AI Automation / AI Agents

Build an agent that reviews pull requests for style, bugs, and security issues before human review.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Code Quality, PR Review Automation, Dev Productivity
Updated: May 2026
Why This Prompt Exists
Code review is essential but time-consuming. AI can catch common issues — leaving human reviewers to focus on architecture and design.

You get:

- human reviewers spending time on style nitpicks (should be automated)
- common bugs that could have been caught pre-review
- inconsistent style across the codebase (no automation)
- security vulnerabilities that slip through
- slow review cycles (waiting for humans to catch trivial issues)

But code review agents can help:

- style checking: formatting, naming, linter rules
- bug detection: null pointer, off-by-one, unclosed resources
- security scanning: injection risks, hardcoded secrets, unsafe functions
- test coverage: which lines are untested?
- complexity warnings: functions that are too long or nested

Without automation, human reviewers drown in trivial issues.

This prompt designs effective code review assistant agents.

The Prompt

```
Assume the role of a code review automation architect who designs AI
review assistants.

Your task is to create an agent that reviews pull requests
automatically.

Generate:

1. CODEBASE CONTEXT
   - Languages: [e.g., Python, TypeScript, Go]
   - Frameworks: [e.g., React, Django, FastAPI]
   - Style guide: [e.g., PEP 8, Airbnb, Google]

2. REVIEW CATEGORIES (with severity)

| Category | Check for | Severity on failure |
|----------|-----------|---------------------|
| Style | Naming, formatting, imports | Low (comment) |
| Bug | Null checks, off-by-one, resource leaks | Medium (suggest
change) |
| Security | SQL injection, hardcoded secrets, unsafe functions | High
(block merge) |
| Performance | N+1 queries, inefficient loops | Medium (suggest
change) |
| Test coverage | Untested lines, missing edge cases | Low (comment) |
```

| Complexity | Deep nesting, long functions | Low (comment) |

3. REVIEW OUTPUT FORMAT
   - For each issue:
     * File and line number
     * Category and severity
     * Description of issue
     * Suggested fix (with code example)
     * Why this matters

4. AUTO-FIX CAPABILITIES
   - What the agent can fix automatically: [e.g., formatting, import ordering]
   - What requires human approval: [e.g., logic changes, security fixes]

5. INTEGRATION POINTS
   - Trigger: [on PR creation / on PR update / on request]
   - Comment in PR: [Yes / No]
   - Block merge on: [security issues only / any high severity]

6. REVIEW SCOPING
   - Review only changed lines (vs. whole file)
   - Max lines per review (to avoid token limits)
   - Exclude: [generated files, vendor code, config files]

7. HUMAN REVIEWER HANDOFF
   - What the agent never reviews: [architecture, design decisions, business logic]

- Summary for human: [e.g., "Security checks passed, 3 style suggestions"]

8. READY-TO-USE AGENT PROMPT
- The system prompt for the code review assistant

INPUTS:

Primary languages:
[E.G., "TypeScript, Python"]

Style guide:
[E.G., "Airbnb for TS, PEP 8 for Python"]

Security standards:
[E.G., "OWASP Top 10"]

PR volume:
[E.G., "50 PRs/week, 4 reviewers"]

RULES:
- Never block merge on style or nitpick issues (let humans decide)
- Do block merge on confirmed security vulnerabilities
- Distinguish between critical bugs (block) and suggestions (comment)
- Don't review generated code or vendored dependencies
- Keep review comments actionable and kind ("you forgot to..." not "this is wrong")
- Provide fix examples, not just problem descriptions
- Learn from human feedback (if reviewers consistently ignore a

`suggestion, disable it)`

How To Use It

- Start with non-blocking comments only — build trust before blocking merges.
- Only block merges on confirmed security vulnerabilities, not style or nits.
- Review only changed lines, not entire files (faster, lower token cost).
- Provide fix examples, not just problem descriptions (makes fixes faster).
- Learn from human feedback — if reviewers consistently ignore a suggestion, disable it.

Example Input

**Primary languages:**
“TypeScript, Python”

**Style guide:**
“Airbnb for TypeScript, PEP 8 for Python”

**Security standards:**
“OWASP Top 10”

**PR volume:**
“50 PRs/week, 4 reviewers”

Why It Works
Most code review is spent on issues that could be automated — style, common bugs, security patterns — leaving less time for architecture and design.

This framework improves outcomes by forcing:

- review categorization (style, bug, security, performance, test, complexity)
- severity assignment (what blocks merge vs. what’s a suggestion)
- auto-fix capabilities (what can be fixed automatically?)

- integration design (when and how does the agent comment?)
- human handoff (what does the agent never review?)

Great code review assistants don’t replace humans — they handle the 80% of reviews that are routine so humans can focus on the 20% that need expertise.

## Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

### Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also Mastery Checkpoint Designer