Prompt Engineering Mega-Pack

100+ battle-tested prompts for developers · GPT-4, Claude, Gemini

Code Generation

■ Generate with tests

```
Write a {language} function that {task}.

Requirements:
- Include comprehensive unit tests
- Handle edge cases: {list edge cases}
- Follow {style guide} conventions
- Add JSDoc/docstring comments

Output format:
1. Implementation
2. Tests
3. Usage example
```

■ Refactor legacy code

```
Refactor this code to follow SOLID principles.
Preserve all existing behavior (no functional changes).

Prioritize:
1. Single Responsibility
2. Dependency Injection
3. Extract interfaces for testability

For each change, explain WHY in a comment.

```{language}
{paste code}
```
```

■ Generate API endpoint

```
Create a REST API endpoint:

Method: {GET/POST/PUT/DELETE}
Path: /api/{resource}
Auth: {JWT/API key/none}

Request body: {schema}
Response: {schema}

Include:
- Input validation with error messages
- Rate limiting headers
- OpenAPI/Swagger doc comments
- Integration test
```

Debugging & Analysis

■ Root cause analysis

```
I'm seeing this error:
```
```
```

````
{error message}
```

Context:
- Language/framework: {tech}
- What I was doing: {action}
- What I expected: {expected}
- What happened: {actual}

Analyze possible root causes ranked by
likelihood. For each, give a diagnostic
command or code snippet to confirm.
````

■ Performance review

````
Review this code for performance issues:

```{language}
{code}
```

Analyze:
1. Time complexity (current vs optimal)
2. Memory usage
3. I/O bottlenecks
4. Caching opportunities
5. Specific fix for each issue found
````

■ Security audit prompt

````
Audit this code for security vulnerabilities:

```{language}
{code}
```

Check for:
- Injection (SQL, XSS, command)
- Auth/authz issues
- Data exposure
- OWASP Top 10 violations

For each finding:
- Severity: Critical/High/Medium/Low
- Attack scenario
- Fix with code example
````

Documentation

■ Generate README

````
Write a README.md for this project:

Name: {name}
Purpose: {what it does}
Tech stack: {technologies}
Audience: {who uses it}

Include these sections:
- Badges (build, version, license)
- One-paragraph description
````
````

```
- Quick start (< 5 steps to running)
- Configuration table
- API reference (if applicable)
- Contributing guide
- License
```

■ Architecture decision record

```
Write an ADR (Architecture Decision Record):

Title: {decision title}
Context: {why this decision is needed}
Options considered: {list options}
Decision: {what was chosen}

Format: Michael Nygard's ADR template
Include consequences (positive & negative)
and metrics to evaluate the decision.
```

System Design

■ Design a system

```
Design a {system type} that handles:
- {requirement 1}
- {requirement 2}
- Scale: {users/requests/data volume}

Cover:
1. High-level architecture diagram (ASCII)
2. Data model
3. API design
4. Key algorithms
5. Scaling strategy
6. Failure modes & mitigations
7. Cost estimate (AWS/GCP)
```

■ Database schema design

```
Design a database schema for {application}:

Entities: {list main entities}
Relationships: {describe relationships}
Query patterns: {list common queries}
Scale: {expected data volume}

Provide:
1. ERD (ASCII)
2. CREATE TABLE statements
3. Indexes for query patterns
4. Migration strategy from {current}
```

AI/LLM Integration

■ Build AI feature

```
I want to add {AI feature} to my {app type}.

Tech stack: {current stack}
```

```
Budget: {cost constraints}
Latency requirement: {max response time}

Design the integration:
1. Which model/API to use and why
2. Prompt template with variables
3. Error handling & fallbacks
4. Caching strategy
5. Cost estimation per 1K requests
6. Code implementation
```

■ Structured output extraction

```
Extract structured data from this text:

"{paste text}"

Output JSON schema:
{define expected schema}

Rules:
- Use null for missing fields
- Normalize {specific fields}
- Validate {constraints}
- Include confidence score (0-1)
```

DevOps & Infrastructure

■ CI/CD pipeline

```
Create a {GitHub Actions/GitLab CI} pipeline:

Project type: {language/framework}
Stages needed:
- Lint & format check
- Unit tests (parallel)
- Integration tests
- Build artifact
- Deploy to {staging/prod}

Include:
- Caching for dependencies
- Matrix testing (OS/versions)
- Secrets management
- Slack/Discord notification
- Manual approval for prod
```

■ Infrastructure as Code

```
Write {Terraform/Pulumi} for:

Cloud: {AWS/GCP/Azure}
Resources:
- {list resources needed}

Requirements:
- Environment separation (dev/staging/prod)
- Least privilege IAM
- Encryption at rest
- Monitoring & alerts
- Cost tags
```

```
- Remote state with locking
```

Meta-Prompting (Prompts About Prompts)

■ Improve any prompt

```
Improve this prompt for better results:

"{original prompt}"

Apply these techniques:
1. Add specific constraints
2. Include output format
3. Add few-shot examples
4. Specify what NOT to do
5. Add evaluation criteria

Return the improved prompt and explain
each change.
```

■ Chain-of-thought template

```
Solve {problem type} step by step.

Before answering:
1. Identify the core question
2. List what you know
3. List what you need to figure out
4. Work through the logic
5. Verify your answer
6. State confidence level

Show your reasoning at each step.
If uncertain, say so explicitly.
```

■ Usage Tips

Fill in the {placeholders} — these prompts are templates. Customize for your context.

Chain prompts — use output from one prompt as input to another for complex tasks.

Iterate — if the first result isn't right, add constraints rather than starting over.

Works with any model — tested with GPT-4, Claude 3.5/4, Gemini 2.0, and open-source models.

Save your customized versions — build a personal prompt library over time.