Certificação Claude Certified Architect — Foundations

Guia de Estudos (Baseado no Guia Oficial do Exame)

Introdução

A certificação **Claude Certified Architect — Foundations** confirma que um especialista é capaz de tomar decisões fundamentadas de arquitetura (*trade-offs*) ao implementar soluções reais baseadas no Claude. O exame avalia conhecimentos fundamentais sobre o Claude Code, o Claude Agent SDK, a API do Claude e o Model Context Protocol (MCP) — as tecnologias centrais para a criação de aplicações em produção com o Claude.

As questões do exame são baseadas em cenários reais da indústria: construção de sistemas agênticos para suporte ao cliente, design de pipelines de pesquisa multi-agente, integração do Claude Code em CI/CD, criação de ferramentas de produtividade para desenvolvedores e extração de dados estruturados a partir de documentos não estruturados.

Candidato-Alvo

O candidato ideal é um **arquiteto de soluções** (*solution architect*) que projeta e coloca em produção aplicações com o Claude. Você deve ter pelo menos 6 meses de experiência prática com:

- **Claude Agent SDK** — orquestração multi-agente, delegação para subagentes, integração de ferramentas (*tools*), *lifecycle hooks*
- **Claude Code** — CLAUDE.md, servidores MCP, Agent Skills, modo de planejamento (*planning mode*)
- **Model Context Protocol (MCP)** — ferramentas e recursos (*resources*) para integração de backend
- **Engenharia de prompts** — esquemas JSON, exemplos *few-shot*, templates de extração de dados
- **Janelas de contexto** — trabalho com documentos longos, passagem de contexto entre múltiplos agentes
- **Pipelines de CI/CD** — revisão automatizada de código, geração de testes
- **Escalação e confiabilidade** — tratamento de erros, intervenção humana (*human-in-the-loop*)

Formato do Exame

Parâmetro	Valor
Tipo de questão	Múltipla escolha (1 correta entre 4 opções)
Pontuação	Escala de 100 a 1000, nota de corte **720**

Penalidade por erro	Nenhuma (responda a todas as questões!)
Cenários	4 de 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 problemas de conta usando o Claude Agent SDK. O agente utiliza ferramentas MCP (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`). O objetivo é atingir mais de 80% de resolução no primeiro contato com escalação adequada.

Cenário 2: Geração de Código com o Claude Code

Você utiliza o Claude Code para acelerar o desenvolvimento: geração de código, refatoração, depuração, documentação. Você precisa integrá-lo com comandos *slash* customizados e configurações no CLAUDE.md, além de entender quando usar o modo de planejamento.

Cenário 3: Sistema de Pesquisa Multi-Agente

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

Cenário 4: Ferramentas de Produtividade para Desenvolvedores

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

Cenário 5: Claude Code para Integração Contínua (CI/CD)

Integre o Claude Code em um pipeline de CI/CD para revisões automatizadas de código, geração de testes e feedback em *pull requests*. Prompts devem ser projetados para minimizar falsos positivos.

Cenário 6: Extração de Dados Estruturados

O sistema extrai informações de documentos não estruturados, valida saídas com esquemas JSON e mantém alta precisão. Deve tratar corretamente casos de borda (*edge cases*).

Cenário 7: Padrões de Arquitetura de IA Conversacional

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

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

Este cenário foi relatado por candidatos do exame, mas ainda não está coberto neste guia. Se você encontrou questões deste cenário no exame real, por favor compartilhe nas GitHub Issues para que possamos adicionar cobertura completa. Sua contribuição ajudará todos que estão se preparando 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 — Visão Geral	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 — Subagentes	https://platform.claude.com/docs/en/agent-sdk/subagents
Claude Agent SDK — Sessões	https://platform.claude.com/docs/en/agent-sdk/sessions
Model Context Protocol (MCP)	https://modelcontextprotocol.io/
MCP — Ferramentas (Tools)	https://modelcontextprotocol.io/docs/concepts/tools
MCP — Recursos (Resources)	https://modelcontextprotocol.io/docs/concepts/resources
MCP — Servidores (Servers)	https://modelcontextprotocol.io/docs/concepts/servers
Claude Code — Documentação	https://code.claude.com/docs/en/overview

Claude Code — CLAUDE.md e Memória	https://code.claude.com/docs/en/memory
Claude Code — Skills (incl. comandos slash)	https://code.claude.com/docs/en/skills
Claude Code — Hooks	https://code.claude.com/docs/en/hooks
Claude Code — Sub-agentes	https://code.claude.com/docs/en/sub-agents
Claude Code — Integração com MCP	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
Pensamento Estendido (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 é organizado por tecnologias e conceitos em vez de domínios de exame — isso ajuda a construir uma compreensão mais profunda de cada tópico.

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

> Documentação: Messages API | Prompt Engineering

1.1 Estrutura de Requisição da API

A API do Claude segue um modelo de requisição-resposta. Cada requisição para a Messages API do Claude inclui:

```
{
  "model": "claude-sonnet-4-6",
  "max_tokens": 1024,
  "system": "Você é um assistente prestativo.",
  "messages": [
    {"role": "user", "content": "Olá!"},
    {"role": "assistant", "content": "Olá! Como posso ajudar?"},
    {"role": "user", "content": "Como você está?"}
  ],
  "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 prompt do sistema (define o comportamento do modelo)
- `messages` — histórico da conversa (**você deve enviar o histórico completo** para manter a coerência)
- `tools` — definições das ferramentas disponíveis
- `tool_choice` — estratégia de seleção de ferramentas

1.2 Papéis de Mensagem (Roles)

O array `messages` utiliza dois papéis conversacionais mais um papel instrucional:

- `user` — mensagens do usuário, incluindo resultados de ferramentas (enviados como um bloco de conteúdo `tool_result` dentro de uma mensagem com papel `user`, e não como um papel `tool` separado)
- `assistant` — respostas do modelo (incluídas ao enviar o histórico), incluindo requisições de uso de ferramentas (blocos de conteúdo `tool_use`)
- `system` — pode ser definido via campo de nível superior `system` (aplica-se a partir do primeiro turno) ou *inline* em `messages` como `{"role": "system", ...}` (aplica-se a partir daquele ponto em diante, sujeito a regras de posicionamento — veja abaixo)

Resultados de ferramentas não são enviados sob o papel `"tool"`. Eles são enviados em uma mensagem de papel `user` cujo conteúdo inclui um bloco `tool_result`:

```
{
  "role": "user",
  "content": [
    {
      "type": "tool_result",
      "tool_use_id": "toolu_01...",
      "content": "..."
    }
  ]
}
```

O papel `system` também pode aparecer diretamente no array `messages`, e não apenas pelo parâmetro `system` no nível superior. Isso serve para adicionar instruções no meio da conversa sem invalidar o prefixo em cache do campo `system` superior. Há regras de posicionamento específicas:

- Deve seguir imediatamente um turno de `user` (incluindo turnos com blocos `tool_result`) ou um turno de `assistant` que termina em uso de ferramenta no servidor.
- Deve anteceder um turno de `assistant` ou encerrar o array.
- Não pode ficar entre um bloco `tool_use` e seu respectivo `tool_result` — fazer isso retorna um erro 400.
- Mensagens `system` posteriores (mesmo no meio da conversa) têm precedência sobre mensagens anteriores e sobre o campo `system` superior para os turnos seguintes.

Criticamente importante: em toda requisição à API, você deve enviar 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 API do Claude inclui o campo `stop_reason`, que indica o motivo pelo qual o modelo parou de gerar texto:

Valor	Descrição	Ação
`"end_turn"`	O modelo finalizou sua resposta	Exibe o resultado ao usuário
`"tool_use"`	O modelo deseja chamar uma ferramenta	Executa a ferramenta e retorna 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	Trata de acordo com 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 (Prompt do Sistema)

O prompt do sistema é uma instrução especial que define o contexto e as regras comportamentais. 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 aplica-se a toda a conversa
- É usado para definir papéis, restrições e formatos de saída

Importante para o exame: a redação do system prompt pode criar associações não intencionais com ferramentas. Por exemplo, uma instrução como "sempre verifique o cliente" pode fazer com que o modelo use excessivamente `get_customer`, mesmo quando desnecessário.

1.5 Janela de Contexto (Context Window)

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

- O system prompt
- O histórico completo de mensagens
- As definições de ferramentas
- Os resultados de chamadas de ferramentas

Principais problemas de janela de contexto:

1. **Efeito "Lost in the Middle" (Perdido no Meio):** os modelos processam de forma confiável as informações no início e no final de uma entrada longa, mas podem perder detalhes no meio. Mitigação: coloque as informações cruciais perto do início ou do fim.
2. **Acúmulo de resultados de ferramentas:** cada chamada de ferramenta adiciona dados ao contexto. Se uma ferramenta retorna 40+ campos, mas apenas 5 importam, a maior parte do contexto é desperdiçada.
3. **Sumarização progressiva:** ao comprimir o histórico, valores numéricos, porcentagens e datas frequentemente se perdem ou se tornam vagos ("cerca de", "aproximadamente", "alguns").

Capítulo 2: Ferramentas e `tool_use`

> Documentação: Tool Use

2.1 O que é `tool_use`

`tool_use` é um 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 esquema JSON (JSON schema):

```
{
  "name": "get_customer",
  "description": "Localiza um cliente por email ou ID. Retorna o perfil do cliente,
incluindo nome, email, histórico de pedidos e status da conta. Use esta ferramenta
ANTES de lookup_order para verificar a identidade do cliente. Aceita um email
(formato: usuario@dominio.com) ou um customer_id numérico.",
  "input_schema": {
    "type": "object",
    "properties": {
      "email": {"type": "string", "description": "Email do cliente"},
      "customer_id": {"type": "integer", "description": "ID numérico do cliente"}
    },
    "required": []
  }
}
```

Aspectos criticamente importantes da descrição de uma ferramenta:

1. **A descrição é o mecanismo primário de seleção.** Um LLM escolhe ferramentas com base em suas descrições. Descrições mínimas ("Recupera informações do cliente") levam a erros quando há ferramentas sobrepostas.
2. **Inclua na descrição:**
   - O que a ferramenta faz e o que ela retorna
   - Formatos de entrada e valores de exemplo
   - Casos de borda e restrições
   - Quando usar esta ferramenta em relação a alternativas semelhantes
3. **Evite** descrições idênticas ou sobrepostas entre ferramentas. Se `analyze_content` e `analyze_document` possuem descrições quase idênticas, o modelo irá confundificá-las.
4. **Ferramentas nativas (*built-in*) vs ferramentas MCP:** os agentes podem preferir ferramentas nativas (Read, Grep) em vez de ferramentas MCP com funcionalidade similar. Para evitar isso, fortaleça as descrições das ferramentas MCP — destaque vantagens concretas, dados únicos ou contexto que as ferramentas nativas não conseguem fornecer.

2.3 O Parâmetro `tool_choice`

O `tool_choice` controla como o modelo seleciona as 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 uma saída estruturada
`{"type": "tool", "name": "extract_metadata"}`	O modelo **deve** chamar uma ferramenta específica	Quando você precisa forçar um primeiro passo / ordem de execução

Cenários importantes:

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

2.4 Esquemas JSON para Saída Estruturada

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

- Garante JSON sintaticamente válido (sem chaves faltando, sem vírgulas sobrando)
- Impõe a estrutura exigida (campos obrigatórios presentes)
- **Não** garante correção semântica (os valores ainda podem estar incorretos)

Design de esquema — princípios chave:

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": ["bug", "feature", "docs", "unclear", "other"]
    },
    "category_detail": {
      "type": ["string", "null"],
      "description": "Detalhes se category = 'other' ou 'unclear'"
    },
    "severity": {
      "type": "string",
      "enum": ["critical", "high", "medium", "low"]
    },
    "confidence": {
      "type": "number",
      "minimum": 0,
      "maximum": 1
    },
    "optional_field": {
      "type": ["string", "null"],
      "description": "Null se a informação não foi encontrada na fonte"
    }
  },
  "required": ["category", "severity"]
}
```

Regras de design de esquema:

1. **Obrigatório vs Opcional:** marque campos como obrigatórios (`required`) apenas se a informação estiver sempre disponível. Campos obrigatórios forçam o modelo a inventar valores quando os dados estão ausentes.
2. **Campos anuláveis (*nullable*):** use `"type": ["string", "null"]` para informações que possam estar ausentes. O modelo pode retornar `null` em vez de alucinar.
3. **Enums com `"other"`:** adicione `"other"` + uma string de detalhamento para evitar a perda de dados fora de suas categorias pré-definidas.
4. **Enum `"unclear"`:** para casos onde o modelo não consegue escolher com certeza uma categoria — um `"unclear"` honesto é melhor que uma categoria errada.

2.5 Erros Sintáticos vs Erros Semânticos

Tipo de erro	Exemplo	Mitigação
Sintático	JSON inválido, tipo de campo errado	`tool_use` com esquema JSON (elimina totalmente)
Semântico	Totais não batem, valor no campo errado, alucinação	Validações programáticas, retentativa 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 (Agentic Loop)

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

```
1. Envia uma requisição ao Claude com as ferramentas disponíveis
2. Recebe a resposta
3. Verifica o stop_reason:
   - "tool_use" -> executa a ferramenta, anexa o resultado ao histórico, volta ao passo
1
   - "end_turn" -> a tarefa está concluída, exibe o resultado ao usuário
4. Repete até a conclusão
```

Abordagem dirigida pelo modelo: o Claude decide qual ferramenta chamar a seguir com base no contexto e nos resultados anteriores. Isso difere de árvores de decisão codificadas rigidamente, onde a sequência de ações é fixa.

Anti-padrões (evite):

- Analisar 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 de conclusão confiável é `stop_reason == "end_turn"`.

3.2 Configuração com `AgentDefinition`

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

```
agent = AgentDefinition(
    name="customer_support",
    description="Lida com requisições de clientes sobre devoluções e problemas em
pedidos",
    system_prompt="Você é um agente de suporte ao cliente...",
    allowed_tools=["get_customer", "lookup_order", "process_refund",
"escalate_to_human"],
    # Para um coordenador:
    # allowed_tools=["Task", "get_customer", ...]
)
```

Parâmetros chave:

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

3.3 Hub-and-Spoke: Coordenador e Subagentes

Uma arquitetura multi-agente é tipicamente construída como uma topologia *hub-and-spoke*:

```
        Coordenador
      /      |      \
Subagente1 Subagente2 Subagente3
(pesquisa)  (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 os resultados
- Tratar erros e re-tentativas
- Comunicar os resultados ao usuário

Princípio crítico: subagentes têm contexto isolado.

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

3.4 A Ferramenta `Task` para Criar Subagentes

Subagentes são disparados via ferramenta `Task`:

```
# O allowedTools do coordenador deve incluir "Task"
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

A passagem explícita de contexto é obrigatória:

```
# Ruim: o subagente não tem contexto
Task: "Analise o documento"

# Bom: contexto completo no prompt
Task: "Analise o seguinte documento.
Documento: [texto completo do documento]
Resultados de pesquisa anteriores: [resultados de busca web]
Requisitos de formato de saída: [esquema]"
```

Disparo em paralelo: um coordenador pode chamar múltiplas `Task`s em uma única resposta — os subagentes executam em paralelo:

```
# Uma resposta do coordenador contém:
Task 1: "Pesquise artigos sobre X"
Task 2: "Analise o documento Y"
Task 3: "Pesquise artigos sobre Z"
# Todas as três executam concorrentemente
```

3.5 Hooks no Agent SDK

Hooks permitem a interceptação e transformação em pontos específicos do ciclo de vida do agente.

PostToolUse intercepta o resultado de uma ferramenta antes que ele seja entregue ao modelo:

```python
# Exemplo: normalizar formatos de data vindos de diferentes ferramentas MCP
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Converte timestamp Unix -> ISO 8601
    # Converte "5 de Março de 2025" -> "2025-03-05"
    return normalized_result
```

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

```python
# Exemplo: bloquear reembolsos acima de R$ 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 de prompt

Atributo	Hooks	Instruções de Prompt
Garantia	**Determinístico** (100%)	**Probabilístico** (>90%, não 100%)
Quando usar	Regras de negócio críticas, operações financeiras, conformidade	Preferências gerais, recomendações, formatação

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

Regra: quando a falha traz consequências financeiras, legais ou de segurança — use hooks, e 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. **Ferramentas (Tools)** — funções que o agente pode chamar para realizar ações (operações CRUD, chamadas de API, execução de comandos)
2. **Recursos (Resources)** — dados que o agente pode ler para obter contexto (documentação, esquemas de banco de dados, catálogos de conteúdo)
3. **Prompts** — templates de prompt pré-definidos para tarefas comuns

4.2 Servidores MCP

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

- Todas as ferramentas são descobertas automaticamente
- Ferramentas de todos os servidores conectados ficam disponíveis simultaneamente
- As descrições das ferramentas determinam como o modelo as utilizará

4.3 Configurando Servidores MCP

Configuração do projeto (`.mcp.json`) — para uso da 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:

- O arquivo `.mcp.json` é armazenado na raiz do projeto e mantido no controle de versão (VCS)
- Variáveis de ambiente (`${GITHUB_TOKEN}`) são usadas para segredos — os tokens em si não são commitados
- Disponível para todos os contribuidores do projeto

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

- Armazenado no diretório *home* do usuário
- Não é compartilhado via controle de versão
- Adequado para experimentos individuais e testes

Escolhendo servidores:

- Para integrações padrão (Jira, GitHub, Slack), prefira servidores MCP comunitários existentes
- Crie servidores próprios apenas para fluxos de trabalho únicos da sua equipe

4.4 A Flag `isError` no MCP

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

Erro estruturado (bom):

```
{
  "isError": true,
  "content": {
    "errorCategory": "transient",
    "isRetryable": true,
    "message": "O serviço está temporariamente indisponível. Timeout ao chamar a API
de pedidos.",
    "attempted_query": "order_id=12345",
    "partial_results": null
  }
}
```

Erro genérico (anti-padrão):

```
{
  "isError": true,
  "content": "Falha na operação"
}
```

Um erro genérico não dá informações ao agente para tomar decisões — ele deve tentar novamente, alterar a busca ou escalar?

4.5 Recursos MCP (Resources)

Recursos são dados que um agente pode solicitar para obter contexto sem realizar ações ativas:

- Catálogos de conteúdo (ex: lista de todas as tarefas do projeto, navegação hierárquica)
- Esquemas de banco de dados (compreensão da estrutura de dados)
- Documentação (referências de API, guias internos)
- Resumos de problemas/tarefas

Vantagem dos recursos: o agente não precisa fazer chamadas exploratórias de ferramentas para entender quais dados existem. Um recurso 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 do CLAUDE.md

O CLAUDE.md é o arquivo de instruções do Claude Code. Existe uma hierarquia de três níveis:

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

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

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

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

5.2 Sintaxe `@path` (Importação de Arquivos)

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

```
# CLAUDE.md do Projeto

Os padrões de código estão descritos em @./standards/coding-style.md
Os requisitos de testes estão em @./standards/testing-requirements.md
A visão geral do projeto está em @README.md e as dependências em @package.json
```

Regras para `@path`:

- Use `@` imediatamente antes do caminho do arquivo (sem espaço)
- Caminhos relativos e absolutos são suportados
- Caminhos relativos são resolvidos a partir do arquivo que contém a importação
- Profundidade máxima de aninhamento de importação é 5

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

5.3 O Diretório `.claude/rules/`

O diretório `.claude/rules/` é uma alternativa ao 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 implantação
  react-patterns.md   -- padrões React
```

Recurso chave: Frontmatter YAML com `paths` para carregamento condicional:

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

Para arquivos de API, use async/await com tratamento explícito de erros.
Cada endpoint deve retornar um wrapper padrão de resposta.
```

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

Os testes devem usar blocos describe/it.
Use data factories em vez de valores hardcoded.
Não faça mock do banco de dados — use um banco de testes.
```

Como funciona:

- Uma regra é carregada **apenas** quando o Claude Code edita um arquivo correspondente ao 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, migrações)
- CLAUDE.md em nível de diretório — quando convenções estão presas a um diretório específico e não são necessárias em outros lugares

5.4 Comandos Slash Customizados e Skills

> **Nota:** na versão atual do Claude Code, comandos customizados (`.claude/commands/`) foram unificados com skills (`.claude/skills/`). Ambos criam comandos `/nome`. O guia oficial do exame referencia `.claude/commands/` — esse formato continua sendo suportado.

Comandos *slash* são templates de prompt reutilizáveis invocados via `/nome`:

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

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

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

```
.claude/skills/
  review/SKILL.md  -- /review -- com configuração no frontmatter
  test-gen/SKILL.md
```

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

- Armazenados no VCS e disponíveis para todos ao clonar o repositório
- Garantem fluxos de trabalho consistentes na equipe

Comandos do 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 no SKILL.md:

```
---
context: fork
allowed-tools: ["Read", "Grep", "Glob"]
argument-hint: "Caminho para o diretório a ser analisado"
---

Analise a estrutura de código no diretório especificado.
Gere um relatório sobre dependências e padrões arquiteturais.
```

Parâmetros do frontmatter:

Parâmetro	Descrição
`context: fork`	Executa a skill em um subagente isolado. Saídas detalhadas não poluem a sessão principal
`allowed-tools`	Restringe quais ferramentas estão disponíveis (segurança — ex: a skill não pode deletar arquivos se não permitido)
`argument-hint`	Dica que solicita um argumento ao invocar sem parâmetros

Quando usar uma Skill vs CLAUDE.md:

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

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

- Crie variantes pessoais sob nomes diferentes para não afetar colegas de equipe

5.6 Modo de Planejamento vs Execução Direta

Modo de planejamento (*Planning mode*):

- O modelo apenas investiga e planeja; não faz alterações no código
- 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:

- Grandes alterações (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 alterar)
- Migrações de bibliotecas afetando 45+ arquivos

Quando usar execução direta:

- Correções em arquivo único com um stack trace claro
- Adição de uma validação pontual
- Alterações bem compreendidas e inequívocas

Abordagem combinada:

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

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

- Isola a saída detalhada do contexto principal
- Retorna apenas um resumo
- Evita o esgotamento da janela de contexto em tarefas de múltiplas fases

5.7 O Comando `/compact`

`/compact` é um comando nativo para compressão de contexto:

- Sumariza o histórico anterior para liberar a janela de contexto
- Usado em longas sessões de investigação quando o contexto enche com saídas detalhadas de ferramentas
- Risco: valores numéricos exatos, datas e detalhes específicos podem ser perdidos durante a sumarização

5.8 O Comando `/memory`

`/memory` é um comando nativo para gerenciamento de memória entre sessões:

- Abre o arquivo `CLAUDE.md` para edição, permitindo salvar notas, preferências e contexto
- As informações persistem entre sessões e são carregadas automaticamente na inicialização
- Útil para armazenar convenções de projeto, preferências do usuário, comandos frequentes e contexto de trabalho atual
- Alternativa a re-explicar as mesmas instruções a cada sessão

5.9 CLI do Claude Code para CI/CD

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

```
claude -p "Analise este pull request em busca de problemas de segurança"
```

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

Saída estruturada para CI:

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

- `--output-format json` — saída em formato JSON
- `--json-schema` — valida a saída contra um esquema
- O resultado pode ser analisado para publicar automaticamente comentários *inline* no PR

Isolamento de contexto de sessão: A mesma sessão do Claude que gerou o código é frequentemente menos eficaz em revisá-lo (o modelo retém o contexto do seu raciocínio e é menos inclinado a desafiar as próprias decisões). Use uma instância independente para revisão.

Prevenção de comentários duplicados: Ao re-revisar após novos commits, inclua os resultados da revisão anterior no contexto e instrua o Claude a relatar apenas problemas novos ou não resolvidos.

5.10 `fork_session` e Gerenciamento de Sessão

`--resume <nome-da-sessao>` retoma uma sessão nomeada:

```
claude --resume investigacao-bug-auth
```

- Continua uma conversa anterior com o contexto salvo

- Útil para investigações longas divididas em várias sessões
- Risco: se os arquivos mudaram desde a sessão anterior, os resultados das ferramentas podem estar desatualizados

`fork_session` cria uma ramificação independente a partir do contexto compartilhado:

```
Investigação da base de código
              |
         fork_session
        /            \
Abordagem A:      Abordagem B:
   Redux          Context API
```

- Ambos os forks herdam o contexto até o ponto de ramificação
- Depois disso, eles divergem independentemente
- Útil para comparar abordagens ou testar estratégias

Quando iniciar uma nova sessão em vez de retomar:

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

---

Capítulo 6: Engenharia de Prompts — Técnicas Avançadas

> Documentação: Prompt Engineering | Anthropic Cookbook

6.1 Prompting Few-shot

Prompting *few-shot* é a inclusão de 2 a 4 exemplos de entrada/saída em um prompt para demonstrar o comportamento esperado.

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

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

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

1. **Exemplos para cenários ambíguos:**

```
Solicitação: "Meu pedido veio com defeito"
Ação: Chamar get_customer -> lookup_order -> verificar status.
Justificativa: "defeito" significa um item danificado; você precisa dos detalhes do
pedido.

Solicitação: "Fale com um gerente"
Ação: Chamar imediatamente escalate_to_human.
Justificativa: O cliente solicita explicitamente um humano. Não tente resolver
autonomamente.
```

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

```
Exemplo de apontamento:
{
  "location": "src/auth/login.ts:42",
  "issue": "Injeção de SQL no parâmetro username",
  "severity": "critical",
  "suggested_fix": "Use uma query parametrizada"
}
```

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

```
// Aceitável (não apontar):
const items = data.filter(x => x.active);

// Problema (apontar):
const items = data.filter(x => x.active == true); // Use igualdade estrita ===
```

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

```
Documento com citações inline:
"Como mostrado no estudo (Smith, 2023), a taxa é de 42%."
-> {"value": "42%", "source": "Smith, 2023", "type": "inline_citation"}

Documento com referências bibliográficas:
"A taxa é de 42%. [1]"
-> {"value": "42%", "source": "reference_1", "type": "bibliography"}
```

5. **Exemplos para medições informais:**

```
Texto: "cerca de dois punhados de arroz"
-> {"amount": "~100g", "original_text": "dois punhados", "precision": "approximate"}

Texto: "uma pitada de sal"
-> {"amount": "~1g", "original_text": "uma pitada", "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 esquemas JSON estritos para saída estruturada, adicione regras de normalização no prompt:

```
Normalização:
- Datas: sempre ISO 8601 (AAAA-MM-DD); "ontem" -> calcular uma data absoluta
- Moeda: valor numérico + código da moeda; "cinco contos" -> {"amount": 5, "currency":
"BRL"}
- Porcentagens: fração decimal; "metade" -> 0.5
```

Isso previne erros semânticos onde o JSON é sintaticamente válido, mas os valores são inconsistentes.

6.2 Critérios Explícitos vs Instruções Vagas

Ruim (vago):

```
Verifique a precisão dos comentários do código.
Seja conservador — relata apenas descobertas de alta confiança.
```

Bom (critérios explícitos):

```
Aponte um comentário como problemático APENAS se:
1. O comentário descreve um comportamento que CONTRADIZ o comportamento real do código
2. O comentário referencia uma função ou variável inexistente
3. Um comentário TODO/FIXME refere-se a um bug que já foi corrigido no código

NÃO aponte:
- Comentários que estão apenas estilisticamente desatualizados
- Comentários com pequenas imprecisões de redação
- Comentários ausentes (essa é uma categoria separada)
```

Defina critérios de severidade com exemplos:

```
CRITICAL: Falha em tempo de execução para os usuários
  Exemplo: NullPointerException ao processar um pagamento

HIGH: Vulnerabilidade de segurança
  Exemplo: SQL injection, XSS, falta de checagens de autorização

MEDIUM: Bug de lógica sem impacto imediato
  Exemplo: Ordenação errada, erro off-by-one

LOW: Qualidade de código
  Exemplo: Duplicação, algoritmo subótimo para dados pequenos
```

6.3 Encadeamento de Prompts (Prompt Chaining)

O encadeamento de prompts divide uma tarefa complexa em uma sequência de passos focados:

```
Passo 1: Analisar auth.ts (apenas problemas locais)
        -> Saída: lista de problemas em auth.ts

Passo 2: Analisar database.ts (apenas problemas locais)
        -> Saída: lista de problemas em database.ts

Passo 3: Passo de integração (dependências entre arquivos)
        -> Saída: problemas nas fronteiras dos módulos
```

Por que isso importa:

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

Quando usar encadeamento de prompts vs decomposição dinâmica:

- **Encadeamento de prompts** — tarefas previsíveis e repetíveis (revisão de código, migração de arquivos)
- **Decomposição dinâmica** — investigações abertas onde as subtarefas se tornam claras apenas durante a execução

6.4 O Padrão "Entrevista" (Interview Pattern)

Antes de implementar uma solução, o Claude faz perguntas de esclarecimento:

```
Claude: "Antes de implementar o cache para a API, algumas perguntas:
1. Qual estratégia de invalidação de cache você prefere — TTL ou baseada em eventos?
2. Dados desatualizados (stale) são aceitáveis quando o cache está indisponível?
3. O cache deve ser por usuário ou global?
4. Qual é o volume de dados esperado para o 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 onde a melhor escolha depende do contexto

6.5 Validação e Retentativa com Feedback (Retry-with-Feedback)

Quando os dados extraídos falham na validação:

```
Passo 1: Extrair dados do documento
Passo 2: Validar (Pydantic, JSON Schema, regras de negócio)
Passo 3: Se houver erro — tentar novamente com contexto:
  - O documento original
  - A extração anterior (incorreta)
  - O erro específico: "Campo 'total' = 150, mas a soma dos itens = 145. Re-verifique
os valores."
```

Quando a retentativa será eficaz:

- Erros de formato (data no formato errado)
- Erros estruturais (um campo colocado no local errado)
- Inconsistências aritméticas (o modelo pode re-verificar)

Quando a retentativa NÃO ajudará:

- A informação está ausente no documento de origem
- O contexto necessário é externo (os dados estão em outro documento não fornecido)

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

- **Validação estrutural:** tipos, obrigatoriedade, restrições de enum checadas no código após receber o JSON do Claude
- **Validação semântica:** validadores customizados aplicam regras de negócio (soma dos itens igual ao total; data_inicio < data_fim)
- **Loops de validação-retentativa:** na falha de validação do Pydantic, construa uma mensagem de erro e peça novamente ao Claude fornecendo o contexto do erro
- **Geração de JSON Schema:** modelos Pydantic podem gerar JSON Schema para `tool_use`, fornecendo uma única fonte de verdade

6.6 Autocorreção (Self-correction)

Um padrão para detectar contradições internas:

```json
{
  "stated_total": "R$ 150,00",
  "calculated_total": "R$ 145,00",
  "conflict_detected": true,
  "line_items": [
    {"name": "Item A", "price": 75.00},
    {"name": "Item B", "price": 70.00}
  ]
}
```

O modelo extrai tanto o valor declarado quanto o valor calculado — se divergirem, `conflict_detected` permite que você trate a discrepância.

Capítulo 7: API de Message Batches (Lotes de Mensagens)

> Documentação: Message Batches

7.1 Visão Geral

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

Atributo	Valor
Economia	**50%** em relação às chamadas síncronas
Janela de processamento	Até **24 horas** (sem garantia de SLA de latência)
Chamadas de ferramentas multi-turnos	**Não suportado** (uma requisição = uma resposta)
Correlação	Campo `custom_id` para vincular requisição e resposta

7.2 Quando usar a Batch API vs API Síncrona

Tarefa	API	Por que
Checagem de PR pré-merge	**Síncrona**	O desenvolvedor está esperando; 24 horas é inaceitável
Relatório noturno de débito técnico	**Batch**	O resultado é necessário pela manhã; 50% de economia
Auditoria semanal de segurança	**Batch**	Não urgente; 50% de economia
Revisão de código interativa	**Síncrona**	Resposta imediata necessária
Processamento de 10.000 documentos	**Batch**	Processamento em lote; 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": "Extraia dados de: ..."}]
  }
}
```

O `custom_id` permite:

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

7.4 Tratamento de Falhas em Batches

1. Envie um lote de 100 documentos
2. 95 são bem-sucedidos; 5 falham (limite de contexto excedido)
3. Identifique as falhas pelo `custom_id`
4. Modifique a estratégia (ex: divida documentos longos em partes)
5. Re-envie apenas os 5 documentos que falharam

7.5 Planejamento de SLA

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

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

Capítulo 8: Estratégias de Decomposição de Tarefas

8.1 Pipelines Fixos (Encadeamento de Prompts)

Cada passo é definido com antecedência:

```
Documento -> Extração de metadados -> Extração de dados -> Validação -> Enriquecimento
-> Saída final
```

Quando usar:

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

8.2 Decomposição Dinâmica Adaptativa

As subtarefas são geradas com base em resultados intermediários:

```
1. "Adicione testes para uma base de código legada"
2. -> Primeiro: mapear a estrutura (Glob, Grep)
3. -> Encontrado: 3 módulos sem testes, 2 com cobertura parcial
4. -> Priorizar: começar pelo módulo de pagamentos (alto risco)
5. -> Durante o trabalho: descoberta dependência de uma API externa
6. -> Adaptar: criar um mock para a API externa antes de escrever os testes
```

Quando usar:

- Tarefas investigativas abertas
- Quando o escopo completo é desconhecido no início
- Quando cada passo depende dos resultados do passo anterior

8.3 Revisão de Código Multi-Passos (Multi-pass)

Para *pull requests* com 10+ arquivos:

```
Passo 1 (por arquivo): Analisar auth.ts -> listar problemas locais
Passo 1 (por arquivo): Analisar database.ts -> listar problemas locais
Passo 1 (por arquivo): Analisar routes.ts -> listar problemas locais
...
Passo 2 (integração): Analisar relacionamentos entre arquivos
  -> Problemas entre arquivos: tipos inconsistentes, dependências circulares
```

Por que uma única passagem sobre 14 arquivos é ruim:

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

---

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

9.1 Quando Escalar para um Humano

Gatilhos de escalação (regras claras):

Situação	Ação
O cliente pede explicitamente "fale com um gerente"	Escalar imediatamente; não tente resolver
A política da empresa não cobre a solicitação	Escalar (ex: cobrir preço do concorrente quando a política é omissa)
O agente não consegue progredir	Escalar após um número razoável de tentativas
Operação financeira acima de um limite	Escalar (preferencialmente via hook, não por prompt)
Múltiplas correspondências ao buscar um cliente	Pedir identificadores adicionais; não adivinhar

O que NÃO é um gatilho confiável:

Método não confiável	Por que falha
Análise de sentimento	O humor do cliente não correlaciona com a complexidade do caso
Confiança auto-avaliada do modelo (1–10)	O modelo pode estar confiantemente errado; má calibração
Um classificador automático	Complexidade excessiva; pode exigir dados de treino inexistentes

9.2 Padrões de Escalação

Escalação imediata:

```
Cliente: "Quero falar com um gerente"
Agente: [chama imediatamente escalate_to_human]
NÃO: "Eu posso ajudar com seu problema, me diga..."
```

Escalação após tentativa de resolução:

```
Cliente: "Minha geladeira quebrou dois dias após a compra"
Agente: [verifica o pedido, oferece troca na garantia]
Se o cliente não ficar satisfeito -> escalar
```

Escalação diferenciada (reconhecer → resolver → escalar na reiteração):

```
Cliente: "Isso é um absurdo, estou muito chateado com a qualidade!"
Agente: [reconhece a frustração] "Entendo perfeitamente sua frustração."
        [oferece solução] "Posso oferecer uma substituição ou o reembolso."
Cliente: "Não, eu quero falar com alguém!"
Agente: [cliente insiste novamente -> escalação imediata]
```

Princípio chave: reconheça a emoção primeiro, proponha uma solução concreta e apenas escale se o cliente reiterar o desejo de falar com um humano. Não escale no primeiro sinal de insatisfação (isso não é

o mesmo que pedir um gerente).

Escalação por lacuna na política:

```
Cliente: "O concorrente X vende isso 30% mais barato — me dê um desconto"
Política: cobre ajustes de preço apenas no próprio site
Agente: [escala — a política não cobre preços de concorrentes]
```

9.3 Protocolos de Transição Estruturada (Handoff)

Ao escalar, o agente deve passar um resumo estruturado para o atendente humano:

```
{
  "customer_id": "CUST-12345",
  "customer_name": "Ivan Petrov",
  "issue_summary": "Solicitação de reembolso por item danificado",
  "order_id": "ORD-67890",
  "root_cause": "Item chegou danificado; fotos anexadas",
  "actions_taken": [
    "Cliente verificado via get_customer",
    "Pedido confirmado via lookup_order",
    "Oferecida substituição padrão — cliente insiste em reembolso"
  ],
  "refund_amount": "R$ 89,99",
  "recommended_action": "Aprovar reembolso total",
  "escalation_reason": "Cliente solicitou falar com o gerente"
}
```

O operador humano não tem acesso a toda a transcrição da conversa — ele vê apenas este resumo. Portanto, ele deve ser completo e autossuficiente.

9.4 Calibração de Confiança e Supervisão Humana

Para sistemas de extração de dados:

1. **Pontuações de confiança em nível de campo:** o modelo gera uma pontuação de confiança para cada campo extraído
2. **Calibração:** use conjuntos de validação rotulados para ajustar os limiares (*thresholds*)
3. **Roteamento:**
   - Alta confiança + precisão estável -> processamento automatizado
   - Baixa confiança ou fontes ambíguas -> revisão humana

Aostragem aleatória estratificada (*Stratified random sampling*):

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

---

Capítulo 10: Tratamento de Erros em Sistemas Multi-Agente

10.1 Categorias de Erro

Categoria	Exemplos	Passível de retentativa	Ação do agente
Transitório	Timeout, 503, falha de rede	Sim	Tentar novamente com backoff exponencial
Validação	Formato de entrada inválido, campo obrigatório ausente	Não (corrigir entrada)	Modificar a requisição e tentar novamente
Negócio	Violação de política, limite excedido	Não	Explicar ao usuário; propor alternativa
Permissão	Acesso negado	Não	Escalar

10.2 Anti-padrões no Tratamento de Erros

Anti-padrão	Problema	Abordagem correta
Status genérico "busca indisponível"	O coordenador não consegue decidir como recuperar	Retornar tipo de erro, query, resultados parciais, alternativas
Supressão silenciosa (resultado vazio = sucesso)	O coordenador acha que não houve resultados, quando foi uma falha	Distinguir "sem resultados" de "falha na busca"
Abortar todo o fluxo em uma única falha	Perdem-se todos os resultados parciais	Continuar com resultados parciais; anotar as lacunas
Retentativas infinitas dentro do subagente	Latência e recursos desperdiçados	Recuperação local (1–2 retentativas), depois propagar ao coordenador

10.3 Erro Estruturado de Subagente

```
{
  "status": "partial_failure",
  "failure_type": "timeout",
  "attempted_query": "Impacto da IA na indústria da música 2024",
  "partial_results": [
    {"title": "Relatório sobre Geração de Música por IA", "url": "...", "relevance":
0.8}
  ],
  "alternative_approaches": [
    "Tentar uma busca mais específica: 'ferramentas de composição musical com IA'",
    "Usar uma fonte de dados alternativa"
  ],
  "coverage_impact": "Não coberto: impacto da IA na produção musical"
}
```

Isso fornece ao coordenador as informações necessárias para decidir:

- Tentar novamente com uma busca modificada?
- Usar os resultados parciais?
- Delegar a um subagente diferente?
- Continuar sem esta seção e anotar a lacuna?

10.4 Anotações de Cobertura na Síntese Final

```
## Relatório: Impacto da IA nas Indústrias Criativas

### Arte Visual (COBERTURA COMPLETA)
[resultados da pesquisa]

### Música (COBERTURA PARCIAL — timeout no agente de busca)
[resultados parciais]
⚠️ Nota: a cobertura desta seção é limitada devido a um timeout no agente de busca.

### Literatura (COBERTURA COMPLETA)
[resultados da pesquisa]
```

Capítulo 11: Gerenciamento de Contexto em Sistemas de Produção

11.1 Extrair Fatos para um Bloco Separado

Em vez de confiar no histórico da conversa (que se degrada durante a sumarização), extraia fatos chave para um bloco estruturado:

```
=== FATOS DO CASO (atualizado sempre que surge um novo fato) ===
ID do Cliente: CUST-12345
ID do Pedido: ORD-67890
Data do Pedido: 2025-01-15
Valor do Pedido: R$ 89,99
Problema: Item danificado na entrega
Solicitação do Cliente: Reembolso total
Status: Aguardando aprovação do gerente
===
```

Inclua este bloco em todo prompt, independentemente de como o histórico seja sumarizado.

11.2 Filtragem de Resultados de Ferramentas (*Trimming*)

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

```python
# Hook PostToolUse: mantém apenas campos relevantes
@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 economiza contexto e reduz ruídos.

11.3 Entrada Consciente da Posição (*Position-aware*)

Posicione informações críticas tendo em mente o efeito *lost-in-the-middle*:

```
[PRINCIPAIS DESCOBERTAS — no topo]
Encontradas 3 vulnerabilidades críticas...

[RESULTADOS DETALHADOS — no meio]
=== Arquivo auth.ts ===
...
=== Arquivo database.ts ===
...

[AÇÕES RECOMENDADAS — no final]
Prioridade: corrigir vulnerabilidades em auth.ts antes do merge.
```

11.4 Arquivos de Rascunho (*Scratchpad Files*)

Em investigações longas, o agente pode escrever descobertas principais em um arquivo de rascunho:

```
# investigacao-rascunho.md
## Principais descobertas
- PaymentProcessor em src/payments/processor.ts herda de BaseProcessor
- refund() é chamado em 3 lugares: OrderController, AdminPanel, CronJob
- API externa PaymentGateway possui rate limit de 100 req/min
- Migração #47 adicionou refund_reason (NOT NULL) — 01/12/2024
```

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

11.5 Delegar a Subagentes para Proteger o Contexto

```
Agente principal: "Investigue as dependências do módulo de pagamentos"
  -> Subagente (Explore): lê 15 arquivos, rastreia importações
  -> Retorna: "Pagamentos depende de AuthService, OrderModel e da API externa
PaymentGateway"

Agente principal: mantém uma linha no contexto em vez de 15 arquivos
```

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

Orçamentos de contexto restritos para subagentes:

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

11.6 Persistência de Estado Estruturado (para recuperação de falhas)

Cada agente exporta seu estado para um local conhecido:

```
// agent-state/web-search-agent.json
{
  "status": "completed",
  "queries_executed": ["IA música 2024", "composição musical IA"],
  "results_count": 12,
  "key_findings": [...],
  "coverage": ["composição musical", "produção musical"],
  "gaps": ["distribuição musical", "licenciamento de música"]
}
```

O coordenador carrega um manifesto ao retomar a execução:

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

Capítulo 12: Preservando a Proveniência dos Dados

12.1 O Problema da Perda de Atribuição

Ao sumarizar resultados de múltiplas fontes, a ligação "afirmação → fonte" pode se perder:

```
Ruim: "O mercado de música por IA é estimado em US$ 3,2 bi." (Sem fonte, sem ano.)

Bom:
{
  "claim": "O mercado de música por IA é estimado em US$ 3,2 bi.",
  "source_url": "https://exemplo.com/relatorio",
  "source_name": "Global AI Music Report 2024",
  "publication_date": "2024-06-15",
  "confidence": 0.9
}
```

12.2 Tratamento de Dados Conflitantes

Quando duas fontes fornecem valores diferentes:

```
{
  "claim": "Participação de música gerada por IA em plataformas de streaming",
  "values": [
    {
      "value": "12%",
      "source": "Relatório Anual Spotify 2024",
      "date": "2024-03",
      "methodology": "Classificação automatizada"
    },
    {
      "value": "8%",
      "source": "Pesquisa da Associação da Indústria da Música",
      "date": "2024-07",
      "methodology": "Pesquisa com 500 gravadoras"
    }
  ],
  "conflict_detected": true,
  "possible_explanation": "Diferença na metodologia e no período analisado"
}
```

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

12.3 Incluir Datas para Interpretação Correta

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

```
Ruim: "Fonte A diz 10%, fonte B diz 15%. Contradição."
Bom: "Fonte A (2023) diz 10%, fonte B (2024) diz 15%. Crescimento provável de +5% em um ano."
```

12.4 Renderizar por Tipo de Conteúdo

Não force tudo em um único formato:

- Dados financeiros -> tabelas
- Notícias e análises -> texto corrido (*prose*)
- Descobertas técnicas -> listas estruturadas
- Séries temporais -> ordem cronológica

Capítulo 13: Ferramentas Nativas (*Built-in*) do Claude Code

13.1 Referência de Seleção de Ferramentas

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

13.2 Estratégia de Investigação Incremental

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

```
1. Grep: encontrar pontos de entrada (definição de função, export)
2. Read: ler os arquivos encontrados
3. Grep: encontrar usos (imports, chamadas)
4. Read: ler arquivos consumidores
5. Repita até ter um panorama completo
```

13.3 Fallback: Read + Write em vez de Edit

Quando o Edit falha devido a correspondências de texto não únicas:

1. Read — carrega o conteúdo completo do arquivo
2. Modifica o conteúdo programaticamente
3. Write — escreve 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 de conversa para que o modelo decida a próxima ação
- Tomada de decisão dirigida pelo modelo (o Claude escolhe a próxima ferramenta) vs árvores de decisão codificadas rigidamente

Habilidades chave:

- Controle de fluxo: continuar o loop quando `stop_reason = "tool_use"` e parar em `"end_turn"`
- Anexar resultados de ferramentas ao contexto entre iterações
- Anti-padrões a evitar: analisar texto do assistente para detecção de 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 a comunicação entre 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 a cobertura da pesquisa entre subagentes para minimizar duplicações
- Implementar loops de refinamento iterativo (o coordenador avalia a síntese e redireciona tarefas)
- Rotear toda a comunicação através do coordenador para garantia de observabilidade

1.3 Configurando Chamadas de Subagentes, Passagem de Contexto e Disparo

Conhecimento chave:

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

Habilidades chave:

- Incluir saídas completas de agentes anteriores no prompt do subagente
- Usar formatos estruturados para separar dados de metadados ao passar contexto
- Disparar subagentes em paralelo via múltiplas chamadas `Task` em um único turno do coordenador
- Escrever prompts de coordenador focados em metas e critérios de qualidade, e não em instruções passo a passo

1.4 Implementando Fluxos de Trabalho Multi-passos com Regras de Aplicação e Padrões de Handoff

Conhecimento chave:

- A diferença entre **aplicação programática** (hooks, pré-condições) e **orientação por prompt** para ordenação de fluxos
- Quando você precisa de garantias determinísticas (ex: verificação de identidade antes de operações financeiras), prompts isolados são insuficientes
- Protocolos de handoff estruturados durante escalação (ID do cliente, motivo, ação recomendada)

Habilidades chave:

- Pré-condições programáticas que bloqueiam chamadas subsequentes até que os passos anteriores estejam concluídos (ex: bloquear `process_refund` até que `get_customer` retorne um ID verificado)
- Decompor solicitações de clientes com múltiplos aspectos em itens separados
- Produzir resumos estruturados ao escalar para um atendente humano

1.5 Hooks do Agent SDK para Interceptação de Chamadas e Normalização de Dados

Conhecimento chave:

- Padrões de hook (ex: `PostToolUse`) para interceptar resultados de ferramentas antes que o modelo os consuma

- Hooks que interceptam chamadas de saída para aplicar regras de conformidade (ex: bloquear reembolsos acima de um limite)
- Hooks fornecem **garantias determinísticas** vs instruções de prompt que fornecem **conformidade probabilística**

Habilidades chave:

- Hooks `PostToolUse` para normalização de formatos de dados (timestamps Unix, ISO 8601, códigos numéricos de status)
- Hooks de interceptação para bloquear ações que violam políticas com redirecionamento para escalação
- Escolher hooks em vez de prompts quando as regras de negócio exigem conformidade garantida

1.6 Estratégias de Decomposição de Tarefas para Fluxos Complexos

Conhecimento chave:

- **Pipelines fixos** (encadeamento de prompts) vs **decomposição dinâmica adaptativa** baseada em resultados intermediários
- Encadeamento de prompts: passos sequenciais (analisar cada arquivo separadamente, depois executar um passo de integração)
- Planos de investigação adaptativos que geram subtarefas baseadas no que foi descoberto

Habilidades chave:

- Usar encadeamento de prompts para revisões previsíveis de múltiplos aspectos; usar decomposição dinâmica para investigações abertas
- Dividir grandes revisões de código em análises por arquivo mais uma passagem de integração entre arquivos
- Decompor tarefas abertas: mapear a estrutura primeiro, depois criar um plano priorizado

1.7 Estado de Sessão, Retomada e Forking

Conhecimento chave:

- `--resume <nome-da-sessao>` para continuar sessões nomeadas
- `fork_session` para criar ramificações de investigação independentes a partir de um contexto compartilhado
- A importância de informar o agente sobre alterações em arquivos ao retomar sessões
- Uma nova sessão com um resumo estruturado pode ser mais confiável do que retomar com resultados obsoletos

Habilidades chave:

- Usar `--resume` para continuar sessões de investigação nomeadas

- Usar `fork_session` para comparar abordagens em paralelo
- Escolher entre retomar (contexto ainda atual) vs iniciar nova sessão (resultados obsoletos)

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

2.1 Projetando Interfaces de Ferramentas com Descrições Claras

Conhecimento chave:

- Descrições de ferramentas são o **mecanismo primário** que um LLM usa para selecionar ferramentas; descrições mínimas levam a seleções não confiáveis
- A importância de incluir formatos de entrada, exemplos de busca, casos de borda e limites de aplicabilidade
- Descrições ambíguas ou sobrepostas causam roteamento incorreto
- A redação do system prompt pode criar associações não intencionais com ferramentas

Habilidades chave:

- Escrever descrições que diferenciem claramente cada ferramenta de alternativas semelhantes
- Renomear ferramentas para eliminar sobreposição funcional (ex: `analyze_content` -> `extract_web_results`)
- Dividir ferramentas de propósito geral em ferramentas personalizadas 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 ("Falha na operação") impedem decisões corretas de recuperação
- A diferença entre erros passíveis e não passíveis de retentativa

Habilidades chave:

- Retornar metadados estruturados como `errorCategory` (transient/validation/permission), `isRetryable` e uma mensagem legível por humanos
- Usar `retryable: false` para violações de regras de negócio com explicações claras para o usuário
- Fazer recuperação local dentro de subagentes para falhas transitórias; propagar apenas erros que eles não conseguem resolver

- Distinguir falhas de acesso (decisão de retentativa) de resultados vazios válidos (sem correspondências)

2.3 Alocando Ferramentas entre Agentes e Configurando `tool_choice`

Conhecimento chave:

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

Habilidades chave:

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

2.4 Integrando Servidores MCP no Claude Code e nos Fluxos de Trabalho do Agente

Conhecimento chave:

- Escopo do servidor MCP: projeto (`.mcp.json`) para equipes vs usuário (`~/.claude.json`) para experimentos
- Substituição de variáveis de ambiente no `.mcp.json` (ex: `${GITHUB_TOKEN}`) para gerenciamento de segredos
- Ferramentas de todos os servidores MCP conectados são descobertas na conexão e ficam disponíveis simultaneamente
- Recursos MCP como "catálogos de conteúdo" (resumos de tarefas, esquemas de banco de dados) para reduzir chamadas exploratórias de ferramentas

Habilidades chave:

- Configurar servidores MCP compartilhados no `.mcp.json` do projeto com tokens baseados em variáveis de ambiente
- Manter servidores pessoais/experimentais em `~/.claude.json`
- Dar preferência a servidores MCP da comunidade em vez de servidores customizados para integrações padrão

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

Conhecimento chave:

- **Grep**: busca dentro do conteúdo dos arquivos (nomes de funções, mensagens de erro, importações)
- **Glob**: localiza arquivos por padrões de nome/extensão
- **Read/Write**: operações em arquivos completos; **Edit**: alterações precisas via correspondências únicas de texto
- Se o Edit falhar devido a correspondências não únicas, faça fallback para Read + Write

Habilidades chave:

- Usar Grep para busca de conteúdo e Glob para descoberta de arquivos por padrões
- Construir o entendimento incrementalmente: Grep em pontos de entrada, depois Read para rastrear fluxos
- Rastrear o uso de funções através de módulos wrappers

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

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

Conhecimento chave:

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

Habilidades chave:

- Diagnosticar problemas de hierarquia (um novo membro da equipe perde instruções porque estão em nível de usuário em vez de nível de projeto)
- Usar `@path` (ex: `@./standards/testing.md`) para incluir seletivamente padrões no CLAUDE.md de cada pacote
- Dividir um CLAUDE.md grande em múltiplos arquivos em `.claude/rules/` (testing.md, api-conventions.md, deployment.md)

3.2 Criando e Configurando Comandos Slash Customizados e Skills

Conhecimento chave:

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

Habilidades chave:

- Armazenar comandos slash de projeto em `.claude/commands/` para que toda a equipe os receba
- Usar `context: fork` para isolar skills com saída detalhada
- Usar `allowed-tools` para restringir quais ferramentas uma skill pode usar
- Usar `argument-hint` para solicitar parâmetros obrigatórios aos desenvolvedores

3.3 Usando Regras Específicas por Caminho para Carregamento Condicional de Convenções

Conhecimento chave:

- Arquivos em `.claude/rules/` podem incluir no frontmatter YAML o campo `paths` para ativar regras com base em padrões glob
- Regras delimitadas por caminho carregam **apenas** ao editar arquivos correspondentes, economizando contexto e tokens
- Regras de caminho baseadas em glob podem ser preferíveis ao CLAUDE.md em nível de diretório quando convenções se aplicam a muitos diretórios (ex: testes)

Habilidades chave:

- Criar arquivos em `.claude/rules/` com `paths: ["terraform/**/*"]` para carregar apenas ao trabalhar em arquivos correspondentes
- Usar padrões glob (`**/*.test.tsx`) para aplicar convenções por tipo de arquivo independentemente da localização
- Preferir regras específicas por caminho ao CLAUDE.md em nível de diretório quando convenções abrangem toda a base de código

3.4 Decidindo Quando Usar o Modo de Planejamento vs Execução Direta

Conhecimento chave:

- **Modo de planejamento**: para tarefas complexas com grandes alterações, múltiplas abordagens viáveis e decisões arquiteturais
- **Execução direta**: para alterações simples e bem compreendidas (ex: adicionar uma única validação)
- O modo de planejamento permite a exploração segura da base de código antes de fazer alterações
- O subagente Explore isola a saída de descoberta detalhada

Habilidades chave:

- Usar o modo de planejamento para tarefas com consequências arquiteturais (microsserviços, migrações tocando 45+ arquivos)
- Usar execução direta para correções com um stack trace claro em um único arquivo
- Usar o subagente Explore para evitar o esgotamento da janela de contexto em tarefas de múltiplas fases
- Combinar abordagens: planejar para descoberta, depois executar para implementação

3.5 Refinamento Iterativo para Melhoria Progressiva

Conhecimento chave:

- Exemplos concretos de entrada/saída são a forma mais eficaz de comunicar expectativas
- **Iteração guiada por testes**: escreva os testes primeiro, depois itere com base nas falhas
- O padrão "entrevista": o Claude faz perguntas para trazer à tona considerações de design não óbvias
- Quando fornecer todos os problemas em uma única mensagem (interdependentes) vs sequencialmente (independentes)

Habilidades chave:

- Fornecer 2 a 3 exemplos concretos de entrada/saída para esclarecer requisitos de transformação
- Construir conjuntos de testes com comportamento esperado, casos de borda e requisitos de desempenho antes da implementação
- Usar o padrão entrevista para trazer à tona aspectos de design (invalidação de cache, modos de falha)
- Fornecer casos de teste concretos com entradas de amostra e saídas esperadas para casos de borda

3.6 Integrando o Claude Code em Pipelines de CI/CD

Conhecimento chave:

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

- O CLAUDE.md fornece contexto do projeto (padrões de testes, critérios de revisão) para o Claude Code disparado em CI
- **Isolamento de contexto de sessão**: a mesma sessão que gerou o código é menos eficaz em revisá-lo do que uma instância independente

Habilidades chave:

- Rodar o Claude Code em CI com `-p` para evitar travamentos por entrada interativa
- Usar `--output-format json` + `--json-schema` para resultados estruturados (ex: comentários *inline* em PRs)
- Incluir resultados de revisões anteriores ao re-executar após novos commits (relatar apenas problemas novos/não corrigidos)
- Documentar padrões de testes e *fixtures* disponíveis no CLAUDE.md para melhorar a qualidade da geração de testes
- Incluir arquivos de teste existentes no contexto ao gerar novos testes para evitar duplicação e manter o estilo consistente

---

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

4.1 Projetando Prompts com Critérios Explícitos para Aumentar a Precisão

Conhecimento chave:

- Critérios explícitos são mais eficazes do que instruções vagas (ex: "aponte comentários apenas quando contradizem o código" vs "verifique a precisão dos comentários")
- Orientações genéricas como "seja mais conservador" funcionam pior do que critérios categóricos concretos
- O efeito de falsos positivos na confiança dos desenvolvedores: altas taxas de falsos positivos em algumas categorias minam a confiança em categorias precisas

Habilidades chave:

- Definir critérios de revisão: o que relatar (bugs, segurança) vs o que ignorar (estilo menor)
- Desativar temporariamente categorias com altas taxas de falsos positivos
- Definir critérios explícitos de severidade com exemplos de código para cada nível

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

Conhecimento chave:

- Exemplos few-shot são o método mais eficaz para produzir saídas consistentemente formatadas e acionáveis
- Few-shot pode demonstrar o tratamento de casos ambíguos (seleção de ferramentas, lacunas na cobertura de testes)
- Few-shot ajuda o modelo a generalizar para novos padrões em vez de apenas repetir padrões padrão
- Few-shot pode reduzir alucinações em tarefas de extração

Habilidades chave:

- Fornecer de 2 a 4 exemplos direcionados para cenários ambíguos com justificativa
- Incluir exemplos few-shot que demonstrem o formato de saída (localização, problema, severidade, correção sugerida)
- Fornecer exemplos que diferenciem padrões de código aceitáveis de problemas reais
- Fornecer exemplos de extração correta a partir de documentos com estruturas diferentes

4.3 Impondo Saída Estruturada com `tool_use` e Esquemas JSON

Conhecimento chave:

- `tool_use` com Esquemas JSON é a maneira mais confiável de garantir uma saída em conformidade com o esquema e eliminar erros de sintaxe JSON
- Com `tool_choice: "auto"` o modelo pode retornar texto; com `"any"` ele deve chamar uma ferramenta; a seleção forçada escolhe uma ferramenta específica
- Esquemas JSON estritos eliminam erros sintáticos, mas não evitam erros semânticos (totais não batem; valores nos campos errados)
- Design de esquema: campos obrigatórios vs opcional; enums com "other" mais uma string de detalhamento para extensibilidade

Habilidades chave:

- Definir ferramentas de extração com esquemas JSON e analisar dados dos resultados de `tool_use`
- Usar `tool_choice: "any"` para garantir saída estruturada quando existem múltiplos esquemas
- Forçar uma chamada de ferramenta específica: `tool_choice: {"type": "tool", "name": "extract_metadata"}`
- Tornar campos opcionais/anuláveis quando a fonte pode não conter a informação, evitando a fabricação de valores
- Usar valores de enum como `"unclear"` e `"other"` mais campos de detalhamento para categorização extensível

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

Conhecimento chave:

- Retentativa com feedback de erro: inclua erros de validação concretos no prompt de retentativa para guiar as correções
- Retentativas são ineficazes quando a informação está simplesmente ausente na fonte
- Design de loop de feedback: rastreie o padrão que disparou o achado (`detected_pattern`)
- Erros semânticos (totais não batem) vs erros sintáticos (resolvidos por `tool_use`)

Habilidades chave:

- Prompts de acompanhamento com o documento original, uma extração incorreta e erros específicos de validação
- Identificar quando a retentativa será ineficaz (a informação exigida está apenas em um documento externo)
- Incluir campos `detected_pattern` em achados para analisar falsos positivos
- Projetar autocorreção extraindo tanto o `calculated_total` quanto o `stated_total` para detectar divergências

4.5 Projetando Estratégias Eficientes de Processamento em Lote (Batch)

Conhecimento chave:

- API de Message Batches: 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 inadequado para tarefas bloqueantes (checagens pré-merge)
- A Batch API não suporta chamadas de ferramentas multi-turnos dentro de uma única requisição
- Campos `custom_id` correlacionam requisições/respostas dentro dos lotes

Habilidades chave:

- Usar API síncrona para checagens bloqueantes; usar Batch API para cargas de trabalho noturnas/semanais
- Planejar a frequência de envio de lotes com base nas necessidades de SLA (ex: janelas de 4 horas para uma garantia de 30 horas com processamento de 24 horas)
- Tratar falhas re-enviando apenas os documentos que falharam (identificados por `custom_id`)
- Iterar em prompts usando uma amostra antes de rodar o processamento em larga escala

4.6 Projetando Arquiteturas de Revisão Multi-instância e Multi-passos

Conhecimento chave:

- Limitações de autorevisão: o modelo retém o contexto do seu raciocínio e é menos provável que desafie suas próprias decisões
- Instâncias de revisão independentes (sem o contexto de geração) são melhores para encontrar problemas sutis
- Revisão multi-passos: análise local por arquivo mais um passo de integração entre arquivos para evitar a diluição da atenção

Habilidades chave:

- Usar uma segunda instância independente do Claude para revisar alterações sem o contexto de geração
- Dividir revisões multi-arquivos em passagens por arquivo mais passagens de integração para análise de fluxo de dados entre arquivos
- Usar passos de verificação com confiança auto-avaliada para rotear revisões de forma calibrada

---

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

5.1 Gerenciando o Contexto da Conversa para Preservar Informações Críticas

Conhecimento chave:

- Riscos de sumarização progressiva: valores numéricos, porcentagens e datas são condensados em resumos vagos
- Efeito *lost-in-the-middle*: modelos processam de forma confiável o início e o fim de entradas longas, mas podem perder achados no 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 nas requisições subsequentes à API

Habilidades chave:

- Extrair fatos transacionais para um bloco persistente de "fatos do caso" fora do histórico sumarizado
- Filtrar saídas detalhadas de ferramentas para manter apenas campos relevantes
- Colocar descobertas principais no início dos dados agregados com cabeçalhos de seção explícitos
- Exigir que subagentes incluam metadados (datas, fontes) nas saídas estruturadas

5.2 Projetando Padrões Eficazes de Escalação e Resolvendo Ambiguidades

Conhecimento chave:

- Gatilhos de escalação adequados: solicitação explícita por um humano, lacunas/exceções em políticas, incapacidade de progredir
- Escalação imediata (solicitação explícita) vs tentativa de resolução (dentro do escopo do agente)
- Análise de sentimento e auto-avaliação de confiança do modelo são indicadores não confiáveis para a complexidade do caso
- Múltiplas correspondências de clientes exigem a solicitação de identificadores adicionais, e não adivinhação heurística

Habilidades chave:

- Critérios explícitos de escalação com exemplos few-shot no system prompt
- Executar solicitações explícitas por um humano imediatamente sem investigações adicionais
- Escalar quando a política for ambígua ou omissa para uma solicitação específica
- Pedir identificadores adicionais quando os 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, busca realizada, resultados parciais, alternativas) permite uma recuperação mais inteligente pelo coordenador
- Distinguir falhas de acesso (timeouts exigem decisão de retentativa) de resultados vazios válidos (sem correspondências)
- Status de erro genéricos ("busca indisponível") escondem contexto valioso do coordenador
- Supressão silenciosa ou abortar todo o fluxo em uma única falha são ambos anti-padrões

Habilidades chave:

- Retornar contexto de erro estruturado: tipo de falha, o que foi tentado, resultados parciais, alternativas possíveis
- Distinguir falhas de acesso de resultados vazios válidos
- Executar recuperação local em subagentes para falhas transitórias; propagar apenas erros não recuperáveis com resultados parciais
- Anotar a cobertura na síntese: o que é bem fundamentado vs onde permanecem lacunas

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

Conhecimento chave:

- Degradação de contexto em sessões longas: o modelo começa a produzir respostas instáveis e a se referir a "padrões típicos" em vez de classes específicas
- Arquivos de rascunho (*scratchpad*) preservam descobertas chave através das fronteiras de contexto
- Delegar a subagentes isola a saída de descoberta detalhada
- Persistência de estado estruturado permite a recuperação contra falhas (*crashes*)

Habilidades chave:

- Disparar subagentes para questões específicas enquanto mantém a coordenação de alto nível no agente principal
- Usar arquivos de rascunho para armazenar descobertas principais e referenciá-las posteriormente
- Sumarizar descobertas principais antes de disparar subagentes da próxima fase
- Usar `/compact` para reduzir o uso de contexto durante investigações longas

5.5 Projetando Fluxos com Supervisão Humana e Calibração de Confiança

Conhecimento chave:

- Métricas agregadas (ex: 97% de precisão geral) podem mascarar desempenho ruim em tipos específicos de documentos ou 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
- Validar a precisão por tipo de documento e segmento de campo antes de automatizar

Habilidades chave:

- Implementar amostragem aleatória estratificada para detectar novos padrões de erro
- Analisar a precisão por tipo de documento e campo para validar desempenho estável
- Gerar pontuações de confiança em nível de campo e calibrar limiares de revisão usando dados rotulados
- Rotear extrações de baixa confiança ou fontes ambíguas para revisão humana

5.6 Preservando a Proveniência e Tratando a Incerteza em Sínteses de Múltiplas Fontes

Conhecimento chave:

- A atribuição é perdida durante a sumarização se os mapeamentos "afirmação → fonte" não forem preservados

- Mapeamentos estruturados devem ser preservados durante a agregação
- Tratar estatísticas conflitantes anotando os conflitos com atribuição em vez de escolher arbitrariamente um valor
- Incluir datas de publicação/coleta para evitar mal-entender diferenças temporais como contradições

Habilidades chave:

- Exigir que subagentes gerem mapeamentos "afirmação → fonte" (URL, nome do documento, citações)
- Estruturar relatórios para separar descobertas consolidadas daquelas sob disputa
- Preservar valores conflitantes com anotações e passá-los ao coordenador para reconciliação
- Incluir datas de publicação para interpretação temporal correta
- Renderizar conteúdo por tipo: dados financeiros como tabelas, notícias como texto corrido, descobertas técnicas 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 ignora `get_customer` e chama `lookup_order` usando apenas o nome do cliente, o que leva a reembolsos incorretos.

Qual alteração é mais eficaz?

- 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 A: Quando a lógica de negócio crítica exige uma sequência específica de ferramentas, o código de software fornece **garantias determinísticas** que abordagens baseadas em prompt (B, C) não conseguem oferecer. D aborda 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 dúvidas relacionadas a pedidos. As descrições das ferramentas são mínimas e semelhantes.

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 a B: As descrições das ferramentas são o mecanismo primário de seleção do modelo. Esta é a correção de menor esforço e maior impacto. A adiciona tokens sem resolver a causa raiz. C é sobre-engenharia. D exige mais esforço do que o justificado.

Questão 3 (Cenário: Agente de Suporte ao Cliente)

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

Como melhorar a calibração de escalação?

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

Por que a A: Aborda diretamente a causa raiz — fronteiras de decisão não claras. B não é confiável (o modelo pode estar confiantemente errado). C é sobre-engenharia. D resolve um problema diferente (humor != complexidade).

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

Situação: Você precisa de um comando customizado `/review` para revisão padrão de código que esteja disponível para toda a equipe ao clonar o repositório.

Onde você deve criar o arquivo de comando?

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

Por que a A: Comandos de projeto armazenados em `.claude/commands/` estão no controle de versão 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 o Claude Code)

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

Qual abordagem você deve usar?

- A) Modo de planejamento: explorar a base de código, entender dependências, projetar uma abordagem **[CORRETA]**
- B) Execução direta incrementalmente
- C) Execução direta com instruções detalhadas antecipadas
- D) Execução direta e mudar para planejamento quando ficar difícil

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

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

Situação: Uma base de código possui convenções diferentes entre áreas (React, API, banco de dados). Os testes ficam junto com o código. Você quer que as convenções sejam aplicadas automaticamente.

Qual abordagem você deve usar?

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

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

Questão 7 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: O sistema pesquisa sobre "o impacto da IA nas indústrias criativas", mas os relatórios cobrem apenas arte visual. O coordenador decompôs o tópico em: "IA na arte digital", "IA no design gráfico", "IA na fotografia".

Qual é a causa?

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

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

Questão 8 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: Um subagente de busca web sofre timeout ao pesquisar um tópico complexo. Você precisa projetar como a informação de erro é enviada de volta ao coordenador.

Qual abordagem de propagação de erro permite a recuperação mais inteligente?

- A) Retornar contexto de erro estruturado ao coordenador: tipo de falha, busca realizada, resultados parciais e alternativas **[CORRETA]**
- B) Implementar retentativas automáticas com backoff exponencial dentro do subagente, e 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 manipulador de alto nível que encerra todo o fluxo

Por que a A: O contexto de erro estruturado dá ao coordenador o que ele precisa para decidir se tenta novamente com uma busca modificada, tenta uma abordagem alternativa ou continua com resultados parciais. B esconde o contexto por trás de um status genérico. C mascara a falha como sucesso. D aborta todo o fluxo desnecessariamente.

Questão 9 (Cenário: Sistema de Pesquisa Multi-Agente)

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

Como reduzir a sobrecarga mantendo a confiabilidade?

- A) Dar ao agente de síntese uma ferramenta limitada `verify_fact` para checagens simples, e continuar roteando verificações complexas através do coordenador **[CORRETA]**
- B) Acumular todas as necessidades de verificação em um lote e retorná-las ao coordenador no final
- C) Dar ao agente de síntese acesso total a todas as ferramentas de busca web
- D) Fazer cache proativo de contexto adicional em torno de cada fonte

Por que a A: Aplica o princípio do menor privilégio: o agente de síntese recebe exatamente o que precisa para o caso comum de 85% (checagens simples) enquanto preserva o caminho mediado pelo coordenador para investigações complexas. B introduz dependências bloqueantes. C quebra a separação de responsabilidades. D depende de cache especulativo que não consegue prever necessidades com precisão.

Questão 10 (Cenário: Claude Code para CI)

Situação: Um pipeline executa `claude "Analise este pull request em busca de problemas de segurança"`, mas trava aguardando por entrada interativa.

Qual é a abordagem correta?

- A) Usar a flag `-p`: `claude -p "Analise este pull request em busca de problemas de segurança"` **[CORRETA]**
- B) Definir `CLAUDE_HEADLESS=true`
- C) Redirecionar stdin de `/dev/null`
- D) Usar `--batch`

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

Questão 11 (Cenário: Claude Code para CI)

Situação: A equipe deseja reduzir custos de API para análises automatizadas. O Claude atualmente atende dois fluxos de trabalho em tempo real: (1) uma checagem bloqueante pré-merge que deve ser concluída antes que os desenvolvedores façam o merge de um PR, e (2) um relatório de débito técnico gerado da noite para a dia para revisão matinal. Um gerente propõe mover ambos para a Message Batches API para economizar 50%.

Como você deve avaliar esta proposta?

- A) Usar o processamento em lote apenas para relatórios de débito técnico; manter chamadas em tempo real para checagens pré-merge **[CORRETA]**
- B) Mover ambos os fluxos para processamento em lote e fazer polling até a conclusão
- C) Manter chamadas em tempo real para ambos para evitar problemas de ordenação nos resultados em lote
- D) Mover ambos para processamento em lote com um fallback para tempo real se o lote demorar muito

Por que a A: A API de Message Batches economiza 50%, mas o tempo de processamento pode levar até 24 horas sem garantia de SLA de latência. Isso a torna inadequada para checagens bloqueantes pré-merge onde os desenvolvedores estão esperando, mas ideal para cargas em lote noturnas como relatórios de débito técnico.

Questão 12 (Cenário: Revisão de Código Multi-Arquivos)

Situação: Um pull request altera 14 arquivos em um módulo de rastreamento de estoque. Uma revisão em passagem única de todos os arquivos produz resultados inconsistentes: comentários detalhados em

alguns arquivos mas superficiais em outros, bugs óbvios ignorados e feedback contraditório (um padrão é apontado como problemático em um arquivo mas aprovado em código idêntico em outro arquivo).

Como você deve reestruturar a revisão?

- A) Dividir em passagens focadas: analisar cada arquivo individualmente para problemas locais, e depois rodar uma passagem de integração separada para fluxos de dados entre arquivos **[CORRETA]**
- B) Exigir que os desenvolvedores dividam grandes PRs em envios de 3 a 4 arquivos
- C) Mudar para um modelo de nível superior com uma janela de contexto maior para revisar todos os 14 arquivos de uma vez
- D) Rodar três passagens de revisão independentes do PR completo e relatar apenas problemas encontrados em pelo menos duas rodadas

Por que a A: Passagens focadas abordam diretamente a causa raiz — diluição da atenção ao processar muitos arquivos de uma vez. A análise por arquivo garante profundidade consistente, e uma passagem de integração separada captura problemas entre arquivos. B transfere a carga para os desenvolvedores sem melhorar o sistema. C é um equívoco: um contexto maior não melhora a qualidade da atenção. D suprime bugs reais ao exigir consenso entre detecções inconsistentes.

Simulado Prático

> 60 questões cobrindo os cenários do exame. O formato e a dificuldade correspondem ao exame real.
>
> Alternativamente, você pode praticar estas questões em uma interface HTML interativa: Teste Prático (PT-BR)

Cenário: Sistema de Pesquisa Multi-Agente

Questão 1 (Cenário: Sistema de Pesquisa Multi-Agente)

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

Qual abordagem é mais eficaz?

- A) Aplicar heurísticas de credibilidade para escolher o número mais provável de estar 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 que o agente de síntese decida qual usar com base no contexto mais amplo.

- C) Parar a análise e escalar imediatamente para o 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 a atribuição das fontes e deixar que o coordenador decida como reconciliar os dados antes de passar para a síntese. **[CORRETA]**

Por que a 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 possui um contexto mais amplo.

Questão 2 (Cenário: Sistema de Pesquisa Multi-Agente)

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

Qual é o próximo passo mais apropriado?

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

Por que a C: Em uma arquitetura coordenador–subagente, o coordenador encaminha ambos os conjuntos de resultados para o agente de síntese para uma integração centralizada, preservando o controle e garantindo uma fusão de alta qualidade.

Questão 3 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: Um subagente de análise de documentos falha frequentemente ao processar arquivos PDF: alguns possuem 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 encerra imediatamente o subagente e retorna um erro ao coordenador, que deve decidir se tenta novamente, pula ou falha toda a tarefa. Isso causa um envolvimento excessivo do coordenador no tratamento de erros rotineiros. Qual melhoria arquitetural é mais eficaz?

Qual melhoria é mais eficaz?

- A) Criar um agente dedicado ao tratamento de erros que monitore todas as falhas via uma fila compartilhada e decida as ações de recuperação, enviando comandos de reinicialização diretamente aos subagentes.
- B) Configurar o subagente para sempre retornar resultados parciais com status de sucesso, embutindo detalhes do erro nos 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 os erros que ele não puder resolver, incluindo os passos tentados e os resultados parciais. **[CORRETA]**

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

Questão 4 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: Após executar o sistema sobre "o impacto da IA nas indústrias criativas", você observa que cada subagente é concluído 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 apenas arte visual e ignoram completamente música, literatura e cinema. Nos logs do coordenador, você vê que ele decompôs o tópico em três subtarefas: "IA na arte digital", "IA no design gráfico" e "IA na fotografia". Qual é a causa raiz mais provável?

Qual é a causa raiz mais provável?

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

Por que a C: O coordenador decompôs um tópico amplo apenas em subtópicos de arte visual, ignorando completamente música, literatura e cinema. Como os subagentes executaram suas atribuições corretamente, a decomposição estreita é a causa raiz evidente.

Questão 5 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: O subagente de busca web retorna resultados para apenas 3 das 5 categorias de fontes solicitadas (sites de concorrentes e relatórios da indústria obtêm 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 deve produzir um resumo a partir dessas entradas de qualidade mista. Qual estratégia de propagação de erro é mais eficaz?

Qual estratégia de propagação de erro é mais eficaz?

- 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 uma retentativa completa ou falha da tarefa devido a dados incompletos.
- C) O subagente de síntese pede ao coordenador para tentar novamente as fontes que sofreram timeout com um tempo limite maior antes de iniciar a síntese.
- D) Estruturar a saída da síntese com anotações de cobertura que indiquem quais conclusões são bem fundamentadas e onde existem lacunas devido a fontes indisponíveis. **[CORRETA]**

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

Questão 6 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: O subagente de análise de documentos encontra um arquivo PDF corrompido que não consegue analisar. Ao projetar o tratamento de erros do sistema, qual é a forma mais eficaz de lidar com esta falha?

Qual abordagem é mais eficaz?

- A) Retornar um erro com contexto ao agente coordenador, permitindo que ele decida como proceder. **[CORRETA]**
- B) Ignorar silenciosamente o documento corrompido e continuar processando os arquivos restantes para evitar a interrupção do fluxo de trabalho.
- C) Tentar novamente analisar o documento automaticamente três vezes com backoff exponencial antes de relatar uma falha.
- D) Lançar uma exceção que encerre todo o fluxo de trabalho de pesquisa.

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

Questão 7 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: Logs de produção mostram um padrão persistente: solicitações como "analise o relatório trimestral enviado" são roteadas para o agente de busca web 45% das vezes em vez de irem para o agente de análise de documentos. Ao revisar as definições das ferramentas, você descobre que o agente de busca web possui uma ferramenta `analyze_content` descrita como "analisa conteúdo e extrai informações chave", enquanto o agente de análise de documentos possui uma ferramenta `analyze_document` descrita como "analisa documentos e extrai informações chave". Como você deve corrigir este problema de roteamento?

Como você deve corrigir o problema de roteamento?

- A) Adicionar um classificador prévio que detecte se o usuário se refere a arquivos enviados ou conteúdo web antes que o coordenador decida sobre a delegação.

- B) Renomear a ferramenta de busca web para `extract_web_results` e atualizar sua descrição para "processa e retorna informações recuperadas de buscas web e URLs." **[CORRETA]**
- C) Adicionar exemplos few-shot ao prompt do coordenador mostrando o roteamento correto: "Usuário envia um relatório trimestral → agente de análise de documentos" e "Usuário pergunta sobre uma página web → agente de busca web".
- D) Expandir a descrição da ferramenta de análise de documentos com exemplos de uso como "Use para PDFs enviados, documentos Word e planilhas", deixando a ferramenta de busca web inalterada.

Por que a B: Renomear a ferramenta de busca web para `extract_web_results` e atualizar sua descrição para referenciar explicitamente busca web e URLs elimina diretamente a causa raiz ao remover a sobreposição semântica entre os nomes e descrições das duas ferramentas. Isso torna o propósito de cada ferramenta inequívoco, permitindo que o coordenador diferencie com precisão a análise de documentos da busca web.

Questão 8 (Cenário: Sistema de Pesquisa Multi-Agente)

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

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

- A) O coordenador pode observar todas as interações, tratar erros de forma uniforme e decidir quais informações cada subagente deve receber. **[CORRETA]**
- B) O coordenador agrupa múltiplas requisições para os subagentes em lote, reduzindo o total de chamadas de API e a latência geral.
- C) O roteamento através do coordenador permite uma lógica de retentativa automática que chamadas diretas entre agentes não conseguem suportar.
- D) Os subagentes usam memória isolada, e a comunicação direta exigiria uma serialização complexa que apenas o coordenador pode realizar.

Por que a A: O padrão coordenador fornece visibilidade centralizada sobre todas as interações, tratamento uniforme de erros em todo o sistema e controle detalhado sobre quais informações cada subagente recebe — essas são as principais vantagens de uma topologia de comunicação em estrela (*star-shaped*).

Questão 9 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: O subagente de busca web sofre timeout ao pesquisar um tópico complexo. Você precisa projetar como as informações sobre esta falha são retornadas ao coordenador. Qual abordagem de propagação de erro permite a recuperação mais inteligente?

Qual abordagem de propagação de erro permite a recuperação mais inteligente?

- A) Retornar contexto de erro estruturado ao coordenador incluindo o tipo de falha, a busca executada, quaisquer resultados parciais e potenciais abordagens alternativas. **[CORRETA]**

- B) Capturar o timeout dentro do subagente e retornar um conjunto de resultados vazio marcado como sucesso.
- C) Implementar retentativas automáticas com backoff exponencial dentro do subagente, retornando apenas um status genérico "busca indisponível" após esgotar as tentativas.
- D) Propagar a exceção de timeout diretamente para o manipulador de nível superior, encerrando todo o fluxo de trabalho de pesquisa.

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

Questão 10 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: No design do seu sistema, você deu ao agente de análise de documentos acesso a uma ferramenta de propósito geral `fetch_url` para que ele pudesse baixar documentos por URL. Logs de produção mostram que este agente agora baixa frequentemente páginas de resultados de motores de busca para realizar buscas web ad-hoc — comportamento que deveria ser roteado pelo agente de busca web —, causando resultados inconsistentes. Qual correção é mais eficaz?

Qual correção é mais eficaz?

- A) Substituir `fetch_url` por uma ferramenta `load_document` que valide se as URLs apontam para formatos de documento. **[CORRETA]**
- B) Remover `fetch_url` do agente de análise de documentos e rotear todo o download de URLs através do coordenador para o agente de busca web.
- C) Implementar uma filtragem que bloqueie chamadas de `fetch_url` para domínios de motores de busca conhecidos enquanto permite outras URLs.
- D) Adicionar instruções ao prompt do agente de análise de documentos informando que `fetch_url` deve ser usada apenas para baixar URLs de documentos, e não para pesquisar.

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

Questão 11 (Cenário: Sistema de Pesquisa Multi-Agente)

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

Qual é a forma mais eficaz de resolver isso?

- A) Permitir que ambos os agentes concluam em paralelo e depois fazer com que o coordenador remova a duplicação dos 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 subtópicos ou tipos de fontes distintos. **[CORRETA]**
- C) Implementar um mecanismo de estado compartilhado onde os agentes registram sua área de foco atual para que outros agentes possam evitar duplicações dinamicamente durante a execução.
- D) Mudar para a execução sequencial onde a análise de documentos roda apenas após a busca web ser concluída, usando os resultados da busca web como contexto para evitar duplicações.

Por que a B: Fazer com que o coordenador particione explicitamente o espaço de pesquisa antes de delegar é o mais eficaz porque resolve a causa raiz — fronteiras de tarefas não claras — antes do início de qualquer trabalho. Preserva o paralelismo enquanto evita esforços duplicados e tokens desperdiçados.

Questão 12 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: Durante a pesquisa, o subagente de busca web consulta três categorias de fontes com resultados diferentes: bancos de dados acadêmicos retornam 15 artigos relevantes, relatórios da indústria retornam "0 resultados" e bancos de dados de patentes retornam "Connection timeout". Ao projetar a propagação de erros para o coordenador, qual abordagem permite as melhores decisões de recuperação?

Qual abordagem permite as melhores decisões de recuperação?

- A) Agregar os resultados em uma única métrica de porcentagem de sucesso (ex: "67% de cobertura de fontes") com logs detalhados disponíveis sob demanda.
- B) Relatar tanto "timeout" quanto "0 resultados" como falhas que exigem a intervenção do coordenador.
- C) Tentar novamente falhas transitórias internamente e relatar apenas erros persistentes.
- D) Distinguir falhas de acesso (timeout), que exigem uma decisão de retentativa, de resultados vazios válidos ("0 resultados"), que representam buscas bem-sucedidas. **[CORRETA]**

Por que a D: Um timeout (falha de acesso) e "0 resultados" (resultado vazio válido) são desfechos semanticamente diferentes que exigem respostas diferentes. Distingui-los permite que o coordenador tente novamente o banco de dados de patentes enquanto aceita os "0 resultados" dos relatórios da indústria como uma descoberta válida e informativa.

Questão 13 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: O monitoramento de produção mostra uma qualidade de síntese inconsistente. Quando os resultados agregados chegam a ~75K tokens, o agente de síntese cita com precisão informações dos primeiros 15K tokens (manchetes/trechos de busca web) e dos últimos 10K tokens (conclusões de análise de documentos), mas frequentemente perde descobertas críticas nos 50K tokens do meio — mesmo quando elas respondem diretamente à pergunta de pesquisa. Como você deve reestruturar a entrada agregada?

Como você deve reestruturar a entrada agregada?

- A) Sumarizar todas as saídas dos subagentes para menos de 20K tokens antes da agregação para manter o conteúdo dentro do alcance de processamento confiável do modelo.
- B) Transmitir os resultados dos subagentes para o agente de síntese incrementalmente, processando os resultados de busca web primeiro até a conclusão, e depois adicionando os resultados de análise de documentos.
- C) Colocar um resumo das principais descobertas no início da entrada agregada e organizar os resultados detalhados com cabeçalhos de seção explícitos para facilitar a navegação. **[CORRETA]**
- D) Implementar uma rotação que alterne qual resultado de subagente aparece primeiro entre as tarefas de pesquisa para garantir que ambas as fontes obtenham posicionamento de topo igual ao longo do tempo.

Por que a C: Colocar um resumo das principais descobertas no início aproveita o efeito de primazia para que informações críticas fiquem na posição processada com maior confiabilidade. Adicionar cabeçalhos de seção explícitos ao longo do texto ajuda o modelo a navegar e atentar ao conteúdo do meio da entrada, mitigando diretamente o fenômeno "lost in the middle".

Questão 14 (Cenário: Sistema de Pesquisa Multi-Agente)

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) soma 155K tokens, mas o agente de síntese atinge o melhor desempenho com entradas abaixo de 50K tokens. Qual solução é mais eficaz?

Qual solução é mais eficaz?

- A) Modificar os agentes de montante (*upstream*) para retornar dados estruturados (fatos chave, citações, pontuações de relevância) em vez de conteúdos e raciocínios verbosos. **[CORRETA]**
- B) Adicionar um agente intermediário de sumarização que condense os achados antes de passá-los para a síntese.
- C) Fazer com que o agente de síntese processe as descobertas em lotes sequenciais, mantendo o estado entre as chamadas.
- D) Armazenar as descobertas em um banco de dados vetorial e dar ao agente de síntese ferramentas de busca para consultar durante seu trabalho.

Por que a A: Modificar os agentes de montante para retornar dados estruturados corrige a causa raiz ao reduzir o volume de tokens na fonte enquanto preserva informações essenciais. Evita a passagem de conteúdos volumosos de páginas e rastros de raciocínio que inflacionam os tokens sem melhorar o passo de síntese.

Questão 15 (Cenário: Sistema de Pesquisa Multi-Agente)

Situação: Em testes, você observa que o agente de síntese frequentemente precisa verificar afirmações específicas enquanto mescla resultados. Atualmente, quando a verificação é necessária, o agente de

síntese devolve o controle ao coordenador, que chama o agente de busca web e depois re-invoca a síntese com os resultados. Isso adiciona 2 a 3 loops extras por tarefa e aumenta a latência em 40%. Sua avaliação mostra que 85% dessas verificações são checagens simples de fatos (datas, nomes, estatísticas) e 15% exigem pesquisas mais profundas. Qual abordagem reduz mais eficazmente a sobrecarga mantendo a confiabilidade do sistema?

Qual abordagem é mais eficaz?

- A) Dar ao agente de síntese acesso a todas as ferramentas de busca web para que ele possa lidar diretamente com qualquer necessidade de verificação sem loops no coordenador.
- B) Fazer com que o agente de síntese acumule todas as necessidades de verificação e as retorne como um lote ao coordenador no final, que então envia todas para o agente de busca web de uma só vez.
- C) Fazer com que o agente de busca web faça cache proativo de contexto extra em torno de cada fonte durante a pesquisa inicial em antecipação à necessidade de verificação da síntese.
- D) Dar ao agente de síntese uma ferramenta de escopo limitado `verify_fact` para checagens simples, enquanto roteia verificações complexas através do coordenador para o agente de busca web. **[CORRETA]**

Por que a D: Uma ferramenta de verificação de fatos com escopo limitado permite que o agente de síntese lide com 85% das checagens simples diretamente, eliminando a maioria dos loops, enquanto preserva o caminho de delegação ao coordenador para os 15% de verificações complexas. Isso aplica o menor privilégio enquanto reduz significativamente a latência.

Cenário: Claude Code para Integração Contínua (CI/CD)

Questão 16 (Cenário: Claude Code para CI/CD)

Situação: Seu pipeline de CI roda a CLI do Claude Code (em modo `--print`) usando o CLAUDE.md para fornecer contexto de projeto para revisão de código, e os desenvolvedores geralmente acham as revisões substanciais. No entanto, eles relatam que integrar as descobertas no fluxo de trabalho é difícil — o Claude gera parágrafos narrativos que devem ser copiados manualmente para comentários no PR. A equipe deseja publicar automaticamente cada descoberta como um comentário *inline* separado no PR no local relevante do código, o que exige dados estruturados com caminho do arquivo, número da linha, nível de severidade e correção sugerida. Qual abordagem é mais eficaz?

Qual abordagem é mais eficaz?

- A) Adicionar uma seção "Output Format for Review" ao CLAUDE.md com exemplos de achados estruturados para que o Claude aprenda o formato esperado a partir do contexto do projeto.
- B) Usar as flags da CLI `--output-format json` e `--json-schema` para impor achados estruturados, e depois analisar a saída para publicar comentários *inline* via API do GitHub. **[CORRETA]**
- C) Incluir instruções explícitas de formatação no prompt de revisão exigindo que cada descoberta siga um template analisável como `[FILE:caminho] [LINE:n] [SEVERITY:nivel] ...`.

- D) Manter o formato de revisão narrativa, mas adicionar um passo de sumarização que use o Claude para gerar um resumo JSON estruturado das descobertas.

Por que a B: Usar `--output-format json` com `--json-schema` impõe uma saída estruturada no nível da CLI, garantindo um JSON bem-formado com os campos exigidos (caminho do arquivo, linha, severidade, sugestão de correção) que pode ser analisado com segurança e publicado como comentários *inline* via API do GitHub. Aproveita recursos nativos da CLI projetados especificamente para saídas estruturadas.

Questão 17 (Cenário: Claude Code para CI/CD)

Situação: Sua equipe usa o Claude Code para gerar sugestões de código, mas você nota um padrão: problemas não óbvios — otimizações de desempenho que quebram casos de borda, limpezas que alteram inesperadamente o comportamento — só são capturados quando outro membro da equipe 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 desta limitação de auto-checagem?

Qual abordagem aborda diretamente a causa raiz?

- A) Executar uma segunda instância independente do Claude Code para revisar as alterações sem acesso ao raciocínio do gerador. **[CORRETA]**
- B) Ativar o modo de pensamento estendido (*extended thinking*) para a etapa de geração para permitir uma deliberação mais aprofundada antes de produzir sugestões.
- C) Adicionar instruções explícitas de autorevisão ao prompt de geração pedindo ao Claude para criticar suas próprias sugestões antes de finalizar a saída.
- D) Incluir arquivos completos de testes e documentação no contexto do prompt para que o Claude entenda melhor o comportamento esperado durante a geração.

Por que a A: Uma segunda instância independente do Claude Code sem acesso ao raciocínio do gerador aborda diretamente a causa raiz ao evitar o viés de confirmação. Esta perspectiva de "olhos frescos" espelha a revisão por pares humana, onde outro revisor captura problemas que o autor rationalizou.

Questão 18 (Cenário: Claude Code para CI/CD)

Situação: Seu componente de revisão de código é iterativo: o Claude analisa o arquivo alterado e depois pode solicitar arquivos relacionados (importações, classes base, testes) via chamadas de ferramentas para entender o contexto antes de fornecer o feedback final. Sua aplicação define uma ferramenta que permite ao Claude solicitar conteúdos de arquivos; o Claude chama a ferramenta, obtém os resultados e continua a análise. Você está avaliando o processamento em lote para reduzir custos de API. Qual é a principal limitação técnica ao considerar o processamento em lote para este fluxo de trabalho?

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 retornar resultados para que o Claude continue a análise. **[CORRETA]**
- C) A Batch API não suporta definições de ferramentas em parâmetros de requisição.
- D) A latência de processamento em lote de até 24 horas é muito lenta para feedback em pull requests, embora o fluxo funcionasse de outra forma.

Por que a B: Um modelo assíncrono de Batch API ("dispare e esqueça") não possui mecanismo para interceptar uma chamada de ferramenta durante uma requisição, executar a ferramenta e retornar resultados para que o Claude continue a análise. Isso é fundamentalmente incompatível com fluxos iterativos de chamadas de ferramentas que exigem múltiplas rodadas de requisição/resposta dentro de uma única interação lógica.

Questão 19 (Cenário: Claude Code para CI/CD)

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

Qual combinação está correta?

- A) Usar a Message Batches API para todas as três tarefas para maximizar a economia de 50%, configurando o pipeline para fazer polling pela conclusão do lote.
- B) Usar chamadas síncronas para checagens de estilo em PRs; usar a Message Batches API para auditorias semanais de segurança e geração noturna de testes. **[CORRETA]**
- C) Usar chamadas síncronas para todas as três tarefas para tempos de resposta consistentes, contando com cache de prompt para reduzir custos nas cargas de trabalho.
- D) Usar chamadas síncronas para checagens de estilo em PRs e geração noturna de testes; usar a Message Batches API apenas para auditorias semanais de segurança.

Por que a B: Checagens de estilo em PRs bloqueiam desenvolvedores e exigem respostas imediatas via chamadas síncronas, enquanto auditorias semanais de segurança e geração noturna de testes são tarefas agendadas com prazos flexíveis que podem tolerar uma janela de lote de até 24 horas — garantindo 50% de economia para ambas.

Questão 20 (Cenário: Claude Code para CI/CD)

Situação: Suas revisões automatizadas encontram problemas reais, mas os desenvolvedores relatam que o feedback não é acionável. Os achados incluem frases como "lógica complexa de roteamento de chamadas" ou "potencial ponteiro nulo" sem especificar o que exatamente alterar. Quando você adiciona instruções detalhadas como "sempre inclua sugestões concretas de correção", o modelo ainda produz saídas inconsistentes — às vezes detalhadas, às vezes vagas. Qual técnica de prompting produz com maior confiabilidade um feedback consistentemente acionável?

Qual técnica de prompting é mais confiável?

- A) Refinar ainda mais as instruções com requisitos mais explícitos para cada parte do formato de feedback (localização, problema, severidade, correção proposta).
- B) Expandir a janela de contexto para incluir mais partes da base de código ao redor para que o modelo tenha informações suficientes para propor correções concretas.
- C) Implementar uma abordagem em dois passos onde um prompt identifica problemas e um segundo gera correções, permitindo especialização.
- D) Adicionar de 3 a 4 exemplos few-shot mostrando o formato exato exigido: problema identificado, localização no código, sugestão concreta de correção. **[CORRETA]**

Por que a D: Exemplos few-shot são a técnica mais eficaz para alcançar um formato de saída consistente quando instruções isoladas produzem resultados variáveis. Fornecer de 3 a 4 exemplos que mostrem a estrutura exata desejada (problema, localização, correção concreta) dá ao modelo um padrão concreto a seguir, o que é mais confiável do que instruções abstratas.

Questão 21 (Cenário: Claude Code para CI/CD)

Situação: Seu pipeline de CI inclui dois modos de revisão de código baseados em Claude: um hook de pre-merge-commit que bloqueia o merge do PR até a conclusão, e uma "análise profunda" que roda da noite para o dia, faz polling até a conclusão do lote e publica sugestões detalhadas no PR. Você deseja reduzir custos de API usando a Message Batches API, que oferece 50% de economia mas exige polling e pode levar até 24 horas. Qual modo deve usar o processamento em lote?

Qual modo deve usar o processamento em lote?

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

Por que a B: A análise profunda é a candidata ideal para processamento em lote porque já roda da noite para o dia, tolera atrasos e usa um modelo de polling antes de publicar resultados — combinando perfeitamente com a arquitetura assíncrona baseada em polling da Message Batches API enquanto garante 50% de economia.

Questão 22 (Cenário: Claude Code para CI/CD)

Situação: Sua revisão automatizada analisa comentários e docstrings. O prompt atual instrui o Claude a "verificar se os comentários estão precisos e atualizados". Os achados frequentemente apontam padrões aceitáveis (marcadores TODO, descrições simples) enquanto perdem comentários que descrevem comportamentos que o código não implementa mais. Qual alteração aborda a causa raiz desta análise inconsistente?

Qual alteração aborda a causa raiz?

- A) Incluir dados de `git blame` para que o Claude consiga identificar comentários anteriores a alterações recentes no código.
- B) Adicionar exemplos few-shot de comentários enganosos para ajudar o modelo a reconhecer padrões semelhantes na base de código.
- C) Filtrar padrões de comentários TODO, FIXME e descritivos antes da análise para reduzir ruídos.
- D) Especificar critérios explícitos: apontar comentários apenas quando o comportamento que afirmam contradiz o comportamento real do código. **[CORRETA]**

Por que a D: Critérios explícitos — apontar comentários apenas quando o comportamento afirmado contradiz o comportamento real do código — abordam diretamente a causa raiz ao substituir uma instrução vaga por uma definição precisa do que constitui um problema. Isso reduz falsos positivos em padrões aceitáveis e omissões de comentários verdadeiramente enganosos.

Questão 23 (Cenário: Claude Code para CI/CD)

Situação: Seu sistema automatizado de revisão de código mostra classificações de severidade inconsistentes — problemas semelhantes como riscos de ponteiro nulo são classificados como "críticos" em alguns PRs mas apenas "médios" em outros. Pesquisas com desenvolvedores mostram uma desconfiança crescente — muitos começam a ignorar apontamentos sem ler porque "metade está errada". Categorias com altos falsos positivos corroem a confiança em categorias precisas. Qual abordagem restaura melhor a confiança dos desenvolvedores enquanto melhora o sistema?

Qual abordagem restaura melhor a confiança dos desenvolvedores?

- A) Desativar temporariamente categorias com altos falsos positivos (estilo, nomes, documentação) e manter apenas categorias de alta precisão enquanto melhora os prompts. **[CORRETA]**
- B) Manter todas as categorias ativas mas exibir pontuações de confiança em cada achado para que os desenvolvedores decidam o que investigar.
- C) Manter todas as categorias ativas e adicionar exemplos few-shot para melhorar a precisão de cada categoria ao longo das próximas semanas.
- D) Aplicar uma redução uniforme de rigor em todas as categorias para diminuir a taxa geral de falsos positivos.

Por que a A: Desativar temporariamente categorias com altos falsos positivos interrompe imediatamente a erosão da confiança ao remover achados ruidosos que levam os desenvolvedores a ignorar tudo, enquanto preserva o valor de categorias de alta precisão como segurança e correção. Também cria espaço para melhorar os prompts das categorias problemáticas antes de reativá-las.

Questão 24 (Cenário: Claude Code para CI/CD)

Situação: Sua revisão automatizada gera sugestões de casos de teste para cada PR. Ao revisar um PR que adiciona rastreamento de conclusão de curso, o Claude sugere 10 casos de teste, mas o feedback dos desenvolvedores mostra que 6 duplicam cenários já cobertos pela suíte de testes existente. Qual alteração reduz mais eficazmente as sugestões duplicadas?

Qual alteração é mais eficaz?

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

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

Questão 25 (Cenário: Claude Code para CI/CD)

Situação: Após uma revisão automatizada inicial identificar 12 achados, um desenvolvedor faz push de novos commits para resolver os problemas. Ao re-executar a revisão, são produzidos 8 achados, mas os desenvolvedores relatam que 5 duplicam comentários anteriores sobre códigos que já foram corrigidos nos novos commits. Qual é a forma mais eficaz de eliminar este feedback redundante mantendo a abrangência da revisão?

Qual é a forma mais eficaz de eliminar o feedback redundante?

- A) Rodar a revisão apenas quando o PR é criado e no estado final pré-merge, pulando commits intermediários.
- B) Adicionar um filtro de pós-processamento que remova achados que correspondam aos anteriores por caminhos de arquivo e descrições de problemas antes de publicar os comentários.
- C) Restringir o escopo da revisão aos arquivos alterados no push mais recente, excluindo arquivos de commits anteriores.
- D) Incluir os achados das revisões anteriores no contexto e instruir o Claude a relatar apenas problemas novos ou ainda não resolvidos. **[CORRETA]**

Por que a D: Incluir os achados das revisões anteriores no contexto permite que o Claude diferencie novos problemas daqueles já resolvidos em commits recentes. Isso preserva a abrangência da revisão enquanto usa o raciocínio do Claude para evitar feedbacks redundantes sobre códigos corrigidos.

Questão 26 (Cenário: Claude Code para CI/CD)

Situação: Seu script de pipeline roda `claude "Analise este pull request em busca de problemas de segurança"`, mas a tarefa trava indefinidamente. Logs mostram que o Claude Code está aguardando por entrada interativa. Qual é a abordagem correta para rodar o Claude Code em um pipeline automatizado?

Qual é a abordagem correta?

- A) Adicionar a flag `--batch`: `claude --batch "Analise este pull request em busca de problemas de segurança"`.
- B) Adicionar a flag `-p`: `claude -p "Analise este pull request em busca de problemas de segurança"`. **[CORRETA]**
- C) Redirecionar stdin de `/dev/null`: `claude "Analise este pull request em busca de problemas de segurança" < /dev/null`.
- D) Definir a variável de ambiente `CLAUDE_HEADLESS=true` antes de rodar o comando.

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

Questão 27 (Cenário: Claude Code para CI/CD)

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

Como você deve reestruturar a revisão?

- A) Rodar três passagens de revisão independentes do PR completo e apontar apenas problemas que apareçam em pelo menos duas das três rodadas.
- B) Dividir em passagens focadas: revisar cada arquivo individualmente para problemas locais, e depois rodar uma passagem orientada a integração separada para examinar fluxos de dados entre arquivos. **[CORRETA]**
- C) Exigir que os desenvolvedores dividam grandes PRs em envios menores de 3 a 4 arquivos antes de rodar a revisão automatizada.
- D) Mudar para um modelo maior com uma janela de contexto maior para que ele consiga prestar atenção suficiente a todos os 14 arquivos em uma única passagem.

Por que a B: Passagens focadas por arquivo abordam a causa raiz — diluição da atenção — garantindo profundidade consistente e detecção confiável de problemas locais. Uma passagem separada orientada a integração cobre preocupações entre arquivos como interações de dependência e fluxo de dados.

Questão 28 (Cenário: Claude Code para CI/CD)

Situação: Sua revisão automatizada de código gera em média 15 achados por pull request, e os desenvolvedores relatam uma taxa de falsos positivos de 40%. O gargalo é o tempo de investigação: os desenvolvedores precisam clicar em cada achado para ler a justificativa do Claude antes de decidir se corrigem ou descartam. Seu CLAUDE.md já contém regras abrangentes para padrões aceitáveis, e as partes interessadas rejeitaram qualquer abordagem que filtre achados antes que os desenvolvedores os vejam. Qual alteração aborda melhor o tempo de investigação?

Qual alteração aborda melhor o tempo de investigação?

- A) Exigir que o Claude inclua sua justificativa e estimativa de confiança diretamente em cada achado. **[CORRETA]**
- B) Adicionar um pós-processador que analise padrões de achados e suprima automaticamente aqueles que correspondam a assinaturas históricas de falsos positivos.
- C) Categorizar achados como "problemas bloqueantes" vs "sugestões", com requisitos de revisão diferentes por nível.
- D) Configurar o Claude para exibir apenas achados de alta confiança, filtrando alertas incertos antes que os desenvolvedores os vejam.

Por que a A: Incluir justificativa e confiança diretamente em cada achado reduz o tempo de investigação permitindo que os desenvolvedores façam a triagem rapidamente sem ter que abrir cada achado. Atende à restrição de "sem filtragem" porque todos os achados permanecem visíveis enquanto acelera a tomada de decisão do desenvolvedor.

Questão 29 (Cenário: Claude Code para CI/CD)

Situação: A análise da sua revisão automatizada de código mostra grandes diferenças nas taxas de falsos positivos por categoria de achado: achados de segurança/correção possuem 8% de falsos positivos, achados de desempenho 18%, achados de estilo/nomes 52% e achados de documentação 48%. Pesquisas com desenvolvedores mostram uma desconfiança crescente — muitos começam a ignorar achados sem ler porque "metade está errada". Categorias com altos falsos positivos corroem a confiança em categorias precisas. Qual abordagem restaura melhor a confiança dos desenvolvedores enquanto melhora o sistema?

Qual abordagem restaura melhor a confiança dos desenvolvedores?

- A) Desativar temporariamente categorias com altos falsos positivos (estilo, nomes, documentação) e manter apenas categorias de alta precisão enquanto melhora os prompts. **[CORRETA]**
- B) Manter todas as categorias ativas mas exibir pontuações de confiança em cada achado para que os desenvolvedores decidam o que investigar.
- C) Manter todas as categorias ativas e adicionar exemplos few-shot para melhorar a precisão de cada categoria ao longo das próximas semanas.
- D) Aplicar uma redução uniforme de rigor em todas as categorias para diminuir a taxa geral de falsos positivos.

Por que a A: Desativar temporariamente categorias com altos falsos positivos interrompe imediatamente a erosão da confiança ao remover achados ruidosos que levam os desenvolvedores a ignorar tudo, enquanto preserva o valor de categorias de alta precisão como segurança e correção. Também cria espaço para melhorar os prompts de categorias problemáticas antes de reativá-las.

Questão 30 (Cenário: Claude Code para CI/CD)

Situação: Sua equipe deseja reduzir custos de API para análises automatizadas. Atualmente, chamadas síncronas ao Claude atendem dois fluxos de trabalho: (1) uma checagem pré-merge bloqueante que deve ser concluída antes que os desenvolvedores façam o merge, e (2) um relatório de débito técnico gerado da noite para o dia para revisão na manhã seguinte. Seu gerente propõe mover ambos para a Message Batches API para economizar 50%. Como você deve avaliar esta proposta?

Como você deve avaliar esta proposta?

- A) Mover o relatório de débito técnico para lote e manter a checagem pré-merge síncrona. **[CORRETA]**
- B) Mover ambos para processamento em lote e fazer polling até a conclusão.
- C) Manter chamadas síncronas para ambos para evitar problemas de ordenação nos resultados.
- D) Mover ambos para lote com um fallback síncrono caso o lote demore muito.

Por que a A: A API de Message Batches oferece 50% de economia mas pode levar até 24 horas sem SLA de latência garantido, tornando-a inadequada para checagens pré-merge em tempo real, mas perfeita para tarefas agendadas sem urgência imediata.

Questão 31 (Cenário: Extração de Dados Estruturados)

Situação: Você está projetando um sistema de extração para faturas que contêm campos opcionais (ex: número da ordem de compra, identificador de isenção fiscal). Seu esquema JSON marca esses campos como obrigatórios (`required`). O que provavelmente acontecerá durante a extração?

O que provavelmente acontecerá?

- A) O modelo falhará ao gerar o JSON e retornará um erro sintático.
- B) O modelo omitirá o objeto inteiro de fatura quando um campo opcional estiver ausente.
- C) O modelo inventará ou alucinará valores para atender às restrições de campos obrigatórios. **[CORRETA]**
- D) O modelo retornará automaticamente `null` para campos ausentes.

Por que a C: Marcar campos como obrigatórios no esquema JSON força o modelo a fornecer um valor para cada campo exigido. Quando a informação está ausente no documento de origem, a restrição obriga o modelo a inventar ou alucinar valores para produzir um JSON válido que atenda ao esquema.

Questão 32 (Cenário: Extração de Dados Estruturados)

Situação: Seu sistema extrai dados de relatórios financeiros onde os valores numéricos são apresentados em diferentes formatos (ex: "US$ 1,2 milhão", "1.200.000 USD", "R$ 1,2M"). Você precisa que a saída contenha valores numéricos padronizados e códigos de moeda ISO. Qual abordagem é mais confiável?

Qual abordagem é mais confiável?

- A) Incluir regras explícitas de normalização no prompt com exemplos few-shot demonstrando a conversão de vários formatos de entrada para valores numéricos padrão e códigos ISO. **[CORRETA]**
- B) Usar um pós-processamento com expressões regulares (*regex*) para identificar e converter os valores numéricos após a extração.
- C) Confiar que o esquema JSON sozinho converterá os formatos de entrada para números.
- D) Solicitar ao modelo que retorne os valores exatamente como aparecem no documento e fazer a conversão em código posteriormente.

Por que a A: O modelo possui excelente capacidade de compreensão contextual para interpretar linguagens naturais e formatos variados de valores numéricos. Fornecer regras explícitas de normalização e exemplos few-shot guia o modelo a realizar a padronização diretamente durante a extração de forma altamente confiável.

Questão 33 (Cenário: Extração de Dados Estruturados)

Situação: Um sistema de extração de currículos precisa categorizar o nível de experiência dos candidatos em um dos quatro níveis: "Júnior", "Pleno", "Sênior" ou "Especialista". Alguns currículos contêm perfis atípicos ou informações insuficientes para determinar o nível com clareza. Qual design de esquema e prompt é mais adequado?

Qual design é mais adequado?

- A) Usar um enum com `["Júnior", "Pleno", "Sênior", "Especialista"]` e instruir o modelo a escolher o nível mais próximo mesmo que duvidoso.
- B) Usar um enum incluindo `"Indeterminado"` mais um campo opcional `justificativa` para explicar a incerteza. **[CORRETA]**
- C) Usar uma string livre para o nível de experiência em vez de um enum.
- D) Marcar o campo de nível de experiência como opcional e omiti-lo quando for incerto.

Por que a B: Permitir uma opção explícita de incerteza (`"Indeterminado"`) juntamente com uma justificativa evita que o modelo force uma classificação incorreta ao se deparar com casos ambíguos, mantendo a integridade dos dados categorizados.

Questão 34 (Cenário: Extração de Dados Estruturados)

Situação: Seu pipeline extrai itens de linha de faturas. Em 5% dos casos, a soma dos valores extraídos dos itens de linha não atinge o valor total declarado na fatura devido a taxas ou descontos não detectados. Como seu pipeline de validação deve tratar essa inconsistência?

Como tratar essa inconsistência?

- A) Falhar o pipeline e descartar a extração.
- B) Ajustar automaticamente o total declarado para igualar a soma dos itens de linha.
- C) Disparar um loop de retentativa com feedback incluindo os itens extraídos, a soma calculada, o total declarado e a instrução para re-verificar o documento em busca de taxas ou descontos omitidos.

[CORRETA]

- D) Aceitar a extração e marcar uma flag global de erro.

Por que a C: Um loop de retentativa com feedback de erro específico orienta o modelo a re-examinar o documento com foco nos itens ou valores que podem ter sido ignorados, corrigindo a inconsistência sem alterar dados declarados de forma arbitrária.

Questão 35 (Cenário: Extração de Dados Estruturados)

Situação: Você processa 50.000 documentos usando a Message Batches API. Ao receber os resultados do lote, você nota que 350 documentos falharam devido a limites de tokens excedidos. Como você deve proceder para concluir o processamento desses documentos com falha?

Como você deve proceder?

- A) Re-enviar todo o lote de 50.000 documentos novamente.
- B) Identificar os 350 documentos pelo `custom_id`, aplicar uma estratégia de divisão (*chunking*) nos documentos longos e re-enviar apenas esses 350 em um novo lote. **[CORRETA]**
- C) Processar os 350 documentos individualmente de forma síncrona sem alterar a estratégia de prompt.
- D) Marcar os 350 documentos como irrecuperáveis e descartá-los.

Por que a B: O uso de `custom_id` permite isolar exatamente os itens que falharam. Modificar a estratégia para tratar documentos muito longos (como dividi-los em partes) e re-enviar apenas o subconjunto afetado otimiza tempo e custos.

Questão 36 (Cenário: Extração de Dados Estruturados)

Situação: Uma empresa precisa extrair dados de formulários médicos digitalizados. A precisão geral do sistema é de 98%, mas para um tipo específico de formulário antigo a taxa de erro nos nomes dos medicamentos chega a 30%. Qual estratégia de supervisão/auditoria é mais eficaz para detectar e mitigar esse problema?

Qual estratégia é mais eficaz?

- A) Realizar amostragem aleatória simples de 1% de todos os formulários processados.
- B) Implementar amostragem aleatória estratificada analisando a precisão dividida por tipo de formulário e por campo específico. **[CORRETA]**
- C) Aumentar o limite de confiança necessário para automação de 90% para 99% em todos os campos.
- D) Exigir revisão humana para 100% dos formulários médicos.

Por que a B: A amostragem estratificada permite segmentar as métricas de qualidade por tipo de documento e campo individual, revelando bolsas de alta taxa de erro que seriam mascaradas por métricas de precisão agregadas globais.

Questão 37 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Um assistente conversacional de suporte financeiro atende a uma conversa longa que atingiu 60 turnos e 80.000 tokens. O usuário mencionou sua renda e metas no 3º turno. No 60º turno, ao pedir uma recomendação de plano de aposentadoria, o assistente ignora as metas mencionadas no início. O histórico é enviado completo a cada turno. Qual é o fenômeno causador deste problema?

Qual é a causa?

- A) O modelo atingiu o limite rígido da API e truncou as mensagens iniciais.
- B) O efeito "Lost in the Middle" (Perdido no Meio), onde informações no meio de um histórico muito longo perdem a atenção do modelo em relação ao início e ao fim. **[CORRETA]**
- C) Falta de um parâmetro de temperatura adequado para conversas longas.
- D) O assistente limpou a memória automaticamente ao ultrapassar 50 turnos.

Por que a B: Em janelas de contexto muito grandes, a atenção do modelo tende a se concentrar no início (system prompt e primeiras mensagens) e no final (mensagens mais recentes), reduzindo a capacidade de resgatar detalhes específicos localizados no meio da sequência de contexto.

Questão 38 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Para mitigar o crescimento descontrolado do histórico em um chat de suporte, sua equipe decide aplicar uma sumarização progressiva das mensagens antigas. Qual é o principal risco dessa abordagem para dados transacionais e de identificação?

Qual é o principal risco?

- A) O tempo de resposta da API aumentará proporcionalmente ao tamanho do resumo.
- B) Dados precisos como números de documentos, códigos de erro, datas e valores podem ser generalizados ou perder precisão durante a sumarização. **[CORRETA]**
- C) O modelo se recusará a responder mensagens que tenham resumos no histórico.
- D) A sumarização altera o papel (*role*) das mensagens de `user` para `system`.

Por que a B: A sumarização abstrativa frequentemente substitui valores exatos e datas específicas por descrições aproximadas ou resumidas (ex: "o cliente mencionou um valor alto"), perdendo detalhes cruciais necessários para chamadas de ferramentas de precisão.

Questão 39 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Como arquiteto, você deseja implementar a melhor estratégia para manter informações transacionais críticas (como ID do cliente, número do pedido e status da conta) sempre disponíveis e

precisas em uma conversa longa de suporte, sem depender da sumarização do histórico. Qual estrutura você deve adotar?

Qual estrutura você deve adotar?

- A) Gravar as informações em um banco vetorial e fazer busca semântica a cada turno.
- B) Manter um bloco estruturado de "Fatos do Caso" (*Case Facts*) atualizado, que é anexado a todo prompt fora do histórico sumarizado. **[CORRETA]**
- C) Repetir as informações cruciais no final de cada mensagem enviada pelo usuário.
- D) Confiar que o system prompt sozinho manterá a memória de todas as variáveis.

Por que a B: Extrair e manter um bloco explícito de "Fatos do Caso" garante que dados exatos permaneçam intactos e visíveis a cada requisição, imunes à perda de precisão que ocorre na sumarização do histórico de conversa.

Questão 40 (Cenário: Padrões de Arquitetura de IA Conversacional)

Situação: Durante uma interação de atendimento, o usuário diz: "Quero alterar o endereço de entrega do meu pedido #9988". Em seguida, o agente chama `update_shipping_address` diretamente. No entanto, o sistema da empresa exige confirmação explícita do usuário com a exibição do novo endereço formatado antes de efetivar qualquer alteração de cadastro. Como estruturar essa interação com segurança?

Como estruturar com segurança?

- A) Confiar na instrução do prompt para que o agente peça confirmação verbal antes de chamar a ferramenta.
- B) Dividir o processo em duas ferramentas: `preview_address_change` (que retorna o endereço formatado e um token de confirmação) e `confirm_address_change` (que exige o token para efetivar a alteração no banco). **[CORRETA]**
- C) Executar a alteração e enviar uma mensagem dizendo "Endereço alterado, me avise se quiser desfazer".
- D) Usar uma ferramenta única com um parâmetro `force: true`.

Por que a B: Garantir a exigência no nível da arquitetura de ferramentas (exigindo um token que só a ferramenta de prévia gera) impede que o modelo execute a alteração final diretamente sem passar pela etapa de exibição e confirmação prévias.

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

Situação: Você precisa configurar o Claude Code em um projeto monorepo para que as regras de linting e estilo do React sejam aplicadas apenas quando arquivos sob a pasta `packages/ui/` forem modificados. O que você deve fazer?

O que você deve fazer?

- A) Criar um arquivo `~/.claude/CLAUDE.md` com as regras do React.
- B) Criar um arquivo `.claude/rules/react-ui.md` com o frontmatter YAML contendo `paths: ["packages/ui/**/*"]`. **[CORRETA]**
- C) Incluir as regras no `package.json` da raiz.
- D) Adicionar as regras diretamente na descrição da ferramenta Edit.

Por que a B: O uso de arquivos em `.claude/rules/` com a propriedade `paths` no frontmatter YAML permite o carregamento condicional e direcionado de convenções com base em padrões de caminho, economizando contexto.

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

Situação: Um desenvolvedor deseja criar um comando personalizado `/benchmark` no Claude Code para uso estritamente pessoal, sem compartilhar com o restante da equipe no repositório Git. Onde esse arquivo de comando deve ser salvo?

Onde o arquivo deve ser salvo?

- A) Em `.claude/commands/benchmark.md` no repositório do projeto.
- B) Em `~/.claude/commands/benchmark.md` ou `~/.claude/skills/benchmark/SKILL.md` no diretório home do usuário. **[CORRETA]**
- C) No arquivo `CLAUDE.md` na raiz do projeto.
- D) Em `/tmp/benchmark.md`.

Por que a B: Arquivos salvos no diretório home do usuário (`~/.claude/`) são pessoais e não entram no controle de versão do projeto, mantendo os comandos individuais isolados.

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

Situação: Ao utilizar o Claude Code em uma sessão longa de refatoração, o modelo começa a apresentar respostas mais lentas e omite convenções definidas no início da sessão. Qual comando nativo da CLI pode ser executado para condensar o histórico e liberar a janela de contexto?

Qual comando nativo usar?

- A) `/reset`
- B) `/compact` **[CORRETA]**
- C) `/clear`
- D) `/clean-context`

Por que a B: O comando `/compact` sumariza o histórico de conversa acumulado até o momento, reduzindo o uso da janela de contexto e restaurando o desempenho e a atenção do modelo.

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

Situação: Sua equipe deseja adicionar um servidor MCP do GitHub para buscar PRs e verificar status de CI via Claude Code. Cada um dos seis desenvolvedores possui seu próprio token de acesso pessoal do GitHub. Você quer ferramentas consistentes em toda a equipe sem commitar credenciais no controle de versão.

Qual abordagem de configuração é mais eficaz?

- A) Fazer com que cada desenvolvedor adicione o servidor no escopo de usuário via `claude mcp add --scope user`.
- B) Criar um wrapper de servidor MCP que leia tokens de um arquivo `.env` e faça proxy das chamadas da 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 autenticação e documentar a variável de ambiente necessária no README do projeto. **[CORRETA]**
- D) Configurar o servidor no escopo do projeto com um token de marcador de posição (placeholder) e dizer aos desenvolvedores para sobrescrevê-lo em sua configuração local.

Por que a C: Um arquivo `.mcp.json` de projeto com substituição de variável de ambiente é a forma idiomática: fornece uma fonte única da verdade versionada para a configuração do MCP enquanto permite que cada desenvolvedor forneça credenciais via variáveis de ambiente. Documentar a variável facilita a integração (*onboarding*) sem commitar segredos.

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

Situação: Você está adicionando wrappers de tratamento de erro em torno de chamadas de API externas em uma base de código de 120 arquivos. O trabalho possui três fases: (1) descobrir todos os locais de chamada e padrões, (2) projetar colaborativamente a abordagem de tratamento de erros, e (3) implementar os wrappers de forma consistente. Na Fase 1, o Claude gera uma saída grande listando centenas de locais de chamada com contexto, preenchendo rapidamente a janela de contexto antes que a descoberta termine.

Qual abordagem é mais eficaz para concluir a tarefa mantendo a consistência da implementação?

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

Por que a A: Um subagente Explore isola a saída verbosa de descoberta em um contexto separado e retorna apenas um resumo conciso para a conversa principal. Isso preserva a janela de contexto principal

para as fases de design colaborativo e implementação consistente onde o contexto mantido é mais valioso.

Cenário: Agente de Suporte ao Cliente

Questão 46 (Cenário: Agente de Suporte ao Cliente)

Situação: Durante os testes, você nota que o agente frequentemente chama `get_customer` quando os usuários perguntam sobre o status do pedido, embora `lookup_order` fosse mais apropriado. O que você deve verificar primeiro para resolver este problema?

O que você deve verificar primeiro?

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

Por que a D: As descrições das ferramentas são a entrada primária que o modelo usa para decidir qual ferramenta chamar. Quando um agente escolhe consistentemente a ferramenta errada, o primeiro passo diagnóstico é verificar se as descrições das ferramentas separam claramente o propósito de cada uma e suas fronteiras de uso.

Questão 47 (Cenário: Agente de Suporte ao Cliente)

Situação: Seu agente lida com solicitações de problema único com 94% de precisão (ex: "preciso de um reembolso para o pedido #1234"). Mas quando os clientes incluem múltiplos problemas em uma única mensagem (ex: "preciso de um reembolso para o pedido #1234 e também quero atualizar o endereço de entrega do pedido #5678"), a precisão da seleção de ferramentas cai para 58%. O agente geralmente resolve apenas um problema ou mistura parâmetros entre as solicitações. Qual abordagem melhora com mais eficácia a confiabilidade para solicitações de múltiplos problemas?

Qual abordagem é mais eficaz?

- A) Implementar uma camada de pré-processamento que use uma chamada de modelo separada para decompor mensagens de múltiplos problemas em solicitações separadas, tratar cada uma independentemente e mesclar os resultados.
- B) Combinar ferramentas relacionadas em menos ferramentas universais.
- C) Adicionar exemplos few-shot ao prompt demonstrando o raciocínio correto e o encadeamento de ferramentas para solicitações de múltiplos problemas. **[CORRETA]**

- D) Implementar validação de resposta que detecte respostas incompletas e reprompte automaticamente o agente para resolver problemas ignorados.

Por que a C: Exemplos few-shot que demonstram o raciocínio correto e o sequenciamento de ferramentas para solicitações de múltiplos problemas são mais eficazes porque o agente já se comporta bem em problemas únicos — o que ele precisa é de orientação sobre o padrão para decompor e rotear múltiplos problemas mantendo os parâmetros separados.

Questão 48 (Cenário: Agente de Suporte ao Cliente)

Situação: Logs de produção mostram que para solicitações simples como "reembolso para o pedido #1234", seu agente resolve o problema em 3–4 chamadas de ferramentas com 91% de sucesso. Mas para solicitações complexas como "fui cobrado duas vezes, meu desconto não foi aplicado e quero cancelar", o agente calcula em média 12+ chamadas de ferramentas com apenas 54% de sucesso — frequentemente investigando problemas de forma sequencial e buscando dados redundantes do cliente para cada um. Qual alteração melhora de forma mais eficaz o tratamento de solicitações complexas?

Qual alteração é mais eficaz?

- A) Adicionar pontos de checagem explícitos de verificação entre as etapas, exigindo que o agente registre o progresso após resolver cada problema antes de passar para o próximo.
- B) Reduzir o número de ferramentas combinando `get_customer`, `lookup_order` e ferramentas de cobrança em uma única ferramenta `investigate_issue`.
- C) Decompor a solicitação em problemas separados, e depois investigar cada um em paralelo usando o contexto compartilhado do cliente antes de sintetizar uma resolução final. **[CORRETA]**
- D) Adicionar exemplos few-shot ao system prompt demonstrando sequências ideais de chamadas de ferramentas para vários cenários complexos de cobrança.

Por que a C: Decompor em problemas separados e investigar em paralelo com contexto compartilhado do cliente corrige ambos os problemas principais: elimina a busca redundante de dados ao reutilizar o contexto compartilhado entre os problemas e reduz os loops totais de chamadas de ferramentas ao paralelizar a 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 produtos danificados com foto comprovante) enquanto tenta lidar autonomamente com situações complexas que exigem exceções de políticas. Qual é a forma mais eficaz de melhorar a calibração de escalação?

Qual é a forma mais eficaz de melhorar a calibração de escalação?

- A) Exigir que o agente auto-avalie a confiança em uma escala de 1–10 antes de cada resposta e roteie automaticamente para humanos quando a confiança cair abaixo de um limite.
- B) Implantar um modelo classificador separado treinado em chamados históricos para prever quais solicitações precisam de escalação antes que o agente principal comece o processamento.

- C) Adicionar critérios explícitos de escalação ao system prompt com exemplos few-shot mostrando quando escalar versus resolver autonomamente. **[CORRETA]**
- D) Implementar análise de sentimento para determinar o nível de frustração do cliente e escalar automaticamente ao ultrapassar um limite de sentimento negativo.

Por que a C: Critérios explícitos de escalação com exemplos few-shot abordam diretamente a causa raiz — fronteiras de decisão não claras entre casos simples e complexos. É a intervenção inicial mais proporcional e eficaz, que ensina 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 possui todos os dados do sistema disponíveis mas ainda enfrenta incerteza. Qual situação é o gatilho mais justificado para chamar `escalate_to_human`?

Qual situação é mais justificada para escalação?

- A) Um cliente deseja cancelar um pedido enviado ontem e que chega amanhã. O agente deve escalar porque o cliente pode mudar de ideia após receber o pacote.
- B) Um cliente afirma que não recebeu um pedido, mas o rastreamento mostra que ele foi entregue e assinado em seu endereço há três dias. O agente deve escalar porque apresentar provas contraditórias pode prejudicar o relacionamento com o cliente.
- C) Um cliente solicita cobrir o preço do concorrente. Suas políticas permitem ajustes de preço para quedas de valor no seu próprio site dentro de 14 dias, mas não dizem nada sobre preços de concorrentes. O agente deve escalar para interpretação da política. **[CORRETA]**
- D) Uma mensagem de cliente contém tanto uma dúvida de cobrança quanto uma devolução de produto. O agente deve escalar para que um humano possa coordenar ambos os problemas em uma única interação.

Por que a C: Esta é uma lacuna genuína de política: as regras da empresa cobrem quedas de preço no próprio site mas não abordam cobrir preços de concorrentes. O agente não deve inventar políticas e deve escalar para julgamento humano sobre como interpretar ou estender as regras existentes.

Questão 51 (Cenário: Agente de Suporte ao Cliente)

Situação: Logs de produção mostram que em 12% dos casos seu agente ignora `get_customer` e chama `lookup_order` diretamente usando apenas o nome fornecido pelo cliente, às vezes levando a contas identificadas incorretamente e reembolsos errados. Qual alteração corrige com mais eficácia este problema de confiabilidade?

Qual alteração é mais eficaz?

- A) Adicionar exemplos few-shot mostrando que o agente sempre chama `get_customer` primeiro, mesmo quando os clientes fornecem voluntariamente detalhes do pedido.

- B) Implementar um classificador de roteamento que analise cada solicitação e habilite apenas um subconjunto de ferramentas apropriadas para aquele tipo de solicitação.
- C) Adicionar uma pré-condição programática que bloqueie `lookup_order` e `process_refund` até que `get_customer` retorne um identificador de cliente verificado. **[CORRETA]**
- D) Reforçar o system prompt declarando que a verificação do cliente via `get_customer` é obrigatória antes de qualquer operação de pedido.

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

Questão 52 (Cenário: Agente de Suporte ao Cliente)

Situação: Métricas de produção mostram que ao resolver disputas de cobrança complexas ou devoluções de múltiplos pedidos, a satisfação do cliente é 15% menor do que em casos simples — mesmo quando a resolução está tecnicamente correta. A análise da causa raiz mostra que o agente fornece soluções precisas mas explica a justificativa de forma inconsistente: às vezes omitindo detalhes relevantes de políticas, às vezes perdendo informações de prazos ou próximos passos. As lacunas específicas de contexto variam de caso para caso. Você deseja melhorar a qualidade da solução sem adicionar supervisão humana. Qual abordagem é mais eficaz?

Qual abordagem é mais eficaz?

- A) Adicionar um estágio de autocrítica onde o agente avalie um rascunho de resposta quanto à completude — garantindo que ele resolva o problema do cliente, inclua contexto relevante e antecipe perguntas de acompanhamento. **[CORRETA]**
- B) Adicionar um estágio de confirmação onde o agente pergunte "Isso resolve totalmente o seu problema?" antes de fechar, permitindo que os clientes solicitem informações adicionais se necessário.
- C) Atualizar o modelo de Haiku para Sonnet para 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 políticas, prazos e próximos passos.

Por que a A: Um estágio de autocrítica (o padrão avaliador-otimizador) aborda diretamente a completude inconsistente da explicação ao forçar o agente a avaliar seu próprio rascunho contra critérios concretos — tais como contexto de políticas, prazos 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 calcula em média 4+ loops de API por resolução. A análise revela que o Claude frequentemente solicita `get_customer` e `lookup_order` em turnos sequenciais separados mesmo quando ambos são necessários inicialmente. Qual é a forma mais eficaz de reduzir o número de loops?

Qual é a forma mais eficaz de reduzir loops?

- A) Implementar execução especulativa que chame automaticamente ferramentas provavelmente necessárias em paralelo com qualquer ferramenta solicitada e retorne todos os resultados independentemente do que foi solicitado.
- B) Aumentar `max_tokens` para dar ao Claude mais espaço para planejar e combinar naturalmente solicitações de ferramentas.
- C) Criar ferramentas compostas como `get_customer_with_orders` que agrupem combinações comuns de busca em chamadas únicas.
- D) Instruir o Claude no prompt a agrupar solicitações de ferramentas em um único turno e retornar todos os resultados juntos antes da próxima chamada de API. **[CORRETA]**

Por que a D: Instruir o Claude a agrupar solicitações de ferramentas relacionadas em um único turno aproveita sua capacidade nativa de solicitar múltiplas ferramentas de uma vez. Corrige diretamente o padrão de chamadas sequenciais com alteração 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. A investigação mostra que esses detalhes foram mencionados 20+ turnos atrás e condensados em resumos vagos como "preços promocionais foram discutidos". Qual correção é mais eficaz?

Qual correção é mais eficaz?

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

Por que a C: A sumarização inerentemente perde detalhes precisos. Extrair fatos transacionais em um bloco estruturado de "fatos do caso" fora do histórico sumarizado preserva informações críticas para que fiquem disponíveis de forma confiável em todo prompt independentemente 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 existem múltiplos resultados, o Claude escolhe o cliente com o pedido mais recente, mas dados de produção mostram que isso seleciona a conta errada 15% das vezes para correspondências ambíguas. Como você deve resolver isso?

Como você deve resolver isso?

- A) Implementar um sistema de pontuação de confiança que atue autonomamente acima de 85% de confiança e solicite esclarecimento abaixo do limite.
- B) Instruir o Claude a solicitar 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 um algoritmo de classificação, eliminando a ambiguidade.
- D) Adicionar exemplos few-shot ao prompt demonstrando o raciocínio correto e o sequenciamento de ferramentas para correspondências ambíguas.

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

Questão 56 (Cenário: Agente de Suporte ao Cliente)

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

Qual é a causa raiz mais provável?

- A) O system prompt contém instruções sensíveis a palavras-chave que direcionam o comportamento com base em termos como "conta", criando padrões não intencionais de seleção de ferramentas. **[CORRETA]**
- B) O treinamento base do modelo cria associações entre a terminologia de "conta" e operações relacionadas ao cliente que sobrescrevem as descrições das ferramentas.
- C) O modelo precisa de mais dados de treinamento em mensagens de múltiplos conceitos e deve passar por fine-tuning em exemplos contendo terminologia de conta e pedido.
- D) As descrições das ferramentas precisam de exemplos negativos adicionais especificando quando NÃO usar cada ferramenta para evitar essa confusão induzida por palavras-chave.

Por que a A: O padrão sistemático impulsionado por palavras-chave (78% vs 93%) indica fortemente uma lógica de roteamento explícita no system prompt reagindo à palavra "conta" e direcionando o agente para ferramentas relacionadas ao cliente. Como as descrições das ferramentas 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 escolhe `get_customer` quando os usuários perguntam sobre pedidos (ex: "verifique meu pedido #12345") em vez de chamar `lookup_order`. Ambas as ferramentas possuem descrições mínimas ("Obtém informações do cliente" / "Obtém detalhes do pedido") e aceitam formatos de identificador com aparência semelhante. Qual é o primeiro passo mais eficaz para melhorar a confiabilidade da seleção de ferramentas?

Qual é o primeiro passo mais eficaz?

- A) Implementar uma camada de roteamento que analise a entrada do usuário antes de cada turno e pré-selecione a ferramenta correta com base em palavras-chave e padrões de ID detectados.
- B) Combinar ambas as ferramentas em uma única `lookup_entity` que aceite qualquer identificador e decida internamente qual backend consultar.
- C) Adicionar exemplos few-shot ao system prompt demonstrando padrões corretos de seleção de ferramentas, com 5–8 exemplos roteando consultas relacionadas a pedidos para `lookup_order`.
- D) Expandir a descrição de cada ferramenta para incluir formatos de entrada, exemplos de consultas, casos de borda e limites explicando quando usá-la em relação a ferramentas semelhantes. **[CORRETA]**

Por que a D: Expandir as descrições das ferramentas com formatos de entrada, exemplos de consultas, casos de borda e limites claros corrige diretamente a causa raiz — descrições mínimas que não dão ao LLM informações suficientes para diferenciar ferramentas semelhantes. É um primeiro passo de baixo esforço e alto impacto que melhora o mecanismo primário que o LLM usa para seleção de ferramentas.

Questão 58 (Cenário: Agente de Suporte ao Cliente)

Situação: Você está implementando o loop do agente para seu agente de suporte. Após cada chamada da API do Claude, você deve decidir se continua o loop (executa ferramentas solicitadas e chama o Claude novamente) ou para (apresenta a resposta final ao cliente). O que determina esta decisão?

O que determina esta decisão?

- A) Verificar o campo `stop_reason` na resposta do Claude — continuar se for `tool_use` e parar se for `end_turn`. **[CORRETA]**
- B) Analisar o texto do Claude em busca de frases como "Terminei" ou "Posso ajudar com mais alguma coisa?" — sinais em linguagem natural indicam a conclusão da tarefa.
- C) Definir uma contagem máxima de iterações (ex: 10 chamadas) e parar quando atingida, independentemente de o Claude indicar que mais trabalho é necessário.
- D) Verificar se a resposta contém conteúdo de texto do assistente — se o Claude gerou texto explicativo, o loop deve ser encerrado.

Por que a A: `stop_reason` é o sinal estruturado explícito do Claude para controle de loop: `tool_use` indica que o Claude deseja executar uma ferramenta e receber os resultados de volta, enquanto `end_turn` indica que o Claude concluiu sua resposta e o loop deve terminar.

Questão 59 (Cenário: Agente de Suporte ao Cliente)

Situação: Logs de produção mostram que o agente interpreta mal as saídas de suas ferramentas MCP: timestamps Unix do `get_customer`, datas ISO 8601 do `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 manutenível?

Qual abordagem é mais manutenível?

- A) Usar um hook PostToolUse para interceptar saídas de ferramentas e aplicar transformações de formatação antes que o agente as processe. **[CORRETA]**
- B) Modificar ferramentas que você controla para retornar formatos legíveis por humanos e criar wrappers para ferramentas de terceiros.
- C) Criar uma ferramenta `normalize_data` que o agente chame após cada busca de dados para transformar valores.
- D) Adicionar documentação detalhada de formato ao system prompt explicando as convenções de dados de cada ferramenta.

Por que a A: Um hook PostToolUse fornece um ponto centralizado e determinístico para interceptar e normalizar todas as saídas de ferramentas — incluindo dados de servidores MCP de terceiros — antes que o agente as processe. É mais manutenível porque as transformações residem no código e são aplicadas uniformemente, em vez de 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 consultas ambíguas como "preciso de ajuda com minha compra recente". Você decide adicionar exemplos few-shot ao system prompt para melhorar a seleção de ferramentas. Qual abordagem aborda o problema com mais eficácia?

Qual abordagem é mais eficaz?

- A) Adicionar orientações explícitas "use quando" e "não use quando" na descrição de cada 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 aos cenários ambíguos, cada um com justificativa de por que uma ferramenta foi escolhida em relação a alternativas plausíveis. **[CORRETA]**
- D) Adicionar 10–15 exemplos de solicitações claras e inequívocas demonstrando a escolha correta de ferramentas para cenários típicos para cada ferramenta.

Por que a C: Direcionar exemplos few-shot para os cenários ambíguos específicos onde ocorrem erros, com justificativa explícita de por que uma ferramenta é preferível a alternativas, ensina ao modelo o processo comparativo de decisão necessário para casos de borda. Isso é mais eficaz do 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. O monitoramento de produção mostra que o agente ignora a etapa de prévia chamando diretamente com `dry_run=false`. Você precisa garantir que cada remoção seja precedida por uma prévia que o usuário confirme explicitamente.

Qual é a abordagem mais confiável?

- A) Adicionar validação no lado do servidor que permita `dry_run=false` apenas quando uma chamada `dry_run=true` com parâmetros idênticos tiver ocorrido nos últimos 60 segundos.
- B) Anotar a ferramenta como exigindo confirmação e configurar a camada de orquestração para solicitar aprovação do usuário antes de encaminhar chamadas para ferramentas anotadas.
- C) Adicionar instruções detalhadas e exemplos few-shot à descrição da ferramenta exigindo que o agente sempre chame com `dry_run=true` primeiro e aguarde a confirmação do usuário antes de chamar novamente.
- D) Substituir por duas ferramentas: `preview_remove_member` retorna detalhes do impacto e um token de confirmação de uso único; `execute_remove_member` exige esse token, vinculando a execução à prévia. **[CORRETA]**

Por que a D: A abordagem de vinculação por token de duas ferramentas torna arquiteturalmente impossível executar sem uma prévia anterior — a ferramenta de execução exige literalmente um token que apenas a ferramenta de prévia pode gerar. Esta é a única abordagem que impõe a restrição no nível do código em vez de depender da conformidade do LLM com instruções (C), heurísticas de tempo (A) ou infraestrutura de orquestração (B).

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

Situação: O monitoramento de produção mostra que sua ferramenta `search_catalog` falha 12% das vezes: 8% são timeouts de rede que obtêm sucesso quando retentados, e 4% são erros de sintaxe de consulta que nunca obtêm sucesso independentemente das retentativas. Atualmente ambos os tipos de erro são retornados de forma idêntica, causando retentativas desperdiçadas.

Como você deve modificar o tratamento de erros da ferramenta?

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

Por que a C: Tratar retentativas no nível da ferramenta para erros transitórios é a fronteira de abstração correta — a ferramenta possui conhecimento definitivo do tipo de erro e pode implementar uma lógica determinística de retentativa sem depender de o agente interpretar uma flag (D) ou seguir instruções no

nível do prompt (A). O backoff uniforme (B) desperdiça tempo em erros de sintaxe que nunca obterão sucesso.

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

Situação: Ao longo de vários turnos discutindo estratégia de investimento, um usuário declarou "tenho uma tolerância ao risco muito baixa" e mais tarde "quero maximizar meus retornos". Ele agora pergunta: "Em que devo investir?"

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

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

Por que a A: Quando as preferências do usuário se contradizem diretamente, trazer o conflito à tona e pedir esclarecimentos é a única forma de garantir que a recomendação se alinhe à verdadeira intenção do usuário. Qualquer outra abordagem envolve fazer uma suposição que pode estar errada — maximizar retornos e ter baixa tolerância ao risco são objetivos fundamentalmente incompatíveis que exigem uma decisão humana.

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

Situação: Os usuários refinam preferências de listas de reprodução (*playlists*) ao longo de múltiplos turnos de conversa. Duas mensagens após um usuário dizer "eu amo jazz", o Claude pergunta "De quais gêneros você gosta?"

Qual é a causa mais provável?

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

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

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

Situação: Após uma sessão de culinária de 40 minutos, a conversa atinge 78.000 tokens. O histórico inclui alergias, redimensionamento de receitas, termos culinários esclarecidos e discussões gerais. Você deve reduzir os tokens enquanto preserva informações importantes.

Qual abordagem equilibra melhor a preservação com a 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 discussões gerais e manter as trocas recentes verbatim. **[CORRETA]**
- D) Armazenar a conversa completa externamente e recuperar partes relevantes via busca semântica.

Por que a C: A abordagem híbrida preserva as informações de maior valor com o menor custo. Fatos críticos como alergias e quantidades de receitas são extraídos para um bloco estruturado compacto (evitando a perda de precisão que ocorre na sumarização), discussões gerais são sumarizadas e trocas recentes são mantidas verbatim para coerência conversacional. As opções A e B arriscam perder informações dietéticas críticas; D é um exagero arquitetural para uma única sessão de culinária.

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

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

Qual é a solução mais eficaz?

- A) Abordagem híbrida: sumarizar mensagens mais antigas enquanto mantém as recentes verbatim. **[CORRETA]**
- B) Busca por similaridade vetorial sobre o histórico completo de conversa.
- C) Aumentar a janela para 50 pares de mensagens.
- D) Sumarizar mensagens descartadas a cada turno e prender o resumo contínuo.

Por que a A: A abordagem híbrida atende a ambas as dimensões do problema: reter contexto recente exato (crítico para a coerência conversacional) enquanto mantém uma representação comprimida das preferências anteriores (evitando a perda total quando pares são descartados). Aumentar a janela (C) apenas adia o mesmo problema. A busca vetorial (B) pode perder contextos importantes que não sejam semanticamente semelhantes à consulta atual. A sumarização total por turno (D) adiciona sobrecarga e acumula erros de sumarização.

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

Situação: Usuários relatam que a latência aumenta e os custos sobem quando as conversas ultrapassam 50 turnos.

Qual é a causa primária?

- A) O histórico completo de conversa é incluído em cada requisição de API. **[CORRETA]**
- B) O modelo gera respostas progressivamente mais longas.
- C) As operações de banco de dados ficam mais lentas à medida que o histórico cresce.
- D) O modelo constrói um perfil interno de usuário exigindo mais processamento.

Por que a A: A API do Claude é totalmente *stateless* — cada requisição deve incluir o histórico completo da conversa no array `messages`. À medida que as conversas crescem, cada requisição carrega mais tokens, o que aumenta diretamente tanto a latência de processamento quanto o custo. O modelo não mantém nenhum estado interno entre chamadas (D é falso), e a extensão da resposta não está inerentemente ligada ao tamanho da conversa (B).

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

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

Qual é a abordagem mais eficaz?

- A) Truncamento de janela deslizante (*rolling window*).
- B) Sumarização progressiva capturando conclusões chaves.
- C) Embeddings semânticos com recuperação de trocas relevantes. **[CORRETA]**
- D) Adicionar tags XML estruturadas marcando conclusões de discussão.

Por que a C: A busca semântica sobre o histórico de conversa é a única abordagem que se adapta a três meses de discussão enquanto consegue trazer à tona trocas específicas relevantes sob demanda. A janela deslizante (A) descartaria a maior parte do histórico. A sumarização progressiva (B) comprime discussões em abstrações que perdem as conclusões específicas sobre as quais os usuários estão perguntando. Tags XML (D) exigem a reestruturação de todo o conteúdo passado e não resolvem o problema de recuperação nesta escala.

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

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

tokens.

Qual é a melhor solução?

- A) Mover diretrizes de comportamento para a primeira mensagem do usuário.
- B) Iniciar uma nova conversa após 20 turnos.
- C) Inserir mensagens no papel de usuário reforçando diretrizes em pontos de quebra da conversa. **[CORRETA]**
- D) Usar validação pós-resposta para regenerar respostas não conformes.

Por que a C: A injeção periódica de lembretes comportamentais combate diretamente a variação de instruções (*instruction drift*), reestabelecendo restrições em intervalos regulares à medida que o histórico da conversa se acumula. Mover diretrizes para a primeira mensagem do usuário (A) reduz sua autoridade. Iniciar uma nova conversa (B) destrói o contexto. A validação pós-resposta (D) é corretiva e não preventiva, e adiciona latência significativa.

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

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

Qual é a correção mais eficaz?

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

Por que a B: Um system prompt de 2.800 tokens com regras declarativas é vulnerável ao desvio porque regras abstratas exigem que o modelo raciocine sobre elas em cada turno. Substituir regras verbosas por exemplos few-shot concretos que demonstrem a adaptação correta do nível de proficiência dá ao modelo padrões comportamentais claros para corresponder — isso é seguido com mais confiabilidade ao longo de muitos turnos do que instruções abstratas. Injeção de lembretes (A) ajuda mas aborda sintomas; posicionamento final (C) ajuda inicialmente mas não com desvios em nível de turno; regeneração (D) é cara e corretiva.

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

Situação: Seu assistente deve manter um tom entusiasmado, explicar seu raciocínio e fazer perguntas de esclarecimento. Onde essas diretrizes comportamentais devem ser definidas?

Onde essas diretrizes comportamentais devem ser definidas?

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

Por que a B: O system prompt foi projetado especificamente para restrições e diretrizes comportamentais persistentes que se aplicam a toda a conversa. Prepender em cada mensagem do usuário (A) é sobrecarga redundante. A primeira mensagem do assistente (C) não é confiável porque o modelo pode se desviar de suas próprias declarações anteriores. Variáveis de ambiente (D) não possuem efeito no comportamento do modelo.

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

Situação: Os usuários relatam aberturas de resposta repetitivas como "Certamente!" e "Ficarei feliz em ajudar!".

Qual é a abordagem mais eficaz?

- A) Anexar uma mensagem parcial de assistente (*prefill*) com uma abertura de resposta direta. **[CORRETA]**
- B) Diminuir a configuração de temperatura.
- C) Fazer pós-processamento das respostas para remover saudações.
- D) Adicionar instruções ao system prompt para evitar essas frases.

Por que a A: Pré-preencher (*prefill*) a resposta do assistente com o início de uma resposta direta evita padrões de saudação no nível da geração — o modelo continua a partir do pré-preenchimento 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 contorno frágil. Temperatura (B) controla a aleatoriedade, não padrões específicos de frases.

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

Situação: Um webhook notifica seu sistema de que o pacote de um usuário foi enviado enquanto o usuário está ativamente conversando no chat. Você deseja que o assistente incorpore isso naturalmente na próxima resposta.

Qual é a melhor abordagem?

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

Por que a D: Adicionar a atualização de status como prefixo para a próxima mensagem do usuário injeta contexto em tempo real em uma fronteira conversacional natural sem interromper o fluxo. Modificar o system prompt (A) exige reconstruir a sessão ou é arquiteturalmente incômodo. Uma mensagem sintética de usuário (B) pode quebrar o fluxo natural do diálogo e confundir a atribuição. Forçar uma chamada de ferramenta a cada turno (C) é um desperdício quando os eventos são raros.

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

Situação: Os usuários enviam frequentemente solicitações vagas como "Reserve um local para a festa". O assistente faz 4+ perguntas de esclarecimento, causando 35% de abandono.

Qual abordagem melhora melhor o compromisso (*trade-off*)?

- A) Prosseguir com padrões ocultos.
- B) Fazer todas as perguntas de esclarecimento em uma mensagem composta.
- C) Declarar suposições explicitamente e prosseguir enquanto convida a correções. **[CORRETA]**
- D) Usar um formulário de entrada estruturado.

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

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

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

Qual é a causa mais provável?

- A) System prompts estabelecem apenas o comportamento inicial.
- B) A atenção do modelo enfraquece à medida que os 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 a C: À medida que as respostas do assistente se acumulam no histórico da conversa, a proporção de texto refletindo as restrições comportamentais do system prompt diminui em relação ao corpo crescente de conteúdo gerado pelo próprio assistente. O modelo passa a corresponder cada vez mais a padrões de suas próprias saídas anteriores em vez do system prompt, agravando o desvio mesmo em comprimentos de tokens curtos. O system prompt é incluído em cada chamada de API (D é falso como explicação isolada), e a degradação de atenção do modelo (B) não atua em 2.500 tokens.

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

Situação: Os usuários fazem solicitações vagas como "Você pode ajudar com o relatório?". O assistente responde fazendo múltiplas perguntas (qual relatório? qual ajuda? qual o prazo?), causando 40% de abandono.

Qual é a melhor solução?

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

Por que a A: Prosseguir com suposições declaradas razoáveis elimina inteiramente o idas e voltas enquanto mantém o usuário informado e no controle. Interpretações silenciosas pré-definidas (C) deixam os usuários confusos quando a resposta não corresponde à sua intenção. Um limite de uma pergunta (D) ainda exige turnos de idas e voltas. 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-ferramentas com Lógica de Escalação

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

Passos:

1. Definir 3–4 ferramentas MCP com descrições detalhadas (inclua duas ferramentas semelhantes para testar a seleção de ferramentas)
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 bloqueie operações acima de um limite e redirecione para escalação
5. Testar com solicitações de múltiplos aspectos

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

Exercício 2: Configurando o Claude Code para Desenvolvimento em Equipe

Objetivo: Configurar CLAUDE.md, comandos customizados, regras específicas por caminho e servidores MCP.

Passos:

1. Criar um CLAUDE.md em nível de projeto com padrões universais
2. Criar arquivos em `.claude/rules/` com frontmatter YAML para diferentes áreas de código (`paths: ["src/api/**/*"]`, `paths: ["**/*.test.*"]`)
3. Criar uma skill de projeto em `.claude/skills/` com `context: fork` e `allowed-tools`
4. Configurar um servidor MCP no `.mcp.json` com variáveis de ambiente + um override pessoal em `~/.claude.json`
5. Testar o modo de planejamento vs execução direta em tarefas de diferentes complexidades

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

Exercício 3: Pipeline de Extração de Dados Estruturados

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

Passos:

1. Definir uma ferramenta de extração com um esquema JSON (campos obrigatórios/opcionais, enums com "other", campos anuláveis)
2. Construir um loop de validação: ao ocorrer erro, tentar novamente 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. Roteamento para humanos: pontuações de confiança em nível de 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 rastreamento de fontes.

Passos:

1. Um coordenador com 2+ subagentes (`allowedTools` inclui `"Task"`, contexto é passado explicitamente em prompts)

2. Rodar subagentes em paralelo via múltiplas chamadas `Task` em uma única resposta
3. Exigir saída estruturada de subagentes: afirmação, citação, URL da fonte, data de publicação
4. Simular um 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 agentes), 2 (Ferramentas e MCP), 5 (Contexto e confiabilidade)

Apêndice: Tecnologias e Conceitos

Tecnologia	Aspectos chave
Claude Agent SDK	AgentDefinition, loops de agentes, `stop_reason`, hooks (PostToolUse), disparo de subagentes via Task, `allowedTools`
Model Context Protocol (MCP)	Servidores MCP, ferramentas, recursos, `isError`, descrições de ferramentas, `.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 esquemas JSON, `tool_choice` ("auto"/"any"/forçada), `stop_reason`, `max_tokens`, system prompts
Message Batches API	50% de economia, janela de até 24h, `custom_id`, sem chamadas de ferramentas multi-turnos
Esquema JSON (JSON Schema)	Obrigatório vs opcional, campos anuláveis, tipos enum, "other" + detalhe, modo estrito
Pydantic	Validação de esquema, erros semânticos, loops de validação/retentativa
Ferramentas Nativas (*Built-in*)	Read, Write, Edit, Bash, Grep, Glob — propósito e critérios de seleção
Prompting Few-shot	Exemplos direcionados para situações ambíguas, generalização para novos padrões
Encadeamento de Prompts	Decomposição sequencial em passagens focadas
Janela de Contexto	Orçamentos de tokens, sumarização progressiva, "lost in the middle", arquivos de rascunho (*scratchpad*)

Gerenciamento de Sessão	Resume, `fork_session`, sessões nomeadas, isolamento de contexto
Calibração de Confiança	Pontuação em nível de campo, calibração em conjuntos rotulados, amostragem estratificada

---

Tópicos Fora do Escopo

Os seguintes tópicos correlatos **NÃO** serão cobrados no exame:

- Ajuste fino (*Fine-tuning*) de modelos Claude ou treinamento de modelos customizados
- Autenticação na API do Claude, faturamento ou gerenciamento de conta
- Implementação detalhada em linguagens de programação ou frameworks específicos (além do necessário para configuração de ferramentas/esquemas)
- Implantação ou hospedagem de servidores MCP (infraestrutura, redes, orquestração de contêineres)
- Arquitetura interna do Claude, processo de treinamento ou pesos do modelo
- IA Constitucional (*Constitutional AI*), RLHF ou metodologias de treinamento de segurança
- Modelos de embedding ou detalhes de implementação de bancos de dados vetoriais
- Uso de computador (*Computer use* — automação de navegador, interação com desktop)
- Capacidades de análise de imagem (Visão)
- API de streaming ou eventos enviados pelo servidor (*server-sent events*)
- Limites de taxa (*rate limiting*), cotas ou cálculos detalhados de custos de API
- OAuth, rotação de chaves de API ou detalhes de protocolos de autenticação
- Configurações específicas de provedores de nuvem (AWS, GCP, Azure)
- Benchmarks de desempenho ou métricas de comparação de modelos
- Detalhes de implementação de cache de prompt (além de saber que ele existe)
- Contagem detalhada de tokens ou algoritmos de tokenização