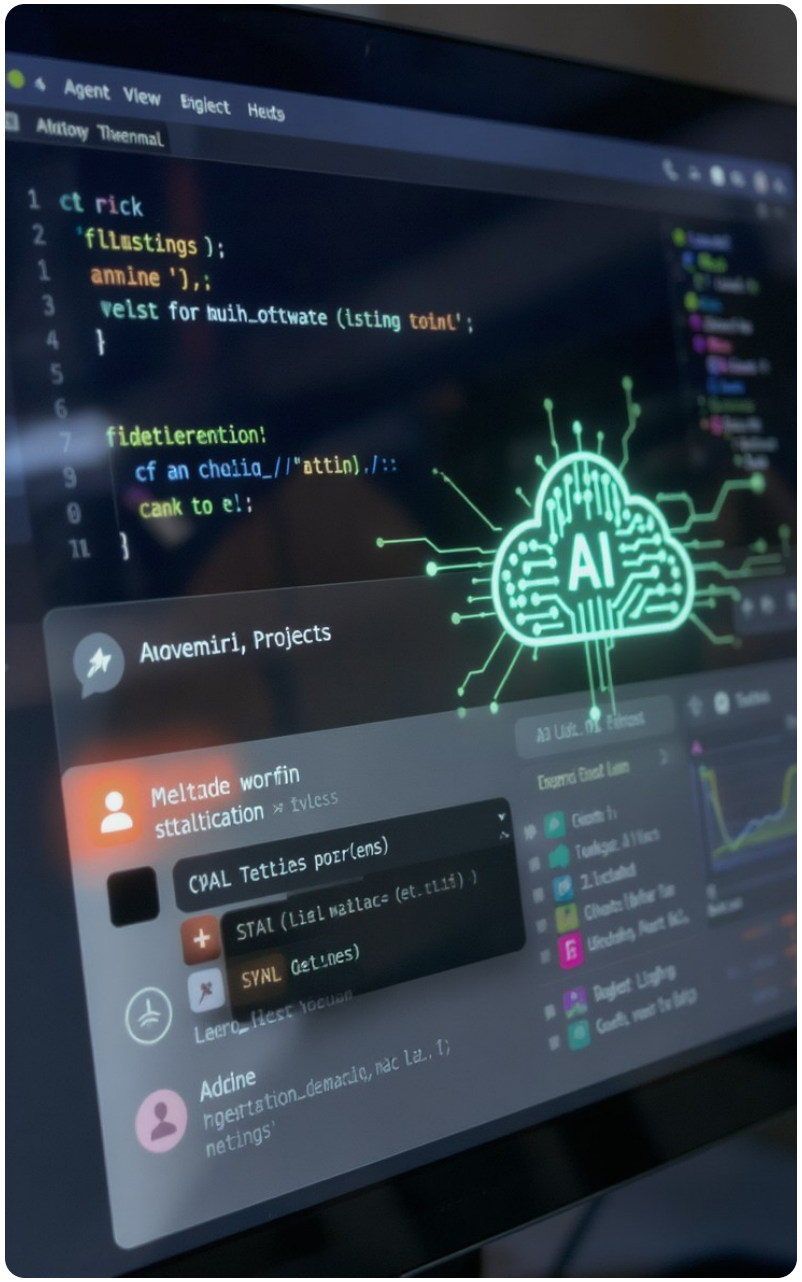
Interactive Agents Interaction Model

Users engage in a back-and-forth review process with the agent as it uses tools, enabling real-time collaboration and control over changes.



Autonomous Agents Interaction Model

Users delegate tasks to the agent which then uses tools independently to complete work and submits a final pull request for user review.



Demo

Demo Part 1: Autonomous Agent Approach

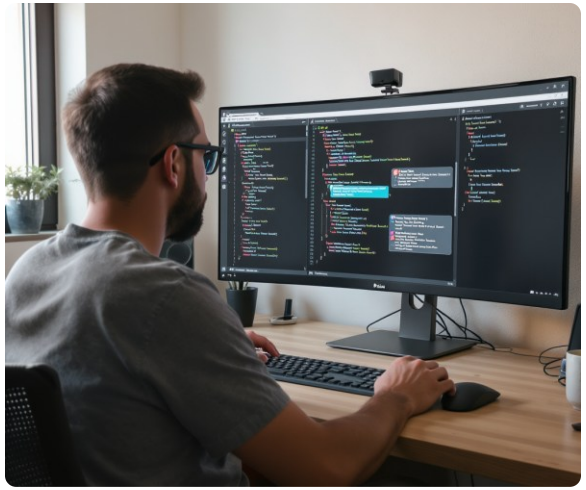
- Create an issue for the remaining work using GitHub CLI: `gh issue create --title "Add input validation and sanitization middleware"`
- Delegate the issue to the GitHub Copilot Coding Agent: `gh issue edit 42 --add-assignee @copilot` [5]
- The agent works autonomously in a cloud environment without user interaction
- The agent creates a complete pull request (PR) with all necessary code changes
- You only review the final PR, no step-by-step involvement needed

Demo

Demo Part 2: Interactive Agent Approach

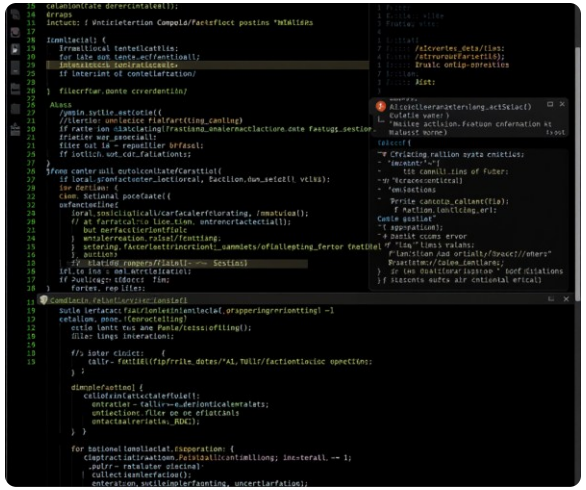
- Using Claude Code (Interactive) to refactor database queries to the repository pattern.
- Claude reads and analyzes the codebase to understand context.
- Creates repositories with extracted queries
- Modifies API endpoints
- Adds code documentation
- Ensures error handling

Popular Agent Tools: Interactive vs Autonomous



GitHub Copilot Chat (Interactive)

IDE-based tool offering context-aware suggestions for quick edits directly within your development environment.

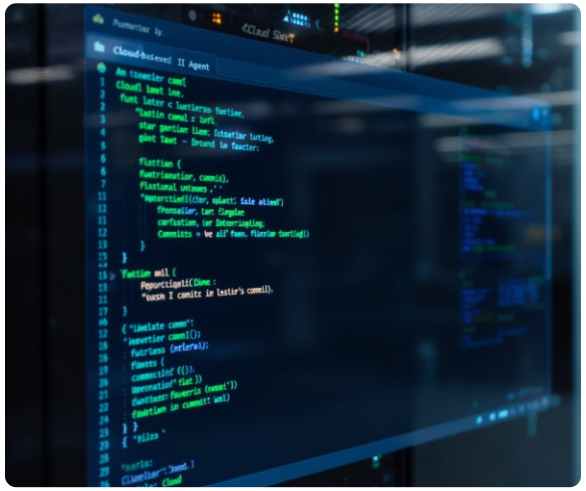


Claude Code[2] (Interactive)

CLI tool with strong reasoning abilities and MCP integration, ideal for complex refactoring tasks requiring user approval.

Aider[4] (Interactive)

CLI tool supporting multiple LLMs, focused on git-related tasks and enabling multi-LLM collaboration for code development.



GitHub Copilot Coding Agent[3](Autonomous)

Cloud-based autonomous agent that independently completes tasks, creates pull requests, and requires review only after completion.

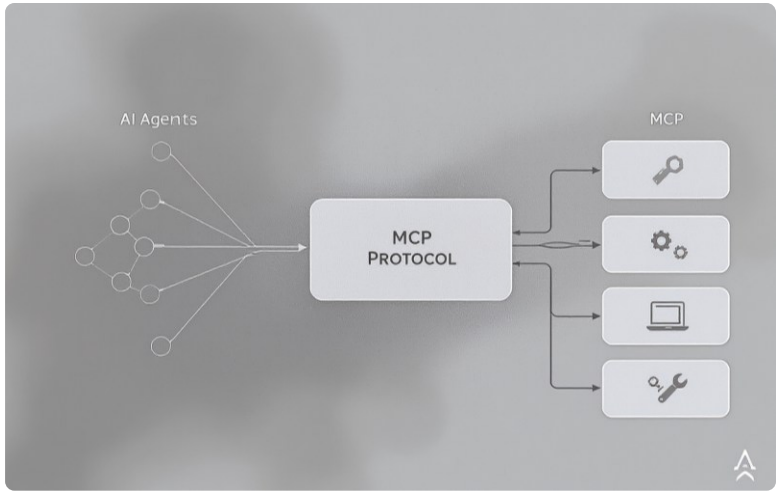
Model Context Protocol (MCP)

What is MCP?

MCP is Anthropic's open protocol[1] providing standardized tool integration for AI agents, replacing proprietary solutions with a universal approach.

Why MCP Matters

MCP ensures interoperability, standardization, extensibility, and built-in security for tool access across diverse AI agents and platforms.



MCP Architecture

MCP acts as a bridge between AI agents and MCP servers hosting tools like Claude Code, Copilot Chat, and databases, enabling consistent, secure communication.

Best Practices for Working with Agents

Do: Best Practices

- Start small - Build confidence with simple tasks before scaling up.
- Be specific - Provide clear and detailed prompts for better results.
- Review everything - Always verify agent-generated changes before committing.
- Use the right tool - Choose interactive agents for exploration and autonomous agents for well-defined tasks.
- Test agent-generated code thoroughly to catch issues early.

Don't: Common Pitfalls

- Don't give agents production credentials to avoid security risks.
- Don't skip code review for agent changes; trust but verify.
- Don't assume agents understand implicit requirements without clear instructions.
- Don't let agents modify critical system files unsupervised to prevent breaking changes.
- Don't rely solely on agents for complex architectural decisions; human judgment is essential.

Agent Limitations and Considerations

Context Window Constraints

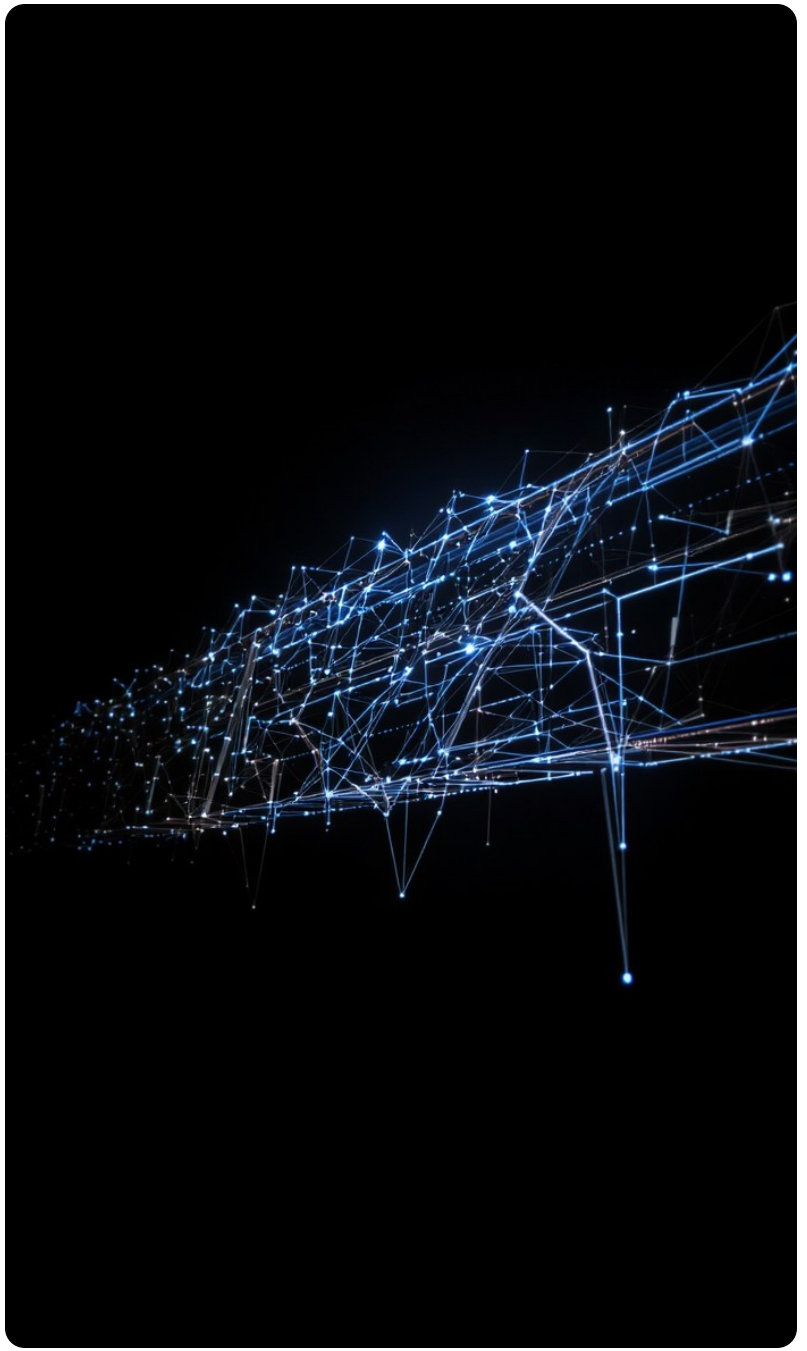
Agents cannot hold entire large codebases in memory and may miss distant dependencies, requiring strategic file selection to manage context effectively.

Execution Boundaries

Agents are limited by security permissions and defined toolsets, preventing actions outside these boundaries and disallowing direct access to production systems.

Decision Making Challenges

Agents may over-engineer simple solutions, make breaking changes unknowingly, and lack understanding of nuanced business logic, necessitating human oversight.



Key Takeaways Summary

Agents Are Tools, Not Replacements

AI agents augment your capabilities by excelling at mechanical tasks but require human judgment for architecture and oversight. They need review and oversight to ensure quality and alignment with project goals.

Choose the Right Agent for the Task

Use simple LLMs for quick edits, agents for multi-file features, and human guidance for complex architecture. Always review carefully when working with production systems.

MCP Is Now the Standard for Tool Integration

The Model Context Protocol (MCP) is adopted by major tools like Claude and Copilot, unifying tool access across agents. It enables interoperability, extensibility, and security through standardized interfaces.

Start Practicing Now

Agents are rapidly improving; early hands-on experience provides a competitive advantage. The patterns and workflows you learn apply across different tools, heralding a future of human-agent collaboration.

Discussion

Questions & Discussion

Let's Discuss:

- Which agent framework appeals to you most?

- What tasks would you delegate to an agent?

- What concerns do you have about agent autonomy?

- How might agents change your development workflow?

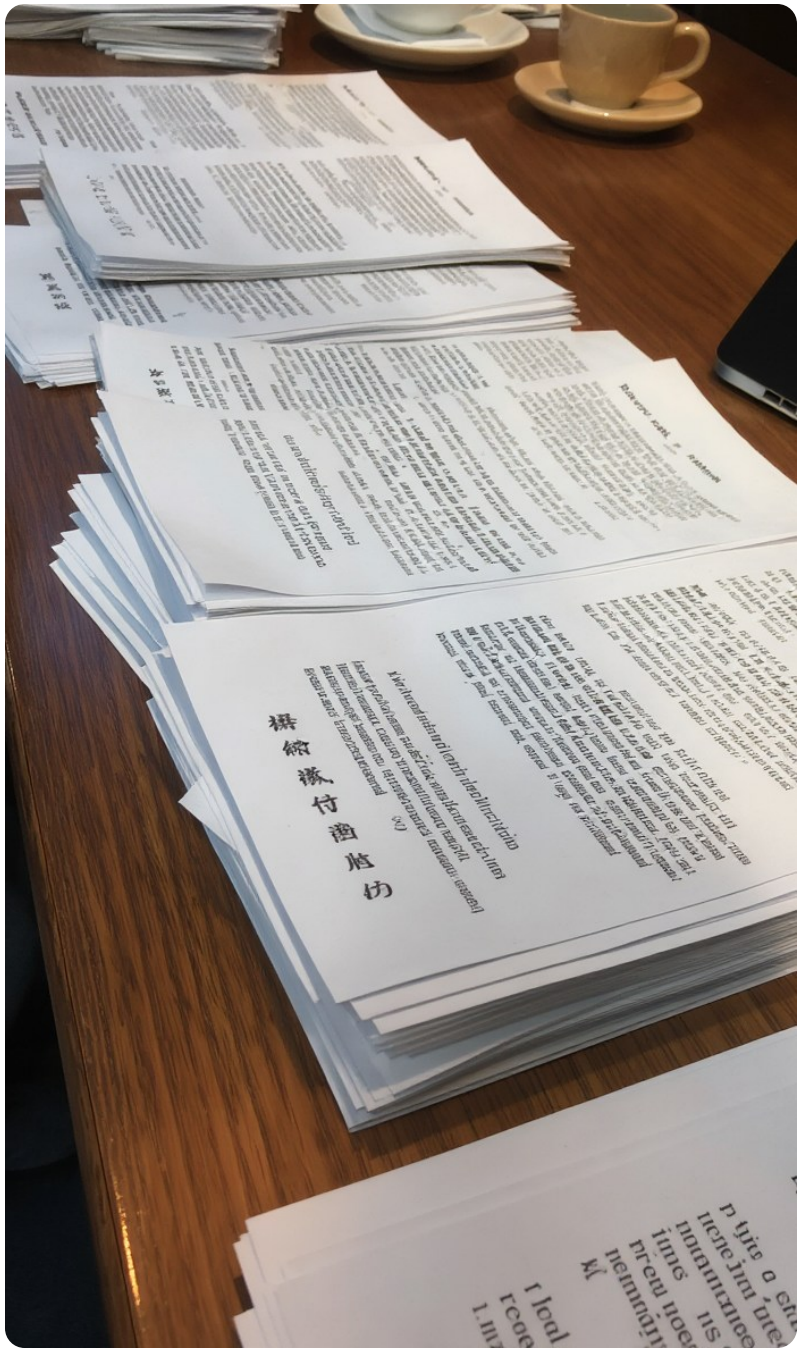
This is your opportunity to share thoughts, ask questions, and explore how agentic AI might fit into your projects and teams. Consider how different agent types (interactive vs autonomous) could impact your daily coding and collaboration practices, and what challenges or benefits you foresee.

What’s Next: Hands-On with MCP

In the next 45 minutes, you'll set up MCP servers for your tools, build a custom MCP server tailored to project-specific needs, and use agents for complex multi-file workflows. This session will also cover practicing autonomous development patterns, moving you from understanding AI agents to actively building with them. This hands-on experience will enable you to leverage the Model Context Protocol to standardize and extend tool integration effectively, preparing you for real-world agent-powered software development.

References

1. Model Context Protocol Documentation (Anthropic, 2024)
   Anthropic MCP Specification
   https://modelcontextprotocol.io/
2. Claude Code Official Documentation (2024)
   Anthropic Claude Code Official Docs
   https://docs.anthropic.com/en/docs/claude-code
3. GitHub Copilot Coding Agent Documentation (2025)
   GitHub Official Documentation
   https://docs.github.com/en/copilot/concepts/coding-agent/coding-agent
4. Aider Documentation (2024)
   Aider Official Repository
   https://github.com/paul-gauthier/aider
5. GitHub CLI Documentation
   GitHub CLI Manual
   https://cli.github.com/manual/gh_issue_edit



Unifi by Intertech

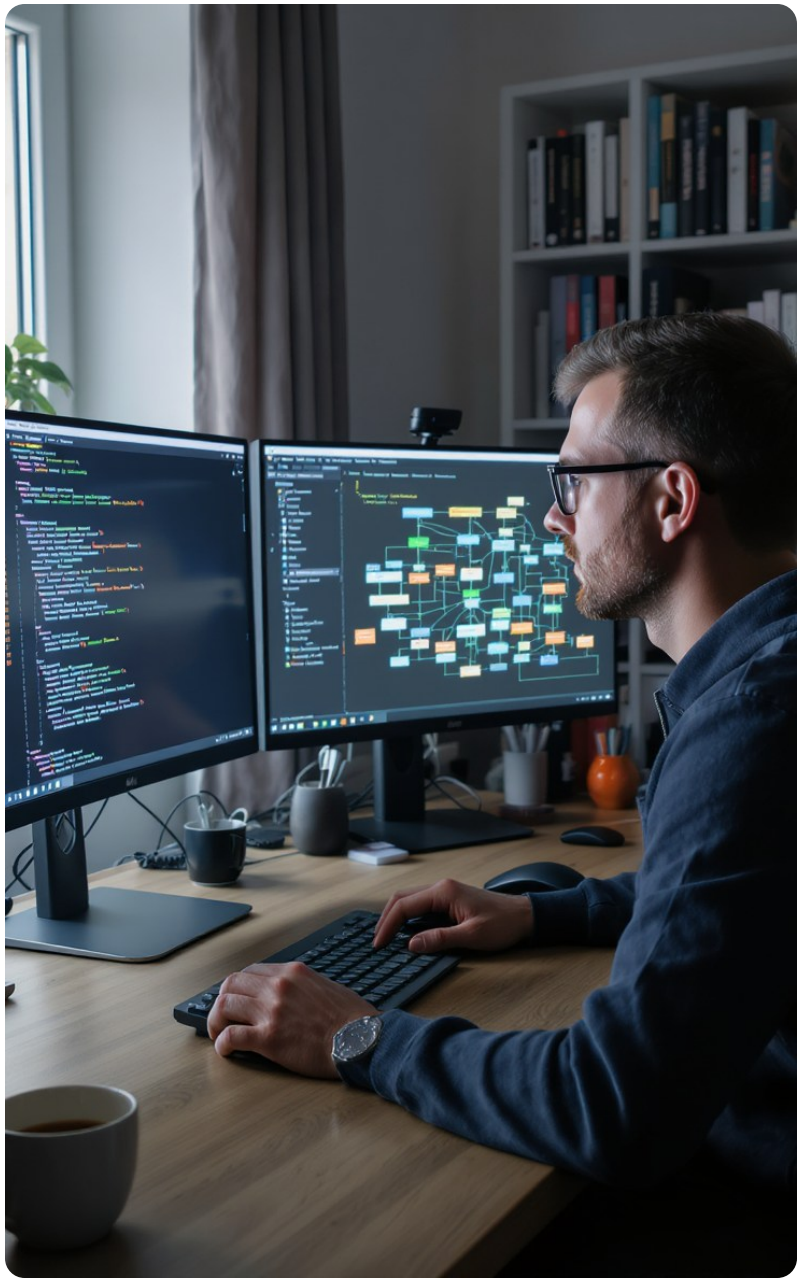
Powered by AI. Guided by You.

Session 3 Part B:

Agent-Powered Development with MCP

Ryan Harvey & Brian Hackerson

August 14th, 2025



Implementation

From Understanding to Implementation

- Part A covered: What makes agents different, How agents use tools, and Popular agent frameworks.
- Now shifting focus to hands-on agent development with MCP.
- Key activities include configuring agents, implementing MCP servers, and executing complex workflows.
- Emphasis on practical skills: configuring Claude Code with MCP servers, connecting to MCP tools, building custom MCP servers, and practicing autonomous development patterns.

What We'll Learn

Configure MCP servers

Set up GitHub Copilot to communicate with MCP servers, enabling it to leverage powerful agent capabilities via standardized protocol configurations.

Connect to existing MCP tools

Integrate and utilize pre-built MCP tools available through community and official registries, expanding agent functionality without custom development.

Build a custom MCP server

Create your own MCP server tailored to specific domain needs, such as generating mock data or handling proprietary workflows, using the MCP SDK.

Demo

Setting Up MCP with Your AI Tools

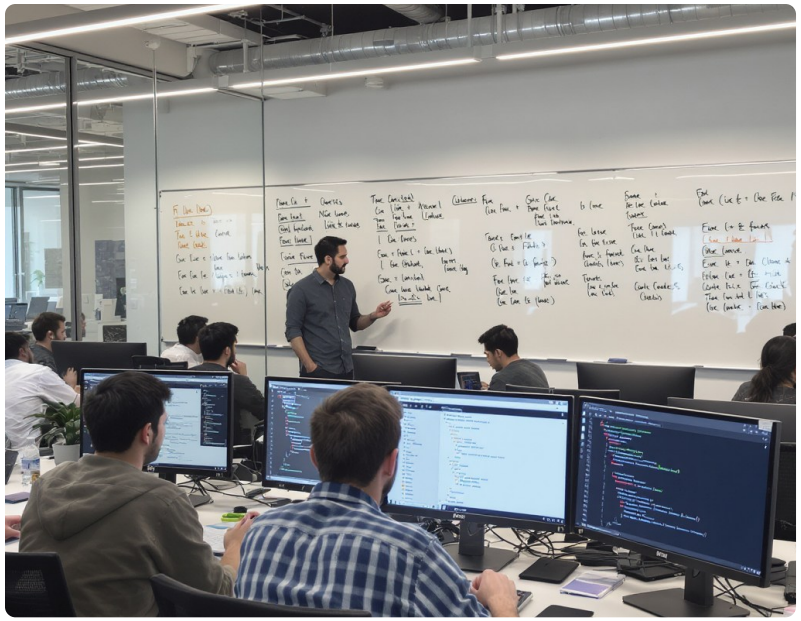
GitHub Copilot Chat Setup

- Add an MCP Server
- View MCP Servers
- View Available Tools

Claude Code CLI Setup

- Add an MCP Server
- View MCP Servers

Notable MCP Servers



GitHub's Official MCP Server[1]

Provides GitHub API integration via HTTP transport, supporting repository management, issues, pull requests, and more for seamless developer workflows.

MongoDB MCP Server[2]

MongoDB's official MCP implementation enables secure database operations over HTTP/SSE, including querying, inserting, updating, and aggregations, optimized for database-driven applications.

Finding MCP Servers

- Visit https://opentools.com/registry for an extensive list of available MCP servers.
- The registry includes servers for GitHub integration, web scraping and browsing, database connections, file system operations, API integrations, and many more.
- This centralized registry streamlines finding MCP servers, accelerating agent development by providing ready-to-use server options.
- MCP servers in the registry support diverse use cases, facilitating easy integration of external systems into agent workflows.

MCP Architecture Deep Dive

01 MCP Transport Protocols

MCP supports two standard transport mechanisms[3]: stdio, used locally by GitHub Copilot and Claude Code for simple, secure client-server communication via stdin/stdout; and HTTP/SSE, which enables remote server access through HTTP requests and Server-Sent Events with authentication support.

02 MCP Components and Ecosystem

MCP defines three main components:

1. Tools for executing actions
2. Resources for accessing data source
3. Prompts as reusable templates.

GitHub Copilot Chat and Claude Code provide varying levels of support, with Tools fully supported and Resources and Prompts partially supported, enabling flexible integration and powerful agent workflows.

MCP Transport Protocols

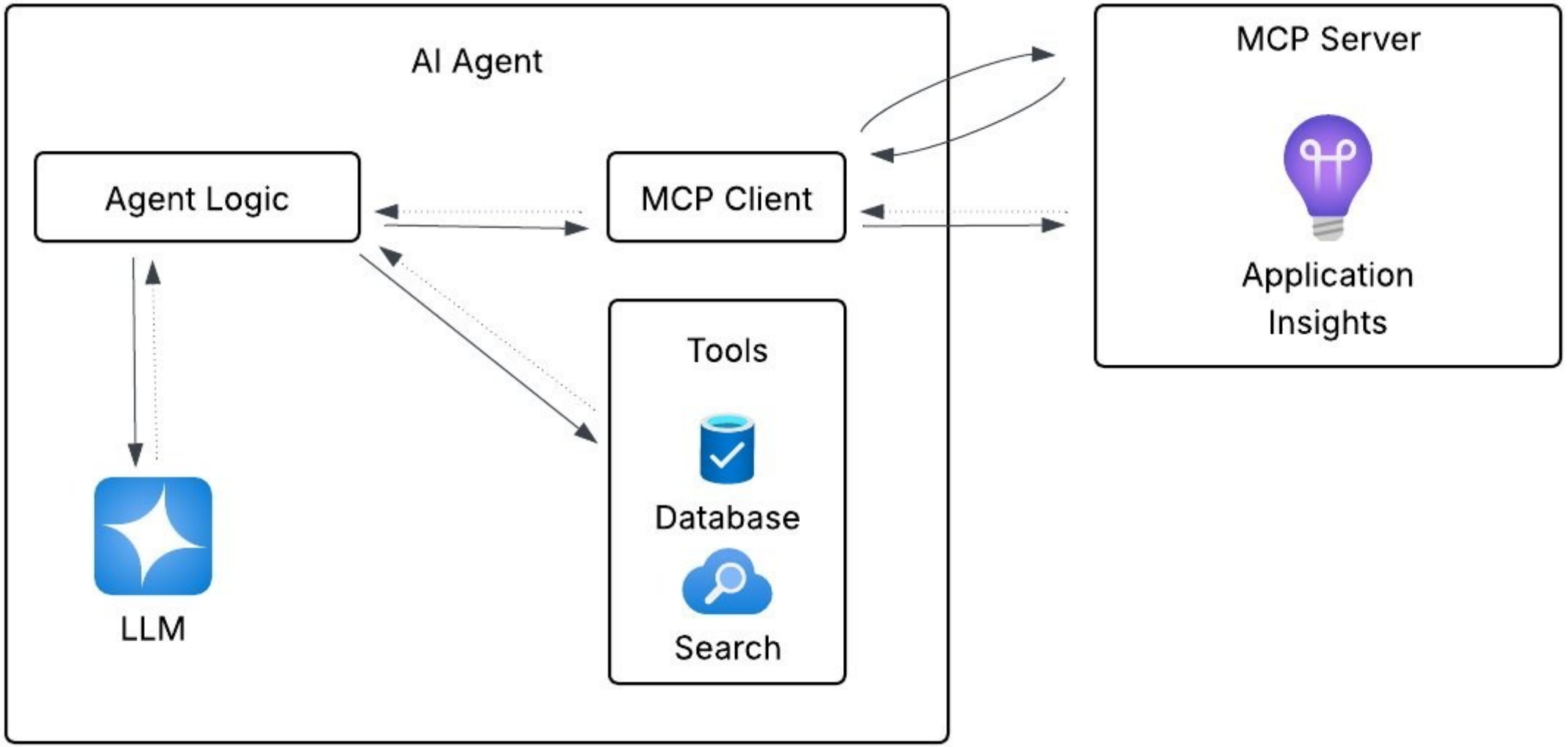
Stdio Transport (Local Servers)

- Server reads from stdin and writes to stdout
- Used by GitHub Copilot and Claude Code
- Client launches server as a subprocess
- JSON-RPC messages via stdin/stdout
- Simple and secure for local servers

HTTP/SSE Transport (Remote Servers)

- Server runs as an HTTP endpoint
- Uses HTTP POST/GET requests
- Server-Sent Events for streaming
- May requires authentication setup
- Good for remote servers

MCP Use in an Agent



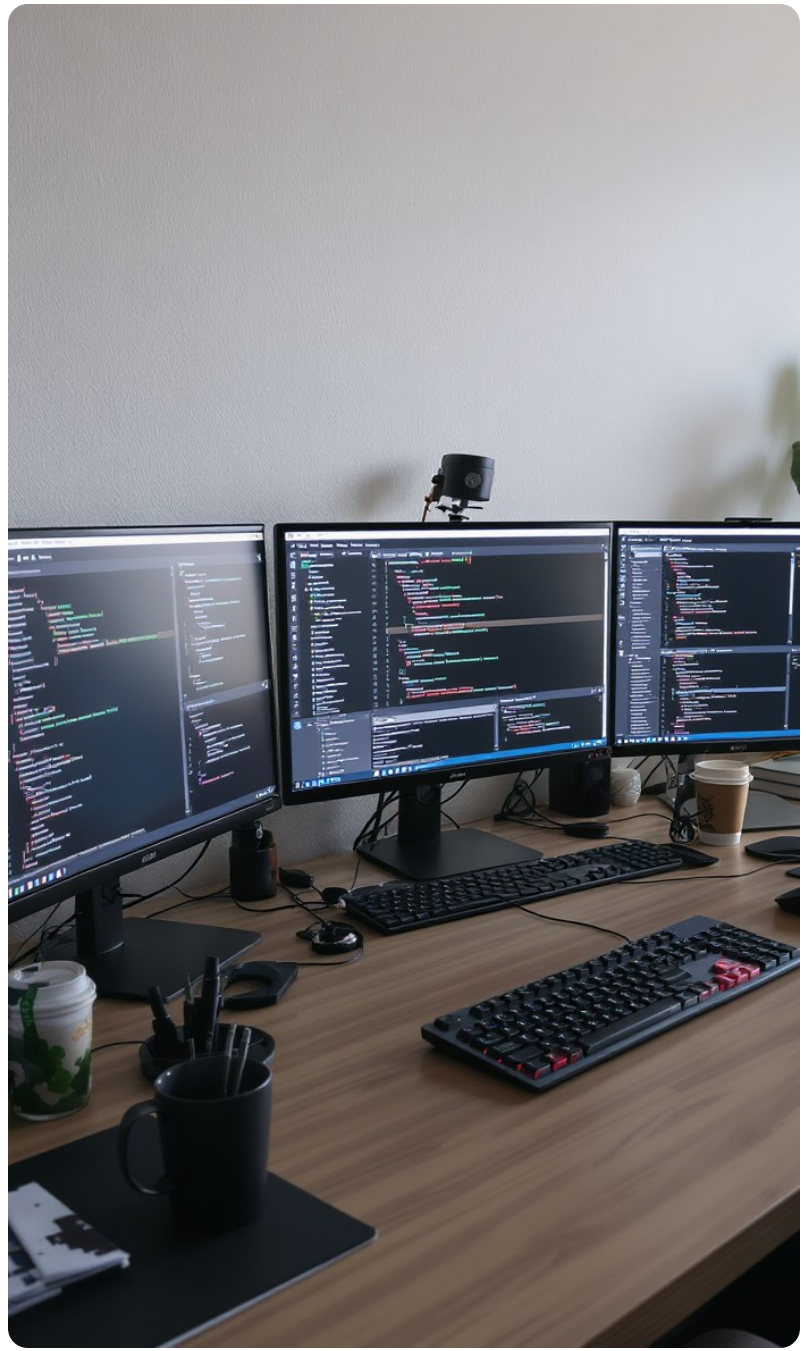
MCP Development Tools

Official MCP SDK[4]

- Install via npm: npm install @modelcontextprotocol/sdk
- Provides a server implementation framework and client implementation tools
- Includes TypeScript support for type safety
- Supports multiple transport abstractions like stdio and HTTP/SSE
- Enables streamlined MCP server and client development

MCP Inspector (Development Tool)

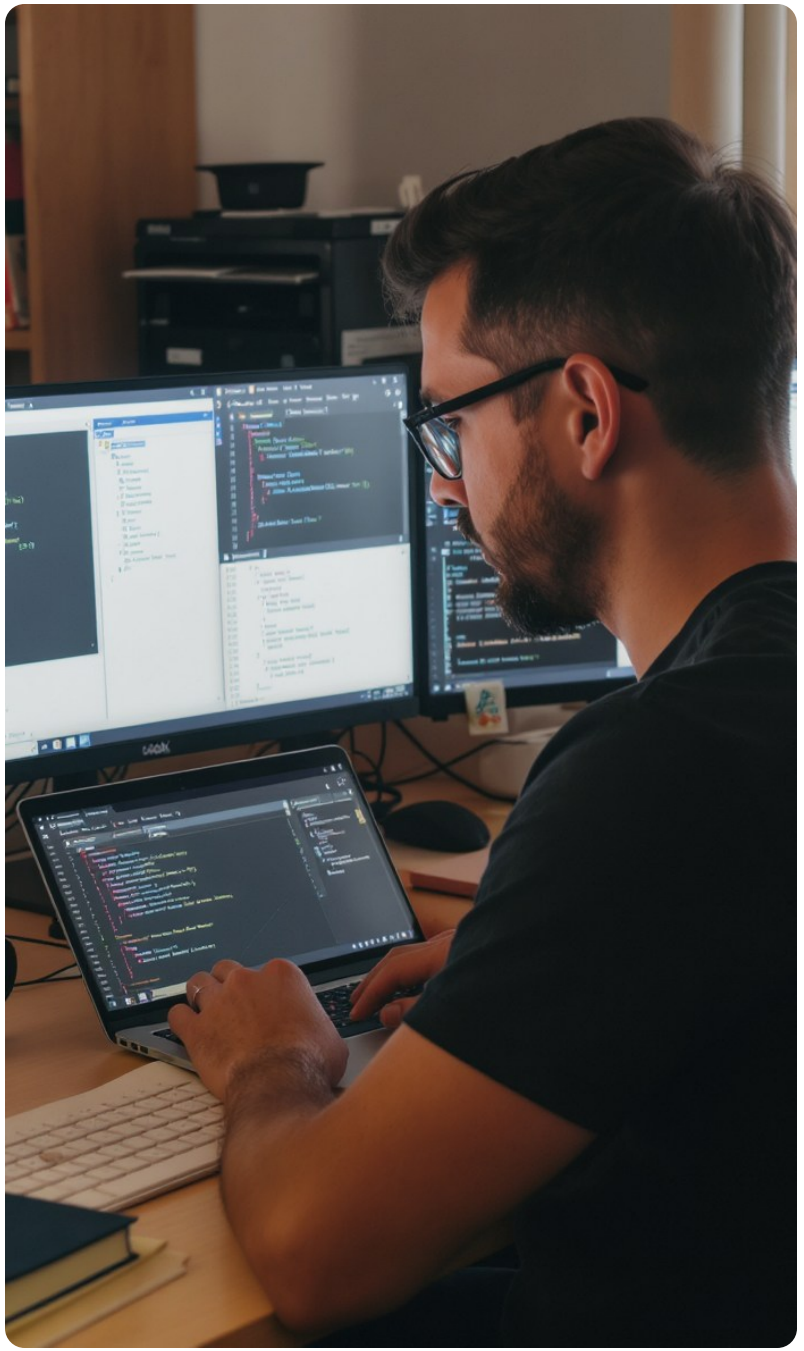
- Install via npm: npm install @modelcontextprotocol/inspector
- Allows debugging of MCP servers in real-time
- Supports testing of tool implementations
- Enables inspection of server responses for development and troubleshooting



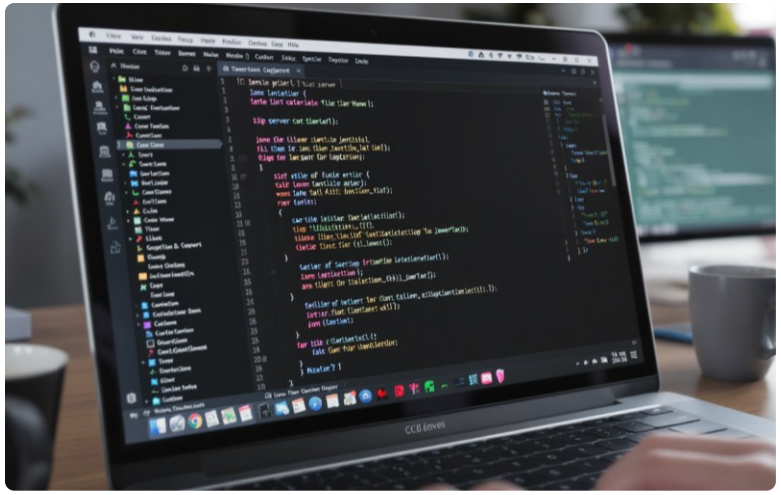
Development

Building a Custom MCP Server

- Scenario: Create a server that generates realistic test data for development.
- Example tools include generating users with realistic profiles, supporting customizable input such as count and address inclusion.
- This approach allows autonomous AI agents to interact with your custom tools via standard MCP protocols.

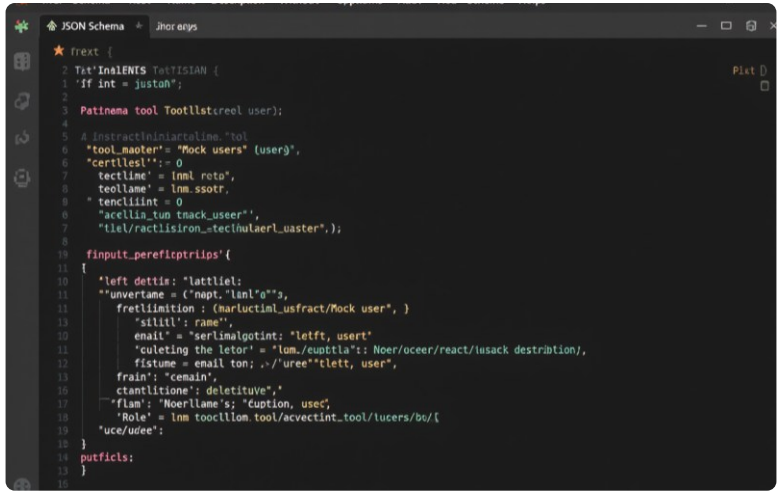


Implementing the Mock Data Server - Code Snippets



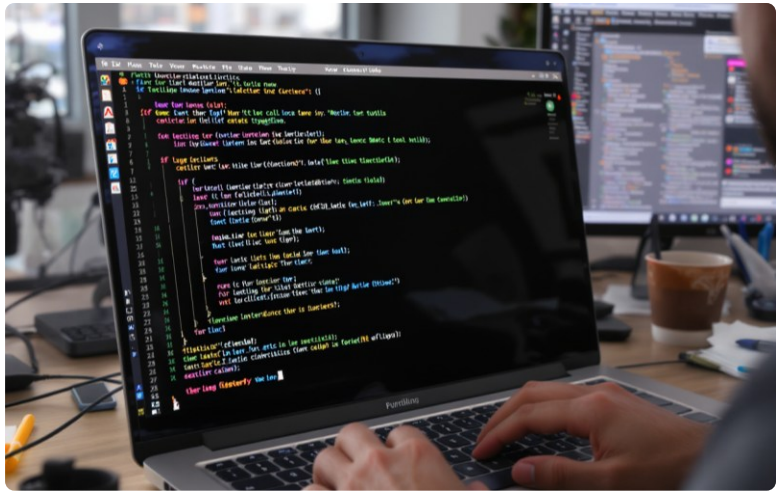
Server Setup

Initialize the MCP server with name, version, and capabilities using the SDK's Server class and StdioServerTransport for communication.



Tools Definition

Define mock data generation tools such as 'generate_users' with detailed input schemas specifying parameters like user count and address inclusion.



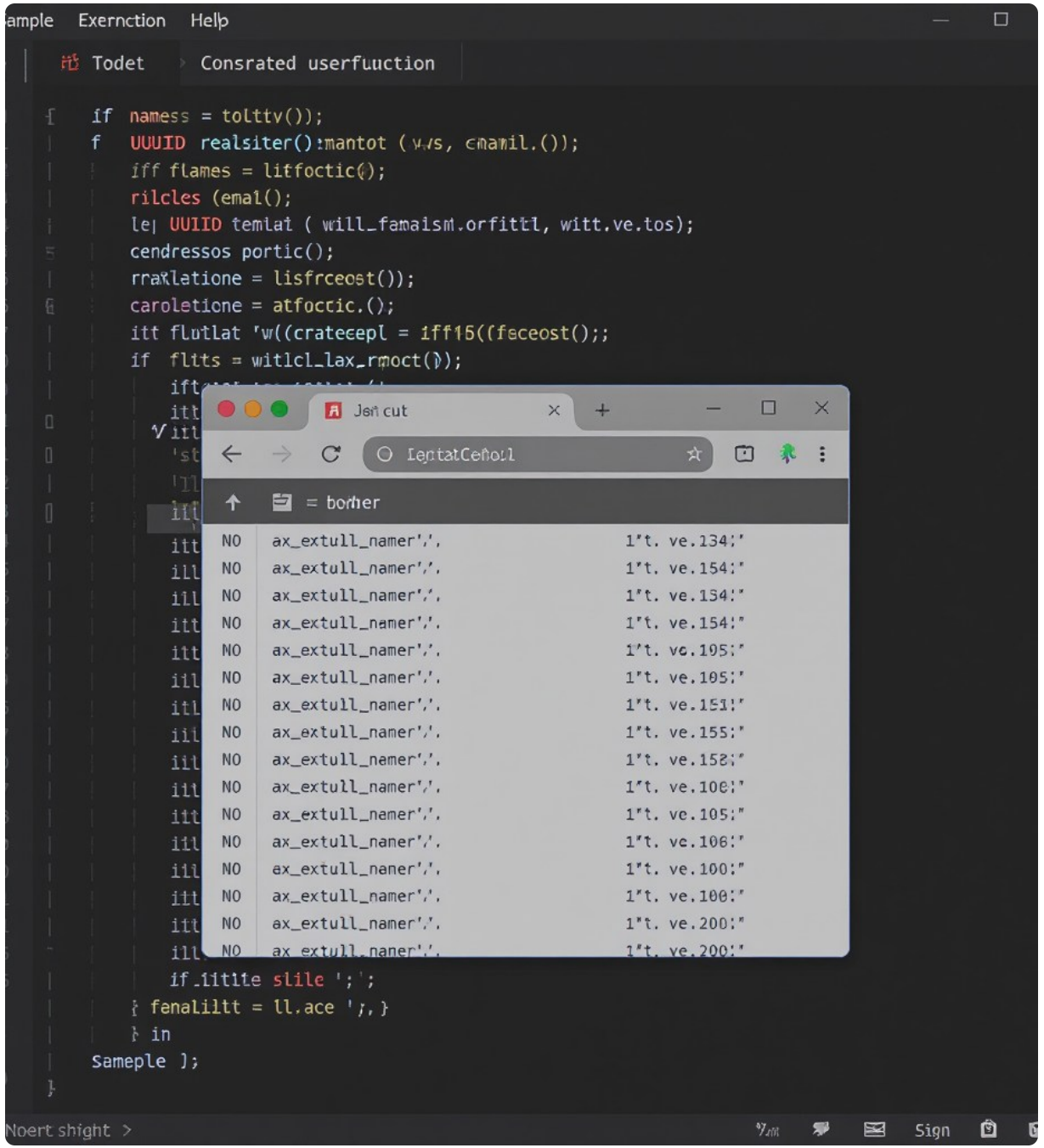
Execution Handler

Implement request handlers for tool execution that process incoming requests, invoke appropriate functions like generateUsers, and handle errors gracefully.

Mock Data Generator Function

generateUsers Function Details

- Generates an array of user objects with unique IDs using crypto.randomUUID().
- Supports input parameters: count (number of users) and includeAddress (boolean to add address info).
- Each user includes fields: id, firstName, lastName, email, age, createdAt.
- Address information is conditionally added with street, city, state, and zip fields.
- Returns user data formatted as JSON text, suitable for MCP tool response.



Configuring the Server

Configuring GitHub Copilot for MCP

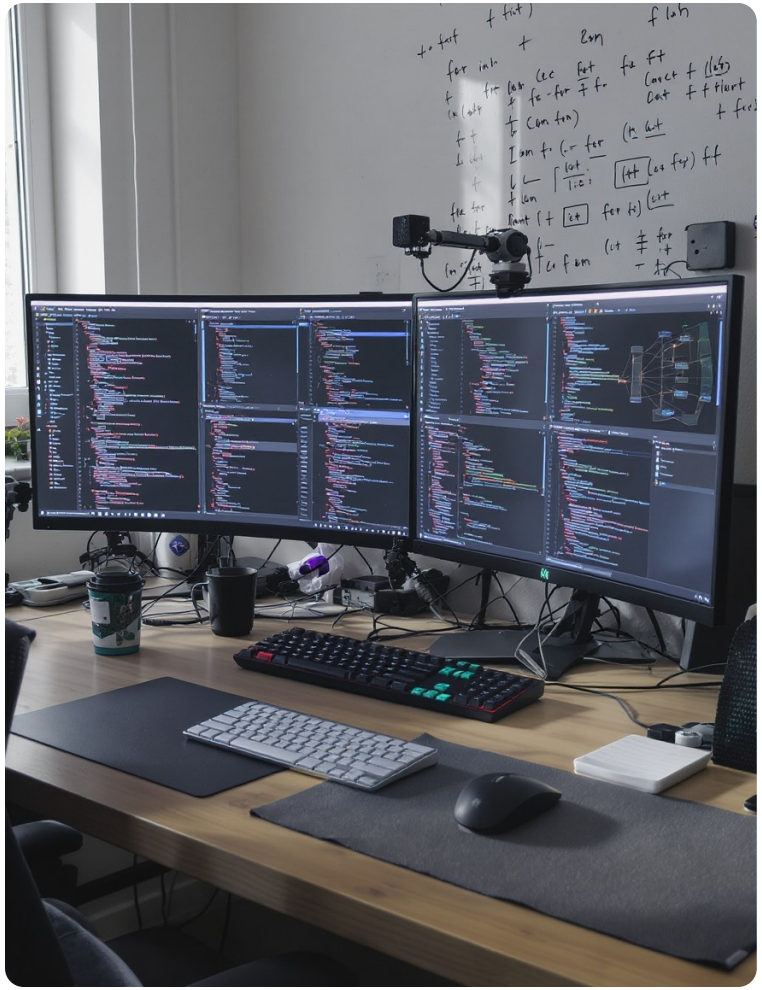
- For GitHub Copilot Chat: Add server config in `.vscode/mcp.json`

```
{
  "servers": {
    "mock-data": {
      "type": "stdio",
      "command": "node",
      "args": ["/workspaces/genai-developer-course/section3/02_mock_data_mcp/index.js"]
    }
  }
}
```

Exercise: Build Your Custom MCP Server

- Part 1: Build Mock Data Server (25 minutes) - Create an MCP server that generates realistic test data for development.
- Implement tools for users, products, transactions, and API responses to simulate comprehensive data workflows.
- Configure your MCP server for integration with GitHub Copilot Chat or Claude Code for seamless AI-powered development.
- Part 2: Use Your Server (10 minutes) - Generate mock data in data.js

Key Takeaways



01 **MCP Standardizes Tool Integration**

MCP provides a universal protocol for all agents, making tools reusable across different AI systems. This community-driven ecosystem represents a future-proof investment for developers.

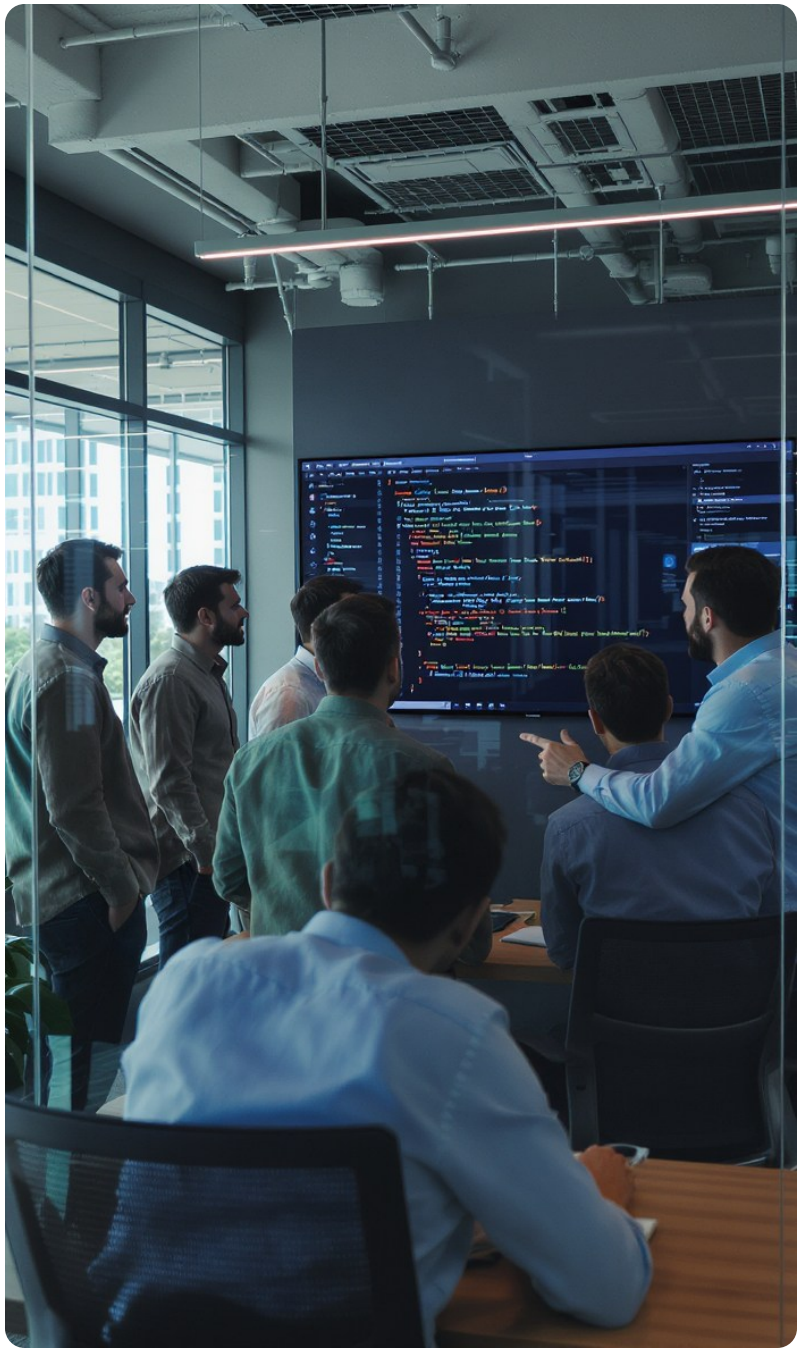
02 **Custom MCP Servers Enable Domain-Specific Tools**

Developers can tailor MCP tools to exact needs, integrating proprietary systems while maintaining security boundaries and optimizing workflows.

Discussion

Questions & Exercise Discussion

- What feature did you implement? Describe the MCP tools or workflows you created.
- What surprised you about agent behavior during development? Share unexpected outcomes or insights.
- What patterns emerged in your workflow? Discuss recurring themes or efficient strategies.
- Did anyone encounter context limitations? How did you manage or mitigate these constraints?
- How did you handle agent mistakes or errors? What safety measures or corrections were effective?

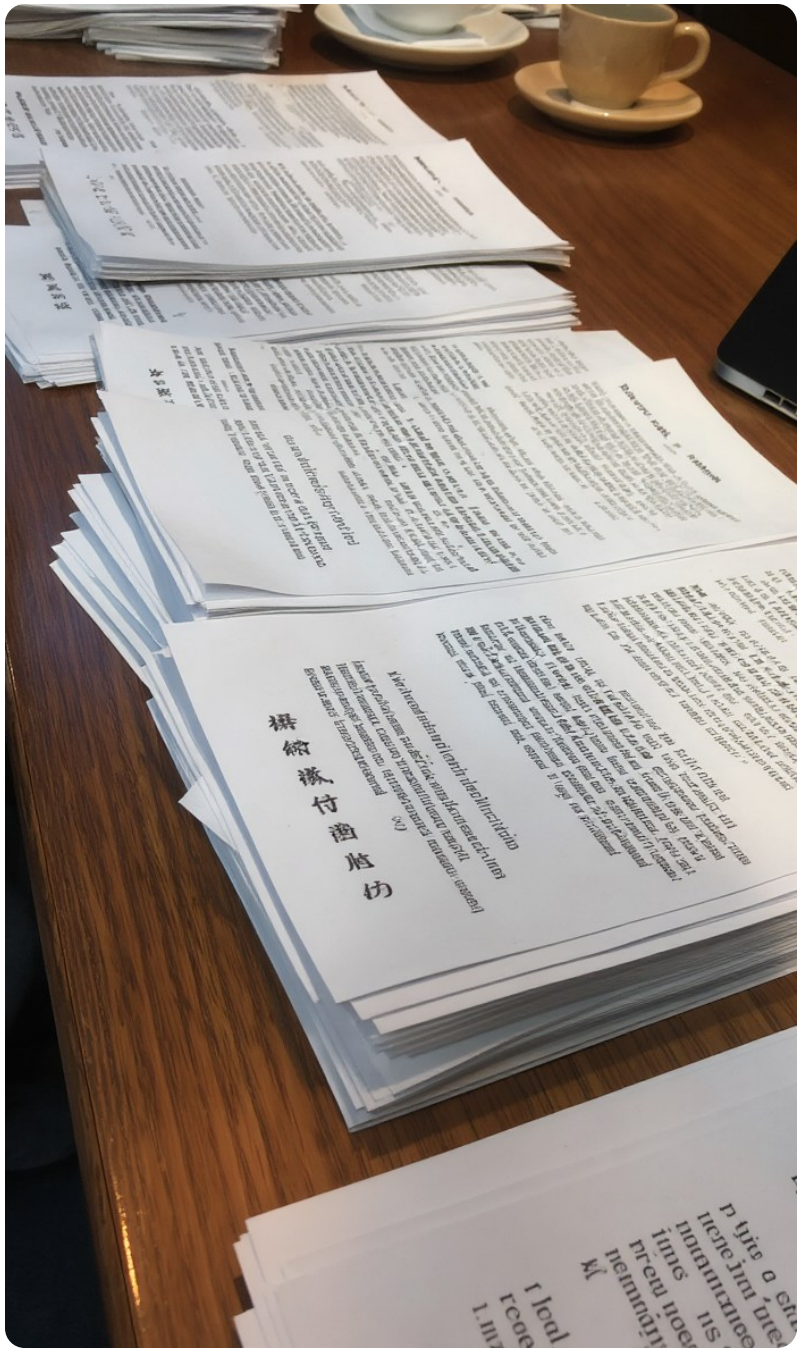


What's Next: Production Considerations

- Strategic model selection tailored for different tasks to optimize performance and costs.
- Cost optimization strategies to manage operational expenses while maintaining efficiency.

References

1. Model Context Protocol Documentation (Anthropic, 2024)
   GitHub Official Repository
   https://github.com/github/github-mcp-server
2. MongoDB MCP Server Announcement
   MongoDB Official Blog
   https://mongodb.com/company/blog/announcing-mongodb-mcp-server
3. Model Context Protocol Specification (Anthropic, 2024)
   MCP Official Documentation
   https://modelcontextprotocol.io/
4. Official MCP SDK Repository (2024)
   Model Context Protocol SDK
   https://github.com/modelcontextprotocol/sdk



Unifi by Intertech

Powered by AI. Guided by You.

Session 4:

Going to Production

Ryan Harvey & Brian Hackerson

August 14th, 2025

Agenda

Model Selection Framework

Choose the right AI model for each development task to optimize cost and performance.

Cost Analysis & Test Generation

Analyze ROI potential and learn how strategic test generation saves time and budget.

Exercise

Hands-on activity to generate tests using strategic model selection for a PaymentProcessor class.

Key Takeaways & Discussion

Summarize key points, review cost savings and ROI, and engage in Q&A discussion.

Challenge

Production Reality: The Strategic Challenge

Your Journey So Far

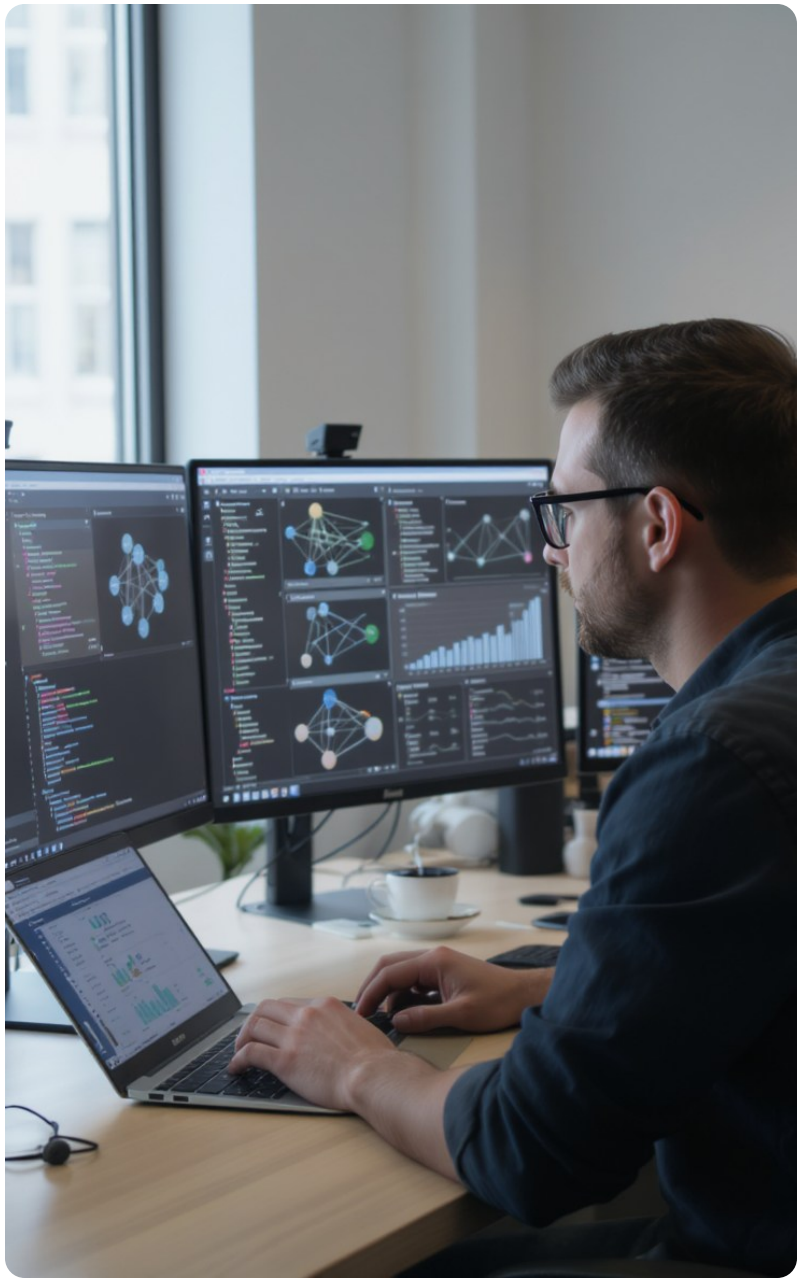
From Sections 1-3, you've learned about prompt engineering, AI agents and tool use, and development with Copilot and Claude Code.

The Strategic Challenge

The key question now is: "Which model should I use, and when?" Choosing the wrong model can mean paying $50/month or $500/month for the same results, impacting both cost and efficiency.

Balancing Cost and Capability

Developers must strategically select models that match task complexity to optimize costs and achieve better results. This session focuses on frameworks and practical test generation to unlock cost savings through smart model selection.



Challenge

The Production Question

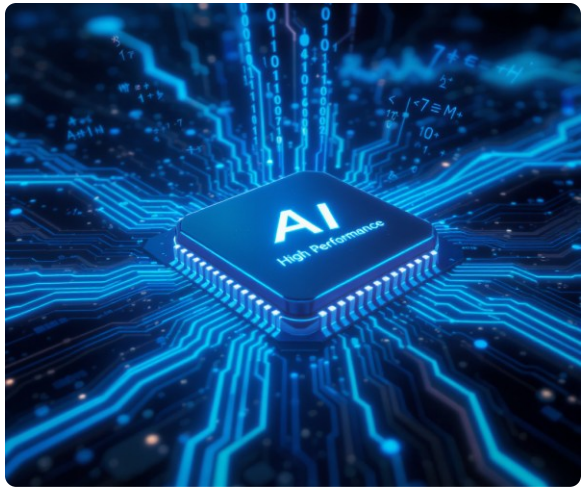
The core production question developers face is: "Which model should I use, and when?"

This decision dramatically affects costs, as the same tasks can range from $50/month to $500/month depending on model choice. Selecting the appropriate AI model balances cost efficiency with task complexity to optimize development resources and outcomes.

The 2025 Model Landscape at a Glance

o3 / Claude Opus

Best for complex reasoning tasks, but slower and highest cost ($$$$). Ideal for architecture and deep debugging.



o4-mini

Optimized for STEM and coding with fast speed and moderate cost ($$). Use for complex tests and algorithms.

GPT-5 & GPT-4.1

GPT-5 is for advanced coding at fast speed and moderate-high cost ($$$). GPT-4.1 supports better coding with fast speed and slightly lower cost ($$+). Both suit most development tasks.

Claude Sonnet 4

Best for structured tasks in complex codebases, offering fast speed and moderate cost ($$).

Advantage

The Strategic Advantage

70% Cost Reduction

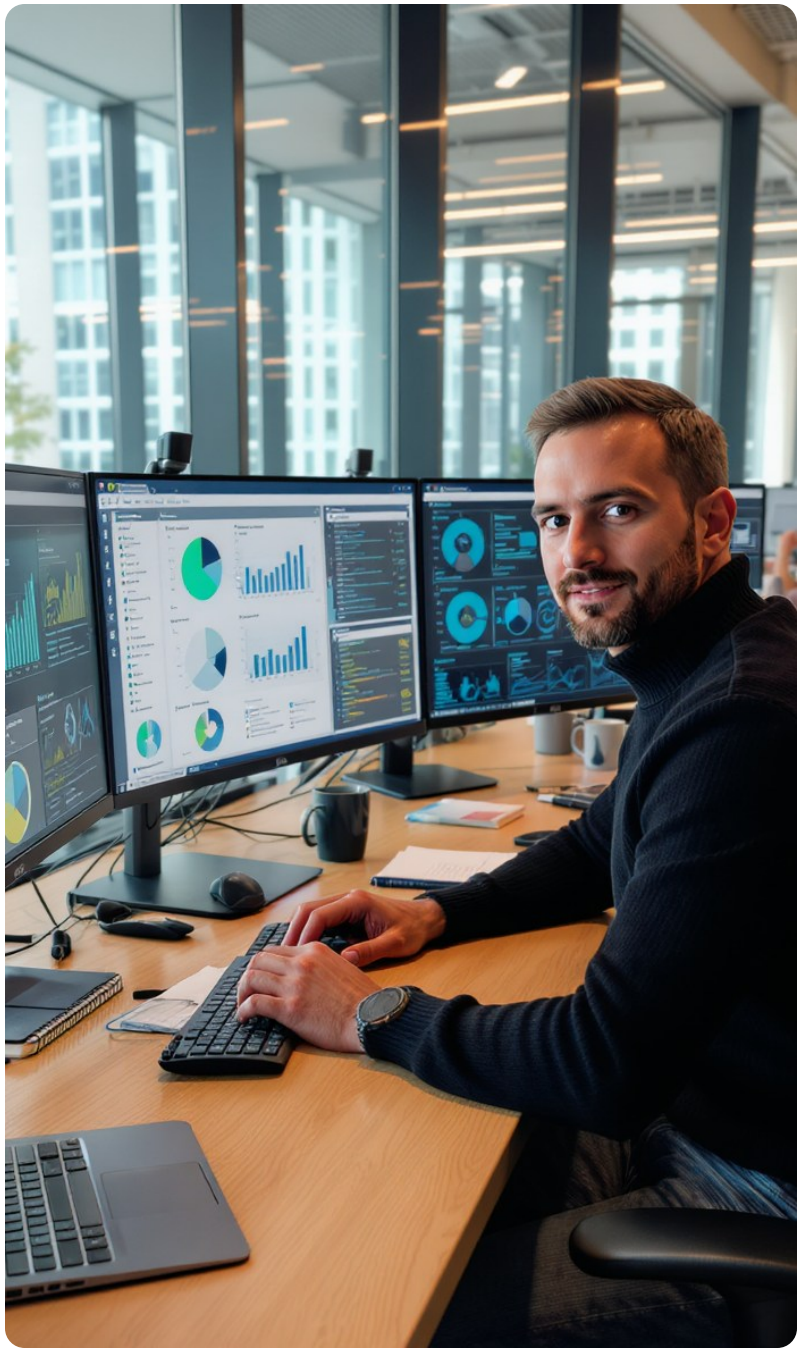
Selecting the right model for each task reduces expenses, significantly lowering monthly AI usage costs compared to using a single high-cost model.

Better Results on Complex Tasks

Using advanced models like o3 or GPT-5 for high complexity tasks ensures superior accuracy and depth, improving outcomes on system architecture, security analysis, and algorithm optimization.

Faster Responses on Simple Tasks

Leveraging faster, lower-cost models such as o3-mini or Claude 3.5 for low complexity tasks speeds up turnaround time without sacrificing quality for formatting, boilerplate, and simple fixes.



Highest ROI Use Case: Test Generation

The Testing Reality

Manual testing is challenging: developers achieve only 30% coverage on average and spend 40% of development time writing tests[1]. Edge cases are frequently missed and tests often become outdated quickly.

AI Benefits in Testing

AI enables thourough test coverage in minutes instead of hours, dramatically reducing manual effort. This accelerates development cycles and improves code quality by catching more edge cases early.

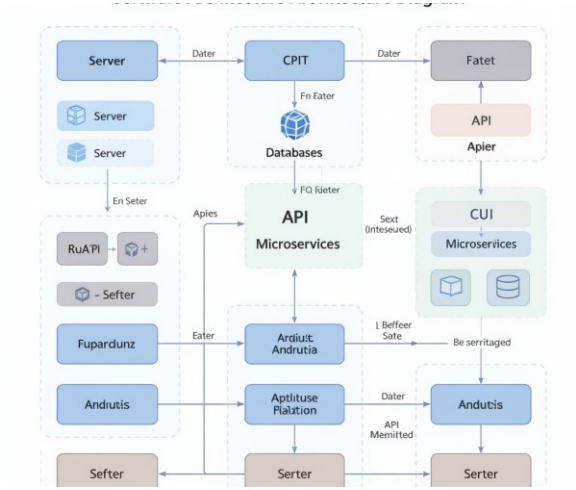
ROI Potential

Test generation with AI can save 8 hours of manual work down to 20 minutes, with API costs around $0.60 creating $400 in value—a 667x ROI. This makes it the highest ROI use case for strategic model selection.

Why Model Selection Matters for Tests

Unit Tests

Use GPT-4.1 for better code patterns



Integration Tests

Use GPT-5 or Claude Sonnet for multi-component testing

Security Tests

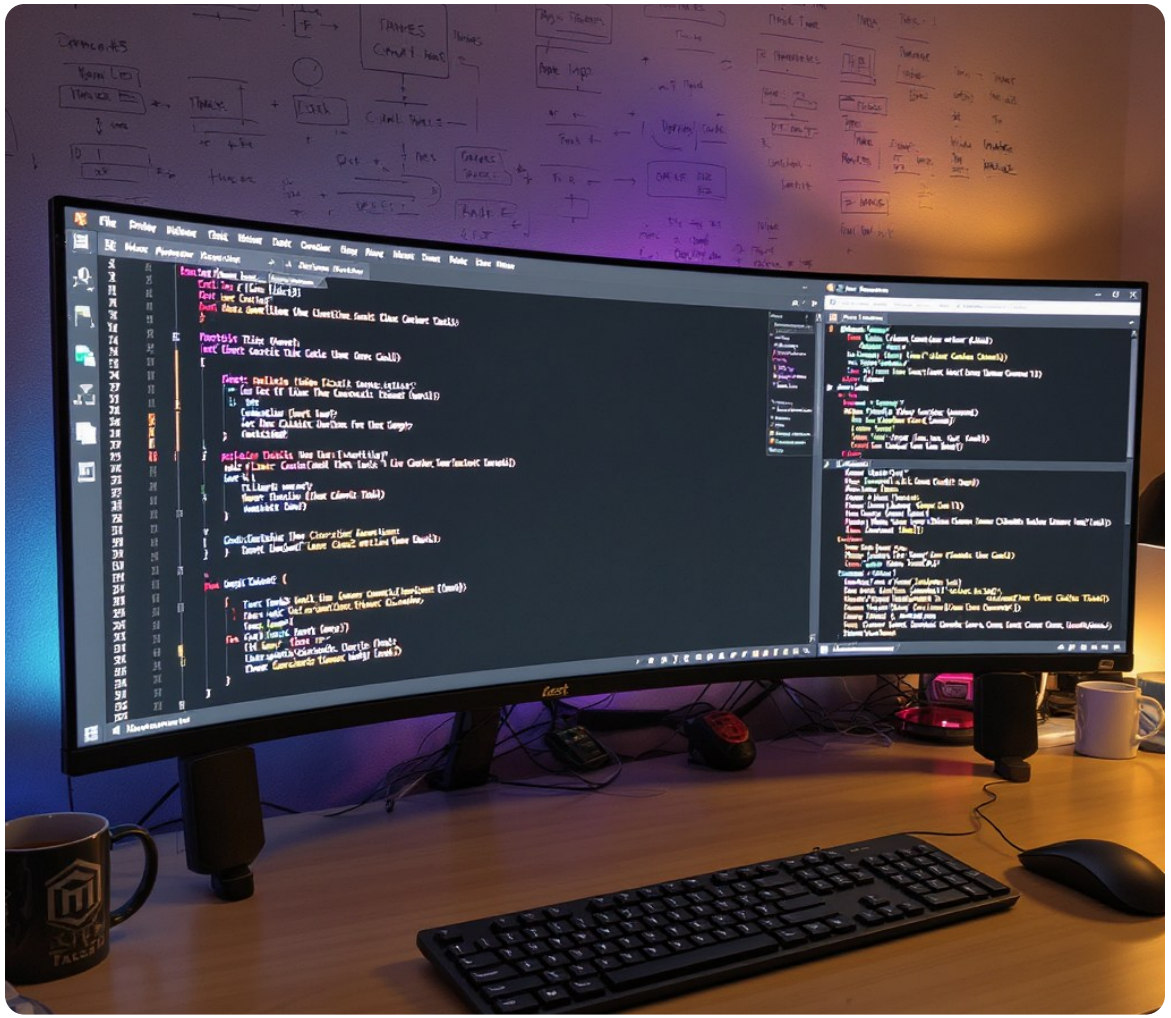
Use GPT-5 or Claude Opus for complex vulnerability patterns

Important Security Testing Disclaimer
AI can help identify common vulnerability patterns but does not replace security experts and tools

Performance Tests

Use GPT-4.1 for metrics-focused testing

Test Generation in Practice: Unit Tests



Example Prompt Pattern for Unit Test Generation

- Context: Payment processing service for an e-commerce platform with high-traffic environment (1000 req/s peak)
- Requirements: Generate comprehensive unit tests using pytest covering all methods, edge cases, and error scenarios
- Include mocking of external dependencies and follow the Arrange-Act-Assert (AAA) pattern for test structure.
-

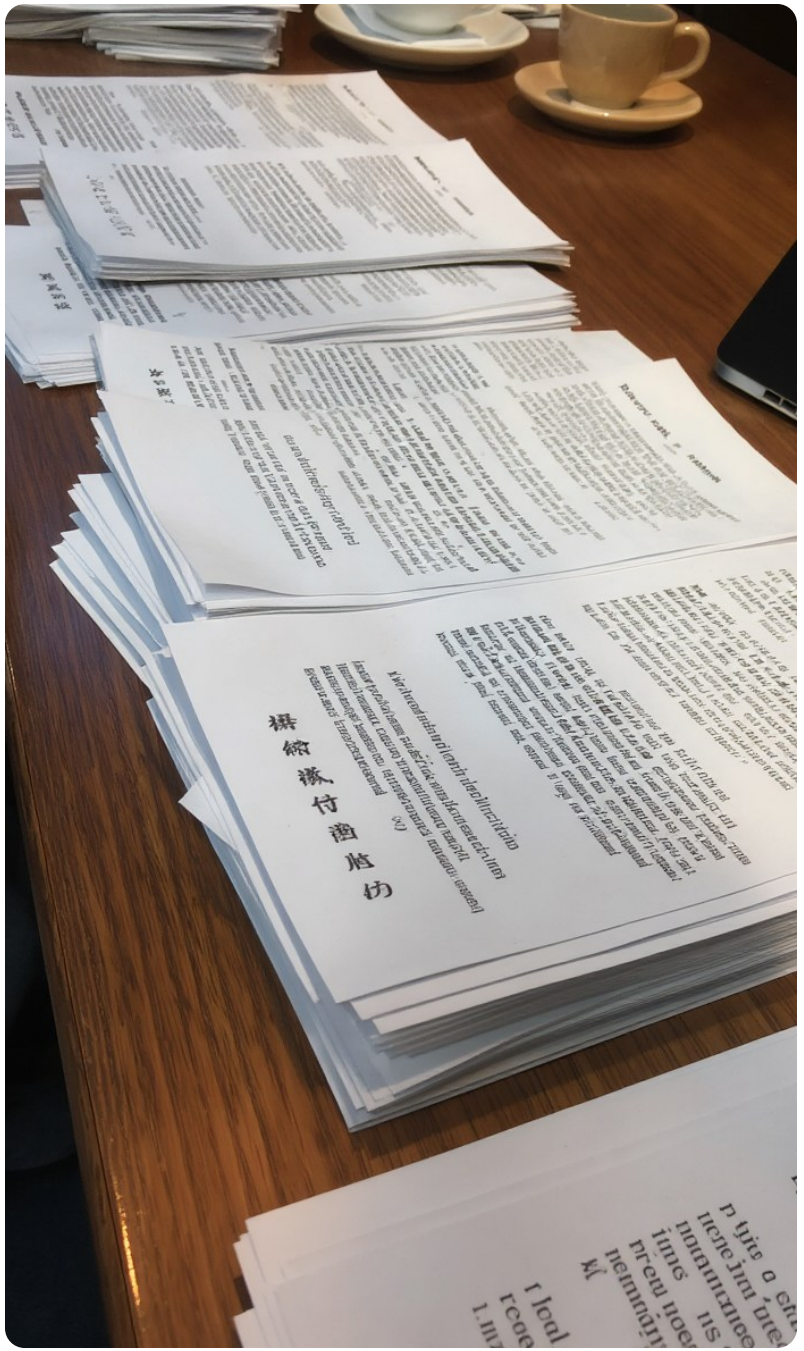
Discussion

Questions & Quick Discussion

- Which model surprised you most in terms of capability or cost?
- What’s your estimated ROI for test generation using strategic model selection?
- What will you implement first based on today’s learnings?
- How do you plan to measure success and adjust your model use over time?
- What challenges do you foresee in applying these models in your development workflow?

References

1. State of Software Testing Report (2024)
   PractiTest & Tea-Time with Testers
   https://www.practitest.com/state-of-testing/
2. MongoDB MCP Server Announcement
   MongoDB Official Blog
   https://mongodb.com/company/blog/announcing-mongodb-mcp-server
3. Model Context Protocol Specification (Anthropic, 2024)
   MCP Official Documentation
   https://modelcontextprotocol.io/
4. Official MCP SDK Repository (2024)
   Model Context Protocol SDK
   https://github.com/modelcontextprotocol/sdk



Unifi by Intertech

Powered by AI. Guided by You.

Session 5:

Course Wrapup

Ryan Harvey & Brian Hackerson

August 14th, 2025

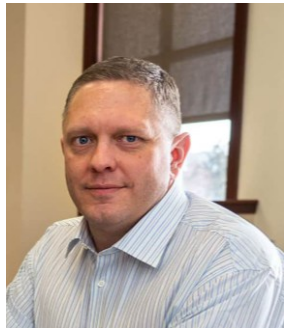
Unifi by Intertech

Powered by AI. Guided by You.

Final Q&A


SIMULATION ACTIVE
INTERTECH
Unifi
Powered by AI. Guided by You.

Let Us Know How We Can Help!



Ryan Harvey
Senior Consultant

https://www.linkedin.com/in/ryan-harvey-32248114/
rharvey@intertech.com

Brian Hackerson
Delivery Manager

https://www.linkedin.com/in/bhackerson/
bhackerson@intertech.com

Course Evaluation: GenAI & Prompt Engineering

Unifi by Intertech

Powered by AI. Guided by You.

Thank you for attending!