10 AI Prompts to 10x Your Coding Speed

Battle-Tested Prompts • Real Examples • Pro Tips

END OF CODING
Build Apps with AI • No Code Experience Needed

Master AI-Assisted Development

These 10 battle-tested prompts will transform how you code with AI. Each prompt has been refined through hundreds of real projects to deliver maximum value. Copy these templates, customize them for your needs, and watch your productivity soar.

Prompt #1: Project Scaffolding from Scratch

When to Use: Starting a new project and need complete setup

The Prompt:

```
Create a [project type] using [tech stack]. Include:
- Complete project structure with all necessary folders
- Package.json with all dependencies
- Environment variable template (.env.example)
- README with setup instructions
- Basic configuration files (tsconfig, eslint, etc.)
- Initial routing structure
- Authentication boilerplate if needed

Make it production-ready with best practices.
```

Example Use: React + TypeScript + Supabase SaaS starter

Pro Tips: Be specific about features you need upfront. Mention any specific packages or patterns you prefer.

Prompt #2: Debugging Complex Errors

When to Use: Stuck on a bug and error messages aren't helping

The Prompt:

```
I'm getting this error: [paste full error]

Here's the relevant code:
[paste code]

Context:
- What I'm trying to do: [explain goal]
- What's happening instead: [describe behavior]
- What I've tried: [list attempts]
- Environment: [framework versions, OS, etc.]

Please explain the root cause and provide a fix with explanation.
```

Example Use: React hydration mismatch in Next.js app

Pro Tips: Include full error stack trace. Provide context about recent changes. Mention environment details.

Prompt #3: Code Review & Optimization

When to Use: Code works but needs improvement

The Prompt:

```
Review this code for:
1. Performance bottlenecks
2. Security vulnerabilities
3. Best practices violations
4. Code smells and anti-patterns
5. Opportunities for optimization

[paste code]

Provide specific improvements with before/after examples. Prioritize by impact.
```

Example Use: Optimizing a React component with multiple useEffect hooks

Pro Tips: Ask for ranked recommendations. Request performance metrics if relevant. Mention your target audience/scale.

Prompt #4: Writing Comprehensive Tests

When to Use: Need test coverage for new or existing code

The Prompt:

```
Write comprehensive tests for this code:
[paste code]

Include:
- Unit tests for all functions
- Integration tests for key workflows
- Edge cases and error scenarios
- Mocking external dependencies
- Setup and teardown

Use [testing framework] and follow [testing pattern] principles.
```

Example Use: Jest tests for API route handlers in Next.js

Pro Tips: Specify your testing framework. Mention coverage goals. Include example of your testing style if you have one.

Prompt #5: API Integration

When to Use: Integrating with external APIs or creating endpoints

The Prompt:

```
Help me integrate [API name]:

Requirements:
- [List specific endpoints needed]
- Authentication method: [OAuth/API key/JWT]
- Response handling and error cases
- TypeScript types for responses
- Rate limiting handling
- Caching strategy if applicable

Provide complete implementation with error handling and types.
```

Example Use: Stripe payment integration with webhook handling

Pro Tips: Include API docs link. Mention authentication details. Specify error handling requirements.

Prompt #6: Database Schema Design

When to Use: Designing database structure for new features

The Prompt:

```
Design a database schema for [feature description].

Requirements:
- Entities needed: [list main objects]
- Relationships: [describe how they connect]
- Scale expectations: [users/data volume]
- Query patterns: [common queries]
- Database: [Postgres/MySQL/MongoDB]
```

```
Include:
- Table definitions with types
- Indexes for performance
- Constraints and validations
- Migration file
- Sample queries
```

Example Use: Multi-tenant SaaS with organizations and team members

Pro Tips: Think through relationships carefully. Mention scale requirements. Include common query patterns.

Prompt #7: Refactoring Legacy Code

When to Use: Modernizing old code without breaking functionality

The Prompt:

```
Refactor this code to modern best practices:
[paste legacy code]

Goals:
- Maintain exact same functionality
- Improve readability and maintainability
- Add TypeScript types if missing
- Remove code smells
- Add comments for complex logic
- Keep backward compatibility with [specify]

Show step-by-step refactoring with explanations.
```

Example Use: Class components to React hooks

Pro Tips: Emphasize maintaining functionality. Ask for incremental steps. Specify any constraints.

Prompt #8: Documentation Generation

When to Use: Need comprehensive docs for code or APIs

The Prompt:

```
Generate complete documentation for:
[paste code/API]

Include:
- Overview and purpose
- Installation/setup steps
- API reference with all parameters
- Usage examples (basic and advanced)
- Common pitfalls and troubleshooting
- TypeScript types/interfaces
- Code examples that work

Target audience: [developers/non-technical/etc.]
```

Example Use: React component library documentation

Pro Tips: Specify your audience level. Request code examples. Mention documentation format (Markdown/JSDoc/etc.).

Prompt #9: Security Audit

When to Use: Checking code for security vulnerabilities

The Prompt:

```
Perform a security audit on this code:
[paste code]

Check for:
- SQL injection vulnerabilities
- XSS attack vectors
- CSRF protection
- Authentication/authorization issues
- Sensitive data exposure
- Dependency vulnerabilities
- Rate limiting needs
- Input validation gaps

Rank by severity with fix recommendations.
```

Example Use: User authentication flow security review

Pro Tips: Include all user-facing code. Mention your framework's built-in protections. Ask for severity ranking.

Prompt #10: Performance Optimization

When to Use: App is slow and needs performance improvements

The Prompt:

```
Optimize this code for performance:
[paste code]

Current issues:
- [describe slow behavior]
- Metrics: [load time, etc.]
- Bottlenecks: [if known]

Target:
- [desired performance goal]
- [constraints: bundle size, etc.]
```

```
Analyze and provide optimizations with expected impact. Include measurement strategy.
```

Example Use: React app with slow initial load

Pro Tips: Include actual metrics if available. Mention your performance goals. Ask for measurement recommendations.

Prompt Engineering Pro Tips

Be Specific: Vague prompts get vague results. Include framework versions, constraints, and exact requirements.

Provide Context: Share what you've tried, error messages, and relevant code. More context = better solutions.

Request Structure: Ask for step-by-step explanations, numbered lists, or before/after examples for clarity.

Iterate: Start broad, then refine. Don't expect perfection on the first try.

Set Constraints: Mention bundle size limits, browser support, performance goals, or other constraints upfront.

Ask for Explanations: Request 'explain why' to learn, not just 'show me how' to copy.

Use Examples: If you have a preferred coding style, include an example so AI matches your patterns.

Specify Output Format: Want TypeScript? Comments? Tests? Spell it out in your prompt.