Coding & Development / Code Reviews

Review code for inefficiencies ($O(n^2)$ algorithms, repeated computations, unnecessary DB queries) and suggest improvements.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Performance Optimization, Code Reviews, Efficiency
Updated: May 2026
Why This Prompt Exists
Most code reviews ignore performance — until it’s too late.

You get:

- code that works on small data but fails at scale
- $O(n^2)$ algorithms that should be O(n log n)
- repeated calculations inside loops (no memoization)
- N+1 database queries (chatty and slow)
- inefficient data structures (list vs. set lookups)

But performance is not premature optimization.

It is scalable design.

- Algorithm complexity: $O(n^2)$ → O(n log n) or O(n)
- Memoization: cache repeated calculations
- Database: batch queries, add indexes, avoid N+1
- Data structures: use sets for lookups, dicts for mapping
- Lazy loading: defer work until needed

Without performance review, you discover scale problems in production.

This framework forces AI to flag performance issues.

The Prompt

```
Assume the role of a performance engineer who reviews code for
efficiency.

Your task is to flag performance issues and suggest improvements.

Generate:

1. ISSUE LIST
   - Inefficient algorithm
   - Repeated computation
   - N+1 database query
   - Wrong data structure
   - Missing index

2. FOR EACH ISSUE:
   - Why it's inefficient
   - Impact at scale (time/complexity)

3. OPTIMIZATION RECOMMENDATION
   - Specific code change

4. ESTIMATED IMPROVEMENT
   - Current vs. optimized complexity

INPUTS:
```

```
Code (paste):
[PASTE CODE]

Data Scale (current and expected):
[E.G., "Currently 100 items, will grow to 100,000"]

Environment:
[API / BATCH / REAL-TIME / WEB]

Measured Slowdown (if known):
[E.G., "Takes 2 seconds for 1,000 items"]

RULES:
- Check for nested loops (O(n²) or worse)
- Check for repeated calculations inside loops (memoization
opportunity)
- Check for database queries inside loops (N+1 problem)
- Check for list lookups when set would be faster (O(n) vs O(1))
- Check for missing database indexes on queried columns
- Check for synchronous operations that could be async
- Check for eager loading of unnecessary data
- Suggest profiling to confirm bottlenecks
```

How To Use It

- Profile before optimizing — don’t guess what’s slow.
- Nested loops are the most common performance issue — check them first.
- Database queries inside loops are the second most common.
- Use sets for membership tests (O(1) vs O(n) for lists).
- Add indexes to columns used in WHERE, JOIN, and ORDER BY clauses.

Example Input

Code:

```
def find_common_items(list_a, list_b):
common = []
for a in list_a:
for b in list_b:
if a == b and a not in common:
common.append(a)
return common
```

Data Scale: Currently 1,000 items, will grow to 100,000

Environment: API endpoint

Measured Slowdown: Takes 0.5 seconds for 1,000 items

Why It Works

Most code reviews ignore performance.

This framework improves outcomes by forcing:

- issue identification (detection)
- complexity analysis (understanding)
- optimization recommendation (action)
- improvement estimation (motivation)
- scale awareness (future-proofing)

Great performance reviews don’t just find slow code — they prevent scalability problems.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, code review 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 The Code Quality Reviewer