Generative AI for Programming

Overview

What We'll Cover Today

- What is generative AI and how does it work?
- Effective prompting strategies for code generation
- Evaluating AI-generated code
- Best practices and limitations
- Ethical considerations
- Practical applications

What is Generative AI?

Core Concepts

Generative AI - AI systems that can create new content (text, code, images) based on patterns learned from training data

Large Language Models (LLMs) - AI trained on vast amounts of text to predict and generate human-like responses

Examples:

- ChatGPT (OpenAI)
- Claude (Anthropic)
- GitHub Copilot
- Google Gemini

How Does It Work?

Pattern Recognition & Prediction

1. **Training Phase**
   - Model learns from billions of examples
   - Recognizes patterns in code structure, syntax, and common solutions
2. **Generation Phase**
   - Takes your prompt as input
   - Predicts most likely next tokens (words/characters)
   - Generates coherent, contextually appropriate code

Key Insight: AI doesn't "understand" code like humans do - it recognizes statistical patterns

Think-Pair-Share

Reflection Question

What programming tasks do you think AI would be good at? What tasks might it struggle with?

Capabilities & Limitations

What AI Does Well

- **Boilerplate code** - repetitive patterns
- **Standard algorithms** - common solutions
- **Code translation** - between languages
- **Documentation** - comments and explanations
- **Debugging help** - identifying common errors

Capabilities & Limitations

Current Limitations

- **Novel algorithms** - truly original solutions
- **Complex architecture** - system design decisions
- **Domain expertise** - specialized knowledge
- **Context understanding** - large codebases
- **Security** - subtle vulnerabilities
- **Edge cases** - unusual scenarios

Effective Prompting Strategies

The PREP Framework

Precise - Be specific about what you want

Relevant - Provide context and constraints

Examples - Include input/output samples

Process - Break complex tasks into steps

Prompting Strategy 1: Be Precise

Vague vs. Specific Prompts

Vague: "Write a function to sort"

Specific: "Write a Java function that sorts an array of integers in ascending order using the quicksort algorithm. Include comments explaining the partitioning step."

Key Elements:

- Programming language
- Data types
- Algorithm choice
- Output requirements
- Documentation needs

Prompting Strategy 2: Provide Context

Add Relevant Information

Without Context: "Create a search function"

With Context: "Create a binary search function in Java for a sorted array of Student objects. Students should be compared by their ID (integer). Return the index if found, -1 if not found."

What to Include:

- Data structures involved
- Constraints (sorted, unique, etc.)
- Expected behavior
- Return values

Prompting Strategy 3: Include Examples

Input/Output Samples

```
Create a Java method that validates email addresses.

Examples:
- "user@example.com" → true
- "invalid.email" → false
- "test@domain.co.uk" → true
- "@example.com" → false

Requirements:
- Must have @ symbol
- Must have domain with extension
- Must have username before @
```

Active Learning: Write a Prompt

Write a prompt to generate a Java method that:

- Finds the maximum value in an ArrayList of doubles
- Returns -1 if the list is empty
- Include at least 2 test cases in your prompt

Prompting Strategy 4: Process (Break It Down)

Iterative Development

Complex Task: "Create a student grade management system"

Better Approach:

1. "Create a Student class with fields for name, ID, and grades"
2. "Add a method to calculate GPA"
3. "Create a StudentDatabase class to store multiple students"
4. "Add search functionality by student ID"

Benefit: Easier to verify and debug each component

Advanced Prompting Techniques

Chain of Thought Prompting

Encourage AI to "think through" the problem:

```
"Before writing the code, explain your approach:
1. What algorithm will you use and why?
2. What are the edge cases to consider?
3. What is the time complexity?

Then provide the implementation."
```

Result: More thoughtful, better-documented code

Advanced Prompting Techniques

Role-Based Prompting

Frame the AI's perspective:

```
"You are a senior Java developer reviewing code
for a banking application. Write a secure password
validation function that:
- Requires 12+ characters
- Includes uppercase, lowercase, digit, special char
- Explain security considerations"
```

Benefit: Encourages best practices and security awareness

Evaluating Generated Code

Correctness

- Does it compile?
- Does it handle edge cases?
- Are there logical errors?

Efficiency

- What's the time/space complexity?
- Are there obvious optimizations?

Style

- Does it follow conventions?
- Is it readable and maintainable?

Evaluating Generated Code

Testing is Essential

Never trust AI output without verification

1. **Write test cases** before accepting code
2. **Test edge cases** - empty inputs, nulls, boundaries
3. **Verify algorithms** - trace through with sample data
4. **Check assumptions** - are invariants maintained?

Remember: AI can confidently produce wrong code!

Think-Pair-Share

Code Review Exercise

Look at this AI-generated code:

```
public int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

What issues might exist? What test cases would you write?

Common Issues in AI Code

Logic errors

- Off-by-one errors
- Incorrect base cases
- Wrong operators

Missing edge cases

- Null handling
- Empty collections
- Negative numbers
- Integer overflow

Performance issues

- Inefficient algorithms
- Unnecessary operations
- Poor data structure choices

Security vulnerabilities:

- SQL injection risks
- Buffer overflows
- Unvalidated input

Best Practices

Using AI as a Programming Tool

DO:

- Use for learning and exploration
- Generate boilerplate and tests
- Get explanations of unfamiliar code
- Brainstorm alternative approaches
- Create documentation

DON'T:

- Copy-paste without understanding
- Skip testing
- Ignore security implications
- Use for critical systems without review
- Assume it's always correct

Best Practices

Iterative Refinement

Follow-up prompting strategy:

1. Generate initial solution
2. Ask for explanation: "Explain how this code works"
3. Request improvements: "Optimize for time complexity"
4. Add features: "Add error handling for X"
5. Generate tests: "Write JUnit tests for this method"

Key: Engage in a dialogue, don't accept first output

Active Learning Exercise

Prompt Engineering Challenge

Scenario: You need a Java method to merge two sorted arrays

1. Write an initial prompt
2. Review generated code (if you have access to AI)
3. Write a follow-up prompt to improve it
4. Identify what test cases you'd need

Academic Integrity

Ethical Considerations

Attribution & Honesty

- Be transparent about AI use
- Cite when required by assignment
- Don't misrepresent your work

Learning vs. Shortcuts

- AI should enhance learning
- Struggle is part of the process
- Build genuine understanding

Professional Ethics

- Employers expect authentic skills
- Your portfolio should reflect your abilities

Real-World Applications

Industry Uses of AI

Code Generation

- GitHub Copilot in IDEs
- Automated test generation
- Boilerplate reduction

Code Review & Analysis

- Bug detection
- Security vulnerability scanning
- Code quality suggestions

Documentation

- Automatic comment generation
- API documentation
- Code explanations

Real-World Applications

AI in Professional Development

Pair programming assistant:

- Suggests completions
- Offers alternative approaches
- Identifies potential bugs

Learning tool:

- Explains unfamiliar code
- Provides examples
- Teaches new languages/frameworks

Productivity multiplier:

- Speeds up routine tasks
- Reduces context switching
- Handles repetitive work

The Future of AI in Programming

Likely developments:

- Better context understanding
- Stronger reasoning capabilities
- Integration with development tools
- Specialized domain models

What won't change:

- Need for human judgment
- Importance of understanding fundamentals
- Value of problem-solving skills
- Critical thinking requirements

Active Learning: Scenario Analysis

Break into groups of 3-4

Each group gets a scenario:

1. Using AI to complete a homework assignment
2. AI assistance during technical interview prep
3. AI-generated code in a production system
4. Using AI to learn a new programming concept

Discuss:

- When is AI use appropriate?
- What risks exist?
- How to use AI effectively?

Tips for Success

Start simple:

- Begin with well-defined problems
- Build complexity gradually
- Learn the tool's strengths

Stay engaged:

- Don't become passive
- Question the output
- Maintain critical thinking

Keep learning:

- Use AI to accelerate, not replace learning
- Explore AI suggestions to learn new patterns
- Understand the "why" behind the code

Practice prompting:

- Experiment with different phrasings
- Learn from successful prompts
- Build a prompt library

Tips for Success

Develop judgment:

- Learn to spot errors quickly
- Understand common pitfalls
- Trust but verify

Maintain fundamentals:

- Strong foundation in CS theory
- Algorithm knowledge
- Problem-solving skills

Key Takeaways

Essential Points

1. **AI is a tool**, not a replacement for understanding
2. **Effective prompting** requires clarity, context, and examples
3. **Always evaluate** generated code critically
4. **Test thoroughly** - AI makes mistakes
5. **Use ethically** - maintain academic integrity
6. **Keep learning** - AI should enhance, not replace skill development