Coding & Development / Debugging

Identify slow code sections (nested loops, repeated DB calls, inefficient algorithms) and recommend optimizations.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Performance Optimization, Slow Code, Profiling
Updated: May 2026
Why This Prompt Exists
Most slow code isn’t obvious — it works fine on small data and fails at scale.

You get:

- code that’s fast in development, slow in production
- nested loops that work for 100 items but crash at 10,000
- database queries inside loops (n+1 problem)
- repeated calculations that could be cached
- no visibility into where time is actually spent

But bottlenecks are not random.

They follow predictable patterns.

- Algorithmic complexity: $O(n^2)$ vs. $O(n \log n)$
- Database: N+1 queries, missing indexes, full table scans
- I/O: synchronous operations blocking the event loop
- Memory: excessive allocations, inefficient data structures
- Caching: missing cache for repeated computations

Without analysis, you optimize the wrong thing.

This framework forces AI to find real bottlenecks.

The Prompt

```
Assume the role of a performance engineer who identifies bottlenecks
in code.

Your task is to find slow code sections and recommend optimizations.

Generate:

1. BOTTLENECK IDENTIFICATION
   - Which code section is slow
   - Why it's slow (complexity, I/O, database)

2. IMPACT ANALYSIS
   - How performance degrades with scale
   - Estimated time for current vs. optimized

3. OPTIMIZATION RECOMMENDATIONS (2-3)
   - Algorithm improvement
   - Caching strategy
   - Database query optimization
   - Parallelization

4. CODE EXAMPLE (optimized version)

5. TESTING SUGGESTIONS
   - How to benchmark before/after
```

```
INPUTS:

Code (paste or describe):
[PASTE CODE OR DESCRIBE]

Data Scale (current and expected):
[E.G., "Currently 1,000 records, will grow to 100,000"]

Observed Slowdown:
[E.G., "Takes 5 seconds for 1,000 items"]

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

RULES:
- Check for nested loops (O(n²) or worse)
- Check for database queries inside loops (N+1)
- Check for repeated calculations (memoization opportunity)
- Check for synchronous I/O in async contexts
- Check for missing indexes on queried columns
- Consider time-space tradeoffs
- Suggest profiling tools (cProfile, Chrome DevTools, etc.)
```

How To Use It

- Describe the data scale (small now, large later).
- Profile your code before asking (know what’s slow).
- Nested loops are the most common bottleneck — check them first.
- Database queries inside loops are the second most common.
- Measure before and after optimization (don’t guess).

Example Input

Code:

```
def find_duplicates(items):
duplicates = []
for i in range(len(items)):
for j in range(i + 1, len(items)):
if items[i] == items[j] and items[i] not in duplicates:
duplicates.append(items[i])
return duplicates
```

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

Observed Slowdown: Takes 0.5 seconds for 1,000 items, would take 5,000 seconds for 100,000

Environment: BATCH (data processing)

Why It Works

Most developers optimize the wrong thing.

This framework improves outcomes by forcing:

- bottleneck identification (precision)
- impact analysis (scale awareness)
- specific optimizations (actionable)
- code examples (executable)
- testing suggestions (verification)

Great performance optimization doesn’t guess — it measures, identifies, and fixes.

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 The Concurrency Bug Analyzer