您的分析非常准确！确实，Claude 4 在架构设计和代码生成方面很强，但上下文限制是个挑战。我来分享一些高效使用 Claude 的策略：

**1. 模块化开发策略**

**文件粒度控制**

* **单文件原则**：每个文件控制在 300-500 行以内
* **单一职责**：每个模块只负责一个核心功能
* **接口优先**：先设计清晰的接口，再实现细节

**分层架构**

项目结构：

├── core/ # 核心接口定义（Claude设计）

├── modules/ # 独立功能模块（Claude实现）

├── integration/ # 模块集成层（Gemini调试）

└── utils/ # 工具函数（Claude优化）

**2. Claude 专注领域**

**架构设计阶段**

* 系统架构图和数据流设计
* API 接口定义和文档
* 核心算法和数据结构设计
* 设计模式选择和实现框架

**代码生成阶段**

* 生成基础框架代码
* 实现核心业务逻辑
* 创建单元测试
* 编写技术文档

**3. 工作流优化**

**开发流程**

1. Claude：需求分析 → 架构设计 → 接口定义

2. Claude：核心模块实现（每次1-2个文件）

3. Gemini：集成测试 → 调试 → 问题定位

4. Claude：针对性优化单个模块

5. Gemini：回归测试确认

**上下文管理技巧**

* **使用 Artifacts**：将关键代码保存为 artifact，方便引用和修改
* **接口文档化**：在每次对话开始时，只提供必要的接口定义
* **增量式开发**：每次只处理一个功能点

**4. 具体实践建议**

**项目初始化模板**

让 Claude 生成一个标准化的项目模板：

* 清晰的目录结构
* 模块间的依赖关系图
* 标准化的接口规范
* 错误处理机制

**代码组织原则**

// 每个模块都遵循这样的结构

export interface ModuleInterface {

// 清晰的接口定义

}

export class ModuleImplementation implements ModuleInterface {

// 实现细节

}

// 导出工厂函数，便于测试和替换

export function createModule(config?: ModuleConfig): ModuleInterface {

return new ModuleImplementation(config);

}

**5. 协作流程优化**

**Claude 负责的任务单**

* 架构设计文档
* 核心算法实现
* 接口契约定义
* 性能关键代码
* 代码重构建议

**Gemini 负责的任务单**

* 多文件集成调试
* 运行时问题排查
* 依赖关系梳理
* 测试用例补充
* 热修复实现

**6. 提高效率的技巧**

**提示词模板**

为不同任务准备标准化的提示词：

"基于以下接口定义，实现 [模块名] 模块：

接口：[接口定义]

依赖：[最小依赖说明]

要求：[具体要求]"

**版本管理策略**

* 每个模块独立版本控制
* 保持接口向后兼容
* 使用语义化版本号

这样的工作流程能让您充分发挥 Claude 在设计和创造性编码方面的优势，同时用 Gemini 补充调试和集成能力。关键是保持模块独立性和接口稳定性，这样可以最大化两个工具的价值。

您在实践中还遇到过什么具体的挑战吗？我可以提供更针对性的建议。