Generative AI for Programming

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): http://chatgpt.com
- Google Gemini: http://gemini.google.com
- Claude (Anthropic): http://claude.ai (requires email)

How Does It Work?

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

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

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!

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

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

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

Scenario: generating permutations

Model: you choose

Initial prompt

```
Write a function to generate all permutations of a list
```

Active Learning

Refine to be more specific

```
Use Java as the programming language.
The list should contain items of generic type T.
```

Active Learning

Specify structure and constraints

```
Use the following structure:
class Permuter<T extends Comparable<T>> {
    List<List<T>> generate(List<T> items) {...}
}
Also, handle duplicates in the list if items.
```

Active Learning

Add style and documentation

```
Add Javadoc explaining the approach and time and space complexity.
Add inline comments for non-obvious logic.
Follow Google Java style guidelines.
```

Active Learning

Add testing

```
Generate a comprehensive test suite using JUnit.
The test cases should cover:
    - Empty list
    - Single item list
    - list with unique items
    - list with duplicates
Each test should have a descriptive name and a comment explaining what it validates
```

Active Learning

Explore scaling

```
When I try to run the code with a list of 15 items I run out of memory.
How can I change the solution so that it uses less memory?
```

Active Learning

Explore alternatives

```
What are some alternative solutions? Compare the alternatives.
```

Key observations:

- Specificity beats brevity: Detailed prompts get better results
- Context is crucial: Provide data structures, constraints, and requirements
- Iterate and refine: Treat prompting as an iterative development process
- Request explanations: Ask the AI to explain its reasoning
- Verify and validate: AI-generated code still needs human review
- Use AI as a teaching tool: Ask for alternatives, trade-offs, and deeper understanding

> Prompt engineering is really about clearly communicating requirements (iteratively)

Recommended Videos

- Practical AI for Instructors and Students, Ethan & Lilach Mollick (Wharton School)
    - https://www.youtube.com/watch?v=t9gmyvf7JYo&list=PLwRdpYzPkkn302_rL5RrXvQE8j0jLP02j&index=1
- AI Cheerleaders Are Entirely Too Unambitious, Sendhil Mullainathan (MIT)
    - https://alum.mit.edu/forum/video-archive/ai-cheerleaders-unambitious

Word of Caution

https://www.media.mit.edu/publications/your-brain-on-chatgpt/