Certificação Claude Certified Architect — Foundations

Guia de Estudo (Baseado no Guia Oficial do Exame)

Introdução

A certificação **Claude Certified Architect — Foundations** atesta que um especialista é capaz de tomar decisões sólidas de trade-off ao implementar soluções reais baseadas em Claude. O exame avalia o conhecimento fundamental de Claude Code, Claude Agent SDK, Claude API e Model Context Protocol (MCP) — as tecnologias centrais para construir aplicações em produção com Claude.

As questões do exame se baseiam em cenários realistas da indústria: construir sistemas agênticos para atendimento ao cliente, projetar pipelines de pesquisa multi-agente, integrar Claude Code em CI/CD, criar ferramentas de produtividade para desenvolvedores e extrair dados estruturados a partir de documentos não estruturados.

Candidato-Alvo

O candidato ideal é um **arquiteto de soluções** que projeta e entrega aplicações em produção com Claude. Espera-se ao menos 6 meses de experiência prática com:

- **Claude Agent SDK** — orquestração multi-agente, delegação para subagentes, integração de ferramentas, hooks de ciclo de vida
- **Claude Code** — CLAUDE.md, servidores MCP, Agent Skills, modo de planejamento
- **Model Context Protocol (MCP)** — ferramentas e recursos para integração de backend
- **Engenharia de prompts** — JSON schemas, exemplos few-shot, templates de extração de dados
- **Janelas de contexto** — trabalho com documentos longos, passagem de contexto multi-agente
- **Pipelines CI/CD** — code review automatizado, geração de testes
- **Escalonamento e confiabilidade** — tratamento de erros, human-in-the-loop

---

Formato do Exame

Parâmetro	Valor
Tipo de questão	Múltipla escolha (1 correta entre 4)
Pontuação	Escala 100–1000, nota de aprovação **720**
Penalidade por chute	Nenhuma (responda todas as questões!)

Cenários	4 entre 8 possíveis (selecionados aleatoriamente)

Conteúdo do Exame: 5 Domínios

Domínio	Peso
1. Arquitetura e orquestração de agentes	**27%**
2. Design de ferramentas e integração com MCP	**18%**
3. Configuração e fluxos de trabalho do Claude Code	**20%**
4. Engenharia de prompts e saída estruturada	**20%**
5. Gerenciamento de contexto e confiabilidade	**15%**

Cenários do Exame

Cenário 1: Agente de Suporte ao Cliente

Você constrói um agente para lidar com devoluções, disputas de cobrança e questões de conta usando o Claude Agent SDK. O agente usa ferramentas MCP (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`). A meta é 80%+ de resolução no primeiro contato com escalonamento apropriado.

Cenário 2: Geração de Código com Claude Code

Você usa Claude Code para acelerar o desenvolvimento: geração de código, refatoração, depuração, documentação. Você precisa integrá-lo a slash commands customizados e à configuração CLAUDE.md, e entender quando usar o modo de planejamento.

Cenário 3: Sistema de Pesquisa Multiagente

Um agente coordenador delega tarefas a subagentes especializados: pesquisa na web, análise de documentos, síntese e geração de relatórios. O sistema deve produzir relatórios completos com citações.

Cenário 4: Ferramentas de Produtividade para Desenvolvedores

O agente ajuda engenheiros a explorar bases de código desconhecidas, gerar boilerplate e automatizar tarefas rotineiras. Ferramentas embutidas (Read, Write, Bash, Grep, Glob) e servidores MCP são utilizados.

Cenário 5: Claude Code para Integração Contínua

Integre Claude Code a um pipeline CI/CD para code reviews automatizados, geração de testes e feedback em pull requests. Os prompts devem ser desenhados para minimizar falsos positivos.

Cenário 6: Extração de Dados Estruturados

O sistema extrai informação de documentos não estruturados, valida a saída com JSON schemas e mantém alta acurácia. Deve tratar corretamente os casos de borda.

Cenário 7: Padrões de Arquitetura de IA Conversacional

Você projeta sistemas conversacionais multi-turno cobrindo gerenciamento da janela de contexto, persistência de instruções entre turnos, estratégias de memória, design de ferramentas para execução segura e tratamento de entradas ambíguas ou conflitantes do usuário.

Cenário 8: Ferramentas de IA Agêntica *(conteúdo faltando — ajude-nos a preencher!)*

Este cenário foi reportado por candidatos do exame mas ainda não está coberto neste guia. Se você se deparou com questões deste cenário no exame real, compartilhe-as nas GitHub Issues para que possamos adicionar cobertura completa. Sua contribuição ajudará todos que se preparam para o exame.

Documentação Oficial

Recurso	URL
Claude API — Messages	https://platform.claude.com/docs/en/api/messages
Claude API — Tool Use	https://platform.claude.com/docs/en/build-with-claude/tool-use
Claude API — Message Batches	https://platform.claude.com/docs/en/build-with-claude/message-batches
Claude Agent SDK — Overview	https://platform.claude.com/docs/en/agent-sdk/overview
Claude Agent SDK — Hooks	https://platform.claude.com/docs/en/agent-sdk/hooks
Claude Agent SDK — Subagents	https://platform.claude.com/docs/en/agent-sdk/subagents
Claude Agent SDK — Sessions	https://platform.claude.com/docs/en/agent-sdk/sessions
Model Context Protocol (MCP)	https://modelcontextprotocol.io/
MCP — Tools	https://modelcontextprotocol.io/docs/concepts/tools
MCP — Resources	https://modelcontextprotocol.io/docs/concepts/resources
MCP — Servers	https://modelcontextprotocol.io/docs/concepts/servers
Claude Code — Documentation	https://code.claude.com/docs/en/overview
Claude Code — CLAUDE.md and Memory	https://code.claude.com/docs/en/memory
Claude Code — Skills (incl. slash commands)	https://code.claude.com/docs/en/skills

Claude Code — Hooks	https://code.claude.com/docs/en/hooks
Claude Code — Sub-agents	https://code.claude.com/docs/en/sub-agents
Claude Code — MCP Integration	https://code.claude.com/docs/en/mcp
Claude Code — GitHub Actions CI/CD	https://code.claude.com/docs/en/github-actions
Claude Code — GitLab CI/CD	https://code.claude.com/docs/en/gitlab-ci-cd
Claude Code — Headless (modo não-interativo)	https://code.claude.com/docs/en/headless
Guia de Engenharia de Prompts	https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview
Extended Thinking	https://platform.claude.com/docs/en/build-with-claude/extended-thinking
Anthropic Cookbook (exemplos de código)	https://github.com/anthropics/anthropic-cookbook

---

PARTE I: FUNDAMENTOS TEÓRICOS

Esta parte cobre toda a teoria necessária para passar no exame com sucesso. O material está organizado por tecnologias e conceitos em vez de pelos domínios do exame — isso ajuda a construir um entendimento mais profundo de cada tópico.

Capítulo 1: Claude API — Fundamentos da Interação com o Modelo

> Documentação: Messages API | Engenharia de Prompts

1.1 Estrutura da Requisição da API

A Claude API segue um modelo requisição–resposta. Cada requisição à Claude Messages API inclui:

```
{
  "model": "claude-sonnet-4-6",
  "max_tokens": 1024,
  "system": "You are a helpful assistant.",
  "messages": [
    {"role": "user", "content": "Hi!"},
    {"role": "assistant", "content": "Hello!"},
    {"role": "user", "content": "How are you?"}
  ],
  "tools": [...],
  "tool_choice": {"type": "auto"}
}
```

Campos principais:

- `model` — seleção do modelo (`claude-opus-4-6`, `claude-sonnet-4-6`, `claude-haiku-4-5`)
- `max_tokens` — número máximo de tokens na resposta
- `system` — o system prompt (define o comportamento do modelo)
- `messages` — histórico da conversa (**você deve enviar todo o histórico** para manter coerência)
- `tools` — definições das ferramentas disponíveis
- `tool_choice` — estratégia de seleção de ferramentas

1.2 Papéis das Mensagens

O array `messages` usa três papéis (roles):

- `user` — mensagens do usuário
- `assistant` — respostas do modelo (incluídas ao enviar histórico)
- `tool` — resultados de chamadas de ferramentas (o role não é setado explicitamente; aparece como bloco de conteúdo `tool_result`)

Criticamente importante: em cada requisição à API você deve enviar todo o **histórico completo da conversa**. O modelo não persiste estado entre requisições — cada chamada é independente.

1.3 O Campo `stop_reason` na Resposta

A resposta da Claude API inclui `stop_reason`, que indica por que o modelo parou de gerar:

Valor	Descrição	Ação
`"end_turn"`	O modelo finalizou sua resposta	Mostre o resultado ao usuário
`"tool_use"`	O modelo quer chamar uma ferramenta	Execute a ferramenta e retorne o resultado
`"max_tokens"`	Limite de tokens atingido	A resposta foi truncada; pode ser necessário aumentar o limite

`"stop_sequence"`	Uma sequência de parada foi encontrada	Trate conforme a lógica da sua aplicação

Para sistemas agênticos, `"tool_use"` e `"end_turn"` são os mais importantes — eles controlam o loop do agente.

1.4 System Prompt

O system prompt é uma instrução especial que define contexto e regras de comportamento. Ele:

- Não faz parte do array `messages`; é passado separadamente no campo `system`
- Tem prioridade sobre as mensagens do usuário
- É carregado uma vez e se aplica ao longo de toda a conversa
- É usado para definir papel, restrições e formato de saída

Importante para o exame: o texto do system prompt pode criar associações não intencionais com ferramentas. Por exemplo, uma instrução como "sempre verifique o cliente" pode levar o modelo a usar `get_customer` em excesso, mesmo quando isso não é necessário.

1.5 Janela de Contexto

A janela de contexto é a quantidade total de texto (em tokens) que o modelo consegue processar de uma vez. Ela inclui:

- O system prompt
- Todo o histórico de mensagens
- Definições de ferramentas
- Resultados de ferramentas

Problemas-chave da janela de contexto:

1. **Efeito "lost-in-the-middle":** os modelos processam de forma confiável a informação no início e no fim de uma entrada longa, mas podem perder detalhes no meio. Mitigação: posicione informação-chave perto do início ou do fim.
2. **Acúmulo de resultados de ferramentas:** cada chamada de ferramenta acrescenta saída ao contexto. Se uma ferramenta retorna 40+ campos mas só 5 importam, a maior parte do contexto é desperdiçada.
3. **Sumarização progressiva:** ao comprimir histórico, valores numéricos, percentuais e datas tendem a se perder e ficar vagos ("cerca de", "aproximadamente", "alguns").

Capítulo 2: Ferramentas e `tool_use`

> Documentação: Tool Use

2.1 O que é `tool_use`

`tool_use` é o mecanismo que permite ao Claude chamar funções externas. O modelo não executa código diretamente — ele gera uma requisição estruturada de chamada de ferramenta; seu código a executa e retorna o resultado.

2.2 Definição de Ferramenta

Cada ferramenta é definida usando um JSON schema:

```
{
  "name": "get_customer",
  "description": "Finds a customer by email or ID. Returns the customer profile,
including name, email, order history, and account status. Use this tool BEFORE
lookup_order to verify the customer's identity. Accepts an email (format:
user@domain.com) or a numeric customer_id.",
  "input_schema": {
    "type": "object",
    "properties": {
      "email": {"type": "string", "description": "Customer email"},
      "customer_id": {"type": "integer", "description": "Numeric customer ID"}
    },
    "required": []
  }
}
```

Aspectos criticamente importantes da descrição de uma ferramenta:

1. **A descrição é o principal mecanismo de seleção.** Um LLM escolhe ferramentas com base nas suas descrições. Descrições mínimas ("Recupera informação do cliente") levam a erros quando ferramentas se sobrepõem.
2. **Inclua na descrição:**
   - O que a ferramenta faz e o que retorna
   - Formatos de entrada e valores de exemplo
   - Casos de borda e restrições
   - Quando usar esta ferramenta vs alternativas similares
3. **Evite** descrições idênticas ou sobrepostas entre ferramentas. Se `analyze_content` e `analyze_document` têm descrições quase idênticas, o modelo as confundirá.
4. **Ferramentas embutidas vs ferramentas MCP:** agentes podem preferir ferramentas embutidas (Read, Grep) em vez de ferramentas MCP com funcionalidade similar. Para evitar isso, reforce as descrições das ferramentas MCP — destaque vantagens concretas, dados únicos ou contexto que ferramentas embutidas não conseguem prover.

2.3 O Parâmetro `tool_choice`

`tool_choice` controla como o modelo seleciona ferramentas:

Valor	Comportamento	Quando usar
`{"type": "auto"}`	O modelo decide se chama uma ferramenta ou responde em texto	Padrão para a maioria dos casos
`{"type": "any"}`	O modelo **deve** chamar alguma ferramenta	Quando você precisa garantir saída estruturada
`{"type": "tool", "name": "extract_metadata"}`	O modelo **deve** chamar uma ferramenta específica	Quando você precisa de um primeiro passo forçado / ordem de execução

Cenários importantes:

- `tool_choice: "any"` + várias ferramentas de extração → o modelo escolhe a melhor, mas você ainda obtém saída estruturada
- Seleção forçada → quando você precisa garantir uma primeira ação específica (ex.: `extract_metadata` antes do enriquecimento)

2.4 JSON Schemas para Saída Estruturada

Usar `tool_use` com JSON schemas é a forma **mais confiável** de obter saída estruturada do Claude. Isso:

- Garante JSON sintaticamente válido (sem chaves faltando, sem vírgulas extras)
- Garante a estrutura exigida (campos obrigatórios estão presentes)
- **Não** garante correção semântica (valores ainda podem estar errados)

Design de schema — princípios-chave:

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": ["bug", "feature", "docs", "unclear", "other"]
    },
    "category_detail": {
      "type": ["string", "null"],
      "description": "Details if category = 'other' or 'unclear'"
    },
    "severity": {
      "type": "string",
      "enum": ["critical", "high", "medium", "low"]
    },
    "confidence": {
      "type": "number",
      "minimum": 0,
      "maximum": 1
    },
    "optional_field": {
      "type": ["string", "null"],
      "description": "Null if the information was not found in the source"
    }
  },
  "required": ["category", "severity"]
}
```

Regras de design de schema:

1. **Required vs opcional:** marque um campo como obrigatório apenas se a informação estiver sempre disponível. Campos obrigatórios empurram o modelo a fabricar valores quando os dados estão ausentes.
2. **Campos nuláveis:** use `"type": ["string", "null"]` para informação que pode estar ausente. O modelo pode retornar `null` em vez de alucinar.
3. **Enums com `"other"`:** adicione `"other"` + uma string de detalhe para evitar perder dados fora das categorias predefinidas.
4. **Enum `"unclear"`:** para casos em que o modelo não consegue escolher uma categoria com confiança — um `"unclear"` honesto é melhor que uma categoria errada.

2.5 Erros de Sintaxe vs Semânticos

Tipo de erro	Exemplo	Mitigação
Sintaxe	JSON inválido, tipo de campo errado	`tool_use` com JSON schema (elimina)
Semântico	Totais não batem, valor no campo errado, alucinação	Validações, retry com feedback, autocorreção

Capítulo 3: Claude Agent SDK — Construindo Sistemas Agênticos

> Documentação: Agent SDK | Hooks | Subagents | Sessions

3.1 O que é um Loop Agêntico

O loop agêntico é o padrão central para execução autônoma de tarefas. O modelo não apenas responde — ele executa uma sequência de ações:

```
1. Envie uma requisição ao Claude com ferramentas
2. Receba a resposta
3. Verifique stop_reason:
   - "tool_use" -> execute a ferramenta, anexe o resultado ao histórico, volte ao passo
1
   - "end_turn" -> a tarefa está concluída, mostre o resultado ao usuário
4. Repita até a conclusão
```

Esta é uma abordagem dirigida pelo modelo: Claude decide qual ferramenta chamar em seguida com base no contexto e em resultados de ferramentas anteriores. Isso difere de árvores de decisão hard-coded em que a sequência de ações é fixa.

Anti-padrões (evite):

- Parsear o texto do assistente para detectar conclusão ("Tarefa concluída")
- Usar um limite arbitrário de iterações (ex.: `max_iterations=5`) como condição primária de parada
- Verificar se o assistente produziu conteúdo textual como sinal de conclusão

Abordagem correta: o único sinal confiável de conclusão é `stop_reason == "end_turn"`.

3.2 Configuração `AgentDefinition`

`AgentDefinition` é o objeto de configuração de agente no Claude Agent SDK:

```python
agent = AgentDefinition(
    name="customer_support",
    description="Handles customer requests for returns and order issues",
    system_prompt="You are a customer support agent...",
    allowed_tools=["get_customer", "lookup_order", "process_refund",
"escalate_to_human"],
    # For a coordinator:
    # allowed_tools=["Task", "get_customer", ...]
)
```

Parâmetros principais:

- `name` / `description` — identificação e descrição do agente
- `system_prompt` — system prompt com instruções
- `allowed_tools` — lista de ferramentas permitidas (princípio do menor privilégio)

3.3 Hub-and-Spoke: Coordenador e Subagentes

Uma arquitetura multi-agente costuma ser construída como uma topologia hub-and-spoke:

```
        Coordenador
       /     |      \
  Subagente1  Subagente2  Subagente3
   (busca)    (análise)   (síntese)
```

O coordenador é responsável por:

- Decompor a tarefa em subtarefas
- Decidir quais subagentes são necessários (seleção dinâmica)
- Delegar trabalho aos subagentes
- Agregar e validar resultados
- Tratar erros e retentativas
- Comunicar resultados ao usuário

Princípio crítico: subagentes têm contexto isolado.

- Subagentes **não** herdam automaticamente o histórico de conversa do coordenador
- Todo contexto necessário deve ser **passado explicitamente** no prompt do subagente
- Subagentes não compartilham memória entre chamadas
- Toda comunicação flui pelo coordenador (para observabilidade e controle de erros)

3.4 A Ferramenta `Task` para Spawnar Subagentes

Subagentes são spawnados via a ferramenta `Task`:

```
# The coordinator's allowedTools must include "Task"
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

Passagem explícita de contexto é obrigatória:

```
# Bad: the subagent has no context
Task: "Analyze the document"

# Good: full context in the prompt
Task: "Analyze the following document.
Document: [full document text]
Prior search results: [web search results]
Output format requirements: [schema]"
```

Spawn em paralelo: um coordenador pode chamar várias `Task`s em uma única resposta — os subagentes rodam em paralelo:

```
# One coordinator response contains:
Task 1: "Search for articles about X"
Task 2: "Analyze document Y"
Task 3: "Search for articles about Z"
# All three run concurrently
```

3.5 Hooks no Agent SDK

Hooks permitem interceptar e transformar pontos específicos do ciclo de vida do agente.

PostToolUse intercepta o resultado de uma ferramenta antes de ser entregue ao modelo:

```python
# Example: normalize date formats from different MCP tools
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Convert Unix timestamp -> ISO 8601
    # Convert "Mar 5, 2025" -> "2025-03-05"
    return normalized_result
```

Hook de interceptação de chamadas de saída bloqueia ações que violem políticas:

```python
# Example: block refunds above $500
@hook("PreToolUse")
def enforce_refund_limit(tool_call):
    if tool_call.name == "process_refund" and tool_call.args.amount > 500:
        return redirect_to_escalation(tool_call)
```

Diferença-chave: hooks vs instruções no prompt

Atributo	Hooks	Instruções no prompt
Garantia	**Determinística** (100%)	**Probabilística** (>90%, não 100%)
Quando usar	Regras de negócio críticas, operações financeiras, compliance	Preferências gerais, recomendações, formatação

Exemplo	Bloquear reembolsos > $500	"Tente resolver antes de escalar"

Regra: quando uma falha tem consequências financeiras, legais ou de segurança — use hooks, não prompts.

Capítulo 4: Model Context Protocol (MCP)

> Documentação: MCP | Tools | Resources | Servers

4.1 O que é MCP

O Model Context Protocol (MCP) é um protocolo aberto para conectar sistemas externos ao Claude. O MCP define três tipos primários de recursos:

1. **Tools** — funções que o agente pode chamar para executar ações (operações CRUD, chamadas de API, execução de comandos)
2. **Resources** — dados que o agente pode ler como contexto (documentação, schemas de banco de dados, catálogos de conteúdo)
3. **Prompts** — templates de prompt predefinidos para tarefas comuns

4.2 Servidores MCP

Um servidor MCP é um processo que implementa o protocolo MCP e fornece tools/resources. Quando você se conecta a um servidor MCP:

- Todas as ferramentas são descobertas automaticamente
- Ferramentas de todos os servidores conectados ficam disponíveis ao mesmo tempo
- As descrições das ferramentas determinam como o modelo as usará

4.3 Configurando Servidores MCP

Configuração de projeto (`.mcp.json`) — para uso em equipe:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "jira": {
      "command": "npx",
      "args": ["-y", "mcp-server-jira"],
      "env": {
        "JIRA_TOKEN": "${JIRA_TOKEN}"
      }
    }
  }
}
```

Pontos-chave:

- `.mcp.json` fica na raiz do projeto e é gerenciado em controle de versão
- Variáveis de ambiente (`${GITHUB_TOKEN}`) são usadas para segredos — os tokens em si não são commitados
- Disponível para todos os contribuidores do projeto

Configuração de usuário (`~/.claude.json`) — para servidores pessoais/experimentais:

- Armazenada no diretório home do usuário
- Não é compartilhada via controle de versão
- Adequada para experimentos pessoais e testes

Escolhendo servidores:

- Para integrações padrão (Jira, GitHub, Slack), prefira servidores MCP comunitários existentes
- Construa seus próprios servidores apenas para fluxos únicos e específicos do time

4.4 A Flag `isError` no MCP

Quando uma ferramenta MCP encontra um erro, ela usa `isError: true` na resposta. Isso sinaliza ao agente que a chamada falhou.

Erro estruturado (bom):

```
{
  "isError": true,
  "content": {
    "errorCategory": "transient",
    "isRetryable": true,
    "message": "The service is temporarily unavailable. Timeout while calling the
orders API.",
    "attempted_query": "order_id=12345",
    "partial_results": null
  }
}
```

Erro genérico (anti-padrão):

```
{
  "isError": true,
  "content": "Operation failed"
}
```

Um erro genérico não dá ao agente nenhuma informação para tomar decisões — deve retentar, mudar a query, ou escalar?

4.5 MCP Resources

Resources são dados que um agente pode requisitar para obter contexto sem realizar ações:

- Catálogos de conteúdo (ex.: lista de todas as tarefas do projeto, navegação hierárquica)
- Schemas de banco de dados (entender a estrutura dos dados)
- Documentação (referências de API, guias internos)
- Sumários de issues/tarefas

Vantagem dos resources: o agente não precisa de chamadas exploratórias de ferramentas para entender quais dados existem. Um resource fornece um "mapa" imediato.

Capítulo 5: Claude Code — Configuração e Fluxos de Trabalho

> Documentação: Claude Code | Memory / CLAUDE.md | Skills | MCP | Hooks | Sub-agents | GitHub Actions | Headless

5.1 A Hierarquia CLAUDE.md

CLAUDE.md é o(s) arquivo(s) de instruções para o Claude Code. Existe uma hierarquia em três níveis:

```
1. Nível usuário: ~/.claude/CLAUDE.md
   - Aplica-se apenas ao usuário
   - NÃO compartilhado via VCS
   - Preferências pessoais e estilo de trabalho

2. Nível projeto: .claude/CLAUDE.md ou um CLAUDE.md na raiz
   - Aplica-se a todos os contribuidores do projeto
   - Gerenciado via VCS
   - Padrões de código, padrões de testes, decisões arquiteturais

3. Nível diretório: CLAUDE.md em subdiretórios
   - Aplica-se ao trabalhar com arquivos naquele diretório
   - Convenções específicas daquela parte da base de código
```

Erro comum: um novo membro do time não recebe as instruções do projeto porque foram colocadas em `~/.claude/CLAUDE.md` (nível usuário) em vez de `.claude/CLAUDE.md` (nível projeto).

5.2 Sintaxe `@path` (Imports de Arquivo)

CLAUDE.md pode referenciar arquivos externos usando `@path`, tornando a configuração modular:

```
# Project CLAUDE.md

Coding standards are described in @./standards/coding-style.md
Test requirements are in @./standards/testing-requirements.md
Project overview is in @README.md and dependencies are in @package.json
```

Regras para `@path`:

- Use `@` imediatamente antes do caminho do arquivo (sem espaço)
- Caminhos relativos e absolutos são suportados
- Caminhos relativos são resolvidos em relação ao arquivo que contém o import
- Profundidade máxima de aninhamento de imports é 5

Isso evita duplicação e permite que cada pacote inclua apenas os padrões relevantes.

5.3 O Diretório `.claude/rules/`

`.claude/rules/` é uma alternativa a um CLAUDE.md monolítico, usado para organizar regras por tópico:

```
.claude/rules/
  testing.md          -- convenções de testes
  api-conventions.md  -- convenções de API
  deployment.md       -- regras de deploy
  react-patterns.md   -- padrões React
```

Recurso-chave: frontmatter YAML com `paths` para carregamento condicional:

```
---
paths: ["src/api/**/*"]
---

For API files, use async/await with explicit error handling.
Each endpoint must return a standard response wrapper.
```

```
---
paths: ["**/*.test.tsx", "**/*.test.ts"]
---

Tests must use describe/it blocks.
Use data factories instead of hardcoding.
Do not mock the database—use a test database.
```

Como funciona:

- Uma regra é carregada **apenas** quando Claude Code edita um arquivo que casa com o padrão `paths`
- Isso economiza contexto e tokens — regras irrelevantes não são carregadas
- Padrões glob permitem aplicar convenções por tipo de arquivo independentemente da localização (ideal para testes espalhados pela base de código)

Quando usar `.claude/rules/` com `paths` vs CLAUDE.md em nível de diretório:

- `.claude/rules/` com `paths` — quando convenções se aplicam a arquivos espalhados por muitos diretórios (testes, migrations)
- CLAUDE.md em nível de diretório — quando convenções estão atreladas a um diretório específico e não são necessárias em outro lugar

5.4 Slash Commands Customizados e Skills

> **Nota:** na versão atual do Claude Code, comandos customizados (`.claude/commands/`) estão unificados com skills (`.claude/skills/`). Ambos os formatos criam comandos `/name`. O guia do exame referencia `.claude/commands/` — este formato continua suportado.

Slash commands são templates de prompt reutilizáveis invocados via `/name`:

Formato `.claude/commands/` (legado, suportado):

```
.claude/commands/
  review.md        -- /review -- code review padrão
  test-gen.md      -- /test-gen -- geração de testes
```

Formato `.claude/skills/` (atual):

```
.claude/skills/
  review/SKILL.md  -- /review -- com configuração via frontmatter
  test-gen/SKILL.md
```

Comandos de projeto (`.claude/commands/` ou `.claude/skills/`):

- Armazenados em VCS e disponíveis para todos ao clonar o repo
- Garantem fluxos consistentes em todo o time

Comandos de usuário (`~/.claude/commands/` ou `~/.claude/skills/`):

- Comandos pessoais não compartilhados via VCS
- Para fluxos individuais

5.5 Skills — `.claude/skills/`

Skills são comandos avançados configurados via frontmatter SKILL.md:

```
---
context: fork
allowed-tools: ["Read", "Grep", "Glob"]
argument-hint: "Path to the directory to analyze"
---

Analyze the code structure in the specified directory.
Output a report on dependencies and architectural patterns.
```

Parâmetros do frontmatter:

Parâmetro	Descrição
`context: fork`	Roda a skill em um subagente isolado. Saídas verbosas não poluem a sessão principal
`allowed-tools`	Restringe quais ferramentas estão disponíveis (segurança — ex.: a skill não pode deletar arquivos se não estiver permitido)
`argument-hint`	Dica que pede um argumento quando invocada sem parâmetros

Quando usar skill vs CLAUDE.md:

- **Skill** — invocação sob demanda para uma tarefa específica (review, análise, geração)
- **CLAUDE.md** — padrões e convenções gerais sempre carregados

Skills pessoais (`~/.claude/skills/`):

- Crie variantes pessoais sob nomes diferentes para não afetar colegas

5.6 Modo de Planejamento vs Execução Direta

Modo de planejamento:

- O modelo apenas investiga e planeja; não faz mudanças
- Usa Read, Grep, Glob para explorar a base de código
- Produz um plano de implementação que o usuário aprova
- Exploração segura sem efeitos colaterais

Quando usar o modo de planejamento:

- Mudanças grandes (dezenas de arquivos)
- Múltiplas abordagens plausíveis (microsserviços: como definir fronteiras?)
- Decisões arquiteturais (qual framework? qual estrutura?)
- Base de código desconhecida (você precisa entender antes de mudar)
- Migrações de biblioteca afetando 45+ arquivos

Quando usar execução direta:

- Correções em um único arquivo com stack trace claro
- Adicionar uma checagem de validação
- Mudanças bem entendidas e não-ambíguas

Abordagem combinada:

1. Modo de planejamento para investigação e desenho
2. Usuário aprova o plano
3. Execução direta para implementar o plano aprovado

Subagente Explore — um subagente especializado para explorar a base de código:

- Isola saídas verbosas do contexto principal
- Retorna apenas um sumário
- Evita esgotar a janela de contexto em tarefas multi-fase

5.7 O Comando `/compact`

`/compact` é um comando embutido para comprimir contexto:

- Sumariza o histórico anterior para liberar a janela de contexto
- Usado em sessões longas de investigação quando o contexto se enche com saída verbosa de ferramentas
- Risco: valores numéricos exatos, datas e detalhes específicos podem se perder na sumarização

5.8 O Comando `/memory`

`/memory` é um comando embutido para gerenciar memória entre sessões:

- Abre o arquivo `CLAUDE.md` para edição, permitindo salvar notas, preferências e contexto
- A informação persiste entre sessões e é carregada automaticamente na inicialização
- Útil para armazenar convenções de projeto, preferências do usuário, comandos frequentes e contexto do trabalho atual
- Alternativa a re-explicar as mesmas instruções em cada sessão

5.9 Claude Code CLI para CI/CD

A flag `-p` (ou `--print`):

```
claude -p "Analyze this pull request for security issues"
```

- Modo não-interativo: processa o prompt, imprime no stdout, encerra
- Não espera input do usuário
- A única forma correta de rodar Claude em pipelines CI/CD

Saída estruturada para CI:

```
claude -p "Review this PR" --output-format json --json-schema
'{"type":"object",...}'
```

- `--output-format json` — saída em JSON
- `--json-schema` — valida a saída contra um schema
- O resultado pode ser parseado para automaticamente postar comentários inline em PRs

Isolamento de contexto entre sessões: A mesma sessão Claude que gerou o código costuma ser menos efetiva ao revisá-lo (o modelo retém seu contexto de raciocínio e tem menos propensão a desafiar suas próprias decisões). Use uma instância independente para review.

Evitando comentários duplicados: Ao rerrevisar após novos commits, inclua os resultados de reviews anteriores no contexto e instrua Claude a reportar apenas problemas novos ou não resolvidos.

5.10 `fork_session` e Gerenciamento de Sessão

`--resume <session-name>` retoma uma sessão nomeada:

```
claude --resume investigation-auth-bug
```

- Continua uma conversa anterior com contexto salvo

- Útil para investigações longas em múltiplas sessões
- Risco: se arquivos mudaram desde a sessão anterior, resultados de ferramentas podem estar desatualizados

`fork_session` cria um branch independente a partir de um contexto compartilhado:

```
Codebase investigation
         |
    fork_session
    /           \
Approach A:      Approach B:
Redux            Context API
```

- Ambos os forks herdam contexto até o ponto de bifurcação
- Em seguida, divergem de forma independente
- Útil para comparar abordagens ou testar estratégias

Quando começar uma nova sessão em vez de retomar:

- Resultados de ferramentas estão desatualizados (arquivos mudaram)
- Muito tempo passou e o contexto se degradou
- É melhor reiniciar com "Aqui está um breve resumo do que descobrimos: ..." do que retomar com dados antigos de ferramentas

---

Capítulo 6: Engenharia de Prompts — Técnicas Avançadas

> Documentação: Prompt Engineering | Anthropic Cookbook

6.1 Few-shot Prompting

Few-shot prompting é a inclusão de 2–4 exemplos de entrada/saída em um prompt para demonstrar o comportamento esperado.

Por que few-shot é mais efetivo do que descrições textuais:

- Uma instrução vaga como "seja mais preciso" pode ser interpretada de muitas formas
- Um exemplo mostra de forma inequívoca o formato esperado e a lógica de decisão
- O modelo generaliza o padrão para novos casos (ele não apenas repete os exemplos)

Tipos de exemplos few-shot e quando usá-los:

1. **Exemplos para cenários ambíguos:**

```
Request: "My order is broken"
Action: Call get_customer -> lookup_order -> check status.
Rationale: "broken" may mean a damaged item; you need order details.

Request: "Get me a manager"
Action: Immediately call escalate_to_human.
Rationale: The customer explicitly requests a human. Do not attempt to solve
autonomously.
```

2. **Exemplos para formatação de saída:**

```
Finding example:
{
  "location": "src/auth/login.ts:42",
  "issue": "SQL injection in the username parameter",
  "severity": "critical",
  "suggested_fix": "Use a parameterized query"
}
```

3. **Exemplos para separar código aceitável vs problemático:**

```
// Acceptable (do not flag):
const items = data.filter(x => x.active);

// Problem (flag):
const items = data.filter(x => x.active == true); // Use strict equality ===
```

4. **Exemplos para extração a partir de diferentes formatos de documento:**

```
Document with inline citations:
"As shown in the study (Smith, 2023), the rate is 42%."
-> {"value": "42%", "source": "Smith, 2023", "type": "inline_citation"}

Document with bibliography references:
"The rate is 42%. [1]"
-> {"value": "42%", "source": "reference_1", "type": "bibliography"}
```

5. **Exemplos para medidas informais:**

```
Text: "about two handfuls of rice"
-> {"amount": "~100g", "original_text": "two handfuls", "precision": "approximate"}

Text: "a pinch of salt"
-> {"amount": "~1g", "original_text": "a pinch", "precision": "approximate"}
```

Few-shot é especialmente eficaz para extrair unidades de medida informais e não-padronizadas que são diversas demais para instruções puramente baseadas em regras.

Regras de normalização de formato em prompts: Ao usar JSON schemas estritos para saída estruturada, adicione regras de normalização no prompt:

```
Normalization:
- Dates: always ISO 8601 (YYYY-MM-DD); "yesterday" -> compute an absolute date
- Currency: numeric amount + currency code; "five bucks" -> {"amount": 5, "currency":
"USD"}
- Percentages: decimal fraction; "half" -> 0.5
```

Isso evita erros semânticos em que o JSON é sintaticamente válido mas os valores são inconsistentes.

6.2 Critérios Explícitos vs Instruções Vagas

Ruim (vago):

```
Check code comments for accuracy.
Be conservative—report only high-confidence findings.
```

Bom (critérios explícitos):

```
Flag a comment as problematic ONLY if:
1. The comment describes behavior that CONTRADICTS the actual code behavior
2. The comment references a non-existent function or variable
3. A TODO/FIXME comment refers to a bug that has already been fixed in code

Do NOT flag:
- Comments that are merely stylistically outdated
- Comments with minor wording inaccuracies
- Missing comments (that is a separate category)
```

Defina critérios de severidade com exemplos:

```
CRITICAL: Runtime failure for users
  Example: NullPointerException while processing a payment

HIGH: Security vulnerability
  Example: SQL injection, XSS, missing authorization checks

MEDIUM: Logic bug without immediate impact
  Example: Wrong sorting, off-by-one error

LOW: Code quality
  Example: Duplication, suboptimal algorithm for small data
```

6.3 Prompt Chaining

Prompt chaining quebra uma tarefa complexa em uma sequência de passos focados:

```
Step 1: Analyze auth.ts (local issues only)
       -> Output: list of issues in auth.ts

Step 2: Analyze database.ts (local issues only)
       -> Output: list of issues in database.ts

Step 3: Integration pass (cross-file dependencies)
       -> Output: issues at module boundaries
```

Por que isso importa:

- Evita **diluição de atenção** — quando o modelo recebe muitos arquivos de uma vez, pode perder bugs em alguns enquanto faz comentários superficiais em outros
- Garante qualidade consistente de análise por arquivo
- Permite análise separada de interações entre arquivos

Quando usar prompt chaining vs decomposição dinâmica:

- **Prompt chaining** — tarefas previsíveis e repetitivas (code review, migrações de arquivos)
- **Decomposição dinâmica** — investigações abertas onde subtarefas só ficam claras durante a execução

6.4 O Padrão "Entrevista"

Antes de implementar uma solução, Claude faz perguntas esclarecedoras:

```
Claude: "Before implementing caching for the API, a few questions:
1. Which cache invalidation strategy do you prefer—TTL or event-based?
2. Is stale data acceptable when the cache is unavailable?
3. Should caching be per-user or global?
4. What is the expected data volume to cache?"
```

Quando isso é útil:

- Domínio desconhecido (fintech, saúde, sistemas jurídicos)
- Tarefas com implicações não óbvias (estratégias de cache, modos de falha)
- Múltiplas abordagens viáveis em que a melhor escolha depende do contexto

6.5 Validação e Retry-com-Feedback

Quando dados extraídos falham na validação:

```
Step 1: Extract data from the document
Step 2: Validate (Pydantic, JSON Schema, business rules)
Step 3: If there's an error—retry with context:
  - The original document
  - The previous (incorrect) extraction
  - The specific error: "Field 'total' = 150, but sum(line_items) = 145. Re-check
values."
```

Quando o retry será efetivo:

- Erros de formato (data em formato errado)
- Erros estruturais (campo no lugar errado)
- Inconsistências aritméticas (o modelo pode reverificar)

Quando o retry NÃO ajudará:

- A informação não está no documento de origem
- O contexto necessário é externo (o dado está em outro documento que não foi fornecido)

Pydantic como ferramenta de validação: Pydantic é uma biblioteca Python para validação de dados baseada em schema. Para o exame, os pontos-chave são:

- **Validação estrutural:** tipos, obrigatoriedade, restrições de enum verificadas em código após receber JSON do Claude
- **Validação semântica:** validadores customizados aplicam lógica de negócio (soma dos itens igual ao total; start_date < end_date)
- **Loops de validação–retry:** ao falhar a validação Pydantic, construa uma mensagem de erro e re-prompte o Claude com o contexto do erro
- **Geração de JSON Schema:** modelos Pydantic podem gerar JSON Schema para `tool_use`, oferecendo uma única fonte de verdade

6.6 Autocorreção

Um padrão para detectar contradições internas:

```
{
  "stated_total": "$150.00",
  "calculated_total": "$145.00",
  "conflict_detected": true,
  "line_items": [
    {"name": "Widget A", "price": 75.00},
    {"name": "Widget B", "price": 70.00}
  ]
}
```

O modelo extrai tanto o valor declarado quanto um valor calculado — se diferirem, `conflict_detected` permite tratar a discrepância.

Capítulo 7: Message Batches API

> Documentação: Message Batches

7.1 Visão Geral

A Message Batches API permite enviar lotes de requisições para processamento assíncrono:

Atributo	Valor
Economia	**50%** comparado a chamadas síncronas
Janela de processamento	Até **24 horas** (sem garantia de SLA de latência)
Multi-turn tool calling	**Não suportado** (uma requisição = uma resposta)
Correlação	Campo `custom_id` para vincular requisição e resposta

7.2 Quando Usar Batch API vs API Síncrona

Tarefa	API	Por quê
Checagem de PR pré-merge	**Síncrona**	O dev está esperando; 24 horas é inaceitável
Relatório de tech-debt noturno	**Batch**	Resultado é necessário pela manhã; 50% de economia
Auditoria semanal de segurança	**Batch**	Não é urgente; 50% de economia
Code review interativo	**Síncrona**	Resposta imediata necessária
Processar 10.000 documentos	**Batch**	Processamento em massa; economia significativa

7.3 Usando `custom_id`

```
{
  "custom_id": "doc-invoice-2024-001",
  "params": {
    "model": "claude-sonnet-4-6",
    "max_tokens": 1024,
    "messages": [{"role": "user", "content": "Extract data from: ..."}]
  }
}
```

`custom_id` permite:

- Vincular o resultado ao documento original
- Em caso de falha, reenviar apenas os documentos que falharam
- Evitar reprocessar documentos bem-sucedidos

7.4 Tratando Falhas em Lotes

1. Envie um lote de 100 documentos

2. 95 obtêm sucesso; 5 falham (limite de contexto excedido)
3. Identifique falhas pelo `custom_id`
4. Modifique a estratégia (ex.: dividir documentos longos em chunks)
5. Reenvie apenas os 5 documentos que falharam

7.5 Planejamento de SLA

Se você precisa do resultado em 30 horas e a Batch API pode levar até 24:

- Janela de envio: 30 - 24 = **6 horas**
- Lotes devem ser enviados no máximo 24 horas antes do prazo
- Para envios frequentes, divida em janelas de 4 horas

Capítulo 8: Estratégias de Decomposição de Tarefas

8.1 Pipelines Fixos (Prompt Chaining)

Cada passo é definido com antecedência:

```
Document -> Metadata extraction -> Data extraction -> Validation -> Enrichment -> Final output
```

Quando usar:

- A estrutura da tarefa é previsível (reviews sempre seguem o mesmo template)
- Todos os passos são conhecidos de antemão
- Você precisa de estabilidade e reprodutibilidade

8.2 Decomposição Adaptativa Dinâmica

Subtarefas são geradas com base em resultados intermediários:

```
1. "Add tests for a legacy codebase"
2. -> First: map the structure (Glob, Grep)
3. -> Found: 3 modules with no tests, 2 with partial coverage
4. -> Prioritize: start with the payments module (high risk)
5. -> During work: discovered a dependency on an external API
6. -> Adapt: add a mock for the external API before writing tests
```

Quando usar:

- Tarefas investigativas abertas
- Quando o escopo completo é desconhecido de início

- Quando cada passo depende dos resultados do anterior

8.3 Code Review em Múltiplas Passagens

Para pull requests com 10+ arquivos:

```
Pass 1 (per-file): Analyze auth.ts -> list local issues
Pass 1 (per-file): Analyze database.ts -> list local issues
Pass 1 (per-file): Analyze routes.ts -> list local issues
...
Pass 2 (integration): Analyze relationships between files
  -> Cross-file issues: inconsistent types, circular dependencies
```

Por que uma única passagem sobre 14 arquivos é ruim:

- Diluição de atenção: análise profunda em alguns arquivos, superficial em outros
- Comentários inconsistentes: um padrão é sinalizado em um arquivo mas aprovado em outro
- Bugs perdidos: erros óbvios são pulados por sobrecarga cognitiva

---

Capítulo 9: Escalonamento e Human-in-the-Loop

9.1 Quando Escalar para um Humano

Gatilhos de escalonamento (regras claras):

Situação	Ação
O cliente pede explicitamente "passe para o gerente"	Escale imediatamente; não tente resolver
A política não cobre a solicitação	Escale (ex.: igualar preço de concorrente quando a política é silente)
O agente não consegue progredir	Escale após um número razoável de tentativas
Operação financeira acima de um limite	Escale (preferivelmente via hook, não via prompt)
Múltiplas correspondências ao buscar um cliente	Peça identificadores adicionais; não chute

O que NÃO é gatilho confiável:

Método não confiável	Por que falha
Análise de sentimento	Humor do cliente não correlaciona com complexidade do caso
Confiança auto-avaliada pelo modelo (1–10)	O modelo pode estar confiantemente errado; calibração é ruim
Classificador automático	Overengineering; pode exigir dados de treino que você não tem

9.2 Padrões de Escalonamento

Escalonamento imediato:

```
Customer: "I want to speak to a manager"
Agent: [immediately calls escalate_to_human]
NOT: "I can help with your issue, let me..."
```

Escalonamento após tentativa de resolução:

```
Customer: "My refrigerator broke two days after purchase"
Agent: [checks the order, offers a warranty replacement]
If the customer is not satisfied -> escalate
```

Escalonamento nuançado (acolher → resolver → escalar na reiteração):

```
Customer: "This is outrageous, I'm very unhappy with the quality!"
Agent: [acknowledges frustration] "I understand your frustration."
       [offers resolution] "I can offer a replacement or a refund."
Customer: "No, I want to talk to someone!"
Agent: [customer insists again -> immediate escalation]
```

Princípio-chave: acolha a emoção primeiro, depois proponha uma solução concreta, e só escale se o cliente reiterar o desejo de falar com humano. Não escale na primeira expressão de insatisfação (não é o mesmo que pedir um gerente).

Escalonamento por lacuna de política:

```
Customer: "Competitor X has this item 30% cheaper—give me a discount"
Policy: covers price adjustments only on your own site
Agent: [escalates — policy does not cover competitor price matching]
```

9.3 Protocolos de Handoff Estruturado

No escalonamento, o agente deve passar um sumário estruturado para o humano:

```
{
  "customer_id": "CUST-12345",
  "customer_name": "Ivan Petrov",
  "issue_summary": "Refund request for a damaged item",
  "order_id": "ORD-67890",
  "root_cause": "Item arrived damaged; photos attached",
  "actions_taken": [
    "Verified customer via get_customer",
    "Confirmed order via lookup_order",
    "Offered a standard replacement — customer insists on a refund"
  ],
  "refund_amount": "$89.99",
  "recommended_action": "Approve a full refund",
  "escalation_reason": "Customer requested to speak with a manager"
}
```

O operador humano não tem acesso ao transcript completo da conversa — vê apenas este sumário. Por isso ele precisa ser completo e auto-contido.

9.4 Calibração de Confiança e Supervisão Humana

Para sistemas de extração de dados:

1. **Scores de confiança por campo:** o modelo emite um score de confiança por campo extraído
2. **Calibração:** use conjuntos de validação rotulados para ajustar limiares
3. **Roteamento:**
   - Alta confiança + acurácia estável -> processamento automatizado
   - Baixa confiança ou fontes ambíguas -> revisão humana

Amostragem aleatória estratificada:

- Mesmo para extrações de alta confiança, audite uma amostra periodicamente
- Uma acurácia agregada de 97% pode esconder 40% de erros em um tipo específico de documento
- Analise acurácia por tipo de documento e por campo, não apenas no agregado

---

Capítulo 10: Tratamento de Erros em Sistemas Multi-agente

10.1 Categorias de Erro

Categoria	Exemplos	Retentável	Ação do agente
Transitório	Timeout, 503, falha de rede	Sim	Retry com backoff exponencial

Validação	Formato de entrada inválido, campo obrigatório ausente	Não (corrija a entrada)	Modifique a requisição e tente de novo
Negócio	Violação de política, limite excedido	Não	Explique ao usuário; proponha alternativa
Permissão	Acesso negado	Não	Escale

10.2 Anti-padrões de Tratamento de Erros

Anti-padrão	Problema	Abordagem correta
Status genérico "busca indisponível"	O coordenador não consegue decidir como recuperar	Retorne tipo de erro, query, resultados parciais, alternativas
Supressão silenciosa (resultado vazio = sucesso)	Coordenador acha que não houve correspondências, mas foi falha	Distinga "sem resultados" de "falha de busca"
Abortar todo o workflow em uma falha	Você perde todos os resultados parciais	Continue com resultados parciais; anote lacunas
Retentativas infinitas dentro de um subagente	Latência e desperdício de recursos	Recuperação local (1–2 retries), depois propague ao coordenador

10.3 Erro Estruturado de Subagente

```
{
  "status": "partial_failure",
  "failure_type": "timeout",
  "attempted_query": "AI impact on music industry 2024",
  "partial_results": [
    {"title": "AI Music Generation Report", "url": "...", "relevance": 0.8}
  ],
  "alternative_approaches": [
    "Try a narrower query: 'AI music composition tools'",
    "Use an alternative data source"
  ],
  "coverage_impact": "Not covered: AI impact on music production"
}
```

Isso fornece ao coordenador a informação necessária para decidir:

- Retentar com query modificada?
- Usar resultados parciais?
- Delegar a outro subagente?
- Continuar sem essa seção e anotar a lacuna?

10.4 Anotações de Cobertura na Síntese Final

```
## Report: AI Impact on Creative Industries

### Visual Art (FULL COVERAGE)
[research results]

### Music (PARTIAL COVERAGE — search agent timeout)
[partial results]
⚠️ Note: coverage for this section is limited due to a timeout in the search agent.

### Literature (FULL COVERAGE)
[research results]
```

Capítulo 11: Gerenciamento de Contexto em Sistemas de Produção

11.1 Extrair Fatos para um Bloco Separado

Em vez de depender do histórico da conversa (que se degrada na sumarização), extraia fatos-chave para um bloco estruturado:

```
=== CASE FACTS (updated whenever a new fact appears) ===
Customer ID: CUST-12345
Order ID: ORD-67890
Order Date: 2025-01-15
Order Amount: $89.99
Issue: Damaged item on delivery
Customer Request: Full refund
Status: Pending manager approval
===
```

Inclua esse bloco em todo prompt, independentemente de como o histórico for sumarizado.

11.2 Aparando Resultados de Ferramentas

Se `lookup_order` retorna 40+ campos mas você precisa de só 5 para a tarefa atual:

```
# PostToolUse hook: keep only relevant fields
@hook("PostToolUse", tool="lookup_order")
def trim_order_fields(result):
    return {
        "order_id": result["order_id"],
        "status": result["status"],
        "total": result["total"],
        "items": result["items"],
        "return_eligible": result["return_eligible"]
    }
```

Isso preserva contexto e reduz ruído.

11.3 Entrada Sensível à Posição

Posicione informação crítica considerando o efeito lost-in-the-middle:

```
[KEY FINDINGS — at the top]
Found 3 critical vulnerabilities...

[DETAILED RESULTS — middle]
=== File auth.ts ===
...
=== File database.ts ===
...

[ACTION ITEMS — at the end]
Priority: fix auth.ts vulnerabilities before merge.
```

11.4 Arquivos de Scratchpad

Em investigações longas, o agente pode escrever achados-chave em um arquivo de scratchpad:

```
# investigation-scratchpad.md
## Key findings
- PaymentProcessor in src/payments/processor.ts inherits from BaseProcessor
- refund() is called from 3 places: OrderController, AdminPanel, CronJob
- External PaymentGateway API has a rate limit of 100 req/min
- Migration #47 added refund_reason (NOT NULL) — 2024-12-01
```

Quando o contexto se degrada (ou em uma nova sessão), o agente pode consultar o scratchpad em vez de refazer a descoberta.

11.5 Delegar a Subagentes para Proteger Contexto

```
Main agent: "Investigate dependencies of the payments module"
  -> Subagent (Explore): reads 15 files, traces imports
  -> Returns: "Payments depends on AuthService, OrderModel, and the external
PaymentGateway API"

Main agent: keeps one line in context instead of 15 files
```

Camada de contexto separada: Em sistemas multi-agente, cada subagente opera dentro de um orçamento limitado de contexto — recebe apenas a informação necessária para sua tarefa. O coordenador atua como uma camada de contexto separada: agrega saídas de subagentes, armazena estado global e aloca contexto. Isso evita "vazamento de contexto", em que um agente consome a janela com informação irrelevante para os outros.

Orçamentos de contexto restritos para subagentes:

- Envie contexto mínimo: uma tarefa específica + dados necessários
- Instrua o subagente a retornar resultados estruturados, não despejos de dados crus
- Use `allowedTools` para limitar o conjunto de ferramentas do subagente — menos ferramentas significa menos distrações e menor custo de contexto

11.6 Persistência Estruturada de Estado (para recuperação de crash)

Cada agente exporta seu estado para um local conhecido:

```
// agent-state/web-search-agent.json
{
  "status": "completed",
  "queries_executed": ["AI music 2024", "AI music composition"],
  "results_count": 12,
  "key_findings": [...],
  "coverage": ["music composition", "music production"],
  "gaps": ["music distribution", "music licensing"]
}
```

O coordenador carrega um manifesto ao retomar:

```
// agent-state/manifest.json
{
  "web-search": "completed",
  "doc-analysis": "in_progress",
  "synthesis": "not_started"
}
```

Capítulo 12: Preservando Proveniência

12.1 O Problema da Perda de Atribuição

Ao sumarizar resultados de várias fontes, o vínculo "afirmação → fonte" pode se perder:

```
Bad: "The AI music market is estimated at $3.2B." (No source, no year.)

Good:
{
  "claim": "The AI music market is estimated at $3.2B.",
  "source_url": "https://example.com/report",
  "source_name": "Global AI Music Report 2024",
  "publication_date": "2024-06-15",
  "confidence": 0.9
}
```

12.2 Tratando Dados Conflitantes

Quando duas fontes fornecem valores diferentes:

```
{
  "claim": "Share of AI-generated music on streaming platforms",
  "values": [
    {
      "value": "12%",
      "source": "Spotify Annual Report 2024",
      "date": "2024-03",
      "methodology": "Automated classification"
    },
    {
      "value": "8%",
      "source": "Music Industry Association Survey",
      "date": "2024-07",
      "methodology": "Survey of 500 labels"
    }
  ],
  "conflict_detected": true,
  "possible_explanation": "Difference in methodology and time period"
}
```

Não escolha um valor arbitrariamente. Preserve ambos com atribuição e deixe o coordenador decidir.

12.3 Inclua Datas para Interpretação Correta

Sem datas, diferenças temporais podem ser interpretadas erroneamente como contradições:

```
Bad: "Source A says 10%, source B says 15%. Contradiction."
Good: "Source A (2023) says 10%, source B (2024) says 15%. Likely +5% growth over a
year."
```

12.4 Renderize por Tipo de Conteúdo

Não force tudo em um único formato:

- Dados financeiros -> tabelas
- Notícias e análises -> prosa
- Achados técnicos -> listas estruturadas
- Séries temporais -> ordenação cronológica

Capítulo 13: Ferramentas Embutidas do Claude Code

13.1 Referência de Seleção de Ferramentas

Tarefa	Ferramenta	Exemplo
Encontrar arquivos por nome/padrão	**Glob**	`**/*.test.tsx`, `src/components/**/*.ts`
Buscar dentro de arquivos	**Grep**	Nome de função, mensagem de erro, import
Ler arquivo na íntegra	**Read**	Carregar arquivo para análise
Escrever arquivo novo	**Write**	Criar arquivo do zero
Editar arquivo existente com precisão	**Edit**	Substituir snippet específico via match único de texto
Executar comando shell	**Bash**	git, npm, rodar testes, build

13.2 Estratégia de Investigação Incremental

Não leia todos os arquivos de uma vez. Construa entendimento incrementalmente:

```
1. Grep: find entry points (function definition, export)
2. Read: read the found files
3. Grep: find usages (import, calls)
4. Read: read consumer files
5. Repeat until you have a complete picture
```

13.3 Fallback: Read + Write em vez de Edit

Quando Edit falha por correspondência de texto não única:

1. Read — carregue o conteúdo completo do arquivo
2. Modifique o conteúdo programaticamente
3. Write — escreva a versão atualizada

PARTE II: NOTAS DOS DOMÍNIOS DO EXAME

Domínio 1: Arquitetura e Orquestração de Agentes (27%)

1.1 Projetando Loops Agênticos para Execução Autônoma de Tarefas

Conhecimento-chave:

- Ciclo de vida do loop do agente: enviar requisição ao Claude, verificar `stop_reason` (`"tool_use"` vs `"end_turn"`), executar ferramentas, retornar resultados para a próxima iteração
- Resultados de ferramentas são anexados ao histórico da conversa para que o modelo possa decidir a próxima ação
- Tomada de decisão dirigida pelo modelo (Claude escolhe a próxima ferramenta) vs árvores de decisão hard-coded

Habilidades-chave:

- Controle de fluxo: continue o loop quando `stop_reason = "tool_use"` e pare em `"end_turn"`
- Anexar resultados de ferramentas ao contexto entre iterações
- Anti-padrões a evitar: parsear texto do assistente para detectar conclusão, usar limites arbitrários de iteração como mecanismo primário de parada

1.2 Orquestrando Sistemas Multi-agente (Coordenador–Subagente)

Conhecimento-chave:

- Arquitetura hub-and-spoke: o coordenador detém toda comunicação inter-agentes, tratamento de erros e roteamento
- Subagentes operam com contexto isolado — não herdam automaticamente o histórico do coordenador

- Responsabilidades do coordenador: decomposição de tarefas, delegação, agregação de resultados, seleção dinâmica de subagentes
- Risco de decomposição excessivamente estreita pelo coordenador

Habilidades-chave:

- Dividir cobertura de pesquisa entre subagentes para minimizar duplicação
- Implementar loops iterativos de refinamento (coordenador avalia síntese e re-roteia tarefas)
- Rotear toda comunicação pelo coordenador para observabilidade

1.3 Configurando Chamadas de Subagente, Passagem de Contexto e Spawn

Conhecimento-chave:

- A ferramenta `Task` spawna subagentes; os `allowedTools` do coordenador devem incluir `"Task"`
- O contexto do subagente deve ser explicitamente incluído no prompt; subagentes não herdam contexto pai
- Configuração `AgentDefinition`: descrições, system prompts, restrições de ferramentas
- Gerenciamento de sessão via `fork_session` para explorar alternativas

Habilidades-chave:

- Inclua saídas completas de agentes anteriores no prompt do subagente
- Use formatos estruturados para separar dados de metadados ao passar contexto
- Spawne subagentes em paralelo via múltiplas chamadas `Task` em um único turno do coordenador
- Escreva prompts do coordenador em termos de objetivos e critérios de qualidade em vez de instruções passo-a-passo

1.4 Implementando Workflows Multi-passo com Padrões de Enforcement e Handoff

Conhecimento-chave:

- A diferença entre **enforcement programático** (hooks, pré-condições) e **orientação por prompt** para ordenar um workflow
- Quando você precisa de garantias determinísticas (ex.: verificação de identidade antes de operações financeiras), prompts sozinhos são insuficientes
- Protocolos de handoff estruturado durante escalonamento (ID do cliente, motivo, ação recomendada)

Habilidades-chave:

- Pré-condições programáticas que bloqueiam chamadas a jusante até que passos anteriores estejam completos (ex.: bloquear `process_refund` até que `get_customer` retorne um ID verificado)
- Decompor pedidos multi-aspecto do cliente em itens separados
- Produzir sumários estruturados ao escalar para um humano

1.5 Hooks do Agent SDK para Interceptar Chamadas de Ferramenta e Normalizar Dados

Conhecimento-chave:

- Padrões de hook (ex.: `PostToolUse`) para interceptar resultados de ferramentas antes de o modelo consumi-los
- Hooks que interceptam chamadas de saída para aplicar regras de compliance (ex.: bloquear reembolsos acima de um limite)
- Hooks fornecem **garantias determinísticas** vs instruções no prompt que fornecem **conformidade probabilística**

Habilidades-chave:

- Hooks `PostToolUse` para normalizar formatos de dados (timestamps Unix, ISO 8601, códigos de status numéricos)
- Hooks de interceptação para bloquear ações que violem políticas com redirecionamento para escalonamento
- Escolha hooks em vez de prompts quando regras de negócio exigem conformidade garantida

1.6 Estratégias de Decomposição de Tarefas para Workflows Complexos

Conhecimento-chave:

- **Pipelines fixos** (prompt chaining) vs **decomposição adaptativa dinâmica** baseada em resultados intermediários
- Prompt chaining: passos sequenciais (analise cada arquivo separadamente, depois rode uma passagem de integração)
- Planos investigativos adaptativos que geram subtarefas baseadas no que foi descoberto

Habilidades-chave:

- Use prompt chaining para reviews previsíveis multi-aspecto; use decomposição dinâmica para investigações abertas
- Divida code reviews grandes em análise por arquivo mais uma passagem cross-file separada
- Decomponha tarefas abertas: mapeie a estrutura primeiro, depois construa um plano priorizado

1.7 Estado de Sessão, Resumo e Forking

Conhecimento-chave:

- `--resume <session-name>` para continuar sessões nomeadas
- `fork_session` para criar branches investigativos independentes a partir de contexto compartilhado
- A importância de informar o agente sobre mudanças em arquivos ao retomar sessões

- Uma nova sessão com sumário estruturado pode ser mais confiável do que retomar com resultados desatualizados

Habilidades-chave:

- Use `--resume` para continuar sessões investigativas nomeadas
- Use `fork_session` para comparar abordagens em paralelo
- Escolha entre retomar (contexto ainda válido) vs iniciar nova sessão (resultados desatualizados)

Domínio 2: Design de Ferramentas e Integração com MCP (18%)

2.1 Projetando Interfaces de Ferramentas com Descrições Claras

Conhecimento-chave:

- As descrições das ferramentas são o **mecanismo principal** que um LLM usa para selecioná-las; descrições mínimas levam a seleção pouco confiável
- A importância de incluir formatos de entrada, queries de exemplo, casos de borda e limites de aplicabilidade
- Descrições ambíguas ou sobrepostas causam roteamento errado
- O texto do system prompt pode criar associações não intencionais com ferramentas

Habilidades-chave:

- Escreva descrições que distingam claramente cada ferramenta de alternativas similares
- Renomeie ferramentas para eliminar sobreposição funcional (ex.: `analyze_content` -> `extract_web_results`)
- Divida ferramentas de uso geral em especializadas com contratos claros de entrada/saída

2.2 Implementando Respostas de Erro Estruturadas para Ferramentas MCP

Conhecimento-chave:

- A flag `isError` em respostas de ferramentas MCP
- A diferença entre **erros transitórios** (timeouts), **erros de validação** (entrada ruim), **erros de negócio** (violação de política) e **erros de acesso/permissão**
- Erros genéricos ("Operation failed") impedem decisões corretas de recuperação
- A diferença entre erros retentáveis e não-retentáveis

Habilidades-chave:

- Retorne metadados estruturados como `errorCategory` (transient/validation/permission), `isRetryable` e mensagem legível
- Use `retryable: false` para violações de regras de negócio com explicações claras para o usuário
- Faça recuperação local em subagentes para falhas transitórias; propague apenas o que não conseguir resolver
- Distinga falhas de acesso (decisão de retry) de resultados vazios válidos (sem correspondência)

2.3 Alocando Ferramentas Entre Agentes e Configurando `tool_choice`

Conhecimento-chave:

- Ferramentas demais por agente (ex.: 18 em vez de 4–5) **reduz** a confiabilidade da seleção
- Agentes com ferramentas fora de sua especialização tendem a usá-las mal
- Acesso de ferramentas com escopo: apenas ferramentas relevantes ao papel mais um conjunto limitado de utilitários cross-role
- `tool_choice`: `"auto"`, `"any"` e seleção forçada de ferramenta (`{"type": "tool", "name": "..."}`)

Habilidades-chave:

- Restrinja o conjunto de ferramentas de cada subagente ao que é relevante para seu papel
- Substitua ferramentas gerais por alternativas restritas (ex.: `fetch_url` -> `load_document`)
- Use `tool_choice: "any"` para garantir uma chamada de ferramenta em vez de resposta em texto
- Force uma ferramenta específica para garantir ordem de execução

2.4 Integrando Servidores MCP em Claude Code e Workflows de Agentes

Conhecimento-chave:

- Escopo de servidor MCP: projeto (`.mcp.json`) para times vs usuário (`~/.claude.json`) para experimentos
- Substituição de variáveis de ambiente em `.mcp.json` (ex.: `${GITHUB_TOKEN}`) para gerenciamento de segredos
- Ferramentas de todos os servidores MCP conectados são descobertas na conexão e ficam disponíveis simultaneamente
- MCP resources como "catálogos de conteúdo" (sumários de tarefas, schemas de banco) para reduzir chamadas exploratórias

Habilidades-chave:

- Configure servidores MCP compartilhados no `.mcp.json` do projeto com tokens via env-var
- Mantenha servidores pessoais/experimentais em `~/.claude.json`

- Prefira servidores MCP da comunidade em vez de servidores customizados para integrações padrão

2.5 Selecionando e Aplicando Ferramentas Embutidas (Read, Write, Edit, Bash, Grep, Glob)

Conhecimento-chave:

- **Grep**: busca dentro do conteúdo de arquivos (nomes de função, mensagens de erro, imports)
- **Glob**: encontra arquivos por padrões de nome/extensão
- **Read/Write**: operações sobre o arquivo inteiro; **Edit**: mudanças precisas via match único de texto
- Se Edit falhar por correspondências não únicas, recorra a Read + Write

Habilidades-chave:

- Use Grep para busca de conteúdo e Glob para descoberta de arquivos por padrões
- Construa entendimento incrementalmente: Grep nos pontos de entrada, depois Read para rastrear fluxos
- Rastreie uso de funções por meio de módulos wrapper

Domínio 3: Configuração e Fluxos de Trabalho do Claude Code (20%)

3.1 Configurando CLAUDE.md com Hierarquia, Escopo e Organização Modular

Conhecimento-chave:

- Hierarquia CLAUDE.md: usuário (`~/.claude/CLAUDE.md`), projeto (`.claude/CLAUDE.md` ou `CLAUDE.md` na raiz) e nível diretório (CLAUDE.md em subdiretórios)
- Configurações de nível usuário se aplicam a um único usuário e não são compartilhadas via VCS
- Sintaxe `@path` para referenciar arquivos externos (ex.: `@./standards/coding-style.md`) modulariza o CLAUDE.md
- Diretório `.claude/rules/` para arquivos de regras focados por tópico em vez de um CLAUDE.md monolítico

Habilidades-chave:

- Diagnostique problemas de hierarquia (um novo membro do time perde instruções porque foram colocadas em nível usuário em vez de projeto)
- Use `@path` (ex.: `@./standards/testing.md`) para incluir padrões seletivamente no CLAUDE.md de cada pacote
- Divida um CLAUDE.md grande em múltiplos arquivos `.claude/rules/` (testing.md, api-conventions.md, deployment.md)

3.2 Criando e Configurando Slash Commands e Skills Customizados

Conhecimento-chave:

- **Comandos de projeto** em `.claude/commands/` (compartilhados via VCS) vs **comandos de usuário** em `~/.claude/commands/`
- Skills em `.claude/skills/` com frontmatter `SKILL.md`: `context: fork`, `allowed-tools`, `argument-hint`
- `context: fork` roda a skill em um contexto de subagente isolado, não poluindo a sessão principal
- Variantes pessoais de skills podem residir em `~/.claude/skills/` sob nomes diferentes

Habilidades-chave:

- Armazene slash commands de projeto em `.claude/commands/` para que todo o time os tenha
- Use `context: fork` para isolar skills com saída verbosa
- Use `allowed-tools` para restringir quais ferramentas a skill pode usar
- Use `argument-hint` para pedir parâmetros aos desenvolvedores

3.3 Usando Regras Específicas por Path para Carregamento Condicional de Convenções

Conhecimento-chave:

- Arquivos em `.claude/rules/` podem incluir frontmatter YAML `paths` para ativar regras com base em padrões glob
- Regras escopadas por path carregam **apenas** ao editar arquivos correspondentes, economizando contexto e tokens
- Regras baseadas em globs podem ser preferíveis a CLAUDE.md em nível de diretório quando convenções se aplicam a vários diretórios (ex.: testes)

Habilidades-chave:

- Crie arquivos `.claude/rules/` com `paths: ["terraform/**/*"]` para carregar apenas ao trabalhar em arquivos correspondentes
- Use padrões glob (`**/*.test.tsx`) para aplicar convenções por tipo de arquivo, independente da localização
- Prefira regras path-specific a CLAUDE.md em nível de diretório quando convenções atravessam a base de código

3.4 Decidindo Quando Usar Modo de Planejamento vs Execução Direta

Conhecimento-chave:

- **Modo de planejamento**: para tarefas complexas com mudanças grandes, várias abordagens viáveis e decisões arquiteturais
- **Execução direta**: para mudanças simples e bem entendidas (ex.: adicionar uma única validação)
- Modo de planejamento permite exploração segura da base antes de fazer mudanças
- Subagente Explore isola saída verbosa de descoberta

Habilidades-chave:

- Use modo de planejamento para tarefas com consequências arquiteturais (microsserviços, migrações afetando 45+ arquivos)
- Use execução direta para correções com stack trace claro e arquivo único
- Use o subagente Explore para evitar esgotar a janela de contexto em tarefas multi-fase
- Combine abordagens: plano para descoberta, execução para implementação

3.5 Refinamento Iterativo para Melhoria Progressiva

Conhecimento-chave:

- Exemplos concretos de entrada/saída são a forma mais efetiva de comunicar expectativas
- **Iteração orientada a teste**: escreva testes primeiro, depois itere com base em falhas
- O padrão "entrevista": Claude faz perguntas para fazer emergir considerações de design não óbvias
- Quando fornecer todos os problemas em uma mensagem (interdependentes) vs sequencialmente (independentes)

Habilidades-chave:

- Forneça 2–3 exemplos concretos de entrada/saída para clarificar requisitos de transformação
- Construa conjuntos de teste com comportamento esperado, casos de borda e requisitos de performance antes da implementação
- Use o padrão de entrevista para fazer emergir aspectos de design (invalidação de cache, modos de falha)
- Forneça casos de teste concretos com entradas-exemplo e saídas esperadas para casos de borda

3.6 Integrando Claude Code em Pipelines CI/CD

Conhecimento-chave:

- A flag `-p` (ou `--print`) para modo não-interativo em pipelines automatizados
- `--output-format json` e `--json-schema` para saída estruturada em CI

- CLAUDE.md fornece contexto de projeto (padrões de teste, critérios de review) para Claude Code disparado por CI
- **Isolamento de contexto entre sessões**: a mesma sessão que gerou o código é menos eficaz ao revisá-lo do que uma instância independente

Habilidades-chave:

- Rode Claude Code no CI com `-p` para evitar travar esperando entrada interativa
- Use `--output-format json` + `--json-schema` para resultados estruturados (ex.: comentários inline em PRs)
- Inclua resultados de reviews anteriores ao rerodar após novos commits (reporte apenas problemas novos/não-corrigidos)
- Documente padrões de teste e fixtures disponíveis no CLAUDE.md para melhorar a qualidade da geração de testes
- Inclua arquivos de teste existentes no contexto ao gerar novos testes para evitar duplicação e manter estilo consistente

---

Domínio 4: Engenharia de Prompts e Saída Estruturada (20%)

4.1 Projetando Prompts com Critérios Explícitos para Melhorar a Acurácia

Conhecimento-chave:

- Critérios explícitos são mais efetivos que instruções vagas (ex.: "marque comentários apenas quando contradizem o código" vs "verifique a acurácia do comentário")
- Orientação genérica como "seja mais conservador" funciona pior que critérios categóricos concretos
- O efeito de falsos positivos sobre a confiança do dev: altas taxas de FP em algumas categorias minam a confiança em categorias acuradas

Habilidades-chave:

- Defina critérios de review: o que reportar (bugs, segurança) vs o que ignorar (pequenos detalhes de estilo)
- Desabilite temporariamente categorias com altas taxas de falsos positivos
- Defina critérios explícitos de severidade com exemplos de código para cada nível

4.2 Usando Few-shot Prompting para Melhorar a Consistência da Saída

Conhecimento-chave:

- Exemplos few-shot são o método mais efetivo para produzir saída acionável e consistentemente formatada
- Few-shot pode demonstrar tratamento de casos ambíguos (seleção de ferramenta, lacunas de cobertura de teste)
- Few-shot ajuda o modelo a generalizar para novos padrões em vez de só repetir padrões default
- Few-shot pode reduzir alucinações em tarefas de extração

Habilidades-chave:

- Forneça 2–4 exemplos direcionados para cenários ambíguos com justificativa
- Inclua exemplos few-shot que demonstrem o formato de saída (location, issue, severity, suggested fix)
- Forneça exemplos que distingam padrões aceitáveis de problemas reais
- Forneça exemplos de extração correta a partir de documentos com diferentes estruturas

4.3 Aplicando Saída Estruturada com `tool_use` e JSON Schemas

Conhecimento-chave:

- `tool_use` com JSON Schemas é a forma mais confiável de garantir saída conforme com schema e eliminar erros de sintaxe JSON
- Com `tool_choice: "auto"` o modelo pode retornar texto; com `"any"` deve chamar uma ferramenta; seleção forçada escolhe uma ferramenta específica
- JSON Schemas estritos eliminam erros de sintaxe, mas não previnem erros semânticos (totais que não batem; valores em campos errados)
- Design de schema: campos obrigatórios vs opcionais; enums com "other" mais string de detalhe para extensibilidade

Habilidades-chave:

- Defina ferramentas de extração com JSON Schemas e parseie dados a partir de resultados de `tool_use`
- Use `tool_choice: "any"` para garantir saída estruturada quando há múltiplos schemas
- Force a chamada de uma ferramenta específica: `tool_choice: {"type": "tool", "name": "extract_metadata"}`
- Torne campos opcionais/nuláveis quando a fonte pode não conter informação para evitar fabricar valores
- Use valores enum como `"unclear"` e `"other"` mais campos de detalhe para categorização extensível

4.4 Implementando Validação, Retries e Feedback Loops para Qualidade de Extração

Conhecimento-chave:

- Retry-com-feedback-de-erro: inclua erros concretos de validação no prompt de retry para guiar correções
- Retries são inefetivos quando a informação simplesmente não está na fonte
- Design de feedback loop: rastreie o padrão que disparou o achado (`detected_pattern`)
- Erros semânticos (totais não batem) vs erros de sintaxe (resolvidos por `tool_use`)

Habilidades-chave:

- Prompts de follow-up com o documento original, uma extração incorreta e erros específicos de validação
- Identifique quando retry será inefetivo (a informação necessária está em documento externo)
- Inclua campos `detected_pattern` em achados para analisar falsos positivos
- Projete autocorreção extraindo `calculated_total` e `stated_total` para detectar discrepâncias

4.5 Projetando Estratégias Eficientes de Processamento em Lote

Conhecimento-chave:

- Message Batches API: 50% de economia, janela de processamento de até 24 horas, sem garantias de SLA de latência
- Processamento em lote é adequado para tarefas não-bloqueantes (relatórios noturnos, auditorias) e não para bloqueantes (checagens pré-merge)
- Batch API não suporta tool calling multi-turn dentro de uma única requisição
- Campos `custom_id` correlacionam request/response em lotes

Habilidades-chave:

- Use API síncrona para checagens bloqueantes; use Batch API para cargas noturnas/semanais
- Planeje cadência de envio de lotes baseada nas necessidades de SLA (ex.: janelas de 4 horas para uma garantia de 30 horas com processamento de até 24)
- Trate falhas reenviando apenas os documentos que falharam (identificados por `custom_id`)
- Itere prompts em uma amostra antes de rodar processamento em larga escala

4.6 Projetando Arquiteturas de Review Multi-instância e Multi-passagem

Conhecimento-chave:

- Limitações da auto-review: o modelo retém seu contexto de raciocínio e tem menos propensão a desafiar suas próprias decisões

- Instâncias de review independentes (sem contexto de geração) são melhores em encontrar problemas sutis
- Review multi-passagem: análise local por arquivo mais uma passagem cross-file de integração para evitar diluição de atenção

Habilidades-chave:

- Use uma segunda instância independente do Claude para revisar as mudanças sem contexto de geração
- Divida reviews de múltiplos arquivos em passagens por arquivo mais passagens de integração para análise de fluxo de dados cross-file
- Use passagens de verificação com confiança auto-avaliada para rotear reviews de forma calibrada

---

Domínio 5: Gerenciamento de Contexto e Confiabilidade (15%)

5.1 Gerenciando Contexto da Conversa para Preservar Informação Crítica

Conhecimento-chave:

- Riscos da sumarização progressiva: valores numéricos, percentuais e datas são condensados em sumários vagos
- Efeito lost-in-the-middle: modelos processam o início e o fim de entradas longas de forma confiável, mas podem perder achados do meio
- Saídas de ferramentas podem se acumular no contexto desproporcionalmente à relevância (40+ campos quando 5 são necessários)
- A importância de enviar o histórico completo da conversa em requisições subsequentes à API

Habilidades-chave:

- Extraia fatos transacionais para um bloco persistente "case facts" fora do histórico sumarizado
- Apare saídas verbosas de ferramentas, mantendo apenas campos relevantes
- Posicione achados-chave no início de dados agregados com cabeçalhos de seção explícitos
- Exija que subagentes incluam metadados (datas, fontes) em saídas estruturadas

5.2 Projetando Padrões Eficazes de Escalonamento e Resolvendo Ambiguidade

Conhecimento-chave:

- Gatilhos de escalonamento adequados: pedido explícito por humano, lacunas/exceções de política, incapacidade de progredir

- Escalonamento imediato (pedido explícito) vs tentativa de resolução (no escopo do agente)
- Análise de sentimento e auto-avaliações de confiança do modelo são proxies não confiáveis de complexidade do caso
- Múltiplas correspondências de cliente exigem pedir identificadores adicionais, não chutar heuristicamente

Habilidades-chave:

- Critérios explícitos de escalonamento com exemplos few-shot no system prompt
- Execute pedidos explícitos por humano imediatamente, sem investigação adicional
- Escale quando a política for ambígua ou silente para um pedido específico
- Peça identificadores adicionais quando resultados de ferramentas contiverem múltiplas correspondências

5.3 Implementando Estratégias de Propagação de Erro em Sistemas Multi-agente

Conhecimento-chave:

- Contexto de erro estruturado (tipo de falha, query, resultados parciais, alternativas) habilita recuperação mais inteligente do coordenador
- Distinga falhas de acesso (timeouts exigem decisão de retry) de resultados vazios válidos (sem correspondência)
- Status de erro genérico ("busca indisponível") esconde contexto valioso do coordenador
- Supressão silenciosa ou abortar todo o workflow em uma única falha são ambos anti-padrões

Habilidades-chave:

- Retorne contexto estruturado de erro: tipo de falha, o que foi tentado, resultados parciais, alternativas possíveis
- Distinga falhas de acesso de resultados vazios válidos
- Faça recuperação local em subagentes para falhas transitórias; propague apenas erros não recuperáveis com resultados parciais
- Anote cobertura na síntese: o que está bem suportado vs onde restam lacunas

5.4 Gerenciando Contexto de Forma Eficiente ao Investigar Bases de Código Grandes

Conhecimento-chave:

- Degradação de contexto em sessões longas: o modelo passa a produzir respostas instáveis e referir-se a "padrões típicos" em vez de classes específicas
- Arquivos de scratchpad preservam achados-chave através de fronteiras de contexto
- Delegar a subagentes isola saída verbosa de descoberta
- Persistência estruturada de estado habilita recuperação de crash

Habilidades-chave:

- Spawne subagentes para perguntas específicas mantendo a coordenação de alto nível no agente principal
- Use arquivos de scratchpad para armazenar achados-chave e referenciá-los depois
- Sumarize achados-chave antes de spawnar subagentes da próxima fase
- Use `/compact` para reduzir uso de contexto em investigações longas

5.5 Projetando Workflows com Supervisão Humana e Calibração de Confiança

Conhecimento-chave:

- Métricas agregadas (ex.: 97% de acurácia geral) podem mascarar desempenho ruim em tipos específicos de documento ou em campos
- Amostragem aleatória estratificada mede taxas de erro em extrações de alta confiança
- Calibração de confiança em nível de campo usando conjuntos de validação rotulados
- Valide acurácia por tipo de documento e segmento de campo antes de automatizar

Habilidades-chave:

- Implemente amostragem aleatória estratificada para detectar novos padrões de erro
- Analise acurácia por tipo de documento e campo para validar performance estável
- Emita scores de confiança por campo e calibre limiares de revisão usando dados rotulados
- Roteie extrações de baixa confiança ou de fontes ambíguas para revisão humana

5.6 Preservando Proveniência e Tratando Incerteza em Síntese Multi-fonte

Conhecimento-chave:

- A atribuição se perde durante sumarização sem preservar mapeamentos "afirmação → fonte"
- Mapeamentos estruturados precisam ser preservados durante a agregação
- Trate estatísticas conflitantes anotando conflitos com atribuição em vez de escolher arbitrariamente um valor
- Inclua datas de publicação/coleta para evitar interpretar diferenças temporais como contradições

Habilidades-chave:

- Exija que subagentes emitam mapeamentos "afirmação → fonte" (URL, nome do documento, citações)
- Estruture relatórios para separar achados estáveis de disputados
- Preserve valores conflitantes com anotações e passe-os ao coordenador para reconciliação
- Inclua datas de publicação para interpretação temporal correta
- Renderize conteúdo por tipo: dados financeiros como tabelas, notícias como prosa, achados técnicos como listas estruturadas

Exemplos de Questões do Exame com Explicações

Questão 1 (Cenário: Agente de Suporte ao Cliente)

Situação: Os dados mostram que em 12% dos casos o agente pula `get_customer` e chama `lookup_order` usando apenas o nome do cliente, o que leva a reembolsos incorretos.

Qual mudança é mais efetiva?

- A) Adicionar uma pré-condição programática que bloqueia `lookup_order` e `process_refund` até que um ID seja obtido de `get_customer` **[CORRETA]**
- B) Melhorar o system prompt
- C) Adicionar exemplos few-shot
- D) Implementar um classificador de roteamento

Por que A: Quando lógica de negócio crítica exige uma sequência específica de ferramentas, software fornece **garantias determinísticas** que abordagens baseadas em prompt (B, C) não conseguem. D trata disponibilidade, não a ordenação de ferramentas.

Questão 2 (Cenário: Agente de Suporte ao Cliente)

Situação: O agente frequentemente chama `get_customer` em vez de `lookup_order` para perguntas relacionadas a pedidos. As descrições das ferramentas são mínimas e parecidas.

Qual é o primeiro passo?

- A) Exemplos few-shot
- B) Expandir a descrição de cada ferramenta com formatos de entrada, exemplos e limites **[CORRETA]**
- C) Adicionar uma camada de roteamento
- D) Mesclar as ferramentas

Por que B: Descrições de ferramentas são o principal mecanismo de seleção do modelo. É a correção de menor esforço e maior impacto. A adiciona tokens sem atacar a causa raiz. C é overengineering. D exige mais esforço do que se justifica.

Questão 3 (Cenário: Agente de Suporte ao Cliente)

Situação: O agente resolve apenas 55% dos problemas com meta de 80%. Ele escala casos simples e tenta lidar com exceções complexas de política autonomamente.

Como melhorar a calibração?

- A) Adicionar critérios explícitos de escalonamento com exemplos few-shot **[CORRETA]**
- B) Confiança auto-avaliada (1–10) com escalonamento automático
- C) Um classificador separado treinado em dados históricos
- D) Análise de sentimento

Por que A: Ataca diretamente a causa raiz — limites de decisão pouco claros. B é não confiável (o modelo pode estar confiantemente errado). C é overengineering. D resolve outro problema (humor != complexidade).

Questão 4 (Cenário: Geração de Código com Claude Code)

Situação: Você precisa de um comando customizado `/review` para code review padrão que esteja disponível para todo o time quando clonarem o repositório.

Onde criar o arquivo do comando?

- A) `.claude/commands/` no repositório do projeto **[CORRETA]**
- B) `~/.claude/commands/`
- C) `CLAUDE.md` na raiz
- D) `.claude/config.json`

Por que A: Comandos de projeto armazenados em `.claude/commands/` são versionados e ficam automaticamente disponíveis para todos. B é para comandos pessoais. C é para instruções, não definições de comando. D não existe.

Questão 5 (Cenário: Geração de Código com Claude Code)

Situação: Você precisa reestruturar um monolito em microsserviços (dezenas de arquivos, decisões sobre fronteiras de serviço).

Qual abordagem usar?

- A) Modo de planejamento: explore a base, entenda dependências, desenhe a abordagem **[CORRETA]**
- B) Execução direta de forma incremental
- C) Execução direta com instruções detalhadas antecipadas
- D) Execução direta e mude para planejamento quando ficar difícil

Por que A: O modo de planejamento foi projetado para mudanças grandes, múltiplas abordagens possíveis e decisões arquiteturais. B arrisca retrabalho caro. C assume que você já conhece a estrutura. D é reativa.

Questão 6 (Cenário: Geração de Código com Claude Code)

Situação: Uma base de código tem convenções diferentes em áreas distintas (React, API, banco de dados). Testes ficam co-localizados com o código. Você quer que as convenções sejam aplicadas automaticamente.

Qual abordagem usar?

- A) Arquivos `.claude/rules/` com frontmatter YAML e padrões glob **[CORRETA]**
- B) Colocar tudo no CLAUDE.md da raiz
- C) Skills em `.claude/skills/`
- D) CLAUDE.md em todo diretório

Por que A: `.claude/rules/` com padrões glob (ex.: `**/*.test.tsx`) habilita aplicação automática de convenções com base em paths — ideal para testes espalhados pela base. B depende de inferência do modelo. C é manual/sob demanda. D não funciona bem quando arquivos relevantes estão em muitos diretórios.

Questão 7 (Cenário: Sistema de Pesquisa Multiagente)

Situação: O sistema pesquisa "impacto da IA nas indústrias criativas", mas os relatórios cobrem apenas artes visuais. O coordenador decompôs o tema em: "IA em arte digital", "IA em design gráfico", "IA em fotografia".

Qual é a causa?

- A) O agente de síntese não detecta lacunas
- B) O coordenador decompôs a tarefa de forma muito estreita **[CORRETA]**
- C) O agente de busca web não busca de forma exaustiva
- D) O agente de análise de documentos filtra fontes não-visuais

Por que B: Os logs mostram que o coordenador decompôs "indústrias criativas" apenas em subtemas visuais, perdendo música, literatura e cinema. Os subagentes executaram corretamente — o problema é o que lhes foi atribuído.

Questão 8 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Um subagente de busca web sofre timeout ao pesquisar um tema complexo. Você precisa desenhar como a informação de erro é passada de volta ao coordenador.

Qual abordagem de propagação de erro melhor habilita recuperação inteligente?

- A) Retornar contexto estruturado de erro ao coordenador: tipo de falha, query, resultados parciais e alternativas **[CORRETA]**
- B) Implementar retentativas automáticas com backoff exponencial dentro do subagente, depois retornar um status genérico "busca indisponível"

- C) Capturar o timeout dentro do subagente e retornar um conjunto de resultados vazio marcado como sucesso
- D) Propagar a exceção de timeout para um handler de topo que termina o workflow inteiro

Por que A: Contexto estruturado de erro dá ao coordenador o necessário para decidir se retenta com query modificada, tenta abordagem alternativa ou continua com resultados parciais. B esconde contexto atrás de um status genérico. C mascara falha como sucesso. D aborta o workflow desnecessariamente.

Questão 9 (Cenário: Sistema de Pesquisa Multiagente)

Situação: O agente de síntese frequentemente precisa verificar afirmações específicas ao mesclar resultados. Atualmente, quando verificação é necessária, o agente de síntese devolve o controle ao coordenador, que chama o agente de busca web e então re-executa a síntese com os novos resultados. Isso adiciona 2–3 round trips extras por tarefa e aumenta a latência em 40%. Sua avaliação mostra que 85% dessas verificações são checagens simples (datas, nomes, estatísticas), enquanto 15% exigem investigação mais profunda.

Como reduzir overhead mantendo a confiabilidade?

- A) Dar ao agente de síntese uma ferramenta limitada `verify_fact` para checagens simples e continuar roteando verificação complexa pelo coordenador **[CORRETA]**
- B) Acumular todas as verificações em lote e devolvê-las ao coordenador no final
- C) Dar ao agente de síntese acesso completo a todas as ferramentas de busca web
- D) Cachear proativamente contexto adicional ao redor de cada fonte

Por que A: Aplica o princípio do menor privilégio: o agente de síntese ganha exatamente o necessário para o caso comum de 85% (checagens simples) preservando o caminho mediado pelo coordenador para investigações complexas. B introduz dependências bloqueantes. C quebra a separação de responsabilidades. D depende de cache especulativo.

Questão 10 (Cenário: Claude Code para CI)

Situação: Um pipeline executa `claude "Analyze this pull request for security issues"`, mas trava esperando entrada interativa.

Qual é a abordagem correta?

- A) Use a flag `-p`: `claude -p "Analyze this pull request for security issues"` **[CORRETA]**
- B) Defina `CLAUDE_HEADLESS=true`
- C) Redirecione stdin de `/dev/null`
- D) Use `--batch`

Por que A: `-p` (ou `--print`) é a forma documentada de rodar Claude Code em modo não-interativo. Processa o prompt, imprime no stdout e encerra. As outras opções ou são funcionalidades inexistentes ou são contornos Unix.

Questão 11 (Cenário: Claude Code para CI)

Situação: O time quer reduzir o custo de API para análise automatizada. O Claude atende dois fluxos em tempo real: (1) checagem bloqueante pré-merge que precisa terminar antes de o dev poder mergear o PR, e (2) relatório de tech-debt gerado de noite para revisão pela manhã. Um gerente propõe mover ambos para a Message Batches API para economizar 50%.

Como avaliar essa proposta?

- A) Use processamento em lote apenas para os relatórios de tech-debt; mantenha chamadas em tempo real para checagens pré-merge **[CORRETA]**
- B) Mova ambos para processamento em lote e faça polling até concluir
- C) Mantenha tempo real para ambos para evitar problemas de ordenação nos resultados em lote
- D) Mova ambos para processamento em lote com fallback para tempo real se um lote demorar

Por que A: A Message Batches API economiza 50%, mas pode levar até 24 horas sem garantia de SLA de latência. Inadequada para checagens pré-merge bloqueantes em que devs estão esperando, mas ideal para cargas em lote noturnas como relatórios de tech-debt.

Questão 12 (Cenário: Code Review Multi-arquivo)

Situação: Um pull request altera 14 arquivos em um módulo de tracking de inventário. Uma review de uma única passagem produz resultados inconsistentes: comentários detalhados em alguns arquivos e superficiais em outros, bugs óbvios não detectados e feedback contraditório (um padrão é marcado como problemático em um arquivo mas aprovado em código idêntico em outro).

Como reestruturar a review?

- A) Divida em passagens focadas: analise cada arquivo individualmente para problemas locais e depois rode uma passagem de integração separada para fluxos de dados cross-file **[CORRETA]**
- B) Exija que devs dividam PRs grandes em submissões de 3–4 arquivos
- C) Mude para um modelo de tier maior com janela de contexto maior para revisar todos os 14 arquivos em uma única passagem
- D) Rode três passagens independentes de review do PR completo e reporte apenas problemas encontrados em pelo menos duas execuções

Por que A: Passagens focadas atacam diretamente a causa raiz — diluição de atenção ao processar muitos arquivos de uma vez. Análise por arquivo garante profundidade consistente, e uma passagem de integração separada captura problemas cross-file. B transfere o ônus para os devs sem melhorar o sistema. C é equívoco: contexto maior não corrige qualidade de atenção. D suprime bugs reais ao exigir consenso entre detecções inconsistentes.

Teste Prático

> 76 questões em 5 cenários. O formato e a dificuldade equiparam-se ao exame real.

> Como alternativa, você pode praticar essas questões em um arquivo HTML estilo exame:
> Teste Prático (PT)

Cenário: Sistema de Pesquisa Multiagente

Questão 1 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Um agente de análise de documentos descobre que duas fontes confiáveis contêm estatísticas diretamente contraditórias para uma métrica-chave: um relatório governamental indica crescimento de 40%, enquanto uma análise da indústria indica 12%. Ambas as fontes parecem confiáveis e a discrepância pode afetar materialmente as conclusões da pesquisa. Como o agente de análise de documentos deve lidar com essa situação da forma mais efetiva?

Qual abordagem é mais efetiva?

- A) Aplicar heurísticas de credibilidade para escolher o número mais provavelmente correto, finalizar a análise com esse valor e adicionar uma nota de rodapé mencionando a discrepância.
- B) Incluir ambos os números na saída da análise sem marcá-los como conflitantes, deixando o agente de síntese decidir qual usar com base em contexto mais amplo.
- C) Parar a análise e escalar imediatamente ao coordenador, pedindo que ele decida qual fonte é mais autoritativa antes de continuar.
- D) Concluir a análise com ambos os números, anotar explicitamente o conflito com atribuição de fonte e deixar o coordenador decidir como reconciliar os dados antes de passá-los à síntese. **[CORRETA]**

Por que D: Esta abordagem preserva a separação de responsabilidades: o agente de análise conclui seu trabalho principal sem bloquear, preserva ambos os valores conflitantes com atribuição clara e passa corretamente a reconciliação ao coordenador, que tem contexto mais amplo.

Questão 2 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Os agentes de busca web e de análise de documentos concluíram suas tarefas e retornaram resultados ao coordenador. Qual é o próximo passo para criar um relatório de pesquisa integrado?

Qual próximo passo é mais apropriado?

- A) Cada agente envia seus resultados diretamente ao agente redator do relatório, contornando o coordenador.
- B) O agente de análise de documentos solicita os resultados de busca web e os funde internamente.
- C) O coordenador passa ambos os conjuntos de resultados ao agente de síntese para uma integração unificada. **[CORRETA]**
- D) O coordenador concatena as saídas brutas de ambos os agentes e as devolve como resultado final.

Por que C: Em uma arquitetura coordenador–subagente, o coordenador encaminha ambos os conjuntos de resultados ao agente de síntese para uma integração centralizada, preservando o controle e

garantindo uma fusão de alta qualidade.

Questão 3 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Um subagente de análise de documentos falha frequentemente ao processar arquivos PDF: alguns têm seções corrompidas que disparam exceções de parsing, outros são protegidos por senha e às vezes a biblioteca de parsing trava em arquivos grandes. Atualmente, qualquer exceção termina imediatamente o subagente e retorna um erro ao coordenador, que precisa decidir se retenta, pula ou falha a tarefa toda. Isso causa envolvimento excessivo do coordenador no tratamento rotineiro de erros. Qual melhoria arquitetural é mais efetiva?

Qual melhoria é mais efetiva?

- A) Criar um agente dedicado a tratamento de erros que monitore todas as falhas via fila compartilhada e decida ações de recuperação, enviando comandos de reinício diretamente aos subagentes.
- B) Configurar o subagente para sempre retornar resultados parciais com status de sucesso, embutindo detalhes do erro em metadados; o coordenador trata todas as respostas como bem-sucedidas.
- C) Fazer com que o coordenador valide todos os documentos antes de enviá-los ao subagente, rejeitando documentos que possam causar falhas.
- D) Implementar recuperação local no subagente para falhas transitórias e escalar ao coordenador apenas erros que ele não consegue resolver, incluindo passos tentados e resultados parciais. **[CORRETA]**

Por que D: Trate erros no nível mais baixo capaz de resolvê-los. A recuperação local reduz a carga do coordenador enquanto ainda escala questões verdadeiramente irrecuperáveis com contexto completo e progresso parcial.

Questão 4 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Após executar o sistema sobre "impacto da IA nas indústrias criativas", você observa que cada subagente conclui com sucesso: o agente de busca web encontra artigos relevantes, o agente de análise de documentos os sumariza corretamente e o agente de síntese produz um texto coerente. No entanto, os relatórios finais cobrem somente artes visuais e perdem totalmente música, literatura e cinema. Nos logs do coordenador, você vê que ele decompôs o tema em três subtarefas: "IA em arte digital", "IA em design gráfico" e "IA em fotografia". Qual é a causa-raiz mais provável?

Qual é a causa-raiz mais provável?

- A) Faltam ao agente de síntese instruções para detectar lacunas de cobertura.
- B) O agente de análise de documentos filtra fontes não-visuais por critérios de relevância excessivamente estritos.
- C) A decomposição de tarefas do coordenador é estreita demais, atribuindo aos subagentes trabalho que não cobre todas as áreas relevantes. **[CORRETA]**
- D) As queries do agente de busca web são insuficientes e deveriam ser ampliadas para cobrir mais setores.

Por que C: O coordenador decompôs um tema amplo apenas em subtarefas de artes visuais, perdendo totalmente música, literatura e cinema. Como os subagentes executaram suas atribuições corretamente, a decomposição estreita é a causa-raiz óbvia.

Questão 5 (Cenário: Sistema de Pesquisa Multiagente)

Situação: O subagente de busca web retorna resultados para apenas 3 das 5 categorias de fontes solicitadas (sites de concorrentes e relatórios da indústria têm sucesso, mas arquivos de notícias e feeds sociais sofrem timeout). O subagente de análise de documentos processa com sucesso todos os documentos fornecidos. O subagente de síntese precisa produzir um sumário a partir de entradas a montante de qualidade mista. Qual estratégia de propagação de erro é mais efetiva?

Qual estratégia de propagação de erro é mais efetiva?

- A) Continuar a síntese usando apenas as fontes bem-sucedidas e produzir uma saída sem mencionar quais dados estavam indisponíveis.
- B) O subagente de síntese retorna um erro ao coordenador, disparando retry total ou falha da tarefa por dados incompletos.
- C) O subagente de síntese pede ao coordenador para retentar fontes com timeout com prazo maior antes de iniciar a síntese.
- D) Estruturar a saída da síntese com anotações de cobertura indicando quais conclusões estão bem suportadas e onde existem lacunas devido a fontes indisponíveis. **[CORRETA]**

Por que D: Anotações de cobertura implementam degradação graciosa com transparência, preservando valor do trabalho concluído enquanto propagam incerteza para permitir decisões informadas sobre confiança.

Questão 6 (Cenário: Sistema de Pesquisa Multiagente)

Situação: O subagente de análise de documentos encontra um arquivo PDF corrompido que ele não consegue parsear. Ao desenhar o tratamento de erros do sistema, qual é a forma mais efetiva de lidar com essa falha?

Qual abordagem é mais efetiva?

- A) Retornar um erro com contexto ao agente coordenador, permitindo que ele decida como prosseguir. **[CORRETA]**
- B) Pular silenciosamente o documento corrompido e continuar processando os demais arquivos para evitar interromper o workflow.
- C) Retentar automaticamente o parsing três vezes com backoff exponencial antes de reportar uma falha.
- D) Lançar uma exceção que termina o workflow inteiro de pesquisa.

Por que A: Retornar um erro com contexto ao coordenador é a abordagem mais efetiva porque o deixa tomar uma decisão informada — pular o arquivo, tentar um método alternativo de parsing ou notificar o usuário — mantendo visibilidade da falha.

Questão 7 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Logs de produção mostram um padrão persistente: pedidos como "analise o relatório trimestral enviado" são roteados ao agente de busca web em 45% das vezes em vez de ao agente de análise de documentos. Revisando as definições das ferramentas, você descobre que o agente de busca web tem uma ferramenta `analyze_content` descrita como "analyzes content and extracts key information", enquanto o agente de análise de documentos tem `analyze_document` descrita como "analyzes documents and extracts key information". Como corrigir o problema de roteamento errado?

Como corrigir o problema?

- A) Adicionar um classificador de pré-roteamento que detecta se o usuário se refere a arquivos enviados ou a conteúdo web antes de o coordenador decidir a delegação.
- B) Renomear a ferramenta de busca web para `extract_web_results` e atualizar sua descrição para "processes and returns information retrieved from web search and URLs." **[CORRETA]**
- C) Adicionar exemplos few-shot ao prompt do coordenador mostrando o roteamento correto: "User uploads a quarterly report → document analysis agent" e "User asks about a web page → web-search agent."
- D) Expandir a descrição da ferramenta de análise de documentos com exemplos de uso como "Use for uploaded PDFs, Word docs, and spreadsheets," deixando a ferramenta de busca web inalterada.

Por que B: Renomear a ferramenta de busca web para `extract_web_results` e atualizar a descrição para referenciar explicitamente busca web e URLs remove diretamente a causa-raiz ao eliminar a sobreposição semântica entre nomes e descrições. Isso torna o propósito de cada ferramenta inequívoco, permitindo ao coordenador distinguir análise de documentos de busca web de forma confiável.

Questão 8 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Um colega propõe que o agente de análise de documentos envie seus resultados diretamente ao agente de síntese, contornando o coordenador. Qual é a vantagem principal de manter o coordenador como hub central para toda comunicação entre subagentes?

Qual é a vantagem principal de manter o coordenador como hub central?

- A) O coordenador pode observar todas as interações, tratar erros de forma uniforme e decidir que informação cada subagente deve receber. **[CORRETA]**
- B) O coordenador agrupa múltiplas requisições aos subagentes, reduzindo o total de chamadas de API e a latência geral.
- C) Roteamento pelo coordenador habilita lógica de retry automática que chamadas inter-agente diretas não suportam.
- D) Subagentes usam memória isolada e a comunicação direta exigiria serialização complexa que só o coordenador consegue realizar.

Por que A: O padrão coordenador fornece visibilidade central sobre todas as interações, tratamento de erros uniforme em todo o sistema e controle granular sobre que informação cada subagente recebe — essas são as principais vantagens de uma topologia de comunicação em estrela.

Questão 9 (Cenário: Sistema de Pesquisa Multiagente)

Situação: O subagente de busca web sofre timeout ao pesquisar um tema complexo. Você precisa desenhar como a informação sobre essa falha é retornada ao coordenador. Qual abordagem de propagação de erro melhor habilita recuperação inteligente?

Qual abordagem de propagação de erro melhor habilita recuperação inteligente?

- A) Retornar contexto estruturado de erro ao coordenador, incluindo o tipo de falha, a query executada, quaisquer resultados parciais e abordagens alternativas potenciais. **[CORRETA]**
- B) Capturar o timeout dentro do subagente e retornar um conjunto de resultados vazio marcado como bem-sucedido.
- C) Implementar retentativas automáticas com backoff exponencial dentro do subagente, retornando apenas um status genérico "busca indisponível" depois de esgotar as tentativas.
- D) Propagar a exceção de timeout diretamente ao handler de topo, terminando o workflow de pesquisa inteiro.

Por que A: Retornar contexto estruturado de erro — incluindo tipo de falha, query executada, resultados parciais e abordagens alternativas — dá ao coordenador tudo que precisa para tomar decisões inteligentes de recuperação (ex.: retentar com query modificada ou continuar com resultados parciais). Preserva o máximo de contexto para decisões informadas no nível de coordenação.

Questão 10 (Cenário: Sistema de Pesquisa Multiagente)

Situação: No seu desenho de sistema, você deu ao agente de análise de documentos acesso a uma ferramenta de uso geral `fetch_url` para que ele pudesse baixar documentos por URL. Logs de produção mostram que esse agente agora frequentemente baixa páginas de resultados de busca para fazer busca web ad hoc — comportamento que deveria passar pelo agente de busca web — causando resultados inconsistentes. Qual correção é mais efetiva?

Qual correção é mais efetiva?

- A) Substituir `fetch_url` por uma ferramenta `load_document` que valida que URLs apontem para formatos de documento. **[CORRETA]**
- B) Remover `fetch_url` do agente de análise de documentos e rotear toda busca por URL pelo coordenador para o agente de busca web.
- C) Implementar filtragem que bloqueia chamadas de `fetch_url` para domínios conhecidos de buscadores enquanto permite outras URLs.
- D) Adicionar instruções no prompt do agente de análise de documentos dizendo que `fetch_url` só deve ser usada para baixar URLs de documentos, não para buscar.

Por que A: Substituir uma ferramenta de uso geral por uma específica de documento que valida URLs contra formatos de documento ataca a causa-raiz ao restringir capacidade no nível da interface. Isso segue o princípio do menor privilégio, tornando o comportamento indesejado de busca impossível em vez de meramente desencorajado.

Questão 11 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Ao pesquisar um tema amplo, você observa que o agente de busca web e o agente de análise de documentos investigam os mesmos subtemas, levando a duplicação substancial em suas saídas. O uso de tokens quase dobra sem aumento proporcional na amplitude ou profundidade da pesquisa. Qual é a forma mais efetiva de tratar isso?

Qual é a forma mais efetiva de tratar isso?

- A) Permitir que ambos os agentes terminem em paralelo e fazer com que o coordenador desduplique resultados sobrepostos antes de passá-los ao agente de síntese.
- B) O coordenador particiona explicitamente o espaço de pesquisa antes de delegar, atribuindo a cada agente subtemas distintos ou tipos de fonte distintos. **[CORRETA]**
- C) Implementar um mecanismo de estado compartilhado em que agentes registrem sua área de foco atual para que outros agentes evitem duplicação dinamicamente durante a execução.
- D) Mudar para execução sequencial em que a análise de documentos rode somente após a busca web concluir, usando os resultados da busca web como contexto para evitar duplicação.

Por que B: Fazer o coordenador particionar explicitamente o espaço de pesquisa antes de delegar é mais efetivo porque ataca a causa-raiz — fronteiras de tarefa pouco claras — antes de qualquer trabalho começar. Preserva paralelismo prevenindo esforço duplicado e tokens desperdiçados.

Questão 12 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Durante a pesquisa, o subagente de busca web consulta três categorias de fontes com resultados diferentes: bases acadêmicas retornam 15 papers relevantes, relatórios da indústria retornam "0 resultados" e bases de patentes retornam "Connection timeout". Ao desenhar a propagação de erro para o coordenador, qual abordagem habilita as melhores decisões de recuperação?

Qual abordagem habilita as melhores decisões de recuperação?

- A) Agregar resultados em uma única métrica de percentual de sucesso (ex.: "67% de cobertura de fontes") com logs detalhados disponíveis sob demanda.
- B) Reportar tanto "timeout" quanto "0 resultados" como falhas que requerem intervenção do coordenador.
- C) Retentar falhas transitórias internamente e reportar apenas erros persistentes.
- D) Distinguir falhas de acesso (timeout) que exigem decisão de retry de resultados vazios válidos ("0 resultados") que representam queries bem-sucedidas. **[CORRETA]**

Por que D: Um timeout (falha de acesso) e "0 resultados" (resultado vazio válido) são desfechos semanticamente diferentes que exigem respostas diferentes. Distingui-los permite ao coordenador retentar a base de patentes enquanto aceita os "0 resultados" dos relatórios da indústria como achado válido e informativo.

Questão 13 (Cenário: Sistema de Pesquisa Multiagente)

Situação: O monitoramento de produção mostra qualidade inconsistente de síntese. Quando os resultados agregados têm ~75K tokens, o agente de síntese cita de forma confiável informações dos primeiros 15K tokens (manchetes/snippets de busca web) e dos últimos 10K (conclusões da análise de documentos), mas frequentemente perde achados críticos nos 50K do meio — mesmo quando respondem diretamente à pergunta de pesquisa. Como reestruturar a entrada agregada?

Como reestruturar a entrada agregada?

- A) Sumarizar todas as saídas dos subagentes para abaixo de 20K tokens antes da agregação para manter o conteúdo dentro do alcance confiável de processamento do modelo.
- B) Streamar incrementalmente os resultados dos subagentes ao agente de síntese, processando primeiro os de busca web até concluir e depois adicionando os de análise de documentos.
- C) Posicionar um sumário de achados-chave no início da entrada agregada e organizar resultados detalhados com cabeçalhos de seção explícitos para navegação mais fácil. **[CORRETA]**
- D) Implementar rotação que alterne quais resultados aparecem primeiro entre tarefas de pesquisa para garantir que ambas as fontes ganhem posição superior ao longo do tempo.

Por que C: Posicionar um sumário de achados-chave no início aproveita o efeito de primazia, colocando informação crítica na posição de processamento mais confiável. Adicionar cabeçalhos de seção explícitos ajuda o modelo a navegar e atender ao conteúdo do meio, mitigando diretamente o fenômeno "lost in the middle".

Questão 14 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Em testes, a saída combinada do agente de busca web (85K tokens incluindo conteúdo de páginas) e do agente de análise de documentos (70K tokens incluindo cadeias de raciocínio) totaliza 155K tokens, mas o agente de síntese performa melhor com entradas abaixo de 50K tokens. Qual solução é mais efetiva?

Qual solução é mais efetiva?

- A) Modificar agentes upstream para retornar dados estruturados (fatos-chave, citações, scores de relevância) em vez de conteúdo verboso e raciocínio. **[CORRETA]**
- B) Adicionar um agente intermediário de sumarização que condense achados antes de passá-los à síntese.
- C) Fazer o agente de síntese processar achados em lotes sequenciais, mantendo estado entre chamadas.
- D) Armazenar achados em uma base vetorial e dar ao agente de síntese ferramentas de busca para consultar durante o trabalho.

Por que A: Modificar agentes upstream para retornar dados estruturados ataca a causa-raiz reduzindo o volume de tokens na origem enquanto preserva informação essencial. Evita passar conteúdo volumoso de páginas e traços de raciocínio que inflam tokens sem melhorar a etapa de síntese.

Questão 15 (Cenário: Sistema de Pesquisa Multiagente)

Situação: Em testes, você observa que o agente de síntese frequentemente precisa verificar afirmações específicas ao mesclar resultados. Atualmente, quando verificação é necessária, o agente de síntese devolve o controle ao coordenador, que chama o agente de busca web e então re-invoca a síntese com os resultados. Isso adiciona 2–3 loops extras por tarefa e aumenta a latência em 40%. Sua avaliação mostra que 85% dessas verificações são checagens simples (datas, nomes, estatísticas) e 15% exigem pesquisa mais profunda. Qual abordagem é mais efetiva para reduzir overhead preservando confiabilidade?

Qual abordagem é mais efetiva?

- A) Dar ao agente de síntese acesso a todas as ferramentas de busca web para que ele possa lidar com qualquer necessidade de verificação diretamente, sem loops do coordenador.
- B) Fazer o agente de síntese acumular todas as necessidades de verificação e devolvê-las como um lote ao coordenador no fim, que então as envia todas de uma vez ao agente de busca web.
- C) Fazer o agente de busca web cachear proativamente contexto extra ao redor de cada fonte durante a pesquisa inicial, antecipando necessidades da síntese.
- D) Dar ao agente de síntese uma ferramenta `verify_fact` de escopo limitado para checagens simples, enquanto roteia verificações complexas pelo coordenador para o agente de busca web. **[CORRETA]**

Por que D: Uma ferramenta de verificação de fato de escopo limitado deixa o agente de síntese lidar diretamente com 85% das checagens simples, eliminando a maioria dos loops, enquanto preserva o caminho de delegação pelo coordenador para os 15% de verificações complexas. Aplica menor privilégio enquanto reduz significativamente a latência.

Cenário: Claude Code para Integração Contínua

Questão 16 (Cenário: Claude Code para Integração Contínua)

Situação: Seu pipeline de CI executa o Claude Code CLI (em modo `--print`) usando CLAUDE.md para fornecer contexto do projeto para code review, e os devs em geral acham as reviews substantivas. Porém, eles relatam que integrar achados ao workflow é difícil — Claude emite parágrafos narrativos que precisam ser copiados manualmente para comentários no PR. O time quer postar automaticamente cada achado como comentário inline separado no local relevante do código, o que exige dados estruturados com path do arquivo, número da linha, nível de severidade e correção sugerida. Qual abordagem é mais efetiva?

Qual abordagem é mais efetiva?

- A) Adicionar uma seção "Output Format for Review" ao CLAUDE.md com exemplos de achados estruturados para que Claude aprenda o formato esperado a partir do contexto do projeto.
- B) Usar as flags do CLI `--output-format json` e `--json-schema` para forçar achados estruturados, depois parsear a saída para postar comentários inline via API do GitHub. **[CORRETA]**

- C) Incluir instruções explícitas de formatação no prompt de review exigindo que cada achado siga um template parseável como `[FILE:path] [LINE:n] [SEVERITY:level] ...`.
- D) Manter o formato narrativo de review mas adicionar um passo de sumarização que use Claude para gerar um sumário JSON estruturado dos achados.

Por que B: Usar `--output-format json` com `--json-schema` força saída estruturada no nível do CLI, garantindo JSON bem formado com os campos exigidos (path, linha, severidade, correção sugerida) que pode ser parseado de forma confiável e postado como comentários inline via API do GitHub. Aproveita capacidades nativas do CLI projetadas justamente para saída estruturada.

Questão 17 (Cenário: Claude Code para Integração Contínua)

Situação: Seu time usa Claude Code para gerar sugestões de código, mas você nota um padrão: questões não-óbvias — otimizações de performance que quebram casos de borda, limpezas que mudam comportamento inesperadamente — só são pegas quando outro membro do time revisa o PR. O raciocínio do Claude durante a geração mostra que ele considerou esses casos mas concluiu que sua abordagem estava correta. Qual abordagem aborda diretamente a causa-raiz dessa limitação de auto-checagem?

Qual abordagem aborda diretamente a causa-raiz?

- A) Rodar uma segunda instância independente do Claude Code para revisar as mudanças sem acesso ao raciocínio do gerador. **[CORRETA]**
- B) Habilitar modo de extended thinking na geração para permitir deliberação mais minuciosa antes de produzir sugestões.
- C) Adicionar instruções explícitas de auto-review ao prompt de geração pedindo que Claude critique suas próprias sugestões antes de finalizar.
- D) Incluir arquivos de teste e documentação completos no contexto do prompt para que Claude entenda melhor o comportamento esperado durante a geração.

Por que A: Uma segunda instância independente do Claude Code sem acesso ao raciocínio do gerador ataca diretamente a causa-raiz evitando viés de confirmação. Essa perspectiva de "olhos novos" espelha a peer review humana, em que outro revisor pega problemas que o autor racionalizou.

Questão 18 (Cenário: Claude Code para Integração Contínua)

Situação: Seu componente de code review é iterativo: Claude analisa o arquivo modificado, depois pode pedir arquivos relacionados (imports, classes-base, testes) via chamadas de ferramenta para entender contexto antes de prover feedback final. Sua aplicação define uma ferramenta que permite a Claude pedir conteúdo de arquivos; Claude chama a ferramenta, recebe resultados e continua a análise. Você está avaliando processamento em lote para reduzir custo de API. Qual é a principal limitação técnica ao considerar processamento em lote para esse fluxo?

Qual é a principal limitação técnica?

- A) O processamento em lote não inclui IDs de correlação para mapear saídas de volta às requisições de entrada.
- B) O modelo assíncrono não consegue executar ferramentas no meio da requisição e devolver resultados para Claude continuar a análise. **[CORRETA]**
- C) A Batch API não suporta definições de ferramentas nos parâmetros da requisição.
- D) A latência de até 24 horas do processamento em lote é lenta demais para feedback de pull request, embora o fluxo funcionasse no resto.

Por que B: A natureza assíncrona da Batch API (uma requisição = uma resposta) significa que ela não consegue executar ferramentas no meio da chamada e devolver resultados para o modelo continuar a análise iterativa. Isso torna fundamentalmente incompatível workflows que precisam de tool calling multi-turn. (A está incorreta: `custom_id` existe. C está incorreta: a Batch API aceita ferramentas nos params. D é uma preocupação real mas não é a limitação fundamental — algumas review podem tolerar 24 horas; a limitação fundamental é não conseguir executar tool calls iterativos em meio à requisição.)

Questão 19 (Cenário: Claude Code para Integração Contínua)

Situação: Seu sistema CI/CD roda três análises baseadas em Claude: (1) checagens rápidas de estilo em todo PR que bloqueiam o merge até concluir, (2) auditorias semanais e abrangentes de segurança em toda a base, e (3) geração noturna de casos de teste para módulos recém-modificados. A Message Batches API oferece 50% de economia, mas o processamento pode levar até 24 horas. Você quer otimizar custo de API mantendo experiência aceitável para devs. Qual combinação casa corretamente cada tarefa com uma abordagem de API?

Qual combinação está correta?

- A) Use a Message Batches API para todas as três tarefas para maximizar a economia de 50%, configurando o pipeline para fazer polling até a conclusão dos lotes.
- B) Use chamadas síncronas para checagens de estilo em PR; use a Message Batches API para auditorias semanais de segurança e geração noturna de testes. **[CORRETA]**
- C) Use chamadas síncronas para todas para tempos de resposta consistentes, contando com prompt caching para reduzir custos.
- D) Use chamadas síncronas para checagens de estilo em PR e geração noturna de testes; use a Message Batches API apenas para auditorias semanais.

Por que B: Checagens de estilo em PR bloqueiam devs e exigem respostas imediatas via síncronas, enquanto auditorias semanais e geração noturna de testes são tarefas agendadas com prazos flexíveis que toleram a janela de até 24 horas — capturando 50% de economia em ambas.

Questão 20 (Cenário: Claude Code para Integração Contínua)

Situação: Suas reviews automatizadas encontram problemas reais, mas devs reportam que o feedback não é acionável. Achados incluem frases como "lógica complexa de roteamento de tickets" ou "potencial null pointer" sem especificar o que exatamente mudar. Quando você adiciona instruções detalhadas como "sempre inclua sugestões concretas de correção", o modelo ainda produz saída inconsistente — às vezes

detalhada, às vezes vaga. Qual técnica de prompting produz feedback consistentemente acionável de forma mais confiável?

Qual técnica de prompting é mais confiável?

- A) Refinar mais as instruções com requisitos mais explícitos para cada parte do formato de feedback (location, issue, severity, proposed fix).
- B) Expandir a janela de contexto para incluir mais código adjacente para que o modelo tenha informação suficiente para propor correções concretas.
- C) Implementar uma abordagem em duas passagens em que um prompt identifica problemas e outro gera correções, permitindo especialização.
- D) Adicionar 3–4 exemplos few-shot mostrando o formato exato exigido: problema identificado, localização no código, sugestão concreta de correção. **[CORRETA]**

Por que D: Exemplos few-shot são a técnica mais efetiva para alcançar formato consistente quando instruções sozinhas produzem resultados variáveis. Fornecer 3–4 exemplos que mostram a estrutura desejada (issue, location, concrete fix) dá ao modelo um padrão concreto a seguir, mais confiável que instruções abstratas.

Questão 21 (Cenário: Claude Code para Integração Contínua)

Situação: Seu pipeline CI inclui dois modos de code review baseados em Claude: um hook pré-merge-commit que bloqueia o merge do PR até concluir, e uma "análise profunda" que roda à noite, faz polling até concluir o lote e posta sugestões detalhadas no PR. Você quer reduzir custo de API usando a Message Batches API, que oferece 50% de economia mas exige polling e pode levar até 24 horas. Qual modo deveria usar processamento em lote?

Qual modo deveria usar processamento em lote?

- A) Apenas o hook pré-merge-commit.
- B) Apenas a análise profunda. **[CORRETA]**
- C) Ambos os modos.
- D) Nenhum dos modos.

Por que B: Análise profunda é candidata ideal para processamento em lote porque já roda à noite, tolera atraso e usa um modelo de polling antes de publicar resultados — o que casa com a arquitetura assíncrona baseada em polling da Message Batches API enquanto captura 50% de economia.

Questão 22 (Cenário: Claude Code para Integração Contínua)

Situação: Sua review automatizada analisa comentários e docstrings. O prompt atual instrui Claude a "verificar que comentários estão precisos e atualizados". Achados frequentemente sinalizam padrões aceitáveis (marcadores TODO, descrições simples) enquanto perdem comentários que descrevem comportamento que o código não implementa mais. Qual mudança ataca a causa-raiz dessa análise inconsistente?

Qual mudança ataca a causa-raiz?

- A) Incluir dados de `git blame` para que Claude possa identificar comentários anteriores a mudanças recentes de código.
- B) Adicionar exemplos few-shot de comentários enganosos para ajudar o modelo a reconhecer padrões similares na base.
- C) Filtrar TODO, FIXME e padrões de comentários descritivos antes da análise para reduzir ruído.
- D) Especificar critérios explícitos: marcar comentários apenas quando o comportamento que afirmam contradiz o comportamento real do código. **[CORRETA]**

Por que D: Critérios explícitos — marcar comentários apenas quando o comportamento afirmado contradiz o comportamento real do código — atacam diretamente a causa-raiz substituindo uma instrução vaga por uma definição precisa do que constitui um problema. Isso reduz falsos positivos em padrões aceitáveis e perdas de comentários genuinamente enganosos.

Questão 23 (Cenário: Claude Code para Integração Contínua)

Situação: Seu sistema de code review automatizado mostra severidades inconsistentes — questões similares como riscos de null pointer são classificadas "critical" em alguns PRs mas só "medium" em outros. Pesquisas com devs mostram desconfiança crescente — muitos começam a descartar achados sem ler porque "metade está errada". Categorias com altas taxas de falsos positivos minam a confiança em categorias acuradas. Qual abordagem melhor restaura a confiança dos devs enquanto melhora o sistema?

Qual abordagem melhor restaura a confiança dos devs?

- A) Desabilitar temporariamente categorias de alto FP (estilo, naming, documentação) e manter apenas categorias de alta precisão enquanto melhora os prompts. **[CORRETA]**
- B) Manter todas as categorias mas exibir scores de confiança junto a cada achado para que devs decidam o que investigar.
- C) Manter todas as categorias e adicionar exemplos few-shot para melhorar a acurácia em cada categoria nas próximas semanas.
- D) Aplicar uma redução uniforme de rigor em todas as categorias para baixar a taxa geral de FP.

Por que A: Desabilitar temporariamente categorias com alto FP interrompe imediatamente a erosão de confiança removendo achados ruidosos que levam devs a descartar tudo, preservando valor de categorias de alta precisão como segurança e correção. Também cria espaço para melhorar prompts das categorias problemáticas antes de reabilitá-las.

Questão 24 (Cenário: Claude Code para Integração Contínua)

Situação: Sua review automatizada gera sugestões de casos de teste para cada PR. Revisando um PR que adiciona tracking de conclusão de curso, Claude sugere 10 casos de teste, mas o feedback dos devs mostra que 6 duplicam cenários já cobertos pela suíte de testes existente. Qual mudança reduz duplicação de forma mais efetiva?

Qual mudança é mais efetiva?

- A) Incluir o arquivo de teste existente no contexto para que Claude possa determinar quais cenários já estão cobertos. **[CORRETA]**
- B) Reduzir o número solicitado de sugestões de 10 para 5, assumindo que Claude prioriza os casos mais valiosos primeiro.
- C) Adicionar instruções direcionando Claude a focar exclusivamente em casos de borda e condições de erro em vez de caminhos felizes.
- D) Implementar pós-processamento que filtra sugestões cujas descrições casam com nomes de testes existentes via sobreposição de palavras-chave.

Por que A: Incluir o arquivo de teste existente corrige a causa-raiz da duplicação: Claude só pode evitar sugerir cenários já cobertos se souber quais testes já existem. Isso dá a Claude a informação necessária para propor testes genuinamente novos e valiosos.

Questão 25 (Cenário: Claude Code para Integração Contínua)

Situação: Após uma review automatizada inicial identificar 12 achados, um dev faz push de novos commits para tratar os problemas. Re-rodar a review produz 8 achados, mas devs reportam que 5 duplicam comentários anteriores em código que já foi corrigido nos novos commits. Qual é a forma mais efetiva de eliminar esse feedback redundante mantendo a profundidade?

Qual é a forma mais efetiva de eliminar feedback redundante?

- A) Rodar review apenas quando o PR é criado e no estado final pré-merge, pulando commits intermediários.
- B) Adicionar um filtro de pós-processamento que remove achados que casam com anteriores por path e descrição antes de postar comentários.
- C) Restringir o escopo da review aos arquivos alterados no push mais recente, excluindo arquivos de commits anteriores.
- D) Incluir achados de reviews anteriores no contexto e instruir Claude a reportar apenas problemas novos ou ainda não resolvidos. **[CORRETA]**

Por que D: Incluir achados de reviews anteriores no contexto deixa Claude distinguir problemas novos dos já tratados em commits recentes. Isso preserva profundidade da review enquanto usa o raciocínio do Claude para evitar feedback redundante em código já corrigido.

Questão 26 (Cenário: Claude Code para Integração Contínua)

Situação: Seu script de pipeline executa `claude "Analyze this pull request for security issues"`, mas o job trava indefinidamente. Logs mostram que o Claude Code está esperando entrada interativa. Qual é a abordagem correta para rodar Claude Code em pipeline automatizado?

Qual é a abordagem correta?

- A) Adicionar a flag `--batch`: `claude --batch "Analyze this pull request for security issues"`.
- B) Adicionar a flag `-p`: `claude -p "Analyze this pull request for security issues"`. **[CORRETA]**
- C) Redirecionar stdin de `/dev/null`: `claude "Analyze this pull request for security issues" < /dev/null`.
- D) Definir a variável de ambiente `CLAUDE_HEADLESS=true` antes de executar o comando.

Por que B: A flag `-p` (ou `--print`) é a forma documentada de rodar Claude Code em modo não-interativo. Processa o prompt, imprime o resultado no stdout e encerra sem esperar entrada do usuário — ideal para pipelines CI/CD.

Questão 27 (Cenário: Claude Code para Integração Contínua)

Situação: Um pull request altera 14 arquivos em um módulo de tracking de inventário. Uma review de única passagem que analisa todos os arquivos juntos produz resultados inconsistentes: feedback detalhado em alguns arquivos mas comentários superficiais em outros, bugs óbvios não detectados e feedback contraditório (um padrão é marcado em um arquivo mas código idêntico aprovado em outro arquivo no mesmo PR). Como reestruturar a review?

Como reestruturar a review?

- A) Rodar três passagens independentes de review do PR completo e marcar apenas problemas que aparecem em pelo menos duas das três execuções.
- B) Dividir em passagens focadas: revisar cada arquivo individualmente para problemas locais, depois rodar uma passagem separada orientada a integração para examinar fluxos de dados cross-file. **[CORRETA]**
- C) Exigir que devs dividam PRs grandes em submissões menores de 3–4 arquivos antes de rodar review automatizada.
- D) Mudar para um modelo maior com janela de contexto maior para que ele possa atender suficientemente todos os 14 arquivos em uma passagem.

Por que B: Passagens focadas por arquivo atacam a causa-raiz — diluição de atenção — garantindo profundidade consistente e detecção confiável de problemas locais. Uma passagem separada de integração cobre questões cross-file como dependências e interações de fluxo de dados.

Questão 28 (Cenário: Claude Code para Integração Contínua)

Situação: Sua review automatizada média 15 achados por PR e devs reportam taxa de FP de 40%. O gargalo é tempo de investigação: devs precisam clicar em cada achado para ler a justificativa do Claude antes de decidir corrigir ou descartar. Seu CLAUDE.md já contém regras abrangentes para padrões aceitáveis e stakeholders rejeitaram qualquer abordagem que filtre achados antes de devs verem. Qual mudança melhor ataca o tempo de investigação?

Qual mudança ataca melhor o tempo de investigação?

- A) Exigir que Claude inclua sua justificativa e estimativa de confiança diretamente em cada achado. **[CORRETA]**
- B) Adicionar um pós-processador que analisa padrões de achados e suprime automaticamente os que casam com assinaturas históricas de FP.
- C) Categorizar achados como "blocking issues" vs "suggestions", com requisitos de revisão diferentes por nível.
- D) Configurar Claude para mostrar apenas achados de alta confiança, filtrando flags incertas antes de devs verem.

Por que A: Incluir justificativa e confiança diretamente em cada achado reduz tempo de investigação ao deixar devs triar rapidamente sem abrir cada achado. Satisfaz a restrição de "não filtrar" porque todos os achados permanecem visíveis enquanto acelera a tomada de decisão.

Questão 29 (Cenário: Claude Code para Integração Contínua)

Situação: A análise da sua review automatizada de código mostra grandes diferenças nas taxas de falsos positivos por categoria de achado: achados de segurança/correção têm 8% de FP, performance 18%, estilo/naming 52% e documentação 48%. Pesquisas com devs mostram desconfiança crescente — muitos começam a descartar achados sem ler porque "metade está errada". Categorias de alto FP minam a confiança em categorias acuradas. Qual abordagem melhor restaura a confiança dos devs enquanto melhora o sistema?

Qual abordagem melhor restaura a confiança dos devs?

- A) Desabilitar temporariamente categorias de alto FP (estilo, naming, documentação) e manter apenas categorias de alta precisão enquanto melhora os prompts. **[CORRETA]**
- B) Manter todas as categorias mas exibir scores de confiança junto a cada achado para que devs decidam o que investigar.
- C) Manter todas as categorias e adicionar exemplos few-shot para melhorar a acurácia em cada categoria nas próximas semanas.
- D) Aplicar uma redução uniforme de rigor em todas as categorias para baixar a taxa geral de FP.

Por que A: Desabilitar temporariamente categorias com alto FP interrompe imediatamente a erosão de confiança removendo achados ruidosos que levam devs a descartar tudo, preservando valor de categorias de alta precisão como segurança e correção. Também cria espaço para melhorar prompts das categorias problemáticas antes de reabilitá-las.

Questão 30 (Cenário: Claude Code para Integração Contínua)

Situação: Seu time quer reduzir custos de API para análise automatizada. Atualmente, chamadas síncronas ao Claude atendem dois fluxos: (1) checagem bloqueante pré-merge que precisa concluir antes de devs poderem mergear, e (2) relatório de tech-debt gerado de noite para revisão na manhã seguinte. Seu gerente propõe mover ambos para a Message Batches API para economizar 50%. Como avaliar essa proposta?

Como avaliar essa proposta?

- A) Mover ambos para processamento em lote com fallback para chamadas síncronas se os lotes demorarem demais.
- B) Mover ambos os fluxos para processamento em lote com polling de status para verificar conclusão.
- C) Use processamento em lote apenas para relatórios de tech-debt; mantenha chamadas síncronas para checagens pré-merge. **[CORRETA]**
- D) Manter chamadas síncronas para ambos os fluxos para evitar problemas com ordenação de resultados em lote.

Por que C: O processamento da Message Batches API pode levar até 24 horas sem SLA de latência, o que é aceitável para relatórios de tech-debt noturnos mas inaceitável para checagens pré-merge bloqueantes em que devs esperam. Isso casa cada fluxo com a API certa baseada em requisitos de latência.

Cenário: Geração de Código com Claude Code

Questão 31 (Cenário: Geração de Código com Claude Code)

Situação: Você pediu ao Claude Code para implementar uma função que transforma respostas de API em um formato interno normalizado. Após duas iterações, a estrutura de saída ainda não casa com as expectativas — alguns campos são aninhados de modo diferente e timestamps formatados incorretamente. Você descreveu requisitos em prosa, mas Claude os interpreta de forma diferente a cada vez.

Qual abordagem é mais efetiva para a próxima iteração?

- A) Escrever um JSON schema descrevendo a estrutura de saída esperada e validar a saída do Claude contra ele após cada iteração.
- B) Fornecer 2–3 exemplos concretos de entrada-saída mostrando a transformação esperada para respostas representativas da API. **[CORRETA]**
- C) Reescrever requisitos com mais precisão técnica, especificando mapeamentos exatos de campos, regras de aninhamento e strings de formato de timestamp.
- D) Pedir ao Claude para explicar seu entendimento atual dos requisitos para identificar onde as interpretações divergem.

Por que B: Exemplos concretos de entrada-saída removem a ambiguidade inerente a descrições em prosa mostrando a Claude o resultado exato esperado da transformação. Isso ataca diretamente a causa-raiz — interpretação errada de requisitos textuais — fornecendo padrões inequívocos para aninhamento e formatação de timestamp.

Questão 32 (Cenário: Geração de Código com Claude Code)

Situação: Você precisa adicionar Slack como novo canal de notificação. A base existente tem padrões claros e estabelecidos para email, SMS e push. Porém, a API do Slack oferece abordagens fundamentalmente diferentes — webhooks de entrada (simples, unidirecional), bot tokens (suporte a confirmação de entrega e controle programático), ou Slack Apps (eventos bidirecionais, requer aprovação do workspace). Sua tarefa diz "adicionar suporte a Slack" sem especificar método de integração ou exigir features avançadas como tracking de entrega.

Como abordar essa tarefa?

- A) Começar em modo de execução direta usando webhooks de entrada para casar com o padrão existente unidirecional.
- B) Mudar para modo de planejamento para explorar opções de integração e implicações arquiteturais, depois apresentar uma recomendação antes da implementação. **[CORRETA]**
- C) Começar em execução direta fazendo scaffolding de uma classe de canal Slack usando padrões existentes, adiando a decisão do método de integração.
- D) Começar em execução direta usando uma abordagem de bot-token para garantir que confirmação de entrega seja possível.

Por que B: Integração com Slack tem múltiplas abordagens válidas com implicações arquiteturais significativamente diferentes, e os requisitos são ambíguos. O modo de planejamento deixa avaliar trade-offs entre webhooks, bot tokens e Slack Apps e alinhar uma abordagem antes de implementar.

Questão 33 (Cenário: Geração de Código com Claude Code)

Situação: Seu CLAUDE.md cresceu para 400+ linhas contendo padrões de código, convenções de testes, um checklist detalhado de PR review, instruções de deploy e procedimentos de migração de banco. Você quer que Claude sempre siga padrões de código e convenções de testes, mas aplique guias de PR review, deploy e migration apenas quando estiver fazendo essas tarefas.

Qual abordagem de reestruturação é mais efetiva?

- A) Mover toda a orientação para arquivos Skills separados organizados por tipo de fluxo, deixando apenas uma breve descrição do projeto no CLAUDE.md.
- B) Manter tudo no CLAUDE.md mas usar sintaxe `@import` para organizar em arquivos mantidos separadamente por categoria.
- C) Dividir o CLAUDE.md em arquivos sob `.claude/rules/` com padrões glob path-bound para que cada regra carregue só para os tipos de arquivo relevantes.
- D) Manter padrões universais no CLAUDE.md e criar Skills para guias específicos de fluxo (PR review, deploy, migrations) com palavras-chave de gatilho. **[CORRETA]**

Por que D: Conteúdo do CLAUDE.md carrega em toda sessão, garantindo que padrões de código e convenções de teste sempre se apliquem, enquanto Skills são invocadas sob demanda quando Claude detecta palavras-chave de gatilho — ideal para guias específicos de fluxo como PR review, deploy e migrations.

Questão 34 (Cenário: Geração de Código com Claude Code)

Situação: Você foi designado para reestruturar a aplicação monolítica do seu time em microsserviços. Isso impacta mudanças em dezenas de arquivos e exige decisões sobre fronteiras de serviço e dependências de módulos.

Qual abordagem você deve escolher?

- A) Mudar para modo de planejamento para explorar a base, entender dependências e desenhar a abordagem de implementação antes de fazer mudanças. **[CORRETA]**
- B) Começar em execução direta e mudar para planejamento somente após encontrar complexidade inesperada durante a implementação.
- C) Começar em execução direta e fazer mudanças incrementais, deixando a implementação revelar fronteiras naturais de serviço.
- D) Usar execução direta com instruções detalhadas antecipadas que especificam a estrutura de cada serviço.

Por que A: Modo de planejamento é a estratégia certa para reestruturação arquitetural complexa como dividir um monolito: permite exploração segura e decisões informadas sobre fronteiras antes de comprometer-se com mudanças potencialmente caras em muitos arquivos.

Questão 35 (Cenário: Geração de Código com Claude Code)

Situação: Seu time criou uma skill `/analyze-codebase` que faz análise profunda de código — varredura de dependências, contagem de cobertura de testes e métricas de qualidade. Após rodar o comando, membros do time relatam que Claude fica menos responsivo na sessão e perde o contexto da tarefa original.

Como você corrige isso de forma mais efetiva mantendo as capacidades plenas de análise?

- A) Adicionar `context: fork` no frontmatter da skill para rodar a análise em contexto isolado de subagente. **[CORRETA]**
- B) Adicionar `model: haiku` no frontmatter para usar um modelo mais rápido e barato para análise.
- C) Dividir a skill em três skills menores, cada uma produzindo menos saída.
- D) Adicionar instruções à skill para comprimir todos os resultados em um sumário curto antes de exibi-los.

Por que A: `context: fork` roda a análise em contexto de subagente isolado, de modo que a saída grande não polui a janela de contexto da sessão principal e Claude não perde o rastro da tarefa original. Preserva capacidade plena de análise mantendo a sessão principal responsiva.

Questão 36 (Cenário: Geração de Código com Claude Code)

Situação: Seu time usa uma skill `/commit` em `.claude/skills/commit/SKILL.md`. Um dev quer customizá-la para seu fluxo pessoal (formato diferente de commit message, checagens extras) sem afetar

colegas.

O que você recomenda?

- A) Criar uma versão pessoal sob `~/.claude/skills/` com um nome diferente, ex.: `/my-commit`.
- B) Adicionar lógica condicional baseada em username no frontmatter da skill do projeto.
- C) Criar uma versão pessoal em `~/.claude/skills/commit/SKILL.md` com o mesmo nome. **[CORRETA]**
- D) Definir `override: true` no frontmatter da skill pessoal para priorizá-la sobre a versão do projeto.

Por que C: Skills pessoais têm precedência sobre skills de projeto com o mesmo nome. Uma skill pessoal em `~/.claude/skills/commit/SKILL.md` sobrescreverá a do time, permitindo ao dev customizar seu fluxo enquanto mantém o nome familiar `/commit` para uso pessoal. Essa abordagem é melhor que A porque preserva o nome de comando original, melhorando o fluxo do dev sem afetar colegas.

Questão 37 (Cenário: Geração de Código com Claude Code)

Situação: Seu time usa Claude Code há meses. Recentemente, três devs reportam que Claude segue a orientação "sempre incluir tratamento abrangente de erros", mas um quarto dev recém-chegado diz que Claude não a segue. Os quatro trabalham no mesmo repo e têm código atualizado.

Qual a causa mais provável e correção?

- A) A orientação está nos arquivos de nível usuário `~/.claude/CLAUDE.md` dos devs originais, não no `.claude/CLAUDE.md` do projeto. Mover a instrução para o arquivo de nível projeto para que todos os membros recebam. **[CORRETA]**
- B) O `~/.claude/CLAUDE.md` do novo dev contém instruções conflitantes que sobrescrevem o projeto; ele deve apagar a seção conflitante.
- C) Claude Code aprende preferências por usuário com o tempo; o novo dev precisa repetir o requisito até Claude "lembrar".
- D) Claude Code cacheia CLAUDE.md após a primeira leitura; os devs originais usam versões em cache. Todos devem limpar o cache.

Por que A: Se a orientação foi adicionada apenas aos configs de nível usuário dos devs originais e não ao `.claude/CLAUDE.md` do projeto, novos membros não a recebem. Mover para a configuração de nível projeto garante que todos os membros atuais e futuros recebam a orientação automaticamente.

Questão 38 (Cenário: Geração de Código com Claude Code)

Situação: Você descobre que incluir 2–3 implementações completas de endpoint como contexto melhora significativamente a consistência ao gerar novos endpoints de API. Porém, esse contexto só é útil ao criar novos endpoints — não ao depurar, revisar código ou outros trabalhos no diretório de API.

Qual abordagem de configuração é mais efetiva?

- A) Adicionar exemplos de endpoint e documentação de padrão ao CLAUDE.md do projeto para que estejam sempre disponíveis.
- B) Referenciar manualmente exemplos de endpoint em cada solicitação de geração copiando código no prompt.
- C) Configurar regras path-specific em `.claude/rules/api/` que incluam exemplos de endpoint e ativem ao trabalhar no diretório de API.
- D) Criar uma skill que referencie os exemplos de endpoint e contenha instruções de seguimento de padrão, invocada sob demanda via slash command. **[CORRETA]**

Por que D: Uma skill invocada sob demanda carrega o contexto de exemplo apenas ao gerar novos endpoints, não em tarefas não relacionadas como depuração ou review. Isso mantém o contexto principal limpo enquanto preserva geração de alta qualidade quando necessária.

Questão 39 (Cenário: Geração de Código com Claude Code)

Situação: Seu time criou uma skill `/migration` que gera arquivos de migração de banco. Recebe o nome da migração via `$ARGUMENTS`. Em produção você observa três problemas: (1) devs frequentemente rodam a skill sem argumentos, causando arquivos mal nomeados, (2) a skill às vezes usa detalhes de schema de conversas anteriores não relacionadas, e (3) um dev rodou acidentalmente cleanup destrutivo de teste quando a skill tinha acesso amplo a ferramentas.

Qual abordagem de configuração corrige todos os três problemas?

- A) Usar parâmetros posicionais `$1` e `$2` em vez de `$ARGUMENTS` para forçar entradas específicas, incluir referências explícitas a schema via sintaxe `@` para controle de contexto, e adicionar uma descrição no frontmatter alertando sobre operações destrutivas.
- B) Adicionar `argument-hint` no frontmatter para pedir parâmetros obrigatórios, usar `context: fork` para isolar a execução, e restringir `allowed-tools` a operações de escrita de arquivo. **[CORRETA]**
- C) Dividir em `/migration-create` e `/migration-apply`, adicionar instruções de validação para pedir o nome da migração se faltar e usar escopos de `allowed-tools` diferentes para cada.
- D) Adicionar instruções de validação no SKILL.md para garantir que `$ARGUMENTS` seja um nome válido, adicionar prompts para ignorar contexto de conversas anteriores e listar operações proibidas.

Por que B: Usa três features de configuração separadas para tratar cada problema: `argument-hint` melhora a entrada de argumentos e reduz argumentos faltando, `context: fork` impede vazamento de contexto de conversas anteriores e `allowed-tools` restringe a skill a operações seguras de escrita de arquivo, prevenindo ações destrutivas.

Questão 40 (Cenário: Geração de Código com Claude Code)

Situação: Sua base contém áreas com convenções de código diferentes: componentes React usam estilo funcional com hooks, handlers de API usam async/await com tratamento específico de erros e modelos de banco seguem o padrão repository. Arquivos de teste estão distribuídos pela base ao lado do código sob

teste (ex.: `Button.test.tsx` ao lado de `Button.tsx`), e você quer que todos os testes sigam as mesmas convenções independentemente da localização.

Qual é a forma mais suportada de garantir que Claude aplique automaticamente as convenções corretas ao gerar código?

- A) Colocar todas as convenções no CLAUDE.md raiz sob cabeçalhos por área e contar com Claude para inferir qual seção se aplica.
- B) Criar skills em `.claude/skills/` para cada tipo de código, embutindo convenções em cada SKILL.md.
- C) Colocar um arquivo CLAUDE.md separado em cada subdiretório com convenções para aquela área.
- D) Criar arquivos de regra em `.claude/rules/` com frontmatter YAML especificando padrões glob para aplicar convenções condicionalmente baseadas em paths. **[CORRETA]**

Por que D: Arquivos `.claude/rules/` com frontmatter YAML e padrões glob (ex.: `**/*.test.tsx`, `src/api/**/*.ts`) permitem aplicação determinística e baseada em path de convenções independente da estrutura de diretórios. É a abordagem mais suportada para padrões cross-cutting como arquivos de teste distribuídos.

Questão 41 (Cenário: Geração de Código com Claude Code)

Situação: Você quer criar um slash command customizado `/review` que rode o checklist padrão de code review do seu time. Ele deve estar disponível para todo dev quando clonar ou atualizar o repositório.

Onde criar o arquivo do comando?

- A) Em `~/.claude/commands/` no diretório home de cada dev.
- B) No repositório do projeto sob `.claude/commands/`. **[CORRETA]**
- C) Em `.claude/config.json` como array de comandos.
- D) No CLAUDE.md raiz do projeto.

Por que B: Colocar slash commands customizados sob `.claude/commands/` dentro do repositório do projeto garante que estejam versionados e disponíveis automaticamente para todo dev que clone ou atualize o repo. É o local previsto para comandos customizados de nível projeto no Claude Code.

Questão 42 (Cenário: Geração de Código com Claude Code)

Situação: O CLAUDE.md do seu time cresceu além de 500 linhas misturando convenções TypeScript, orientação de testes, padrões de API e procedimentos de deploy. Devs têm dificuldade de localizar e atualizar as seções certas.

Qual abordagem o Claude Code suporta para organizar instruções de nível projeto em módulos tópicos focados?

- A) Definir um `.claude/config.yaml` mapeando padrões de arquivo a seções específicas dentro do CLAUDE.md.
- B) Criar arquivos Markdown separados em `.claude/rules/`, cada um cobrindo um tópico (ex.: `testing.md`, `api-conventions.md`). **[CORRETA]**
- C) Dividir instruções em arquivos README.md em subdiretórios relevantes que Claude carrega automaticamente como instruções.
- D) Criar múltiplos arquivos chamados CLAUDE.md em níveis diferentes da árvore de diretórios, cada um sobrescrevendo instruções pai.

Por que B: Claude Code suporta um diretório `.claude/rules/` onde você pode criar arquivos Markdown separados para guias tópicos (ex.: `testing.md`, `api-conventions.md`), permitindo a times organizar grandes conjuntos de instruções em módulos focados e mantíveis.

Questão 43 (Cenário: Geração de Código com Claude Code)

Situação: Você cria uma skill customizada `/explore-alternatives` que seu time usa para fazer brainstorming e avaliar abordagens de implementação antes de escolher uma. Devs reportam que após rodar a skill, respostas subsequentes do Claude são influenciadas pela discussão de alternativas — às vezes referenciando abordagens rejeitadas ou retendo contexto de exploração que interfere na implementação real.

Como configurar essa skill de forma mais efetiva?

- A) Usar o prefixo `!` na skill para rodar lógica de exploração como subprocesso bash.
- B) Adicionar `context: fork` no frontmatter da skill. **[CORRETA]**
- C) Dividir em duas skills — `/explore-start` e `/explore-end` — para marcar fronteiras quando o contexto de exploração deve ser descartado.
- D) Criar a skill em `~/.claude/skills/` em vez de `.claude/skills/`.

Por que B: `context: fork` roda a skill em contexto isolado de subagente para que discussões de exploração não poluam o histórico da conversa principal. Isso impede que abordagens rejeitadas e contexto de brainstorming influenciem trabalho subsequente de implementação.

Questão 44 (Cenário: Geração de Código com Claude Code)

Situação: Seu time quer adicionar um servidor MCP do GitHub para buscar PRs e checar status de CI via Claude Code. Cada um dos seis devs tem seu próprio token pessoal de acesso ao GitHub. Você quer ferramental consistente em todo o time sem commitar credenciais no controle de versão.

Qual abordagem de configuração é mais efetiva?

- A) Que cada dev adicione o servidor em escopo de usuário via `claude mcp add --scope user`.
- B) Criar um wrapper de servidor MCP que leia tokens de um arquivo `.env` e proxie chamadas à API do GitHub, depois adicionar o wrapper ao `.mcp.json` do projeto.

- C) Adicionar o servidor ao `.mcp.json` do projeto usando substituição de variável de ambiente (`${GITHUB_TOKEN}`) para auth e documentar a variável necessária no README. **[CORRETA]**
- D) Configurar o servidor no escopo de projeto com token placeholder, depois pedir aos devs que sobrescrevam no config local.

Por que C: Um `.mcp.json` de projeto com substituição de variável de ambiente é idiomático: fornece uma única fonte versionada de verdade para a configuração MCP enquanto deixa cada dev fornecer credenciais via variáveis de ambiente. Documentar a variável facilita o onboarding sem commitar segredos.

Questão 45 (Cenário: Geração de Código com Claude Code)

Situação: Você está adicionando wrappers de tratamento de erros ao redor de chamadas de API externas em uma base de 120 arquivos. O trabalho tem três fases: (1) descobrir todos os call sites e padrões, (2) desenhar colaborativamente a abordagem de tratamento de erros, e (3) implementar wrappers de forma consistente. Na Fase 1, Claude gera saída grande listando centenas de call sites com contexto, enchendo rapidamente a janela de contexto antes da descoberta terminar.

Qual abordagem é mais efetiva para concluir a tarefa mantendo consistência de implementação?

- A) Usar um subagente Explore para a Fase 1 para isolar saída verbosa de descoberta e retornar um sumário, depois continuar Fases 2–3 na conversa principal. **[CORRETA]**
- B) Fazer todas as fases na conversa principal, periodicamente usando `/compact` para reduzir uso de contexto enquanto avança nos arquivos.
- C) Mudar para modo headless com `--continue`, passando sumários explícitos de contexto entre chamadas em batch para manter continuidade.
- D) Definir o padrão de tratamento de erros no CLAUDE.md, depois processar arquivos em lotes em múltiplas sessões contando com o arquivo de memória compartilhada para consistência.

Por que A: Um subagente Explore isola a saída verbosa de descoberta em contexto separado e retorna apenas um sumário conciso à conversa principal. Isso preserva a janela de contexto principal para as fases de design colaborativo e implementação consistente, em que contexto retido é mais valioso.

Cenário: Agente de Suporte ao Cliente

Questão 46 (Cenário: Agente de Suporte ao Cliente)

Situação: Em testes, você nota que o agente frequentemente chama `get_customer` quando usuários perguntam sobre status de pedido, embora `lookup_order` fosse mais apropriado. O que você deve checar primeiro para tratar esse problema?

O que você deve checar primeiro?

- A) Implementar um classificador de pré-processamento para detectar pedidos relacionados a ordem e roteá-los diretamente a `lookup_order`.
- B) Reduzir o número de ferramentas disponíveis ao agente para simplificar a escolha.
- C) Adicionar exemplos few-shot ao system prompt cobrindo todos os padrões possíveis de pedido para melhorar a seleção de ferramentas.
- D) Verificar as descrições das ferramentas para garantir que diferenciem claramente o propósito de cada uma. **[CORRETA]**

Por que D: Descrições de ferramentas são a entrada principal que o modelo usa para decidir qual chamar. Quando um agente escolhe consistentemente a ferramenta errada, o primeiro passo de diagnóstico é verificar se as descrições separam claramente o propósito e os limites de uso.

Questão 47 (Cenário: Agente de Suporte ao Cliente)

Situação: Seu agente lida com pedidos de questão única com 94% de acurácia (ex.: "preciso de reembolso para o pedido #1234"). Mas quando clientes incluem múltiplas questões em uma mensagem (ex.: "preciso de reembolso para o pedido #1234 e também atualizar o endereço de entrega do pedido #5678"), a acurácia de seleção de ferramenta cai para 58%. O agente normalmente resolve apenas uma questão ou mistura parâmetros entre os pedidos. Qual abordagem melhora confiabilidade para pedidos multi-questão de forma mais efetiva?

Qual abordagem é mais efetiva?

- A) Implementar uma camada de pré-processamento que use uma chamada separada do modelo para decompor mensagens multi-questão em pedidos separados, lidar com cada um independentemente e mesclar resultados.
- B) Combinar ferramentas relacionadas em menos ferramentas universais.
- C) Adicionar exemplos few-shot ao prompt demonstrando raciocínio correto e sequenciamento de ferramentas para pedidos multi-questão. **[CORRETA]**
- D) Implementar validação de resposta que detecta respostas incompletas e re-prompta automaticamente o agente para resolver questões perdidas.

Por que C: Exemplos few-shot que demonstram raciocínio correto e sequenciamento de ferramentas para pedidos multi-questão são mais efetivos porque o agente já performa bem em questão única — o que precisa é orientação sobre o padrão para decompor e rotear múltiplas questões mantendo parâmetros separados.

Questão 48 (Cenário: Agente de Suporte ao Cliente)

Situação: Logs de produção mostram que para pedidos simples como "reembolso para o pedido #1234", seu agente resolve em 3–4 chamadas de ferramenta com 91% de sucesso. Mas para pedidos complexos como "fui cobrado em dobro, meu desconto não foi aplicado e quero cancelar", o agente faz em média 12+ chamadas com apenas 54% de sucesso — frequentemente investigando questões sequencialmente e buscando dados redundantes do cliente para cada uma. Qual mudança melhora o tratamento de pedidos complexos de forma mais efetiva?

Qual mudança é mais efetiva?

- A) Adicionar checkpoints explícitos de verificação entre estágios, exigindo que o agente registre progresso após resolver cada questão antes de passar para a próxima.
- B) Reduzir o número de ferramentas combinando `get_customer`, `lookup_order` e ferramentas relacionadas a cobrança em uma única `investigate_issue`.
- C) Decompor o pedido em questões separadas, depois investigar cada uma em paralelo usando contexto compartilhado do cliente antes de sintetizar uma resolução final. **[CORRETA]**
- D) Adicionar exemplos few-shot ao system prompt demonstrando sequências ideais de chamada de ferramentas para vários cenários multi-faceta de cobrança.

Por que C: Decompor em questões separadas e investigar em paralelo com contexto compartilhado do cliente corrige ambos os problemas-chave: elimina recuperação redundante de dados reutilizando contexto compartilhado entre questões e reduz total de loops de chamada paralelizando investigação antes de sintetizar uma única resolução.

Questão 49 (Cenário: Agente de Suporte ao Cliente)

Situação: Seu agente atinge 55% de resolução no primeiro contato, bem abaixo da meta de 80%. Logs mostram que ele escala casos simples (substituições padrão para mercadorias danificadas com prova fotográfica) enquanto tenta lidar autonomamente com situações complexas que exigem exceções de política. Qual é a forma mais efetiva de melhorar a calibração de escalonamento?

Qual é a forma mais efetiva de melhorar a calibração?

- A) Exigir que o agente auto-classifique confiança em escala 1–10 antes de cada resposta e roteie automaticamente para humanos quando a confiança cair abaixo de um limiar.
- B) Implantar um modelo classificador separado treinado em tickets históricos para prever quais pedidos precisam de escalonamento antes de o agente principal começar.
- C) Adicionar critérios explícitos de escalonamento ao system prompt com exemplos few-shot mostrando quando escalar vs resolver autonomamente. **[CORRETA]**
- D) Implementar análise de sentimento para determinar nível de frustração do cliente e escalar automaticamente acima de um limiar de sentimento negativo.

Por que C: Critérios explícitos de escalonamento com exemplos few-shot atacam diretamente a causa-raiz — limites de decisão pouco claros entre casos simples e complexos. É a primeira intervenção mais proporcional e efetiva, ensinando o agente quando escalar e quando resolver autonomamente sem infraestrutura extra.

Questão 50 (Cenário: Agente de Suporte ao Cliente)

Situação: Após chamar `get_customer` e `lookup_order`, o agente tem todos os dados disponíveis no sistema mas ainda enfrenta incerteza. Qual situação é o gatilho mais justificado para chamar `escalate_to_human`?

Qual situação é mais justificada para escalonamento?

- A) Um cliente quer cancelar um pedido enviado ontem e que chega amanhã. O agente deve escalar porque o cliente pode mudar de ideia depois de receber o pacote.
- B) Um cliente afirma que não recebeu um pedido, mas o tracking mostra que foi entregue e assinado em seu endereço três dias atrás. O agente deve escalar porque apresentar evidência contraditória pode prejudicar o relacionamento.
- C) Um cliente solicita matching de preço de concorrente. Suas políticas permitem ajustes de preço para quedas no próprio site dentro de 14 dias, mas não dizem nada sobre preços de concorrente. O agente deve escalar para interpretação de política. **[CORRETA]**
- D) Uma mensagem do cliente contém tanto uma questão de cobrança quanto uma devolução. O agente deve escalar para que um humano coordene ambas em uma única interação.

Por que C: Esta é uma genuína lacuna de política: regras da empresa cobrem quedas de preço no próprio site mas não tratam matching de concorrente. O agente não deve inventar política e deve escalar para julgamento humano sobre como interpretar ou estender regras existentes.

Questão 51 (Cenário: Agente de Suporte ao Cliente)

Situação: Logs de produção mostram que em 12% dos casos seu agente pula `get_customer` e chama `lookup_order` diretamente usando apenas o nome fornecido pelo cliente, às vezes levando a contas mal identificadas e reembolsos incorretos. Qual mudança corrige esse problema de confiabilidade de forma mais efetiva?

Qual mudança é mais efetiva?

- A) Adicionar exemplos few-shot mostrando que o agente sempre chama `get_customer` primeiro, mesmo quando clientes voluntariamente fornecem detalhes do pedido.
- B) Implementar um classificador de roteamento que analisa cada pedido e habilita só um subconjunto de ferramentas apropriado para o tipo.
- C) Adicionar uma pré-condição programática que bloqueia `lookup_order` e `process_refund` até que `get_customer` retorne um identificador verificado. **[CORRETA]**
- D) Reforçar o system prompt declarando que verificação do cliente via `get_customer` é mandatória antes de qualquer operação de pedido.

Por que C: Uma pré-condição programática fornece garantia determinística de que o sequenciamento exigido seja seguido. É a abordagem mais efetiva porque elimina a possibilidade de pular a verificação, independentemente do comportamento do LLM.

Questão 52 (Cenário: Agente de Suporte ao Cliente)

Situação: Métricas de produção mostram que ao resolver disputas complexas de cobrança ou devoluções multi-pedido, scores de satisfação do cliente são 15% menores que para casos simples — mesmo quando a resolução está tecnicamente correta. Análise de causa-raiz mostra que o agente fornece soluções acuradas mas explica a justificativa de forma inconsistente: às vezes omitindo detalhes relevantes de política, às vezes perdendo info de timeline ou próximos passos. As lacunas de contexto

variam caso a caso. Você quer melhorar qualidade da solução sem adicionar supervisão humana. Qual abordagem é mais efetiva?

Qual abordagem é mais efetiva?

- A) Adicionar uma etapa de auto-crítica em que o agente avalia uma resposta de rascunho quanto à completude — garantindo que resolva o problema, inclua contexto relevante e antecipe perguntas de follow-up. **[CORRETA]**
- B) Adicionar uma etapa de confirmação em que o agente pergunta "Isso resolve totalmente seu problema?" antes de fechar, permitindo que clientes peçam informações adicionais.
- C) Atualizar o modelo de Haiku para Sonnet em casos complexos, roteando com base em uma métrica de complexidade definida.
- D) Implementar exemplos few-shot no system prompt mostrando explicações completas para cinco tipos comuns de casos complexos, demonstrando como incluir contexto de política, timelines e próximos passos.

Por que A: Uma etapa de auto-crítica (padrão evaluator-optimizer) ataca diretamente a completude inconsistente das explicações forçando o agente a avaliar seu próprio rascunho contra critérios concretos — como contexto de política, timelines e próximos passos — antes de apresentá-lo. Isso captura lacunas específicas do caso sem supervisão humana.

Questão 53 (Cenário: Agente de Suporte ao Cliente)

Situação: Métricas de produção mostram que seu agente faz em média 4+ loops de API por resolução. A análise revela que Claude frequentemente solicita `get_customer` e `lookup_order` em turnos sequenciais separados, mesmo quando ambos são necessários inicialmente. Qual é a forma mais efetiva de reduzir o número de loops?

Qual é a forma mais efetiva de reduzir loops?

- A) Implementar execução especulativa que automaticamente chama ferramentas provavelmente necessárias em paralelo a qualquer ferramenta solicitada e retorna todos os resultados, independentemente do que foi pedido.
- B) Aumentar `max_tokens` para dar a Claude mais espaço para planejar e combinar naturalmente solicitações de ferramenta.
- C) Criar ferramentas compostas como `get_customer_with_orders` que agrupam combinações comuns de lookup em uma única chamada.
- D) Instruir Claude no prompt a agrupar solicitações de ferramenta em um turno e retornar todos os resultados juntos antes da próxima chamada de API. **[CORRETA]**

Por que D: Pedir a Claude que agrupe solicitações de ferramenta relacionadas em um único turno aproveita sua capacidade nativa de pedir múltiplas ferramentas de uma vez. Corrige diretamente o padrão de chamadas sequenciais com mudança arquitetural mínima.

Questão 54 (Cenário: Agente de Suporte ao Cliente)

Situação: Logs de produção mostram um padrão: clientes referenciam valores específicos (ex.: "o desconto de 15% que mencionei"), mas o agente responde com valores incorretos. Investigação mostra que esses detalhes foram mencionados 20+ turnos atrás e condensados em sumários vagos como "preços promocionais foram discutidos". Qual correção é mais efetiva?

Qual correção é mais efetiva?

- A) Aumentar o limiar de sumarização de 70% para 85% para que conversas tenham mais espaço antes de a sumarização disparar.
- B) Armazenar histórico completo da conversa em armazenamento externo e implementar recuperação quando o agente detectar referências como "como mencionei".
- C) Extrair fatos transacionais (valores, datas, números de pedido) para um bloco persistente de "case facts" incluído em todo prompt fora do histórico sumarizado. **[CORRETA]**
- D) Revisar o prompt de sumarização para preservar explicitamente todos os números, percentuais, datas e expectativas declaradas pelo cliente verbatim.

Por que C: Sumarização inerentemente perde detalhes precisos. Extrair fatos transacionais para um bloco estruturado "case facts" fora do histórico sumarizado preserva informação crítica para que esteja disponível de forma confiável em todo prompt, independente de quantos turnos foram sumarizados.

Questão 55 (Cenário: Agente de Suporte ao Cliente)

Situação: Sua ferramenta `get_customer` retorna todas as correspondências ao buscar por nome. Atualmente, quando há múltiplos resultados, Claude escolhe o cliente com pedido mais recente, mas dados de produção mostram que isso seleciona a conta errada em 15% dos casos para correspondências ambíguas. Como você deveria tratar isso?

Como você deveria tratar isso?

- A) Implementar um sistema de pontuação de confiança que age autonomamente acima de 85% e pede esclarecimento abaixo do limiar.
- B) Instruir Claude a pedir um identificador adicional (email, telefone ou número do pedido) quando `get_customer` retornar múltiplas correspondências antes de tomar qualquer ação específica do cliente. **[CORRETA]**
- C) Modificar `get_customer` para retornar apenas uma única correspondência mais provável com base em algoritmo de ranking, eliminando ambiguidade.
- D) Adicionar exemplos few-shot ao prompt demonstrando raciocínio correto e sequenciamento de ferramentas para correspondências ambíguas.

Por que B: Pedir ao usuário um identificador adicional é a forma mais confiável de resolver ambiguidade porque o usuário tem conhecimento definitivo da própria identidade. Um turno de conversa extra é um preço pequeno a pagar para eliminar uma taxa de erro de 15% causada por escolher a conta errada.

Questão 56 (Cenário: Agente de Suporte ao Cliente)

Situação: Logs de produção mostram um padrão consistente: quando clientes incluem a palavra "conta" em sua mensagem (ex.: "quero verificar minha conta de um pedido que fiz ontem"), o agente chama `get_customer` primeiro 78% das vezes. Quando clientes formulam pedidos similares sem "conta" (ex.: "quero verificar um pedido que fiz ontem"), chama `lookup_order` primeiro 93% das vezes. As descrições das ferramentas são claras e inequívocas. Qual a causa-raiz mais provável dessa discrepância?

Qual a causa-raiz mais provável?

- A) O system prompt contém instruções sensíveis a palavras-chave que direcionam comportamento baseado em termos como "conta", criando padrões não intencionais de seleção de ferramenta. **[CORRETA]**
- B) O treinamento base do modelo cria associações entre terminologia de "conta" e operações relacionadas ao cliente que sobrepõem descrições de ferramentas.
- C) O modelo precisa de mais dados de treino sobre mensagens multi-conceito e deveria ser fine-tuned em exemplos contendo terminologias de conta e pedido.
- D) Descrições de ferramentas precisam de exemplos negativos adicionais especificando quando NÃO usar cada ferramenta para evitar essa confusão induzida por palavras-chave.

Por que A: O padrão sistemático dirigido por palavra-chave (78% vs 93%) indica fortemente lógica explícita de roteamento no system prompt reagindo à palavra "conta" e direcionando o agente a ferramentas relacionadas ao cliente. Como as descrições já são claras, a discrepância aponta para instruções no nível do prompt criando direcionamento comportamental não intencional.

Questão 57 (Cenário: Agente de Suporte ao Cliente)

Situação: Logs de produção mostram que o agente frequentemente chama `get_customer` quando usuários perguntam sobre pedidos (ex.: "verifique meu pedido #12345") em vez de chamar `lookup_order`. Ambas as ferramentas têm descrições mínimas ("Gets customer information" / "Gets order details") e aceitam formatos de identificador parecidos. Qual é o primeiro passo mais efetivo para melhorar a confiabilidade da seleção de ferramenta?

Qual é o primeiro passo mais efetivo?

- A) Implementar uma camada de roteamento que analisa o input do usuário antes de cada turno e pré-seleciona a ferramenta correta com base em palavras-chave detectadas e padrões de ID.
- B) Combinar ambas em uma única `lookup_entity` que aceita qualquer identificador e decide internamente qual backend consultar.
- C) Adicionar exemplos few-shot ao system prompt demonstrando padrões corretos de seleção, com 5–8 exemplos roteando consultas relacionadas a pedido para `lookup_order`.
- D) Expandir a descrição de cada ferramenta para incluir formatos de entrada, queries de exemplo, casos de borda e fronteiras explicando quando usá-la versus ferramentas similares. **[CORRETA]**

Por que D: Expandir descrições com formatos de entrada, queries de exemplo, casos de borda e fronteiras claras corrige diretamente a causa-raiz — descrições mínimas que não dão ao LLM informação

suficiente para distinguir ferramentas similares. É um primeiro passo de baixo esforço e alto impacto que melhora o mecanismo principal usado pelo LLM para seleção de ferramenta.

Questão 58 (Cenário: Agente de Suporte ao Cliente)

Situação: Você está implementando o loop do agente de suporte. Após cada chamada de API ao Claude, é preciso decidir se continua o loop (executa as ferramentas pedidas e chama Claude novamente) ou para (apresenta a resposta final ao cliente). O que determina essa decisão?

O que determina essa decisão?

- A) Verificar o campo `stop_reason` na resposta do Claude — continuar se for `tool_use` e parar se for `end_turn`. **[CORRETA]**
- B) Parsear o texto do Claude por frases como "Estou pronto" ou "Posso ajudar com mais alguma coisa?" — sinais de linguagem natural indicam conclusão.
- C) Definir um número máximo de iterações (ex.: 10 chamadas) e parar quando atingido, independentemente de Claude indicar mais trabalho.
- D) Verificar se a resposta contém conteúdo de texto do assistente — se Claude gerou texto explicativo, o loop deve terminar.

Por que A: `stop_reason` é o sinal estruturado explícito do Claude para controle de loop: `tool_use` indica que Claude quer rodar uma ferramenta e receber resultados de volta, enquanto `end_turn` indica que Claude completou sua resposta e o loop deve encerrar.

Questão 59 (Cenário: Agente de Suporte ao Cliente)

Situação: Logs de produção mostram que o agente interpreta mal saídas de suas ferramentas MCP: timestamps Unix de `get_customer`, datas ISO 8601 de `lookup_order` e códigos de status numéricos (1=pendente, 2=enviado). Algumas ferramentas são servidores MCP de terceiros que você não pode modificar. Qual abordagem para normalização de formato de dados é mais mantível?

Qual abordagem é mais mantível?

- A) Usar um hook PostToolUse para interceptar saídas de ferramentas e aplicar transformações de formato antes que o agente processe. **[CORRETA]**
- B) Modificar ferramentas que você controla para retornar formatos legíveis para humanos e criar wrappers para ferramentas de terceiros.
- C) Criar uma ferramenta `normalize_data` que o agente chama após cada recuperação de dados para transformar valores.
- D) Adicionar documentação detalhada de formato no system prompt explicando convenções de dado de cada ferramenta.

Por que A: Um hook PostToolUse fornece um ponto centralizado e determinístico para interceptar e normalizar todas as saídas de ferramenta — incluindo dados de servidores MCP de terceiros — antes que o agente as processe. É mais mantível porque transformações vivem em código e se aplicam uniformemente, sem depender da interpretação do LLM.

Questão 60 (Cenário: Agente de Suporte ao Cliente)

Situação: Logs de produção mostram que o agente às vezes escolhe `get_customer` quando `lookup_order` seria mais apropriado, especialmente para queries ambíguas como "preciso de ajuda com minha compra recente". Você decide adicionar exemplos few-shot ao system prompt para melhorar a seleção. Qual abordagem trata o problema de forma mais efetiva?

Qual abordagem é mais efetiva?

- A) Adicionar orientações explícitas "use quando" e "não use quando" em cada descrição de ferramenta cobrindo casos ambíguos.
- B) Adicionar exemplos agrupados por ferramenta — todos os cenários de `get_customer` juntos, depois todos os cenários de `lookup_order`.
- C) Adicionar 4–6 exemplos direcionados a cenários ambíguos, cada um com justificativa de por que uma ferramenta foi escolhida em detrimento de alternativas plausíveis. **[CORRETA]**
- D) Adicionar 10–15 exemplos de pedidos claros e inequívocos demonstrando escolha correta para cenários típicos de cada ferramenta.

Por que C: Direcionar exemplos few-shot aos cenários ambíguos específicos onde os erros ocorrem, com justificativa explícita de por que uma ferramenta é preferível, ensina ao modelo o processo de decisão comparativo necessário para casos de borda. Mais efetivo que exemplos genéricos ou regras declarativas.

Questão 61 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Sua ferramenta `remove_team_member` usa um parâmetro `dry_run: boolean` para visualizar impactos antes da execução. Monitoramento de produção mostra que o agente burla o passo de preview chamando com `dry_run=false` diretamente. Você precisa garantir que toda remoção seja precedida por um preview que o usuário confirme explicitamente.

Qual é a abordagem mais confiável?

- A) Adicionar validação no servidor que permite `dry_run=false` apenas quando uma chamada `dry_run=true` com parâmetros idênticos ocorreu nos últimos 60 segundos.
- B) Anotar a ferramenta como exigindo confirmação e configurar a camada de orquestração para pedir aprovação ao usuário antes de encaminhar quaisquer chamadas a ferramentas anotadas.
- C) Adicionar instruções detalhadas e exemplos few-shot na descrição da ferramenta exigindo que o agente sempre chame com `dry_run=true` primeiro e espere confirmação antes de chamar de novo.
- D) Substituir por duas ferramentas: `preview_remove_member` retorna detalhes de impacto e um token de confirmação de uso único; `execute_remove_member` exige esse token, ligando a execução ao preview. **[CORRETA]**

Por que D: A abordagem de duas ferramentas com binding por token torna arquiteturalmente impossível executar sem um preview prévio — a ferramenta de execução literalmente exige um token que só a

ferramenta de preview pode gerar. É a única abordagem que aplica a restrição no nível de código em vez de depender de conformidade do LLM com instruções (C), heurísticas de timing (A) ou infraestrutura de orquestração (B).

Questão 62 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Monitoramento de produção mostra que sua ferramenta `search_catalog` falha 12% do tempo: 8% são timeouts de rede que sucedem ao retentar e 4% são erros de sintaxe de query que nunca sucedem por mais que se retente. Atualmente ambos os tipos de erro são retornados de forma idêntica, causando retentativas desperdiçadas.

Como você deveria modificar o tratamento de erros da ferramenta?

- A) Adicionar exemplos few-shot ao system prompt demonstrando como distinguir erros de rede de erros de sintaxe.
- B) Aplicar lógica de retry com backoff exponencial uniformemente a todos os erros.
- C) Implementar retry automático com backoff para timeouts de rede dentro da ferramenta; retornar erros de sintaxe imediatamente com detalhes de validação de parâmetro. **[CORRETA]**
- D) Retornar todos os erros com flag booleana `retryable` e detalhes do tipo de erro.

Por que C: Tratar retries no nível da ferramenta para erros transitórios é a fronteira de abstração correta — a ferramenta tem conhecimento definitivo do tipo de erro e pode implementar lógica de retry determinística sem depender do agente para interpretar uma flag (D) ou seguir instruções no nível do prompt (A). Backoff uniforme (B) desperdiça tempo em erros de sintaxe que nunca sucederão.

Questão 63 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Ao longo de vários turnos discutindo estratégia de investimento, um usuário declarou "tenho tolerância a risco muito baixa" e depois "quero maximizar meus retornos". Agora ele pergunta: "Em que devo investir?"

Qual abordagem melhor garante que a recomendação se alinhe com a prioridade real do usuário?

- A) Trazer à tona a contradição e pedir ao usuário que esclareça o que importa mais. **[CORRETA]**
- B) Fornecer recomendações separadas para ambos os cenários.
- C) Prosseguir com a preferência declarada mais recentemente.
- D) Recomendar um portfólio balanceado sem tratar do conflito.

Por que A: Quando preferências do usuário se contradizem diretamente, trazer o conflito à tona e pedir esclarecimento é a única forma de garantir que a recomendação se alinhe com a real intenção do usuário. Qualquer outra abordagem envolve assumir algo que pode estar errado — maximizar retornos e baixa tolerância a risco são objetivos fundamentalmente incompatíveis que exigem decisão humana.

Questão 64 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Usuários refinam preferências de playlist em vários turnos. Duas mensagens depois de o usuário dizer "amo jazz", Claude pergunta "Quais gêneros você gosta?".

Qual a causa mais provável?

- A) Claude exige conexão com base vetorial para manter memória de conversa.
- B) A janela de contexto do modelo foi excedida.
- C) A Claude API exige um parâmetro `session_id`.
- D) Sua aplicação não está incluindo as mensagens anteriores no array `messages`. **[CORRETA]**

Por que D: Claude não tem memória do lado do servidor — toda chamada de API é stateless. Sem incluir o histórico completo no array `messages` de cada requisição, Claude não tem conhecimento de turnos anteriores. Bases vetoriais (A) e `session_id` (C) não fazem parte da arquitetura do Claude; estouro de janela de contexto (B) é impossível em troca de duas mensagens.

Questão 65 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Após uma sessão culinária de 40 minutos, a conversa atinge 78.000 tokens. O histórico inclui alergias, escala de receitas, termos culinários esclarecidos e discussão geral. Você precisa reduzir tokens preservando informação importante.

Qual abordagem melhor balanceia preservação e redução de tokens?

- A) Sumarizar todo o histórico da conversa.
- B) Manter apenas os 20.000 tokens mais recentes.
- C) Extrair dados estruturados críticos (alergias, quantidades, preferências), sumarizar a discussão geral e manter trocas recentes verbatim. **[CORRETA]**
- D) Armazenar a conversa completa externamente e recuperar partes relevantes via busca semântica.

Por que C: A abordagem híbrida preserva a informação de maior valor ao menor custo. Fatos críticos como alergias e quantidades são extraídos para um bloco estruturado compacto (evitando perda de precisão que ocorre na sumarização), discussão geral é sumarizada e trocas recentes mantidas verbatim para coerência conversacional. A e B arriscam perder informação dietética crítica; D é overkill arquitetural para uma única sessão.

Questão 66 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Usuários reportam que durante conversas extensas o assistente perde o rastro de tópicos e preferências anteriores. Sua implementação atual mantém apenas os últimos 25 pares de mensagens.

Qual a solução mais efetiva?

- A) Abordagem híbrida: sumarizar mensagens antigas mantendo as recentes verbatim. **[CORRETA]**
- B) Busca por similaridade vetorial sobre o histórico completo da conversa.
- C) Aumentar a janela para 50 pares de mensagens.
- D) Sumarizar mensagens descartadas a cada turno e prepender o sumário corrente.

Por que A: A abordagem híbrida ataca ambas as dimensões do problema: retém contexto exato recente (crítico para coerência conversacional) enquanto mantém uma representação comprimida das preferências anteriores (evitando perda total quando pares são descartados). Aumentar a janela (C) só atrasa o mesmo problema. Busca vetorial (B) pode perder contexto importante que não é semanticamente similar à query atual. Sumarização total por turno (D) adiciona overhead e acumula erros de sumarização.

Questão 67 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Usuários reportam que latência aumenta e custos sobem quando conversas excedem 50 turnos.

Qual a causa principal?

- A) Todo o histórico da conversa é incluído em cada requisição da API. **[CORRETA]**
- B) O modelo gera respostas progressivamente mais longas.
- C) Operações de banco ficam mais lentas conforme o histórico cresce.
- D) O modelo constrói um perfil interno do usuário que exige mais processamento.

Por que A: A API do Claude é totalmente stateless — toda requisição precisa incluir o histórico completo no array `messages`. Conforme conversas crescem, cada requisição carrega mais tokens, o que aumenta diretamente latência de processamento e custo. O modelo não mantém estado interno entre chamadas (D é falso) e o tamanho da resposta não está intrinsecamente ligado ao tamanho da conversa (B).

Questão 68 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Após três meses de sessões semanais, o histórico cresce para 85.000 tokens. Quando um usuário pergunta "O que concluímos sobre o tema do isolamento?", o assistente dá respostas genéricas em vez de referenciar discussões anteriores.

Qual a abordagem mais efetiva?

- A) Truncamento por janela rolante.
- B) Sumarização progressiva capturando conclusões-chave.
- C) Embeddings semânticos com recuperação de trocas relevantes. **[CORRETA]**
- D) Adicionar tags XML estruturadas marcando conclusões da discussão.

Por que C: Busca semântica sobre o histórico de conversa é a única abordagem que escala para três meses de discussão sendo capaz de surfar trocas específicas relevantes sob demanda. Janela rolante (A) descartaria a maior parte do histórico. Sumarização progressiva (B) comprime discussões em abstrações que perdem as conclusões específicas que os usuários estão pedindo. Tags XML (D) exigem reestruturar todo conteúdo passado e não resolvem o problema de recuperação nessa escala.

Questão 69 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Durante testes de QA, Claude segue diretrizes do system prompt nos primeiros 10–15 turnos, mas respostas posteriores desviam. A conversa ainda está dentro dos limites de tokens.

Qual a melhor solução?

- A) Mover diretrizes comportamentais para a primeira mensagem do usuário.
- B) Iniciar nova conversa após 20 turnos.
- C) Inserir mensagens com role user reforçando diretrizes em pontos de interrupção da conversa. **[CORRETA]**
- D) Usar validação pós-resposta para regenerar respostas não conformes.

Por que C: A injeção periódica de lembretes comportamentais combate diretamente a deriva de instrução restabelecendo restrições em intervalos regulares conforme o histórico se acumula. Mover diretrizes para a primeira mensagem do usuário (A) reduz sua autoridade. Iniciar nova conversa (B) destrói contexto. Validação pós-resposta (D) é corretiva em vez de preventiva e adiciona latência significativa.

Questão 70 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Seu tutor de IA tem um system prompt de 2.800 tokens definindo metodologia de ensino e regras de adaptação. Após 12 turnos, o assistente começa a ignorar níveis de proficiência.

Qual a correção mais efetiva?

- A) Injetar lembretes a cada 4–5 turnos.
- B) Substituir regras verbosas por exemplos few-shot demonstrando adaptação por nível de proficiência. **[CORRETA]**
- C) Posicionar regras críticas no fim do system prompt.
- D) Avaliar respostas e regenerar se o nível de dificuldade não casar.

Por que B: Um system prompt de 2.800 tokens com regras declarativas é vulnerável a deriva porque regras abstratas exigem que o modelo raciocine sobre elas a cada turno. Substituir regras verbosas por exemplos few-shot concretos que demonstram adaptação correta por nível de proficiência dá ao modelo padrões comportamentais claros para casar — isso é seguido de forma mais confiável ao longo de muitos turnos do que instruções abstratas. Injeção de lembretes (A) ajuda mas trata sintomas; posicionamento no fim (C) ajuda inicialmente mas não com deriva por turno; regeneração (D) é cara e corretiva.

Questão 71 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Seu assistente precisa manter um tom entusiástico, explicar seu raciocínio e fazer perguntas esclarecedoras. Onde essas diretrizes comportamentais devem ser definidas?

Onde essas diretrizes comportamentais devem ser definidas?

- A) Prependendo a cada mensagem do usuário.
- B) No system prompt. **[CORRETA]**
- C) Na primeira mensagem do assistente.
- D) Em variáveis de ambiente.

Por que B: O system prompt é especificamente desenhado para restrições e diretrizes comportamentais persistentes que se aplicam ao longo de toda a conversa. Prependê-las a cada mensagem do usuário (A) é overhead redundante. A primeira mensagem do assistente (C) não é confiável porque o modelo pode desviar de suas próprias declarações anteriores. Variáveis de ambiente (D) não têm efeito sobre o comportamento do modelo.

Questão 72 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Usuários reportam aberturas de resposta repetitivas como "Certamente!" e "Ficarei feliz em ajudar!".

Qual a abordagem mais efetiva?

- A) Anexar uma mensagem parcial do assistente com uma abertura direta. **[CORRETA]**
- B) Reduzir o ajuste de temperatura.
- C) Pós-processar respostas para remover saudações.
- D) Adicionar instruções no system prompt para evitar essas frases.

Por que A: Pré-preencher a resposta do assistente com o início de uma resposta direta evita padrões de saudação no nível de geração — o modelo continua a partir do prefill em vez de gerar novas frases de abertura. Instruções no system prompt (D) podem ajudar mas são menos confiáveis pois o modelo ainda pode produzir variantes. Pós-processamento (C) é um workaround frágil. Temperatura (B) controla aleatoriedade, não padrões específicos de frase.

Questão 73 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Um webhook notifica seu sistema de que o pacote do usuário foi enviado enquanto o usuário está conversando ativamente. Você quer que o assistente incorpore isso naturalmente na próxima

resposta.

Qual a melhor abordagem?

- A) Adicionar status de envio ao system prompt.
- B) Enviar imediatamente uma mensagem sintética do usuário.
- C) Forçar o assistente a chamar uma ferramenta de status a cada turno.
- D) Anexar a atualização de status como prefixo à próxima mensagem do usuário. **[CORRETA]**

Por que D: Prefixar a atualização de status à próxima mensagem do usuário injeta contexto em tempo real em uma fronteira natural da conversa sem perturbar o fluxo. Modificar o system prompt (A) exige reconstruir a sessão ou é arquiteturalmente custoso. Uma mensagem sintética do usuário (B) pode quebrar o fluxo natural do diálogo e confundir atribuição. Forçar uma chamada de ferramenta a cada turno (C) é desperdício quando eventos são raros.

Questão 74 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Usuários frequentemente enviam pedidos como "Reserve um local para a festa". O assistente faz 4+ perguntas esclarecedoras, causando 35% de abandono.

Qual abordagem melhora melhor o trade-off?

- A) Prosseguir com defaults ocultos.
- B) Fazer todas as perguntas esclarecedoras em uma única mensagem composta.
- C) Declarar suposições explicitamente e prosseguir convidando correções. **[CORRETA]**
- D) Usar um formulário estruturado de coleta.

Por que C: Declarar suposições explicitamente e prosseguir dá ao usuário uma resposta imediata e útil enquanto preserva sua capacidade de corrigir suposições erradas. Defaults ocultos (A) deixam o usuário sem saber o que foi assumido. Uma lista composta de perguntas (B) ainda exige esforço inicial do usuário. Um formulário estruturado (D) adiciona mais atrito, não menos — contradiz o objetivo de reduzir abandono.

Questão 75 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Seu assistente usa um system prompt de persona "empreiteiro". Os primeiros turnos seguem as regras, mas no turno 7 o assistente passa a dar conselhos genéricos. O comprimento da conversa é de apenas 2.500 tokens.

Qual a causa mais provável?

- A) System prompts apenas estabelecem comportamento inicial.
- B) A atenção do modelo enfraquece à medida que turnos se acumulam.
- C) Respostas acumuladas do assistente diluem a influência do system prompt. **[CORRETA]**

- D) O system prompt é enviado apenas uma vez.

Por que C: Conforme respostas do assistente se acumulam no histórico, a proporção de texto refletindo as restrições comportamentais do system prompt diminui em relação ao volume crescente de conteúdo gerado pelo assistente. O modelo cada vez mais casa padrão com suas próprias saídas anteriores em vez do system prompt, agravando deriva mesmo em comprimentos curtos de tokens. O system prompt é incluído em toda chamada de API (D é falso isoladamente) e degradação de atenção (B) não opera em 2.500 tokens.

Questão 76 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Usuários fazem pedidos vagos como "Pode ajudar com o relatório?". O assistente responde fazendo várias perguntas (qual relatório? que ajuda? prazo?), causando 40% de abandono.

Qual a melhor solução?

- A) Fazer suposições razoáveis, declará-las explicitamente e oferecer ajustes. **[CORRETA]**
- B) Classificar a ambiguidade com um modelo menor antes de responder.
- C) Usar interpretações predefinidas sem declarar suposições.
- D) Limitar o assistente a uma pergunta esclarecedora por turno.

Por que A: Prosseguir com suposições razoáveis declaradas elimina totalmente o vai-e-vem mantendo o usuário informado e no controle. Interpretações silenciosas predefinidas (C) deixam usuários confusos quando a resposta não casa com sua intenção. Um limite de uma pergunta (D) ainda exige turnos de vai-e-vem. Um modelo menor de classificação (B) adiciona latência e complexidade de infraestrutura sem resolver o problema central de UX.

Exercícios Práticos

Exercício 1: Agente Multi-ferramenta com Lógica de Escalonamento

Objetivo: Projetar um loop de agente com integração de ferramentas, tratamento estruturado de erros e escalonamento.

Passos:

1. Definir 3–4 ferramentas MCP com descrições detalhadas (incluir duas ferramentas similares para testar seleção)
2. Implementar um loop de agente verificando `stop_reason` (`"tool_use"` / `"end_turn"`)
3. Adicionar respostas de erro estruturadas: `errorCategory`, `isRetryable`, descrição
4. Implementar um hook interceptador que bloqueia operações acima de um limite e roteia para escalonamento

5. Testar com pedidos multi-aspecto

Domínios: 1 (Arquitetura de agente), 2 (Ferramentas e MCP), 5 (Contexto e confiabilidade)

Exercício 2: Configurando Claude Code para Desenvolvimento em Equipe

Objetivo: Configurar CLAUDE.md, comandos customizados, regras path-specific e servidores MCP.

Passos:

1. Criar um CLAUDE.md de nível projeto com padrões universais
2. Criar arquivos `.claude/rules/` com frontmatter YAML para áreas distintas (`paths: ["src/api/**/*"]`, `paths: ["**/*.test.*"]`)
3. Criar uma skill de projeto sob `.claude/skills/` com `context: fork` e `allowed-tools`
4. Configurar um servidor MCP em `.mcp.json` com variáveis de ambiente + um override pessoal em `~/.claude.json`
5. Testar modo de planejamento vs execução direta em tarefas de complexidades diferentes

Domínios: 3 (Configuração do Claude Code), 2 (Ferramentas e MCP)

Exercício 3: Pipeline de Extração de Dados Estruturados

Objetivo: JSON schemas, `tool_use` para saída estruturada, loops de validação/retry, processamento em lote.

Passos:

1. Definir uma ferramenta de extração com JSON schema (campos obrigatórios/opcionais, enums com "other", campos nuláveis)
2. Construir um loop de validação: em erro, retentar com o documento, a extração incorreta e o erro de validação específico
3. Adicionar exemplos few-shot para documentos com estruturas diferentes
4. Usar processamento em lote via Message Batches API: 100 documentos, tratar falhas via `custom_id`
5. Rotear para humanos: scores de confiança por campo, análise por tipo de documento

Domínios: 4 (Engenharia de prompts), 5 (Contexto e confiabilidade)

Exercício 4: Projetando e Depurando um Pipeline de Pesquisa Multi-agente

Objetivo: Orquestração de subagentes, passagem de contexto, propagação de erro, síntese com tracking de fontes.

Passos:

1. Um coordenador com 2+ subagentes (`allowedTools` inclui `"Task"`, contexto passado explicitamente nos prompts)
2. Rodar subagentes em paralelo via múltiplas chamadas `Task` em uma única resposta
3. Exigir saída estruturada do subagente: afirmação, citação, URL da fonte, data de publicação
4. Simular timeout de subagente: retornar contexto de erro estruturado ao coordenador e continuar com resultados parciais
5. Testar com dados conflitantes: preservar ambos os valores com atribuição; separar achados confirmados vs disputados

Domínios: 1 (Arquitetura de agente), 2 (Ferramentas e MCP), 5 (Contexto e confiabilidade)

Apêndice: Tecnologias e Conceitos

Tecnologia	Aspectos-chave
Claude Agent SDK	AgentDefinition, loops de agente, `stop_reason`, hooks (PostToolUse), spawn de subagentes via Task, `allowedTools`
Model Context Protocol (MCP)	Servidores MCP, tools, resources, `isError`, descrições de ferramenta, `.mcp.json`, variáveis de ambiente
Claude Code	Hierarquia CLAUDE.md, `.claude/rules/` com padrões glob, `.claude/commands/`, `.claude/skills/` com SKILL.md, modo de planejamento, `/compact`, `--resume`, `fork_session`
Claude Code CLI	`-p` / `--print` para modo não-interativo, `--output-format json`, `--json-schema`
Claude API	`tool_use` com JSON schemas, `tool_choice` ("auto"/"any"/forçada), `stop_reason`, `max_tokens`, system prompts
Message Batches API	50% de economia, janela de até 24 horas, `custom_id`, sem multi-turn tool calling
JSON Schema	Required vs opcional, campos nuláveis, tipos enum, "other" + detalhe, modo estrito
Pydantic	Validação por schema, erros semânticos, loops de validação/retry
Ferramentas embutidas	Read, Write, Edit, Bash, Grep, Glob — propósito e critérios de seleção
Few-shot prompting	Exemplos direcionados para situações ambíguas, generalização para novos padrões
Prompt chaining	Decomposição sequencial em passagens focadas

Janela de contexto	Orçamentos de tokens, sumarização progressiva, "lost in the middle", arquivos de scratchpad
Gerenciamento de sessão	Resume, `fork_session`, sessões nomeadas, isolamento de contexto
Calibração de confiança	Scoring por campo, calibração em conjuntos rotulados, amostragem estratificada

---

Tópicos Fora do Escopo

Os seguintes tópicos adjacentes **NÃO** estarão no exame:

- Fine-tuning de modelos Claude ou treinamento de modelos customizados
- Autenticação, cobrança ou gerenciamento de conta da Claude API
- Implementação detalhada em linguagens ou frameworks específicos (além do necessário para configuração de tool/schema)
- Deploy ou hospedagem de servidores MCP (infraestrutura, redes, orquestração de containers)
- Arquitetura interna do Claude, processo de treinamento ou pesos do modelo
- Constitutional AI, RLHF ou metodologias de safety training
- Modelos de embedding ou detalhes de implementação de bases vetoriais
- Computer use (automação de browser, interação com desktop)
- Capacidades de análise de imagem (Vision)
- API de streaming ou server-sent events
- Rate limiting, quotas ou cálculos detalhados de custo de API
- OAuth, rotação de chaves de API ou detalhes de protocolos de autenticação
- Configurações específicas de provedor de nuvem (AWS, GCP, Azure)
- Benchmarks de performance ou métricas de comparação de modelos
- Detalhes de implementação de prompt caching (além de saber que existe)
- Algoritmos de contagem de tokens ou especificidades de tokenização

---

Recomendações de Preparação

1. **Construa um agente com o Claude Agent SDK** — implemente um loop completo de agente com chamada de ferramentas, tratamento de erros e gerenciamento de sessão. Pratique subagentes e passagem explícita de contexto.
2. **Configure Claude Code para um projeto real** — use a hierarquia CLAUDE.md, regras path-specific em `.claude/rules/`, skills com `context: fork` e `allowed-tools`, e integração de servidores MCP.
3. **Projete e teste ferramentas MCP** — escreva descrições que diferenciem ferramentas similares, retorne erros estruturados com categorias e flags de retry, e teste contra pedidos ambíguos do usuário.

4. **Construa um pipeline de extração de dados** — use `tool_use` com JSON schemas, loops de validação/retry, campos opcionais/nuláveis e processamento em lote via Message Batches API.
5. **Pratique engenharia de prompts** — adicione exemplos few-shot para cenários ambíguos, critérios explícitos de revisão e arquiteturas multi-passagem para reviews grandes de código.
6. **Estude padrões de gerenciamento de contexto** — extraia fatos de saídas verbosas, use arquivos de scratchpad e delegue descoberta a subagentes para lidar com limites de contexto.
7. **Entenda escalonamento e human-in-the-loop** — quando escalar (lacunas de política, pedido explícito do usuário, incapacidade de progredir) e fluxos de roteamento baseados em confiança.

8. **Faça um teste prático** antes do real. Ele usa os mesmos cenários e formato.