Claude Certified Architect — Certificazione Foundations

Guida allo Studio (Basata sulla Guida Ufficiale all'Esame)

Introduzione

La certificazione **Claude Certified Architect — Foundations** attesta che uno specialista è in grado di prendere decisioni ponderate sui compromessi progettuali (trade-off) durante l'implementazione di soluzioni reali basate su Claude. L'esame valuta le conoscenze fondamentali di Claude Code, del Claude Agent SDK, della Claude API e del Model Context Protocol (MCP), le tecnologie principali per la realizzazione di applicazioni di produzione con Claude.

Le domande d'esame si basano su scenari realistici del settore: sviluppo di sistemi agentici per l'assistenza clienti, progettazione di pipeline di ricerca multi-agente, integrazione di Claude Code nei processi CI/CD, creazione di strumenti per aumentare la produttività degli sviluppatori ed estrazione di dati strutturati da documenti non strutturati.

Candidato Ideale

Il candidato ideale è un **solution architect** che progetta e rilascia applicazioni in produzione basate su Claude. È consigliata un'esperienza pratica di almeno 6 mesi nei seguenti ambiti:

- **Claude Agent SDK** — orchestrazione multi-agente, delega a sottoagenti, integrazione di strumenti (tool), hook del ciclo di vita (lifecycle hooks)
- **Claude Code** — CLAUDE.md, server MCP, Agent Skills, modalità di pianificazione (planning mode)
- **Model Context Protocol (MCP)** — strumenti e risorse per l'integrazione con sistemi backend
- **Prompt engineering** — schemi JSON, esempi few-shot, modelli per l'estrazione dei dati
- **Finestre di contesto (context windows)** — gestione di documenti lunghi e passaggio del contesto tra più agenti
- **Pipeline CI/CD** — revisione automatizzata del codice e generazione di test
- **Gestione delle escalation e affidabilità** — gestione degli errori e approcci human-in-the-loop

---

Formato dell'Esame

Parametro	Valore
Tipo di domanda	Scelta multipla (1 risposta corretta su 4)
Punteggio	Scala da 100 a 1000, punteggio minimo per il superamento **720**

Penalità per risposta errata	Nessuna (rispondi a tutte le domande!)
Scenari	4 su 8 possibili (selezionati casualmente)

Contenuto dell'Esame: 5 Domini

Dominio	Peso
1. Architettura e orchestrazione degli agenti	**27%**
2. Progettazione di strumenti e integrazione MCP	**18%**
3. Configurazione e workflow di Claude Code	**20%**
4. Prompt engineering e output strutturato	**20%**
5. Gestione del contesto e affidabilità	**15%**

Scenari d'Esame

Scenario 1: Agente per l'assistenza al cliente

Viene costruito un agente per gestire resi, contestazioni di fatturazione e problemi relativi agli account utilizzando il Claude Agent SDK. L'agente utilizza strumenti MCP (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`). L'obiettivo è raggiungere un tasso di risoluzione al primo contatto superiore all'80%, con escalation appropriata quando necessario.

Scenario 2: Generazione di codice con Claude Code

Claude Code viene utilizzato per accelerare lo sviluppo: generazione di codice, refactoring, debug e documentazione. È necessario integrarlo con comandi personalizzati e configurazione tramite CLAUDE.md, oltre a comprendere quando utilizzare la modalità di planning.

Scenario 3: Sistema di ricerca multi-agente

Un agente coordinatore delega attività a subagenti specializzati: ricerca web, analisi documentale, sintesi e generazione di report. Il sistema deve produrre report completi con citazioni.

Scenario 4: Strumenti di produttività per sviluppatori

L'agente supporta gli sviluppatori nell'esplorazione di codebase sconosciute, nella generazione di codice boilerplate e nell'automazione di attività ripetitive. Vengono utilizzati strumenti integrati (Read, Write, Bash, Grep, Glob) e server MCP.

Scenario 5: Claude Code per la Continuous Integration

Integrazione di Claude Code in una pipeline CI/CD per revisione automatica del codice, generazione di test e feedback sulle pull request. I prompt devono essere progettati per ridurre al minimo i falsi positivi.

Scenario 6: Estrazione di Dati Strutturati

Il sistema estrae informazioni da documenti non strutturati, valida l'output tramite JSON schema e mantiene un'elevata accuratezza. Deve gestire correttamente anche i casi limite.

Scenario 7: Pattern di architettura per AI conversazionale

Si progettano sistemi conversazionali multi-turno che includono gestione della finestra di contesto, persistenza delle istruzioni tra i turni, strategie di memoria, progettazione di strumenti per l'esecuzione sicura e gestione di input utente ambigui o contraddittori.

Scenario 8: Strumenti di AI agentica *(contenuto mancante — aiutaci!)*

Questo scenario è stato segnalato dai candidati all'esame ma non è ancora coperto in questa guida. Se hai incontrato domande relative a questo scenario durante l'esame reale, condividili qui GitHub Issues così da aumentare la completezza della guida. Il tuo contributo aiuterà tutti nella preparazione dell'esame.

Official Documentazione

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

Claude Code — Skills (inclusi i comandi slash)	https://code.claude.com/docs/en/skills
Claude Code — Hooks	https://code.claude.com/docs/en/hooks
Claude Code — Sub-agents	https://code.claude.com/docs/en/sub-agents
Claude Code — MCP Integration	https://code.claude.com/docs/en/mcp
Claude Code — GitHub Actions CI/CD	https://code.claude.com/docs/en/github-actions
Claude Code — GitLab CI/CD	https://code.claude.com/docs/en/gitlab-ci-cd
Claude Code — Headless (modalità non interattiva)	https://code.claude.com/docs/en/headless
Prompt Engineering Guide	https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview
Extended Thinking	https://platform.claude.com/docs/en/build-with-claude/extended-thinking
Anthropic Cookbook (esempi di codice)	https://github.com/anthropics/anthropic-cookbook

---

PARTE I: FONDAMENTI TEORICI

Questa parte copre tutta la teoria necessaria per superare con successo l'esame. Il materiale è organizzato per tecnologie e concetti, piuttosto che per domini d'esame: questo approccio aiuta a costruire una comprensione più profonda di ciascun argomento.

Capitolo 1: Claude API — Fondamenti dell'Interazione con il Modello

> Documentazione: Messages API | Prompt Engineering

1.1 Struttura delle Richieste API

La API Claude seguono un modello request–response. Ogni richiesta alla Messages API di Claude include:

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

Campi principali:

- `model` — selezione del modello (`claude-opus-4-6`, `claude-sonnet-4-6`, `claude-haiku-4-5`)
- `max_tokens` — numero massimo di token nella risposta
- `system` — system prompt (definisce il comportamento del modello)
- `messages` — storico della conversazione (**è necessario inviare sempre l'intera cronologia per mantenere la coerenza**)
- `tools` — definizione degli strumenti disponibili
- `tool_choice` — strategia di selezione degli strumenti

1.2 Ruoli dei Messaggi

L'array `messages` utilizza due ruoli conversazionali più un ruolo di istruzione:

- `user` — messaggi dell'utente, inclusi i risultati degli strumenti (inviati come blocco di contenuto `tool_result` all'interno di un messaggio con ruolo `user`, non come ruolo `tool` separato)
- `assistant` — risposte del modello (incluse quando si invia la cronologia), incluse le richieste di utilizzo degli strumenti (blocchi di contenuto `tool_use`)
- `system` — può essere impostato tramite il campo di primo livello `system` (si applica dal primo turno) oppure in linea all'interno di `messages` come `{"role": "system", ...}` (si applica da quel punto in poi, soggetto a regole di posizionamento specifiche — vedi sotto)

I risultati degli strumenti non vengono inviati come messaggio con ruolo `"tool"`. Vengono inviati come messaggio con ruolo `user` il cui contenuto include un blocco `tool_result`:

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

`system` può anche comparire direttamente come ruolo all'interno dell'array `messages`, non solo tramite il parametro di primo livello `system`. Questo serve per aggiungere istruzioni a metà conversazione senza invalidare il prefisso in cache del campo `system` di primo livello. Ha regole di posizionamento specifiche:

- Deve seguire immediatamente un turno `user` (incluso uno con blocchi `tool_result`) o un turno `assistant` che termina con l'uso di uno strumento lato server.
- Deve precedere un turno `assistant` o essere l'ultimo elemento dell'array.
- Non può trovarsi tra un blocco `tool_use` e il suo `tool_result` — farlo restituisce un errore 400.
- I messaggi `system` successivi (inclusi quelli inseriti a metà conversazione) hanno priorità su quelli precedenti e sul campo `system` di primo livello per i turni successivi.

Punto critico: ad ogni richiesta API è necessario inviare **l'intera cronologia della conversazione**. Il modello non mantiene stato tra le richieste: ogni chiamata è indipendente.

1.3 Campo `stop_reason` nella Risposta

La risposta delle API Claude include `stop_reason`, che indica il motivo per cui la generazione si è interrotta:

Valore	Descrizione	Azione
`"end_turn"`	Il modello ha completato la risposta	Mostrare il risultato all'utente
`"tool_use"`	Il modello richiede una chiamata a uno strumento	Eseguire lo strumento e restituire il risultato
`"max_tokens"`	Limite di token raggiunto	La risposta è troncata; può essere necessario aumentare il limite
`"stop_sequence"`	È stata incontrata una sequenza di stop	Gestire in base alla logica dell'applicazione

Per sistemi agentici, `"tool_use"` e `"end_turn"` sono i valori più importanti: governano il ciclo dell'agente.

1.4 System Prompt

Il System Prompt è un'istruzione speciale che definisce contesto e regole comportamentali. Le sue caratteristiche:

- Non fa parte dell'array `messages`; viene passato separatamente nel campo `system`
- Ha priorità rispetto ai messaggi dell'utente
- Viene caricato una sola volta e si applica durante tutta la conversazione
- Serve a definire ruolo, vincoli e formato dell'output

Importante per l'esame: la formulazione del system prompt può creare associazioni indesiderate con gli strumenti. Ad esempio, un'istruzione come "verifica sempre il cliente" può portare il modello a usare eccessivamente `get_customer`, anche quando non necessario.

1.5 Context Window

La context window è la quantità totale di testo (in token) che il modello può elaborare contemporaneamente. Include:

- System prompt
- Intera cronologia dei messaggi
- Definizioni degli strumenti
- Output degli strumenti

Problemi principali della context window:

1. **Effetto "lost-in-the-middle"**: il modello elabora meglio informazioni all'inizio e alla fine di un input lungo, ma può perdere dettagli nella parte centrale. Mitigazione: posizionare le informazioni chiave all'inizio o alla fine.
2. **Accumulazione dei risultati degli strumenti**: ogni chiamata a uno strumento aggiunge output al contesto. Se uno strumento restituisce 40 campi ma solo 5 sono rilevanti, gran parte del contesto viene sprecata.
3. **Summarization progressiva**: quando si comprime la cronologia, valori numerici, percentuali e date tendono a perdersi o diventare vaghi ("circa", "approssimativamente", "alcuni").

Capitolo 2: Strumenti e `tool_use`

> Documentazione: Tool Use

2.1 Cos'è `tool_use`

`tool_use` è un meccanismo che consente a Claude di chiamare funzioni esterne. Il modello non esegue direttamente codice: genera una richiesta strutturata di chiamata a uno strumento; il tuo codice esegue la

funzione e restituisce il risultato.

2.2 Definizione degli Strumenti

Ogni strumento viene definito tramite uno schema JSON:

```
{
  "name": "get_customer",
  "description": "Trova un cliente tramite email o ID. Restituisce il profilo del
cliente, inclusi nome, email, storico ordini e stato dell'account. Utilizzare questo
strumento PRIMA di lookup_order per verificare l'identità del cliente. Accetta una
email (formato: user@domain.com) o un customer_id numerico.",
  "input_schema": {
    "type": "object",
    "properties": {
      "email": {"type": "string", "description": "Email del cliente"},
      "customer_id": {"type": "integer", "description": "ID numerico del cliente"}
    },
    "required": []
  }
}
```

Aspetti critici della descrizione di uno strumento:

1. **La descrizione è il principale criterio di selezione.** Un LLM sceglie gli strumenti in base alle loro descrizioni. Descrizioni minimali ("Recupera informazioni cliente") portano facilmente a errori quando esistono tool simili.
2. **Nella descrizione includere:**
   - Cosa fa lo strumento e cosa restituisce
   - Formati di input ed esempi di valori
   - Casi limite e vincoli
   - Quando usarlo rispetto ad alternative simili
3. **Evitare** descrizioni identiche o sovrapposte tra strumenti. Se `analyze_content` e `analyze_document` hanno descrizioni quasi uguali, il modello tenderà a confonderli.
4. **Strumenti built-in vs MCP:** gli agenti possono preferire strumenti integrati (Read, Grep) rispetto a strumenti MCP con funzionalità simili. Per evitarlo, è necessario rafforzare le descrizioni degli strumenti MCP, evidenziando vantaggi concreti, dati esclusivi o contesti non disponibili nei tool integrati.

2.3 Parametro `tool_choice`

`tool_choice` controlla come il modello seleziona gli strumenti:

Valore	Comportamento	Quando usarlo
`{"type": "auto"}`	Il modello decide se chiamare uno strumento o rispondere in testo	Default nella maggior parte dei casi

`{"type": "any"}`	Il modello **deve** chiamare uno strumento	Quando è necessaria un'uscita strutturata garantita
`{"type": "tool", "name": "extract_metadata"}`	Il modello **deve** chiamare uno strumento specifico	Quando è necessario forzare un'azione iniziale o una sequenza di esecuzione

Scenari importanti:

- `tool_choice: "any"` + più strumenti di estrazione → il modello seleziona il più adatto, ma l'output rimane strutturato
- Selezione forzata → quando è necessario garantire una prima azione specifica (es. `extract_metadata` prima dell'arricchimento)

2.4 JSON Schema per Output Strutturato

L'uso di `tool_use` con JSON schema è il modo **più affidabile** per ottenere output strutturato da Claude. Consente di:

- Garantire JSON sintatticamente valido (nessuna parentesi mancante, nessuna virgola finale errata)
- Forzare una struttura coerente (campi obbligatori sempre presenti)
- **Non garantire la correttezza semantica** (i valori possono comunque essere errati)

Principi chiave di progettazione degli schema:

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": ["bug", "feature", "docs", "unclear", "other"]
    },
    "category_detail": {
      "type": ["string", "null"],
      "description": "Dettagli se category = 'other' o 'unclear'"
    },
    "severity": {
      "type": "string",
      "enum": ["critical", "high", "medium", "low"]
    },
    "confidence": {
      "type": "number",
      "minimum": 0,
      "maximum": 1
    },
    "optional_field": {
      "type": ["string", "null"],
      "description": "Null se l'informazione non è presente nella fonte"
    }
  },
  "required": ["category", "severity"]
}
```

Regole di progettazione degli schema:

1. **Required vs optional**: rendere obbligatori solo i campi sempre disponibili. Campi obbligatori possono spingere il modello a inventare valori quando l'informazione manca.
2. **Campi nullable**: usare `"type": ["string", "null"]` per informazioni potenzialmente assenti. Il modello può restituire null invece di inventare dati.
3. **Enum con `"other"`**: includere `"other"` + un campo descrittivo per non perdere informazioni fuori dalle categorie predefinite.
4. **Enum `"unclear"`**: per casi in cui il modello non può classificare con sicurezza; `"unclear"` è preferibile a una classificazione errata.

2.5 Errori Sintattici vs Semantici

Tipo di errore	Esempio	Mitigazione
Sintattico	JSON non valido, tipo di campo errato	`tool_use` con JSON schema (elimina il problema)
Semantico	Totali incoerenti, valore nel campo sbagliato, allucinazioni	Validazione esterna, retry con feedback, meccanismi di autocorrezione

Capitolo 3: Claude Agent SDK — Costruzione di Sistemi Agentici

> Documentazione: Agent SDK | Hooks | Subagents | Sessions

3.1 Cos’è un Agentic Loop

L’agentic loop è il pattern centrale per l’esecuzione autonoma di task.

```
Il modello non si limita a rispondere: esegue una sequenza di azioni:
1. Invia una richiesta a Claude con strumenti disponibili
2. Riceve una risposta
3. Controlla `stop_reason`:
  - "tool_use" -> esegue lo strumento, aggiunge il risultato alla cronologia e torna
al punto 1
  - "end_turn" -> il task è completato, mostra il risultato all’utente
4. Ripete fino al completamento
```

Approccio guidato dal modello: Claude decide quale strumento chiamare successivamente in base al contesto e ai risultati delle chiamate precedenti. Questo differisce da approcci a logica rigida (decision tree), in cui la sequenza è predefinita.

Anti-pattern da evitare:

- Analizzare il testo del modello per rilevare la conclusione (“Task completato”)
- Usare un limite arbitrario di iterazioni (es. `max_iterations=5`) come condizione primaria di stop
- Considerare la presenza di testo come segnale di completamento

Approccio corretto: l’unico segnale affidabile di completamento è `stop_reason == "end_turn"`.

3.2 Configurazione `AgentDefinition`

`AgentDefinition` è l’oggetto di configurazione di un agente nello Claude Agent SDK:

```
agent = AgentDefinition(
    name="customer_support",
    description="Gestisce richieste clienti relative a resi e problemi d’ordine",
    system_prompt="Sei un agente di assistenza clienti...",
    allowed_tools=["get_customer", "lookup_order", "process_refund",
"escalate_to_human"],
    # Per un coordinatore:
    # allowed_tools=["Task", "get_customer", ...]
)
```

Parametri principali:

- `name` / `description` — identificazione e descrizione dell'agente
- `system_prompt` — prompt di sistema con le istruzioni operative
- `allowed_tools` — elenco degli strumenti consentiti (principio del minimo privilegio)

3.3 Architettura Hub-and-Spoke: Coordinator e Subagent

Un'architettura multi-agente è tipicamente costruita secondo una topologia hub-and-spoke:

```
     Coordinator
    /     |      \
Subagent1 Subagent2 Subagent3
(search) (analysis) (synthesis)
```

Il coordinator è responsabile di:

- Decomposizione del task in sotto-attività
- Scelta dinamica dei subagent necessari
- Delega delle attività ai subagent
- Aggregazione e validazione dei risultati
- Gestione di errori e retry
- Comunicazione del risultato finale all'utente

Principio critico: i subagent hanno contesto isolato.

- I subagent **non** ereditano automaticamente la cronologia del coordinator
- Tutto il contesto necessario deve essere **passato esplicitamente nel prompt**
- I subagent non condividono memoria tra chiamate
- Tutta la comunicazione avviene tramite il coordinator (per osservabilità e controllo degli errori)

3.4 Tool `Task` per la creazione di Subagent

I subagent vengono attivati tramite il tool `Task`:

```
# allowed_tools del coordinator deve includere "Task"
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

Il passaggio esplicito del contesto è obbligatorio:

```
# Errato: il subagent non ha contesto
Task: "Analizza il documento"

# Corretto: contesto completo nel prompt
Task: "Analizza il seguente documento.
Documento: [testo completo]
Risultati di ricerca precedenti: [risultati web]
Formato di output richiesto: [schema]"
```

Esecuzione parallela: un coordinator può invocare più `Task` in una singola risposta — i subagent vengono eseguiti in parallelo:

```
# Un'unica risposta del coordinator contiene:
Task 1: "Cerca articoli su X"
Task 2: "Analizza documento Y"
Task 3: "Cerca articoli su Z"
# Tutti eseguiti in parallelo
```

3.5 Hooks nell'Agent SDK

Gli hook permettono di intercettare e trasformare eventi in punti specifici del ciclo di vita dell'agente.

PostToolUse intercetta il risultato di uno strumento prima che venga fornito al modello:

```python
# Esempio: normalizzazione dei formati data provenienti da diversi tool MCP
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Conversione Unix timestamp -> ISO 8601
    # Conversione "Mar 5, 2025" -> "2025-03-05"
    return normalized_result
```

Hook di intercettazione delle chiamate in uscita: blocca azioni che violano policy:

```python
# Esempio: blocco rimborsi sopra 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)
```

Differenza chiave: hook vs istruzioni nel prompt

Attributo	Hooks	Istruzioni nel prompt
Garanzia	Deterministica (100%)	Probabilistica (>90%, non garantita)

Quando usarli	Regole critiche, operazioni finanziarie, compliance	Preferenze generali, raccomandazioni, formattazione
Esempio	Bloccare rimborsi > 500$	“Prova a risolvere prima di escalation”

Regola: quando un errore ha conseguenze finanziarie, legali o di sicurezza, usare gli hook e non il prompt.

Capitolo 4: Model Context Protocol (MCP)

> Documentazione: MCP | Tools | Resources | Servers

4.1 Cos’è il MCP

Il Model Context Protocol (MCP) è un protocollo aperto per collegare sistemi esterni a Claude. MCP definisce tre tipologie principali di risorse:

1. **Tools** — funzioni che l’agente può chiamare per eseguire azioni (operazioni CRUD, chiamate API, esecuzione di comandi)
2. **Resources** — dati che l’agente può leggere per ottenere contesto (documentazione, schemi di database, cataloghi di contenuti)
3. **Prompts** — template di prompt predefiniti per task ricorrenti

4.2 MCP Servers

Un server MCP è un processo che implementa il protocollo MCP e fornisce tools e resources. Quando ci si connette a un server MCP:

- Tutti i tool vengono scoperti automaticamente
- I tool di tutti i server connessi sono disponibili simultaneamente
- Le descrizioni dei tool determinano come il modello li utilizzerà

4.3 Configurazione dei Server MCP

Configurazione di progetto (`.mcp.json`) — per l'uso in team:

```
{
  "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}"
      }
    }
  }
}
```

Punti chiave:

- `.mcp.json` si trova nella root del progetto ed è gestito tramite versionamento
- Le variabili d'ambiente `(${GITHUB_TOKEN})` vengono usate per i segreti — i token non vengono mai salvati nel repository
- È disponibile per tutti i contributori del progetto

Configurazione utente (`~/.claude.json`) — per server personali o sperimentali:

- Archiviata nella home directory dell'utente
- Non condivisa tramite versionamento
- Adatta a test ed esperimenti personali

Scelta dei server:

- Per integrazioni standard (Jira, GitHub, Slack), preferire server MCP già esistenti nella community
- Costruire server personalizzati solo per workflow specifici e non standard del team

4.4 Flag `isError` in MCP

Quando uno strumento MCP incontra un errore, utilizza `isError: true` nella risposta. Questo segnala all'agente che la chiamata è fallita.

Errore strutturato (corretto):

```
{
  "isError": true,
  "content": {
    "errorCategory": "transient",
    "isRetryable": true,
    "message": "Il servizio è temporaneamente non disponibile. Timeout durante la
chiamata all'API ordini.",
    "attempted_query": "order_id=12345",
    "partial_results": null
  }
}
```

Errore generico (anti-pattern):

```
{
  "isError": true,
  "content": "Operazione fallita"
}
```

Un errore generico non fornisce informazioni utili per il processo decisionale dell'agente: non è possibile capire se ritentare, modificare la query o fare escalation.

4.5 MCP Resources

Le Resources sono dati che un agente può richiedere per ottenere contesto senza eseguire azioni:

- Cataloghi di contenuti (es. elenco di task di progetto, navigazione gerarchica)
- Schemi di database (per comprendere la struttura dei dati)
- Documentazione (reference API, guide interne)
- Sommari di issue o task

Vantaggio delle Resources: l'agente non deve effettuare chiamate esplorative a strumenti per capire quali dati esistono. Una resource fornisce una "mappa" immediata del sistema.

Capitolo 5: Claude Code — Configurazione e Workflow

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

5.1 Gerarchia di CLAUDE.md

CLAUDE.md è il file (o insieme di file) di istruzioni per Claude Code. Esiste una gerarchia su tre livelli:

1. Livello utente: ~/.claude/CLAUDE.md

- Si applica solo a quel singolo utente
- NON viene condiviso tramite VCS
- Preferenze personali e stile di lavoro

1. Livello progetto: .claude/CLAUDE.md oppure CLAUDE.md nella root

- Si applica a tutti i contributori del progetto
- Gestito tramite VCS
- Standard di codice, testing, decisioni architetturali

1. Livello directory: CLAUDE.md nelle sottodirectory

- Si applica quando si lavora sui file di quella directory
- Convenzioni specifiche per quella parte del codebase

Errore comune: un nuovo membro del team non riceve le istruzioni di progetto perché sono state inserite in `~/.claude/CLAUDE.md` (livello utente) invece che in `.claude/CLAUDE.md` (livello progetto).

5.2 Sintassi `@path` (importazione file)

CLAUDE.md può referenziare file esterni tramite `@path`, rendendo la configurazione modulare:

```
# CLAUDE.md del progetto

Gli standard di codice sono descritti in @./standards/coding-style.md
I requisiti di test sono in @./standards/testing-requirements.md
La panoramica del progetto è in @README.md e le dipendenze in @package.json
```

Regole per `@path`:

- Usare `@` immediatamente prima del percorso (senza spazi)
- Sono supportati percorsi relativi e assoluti
- I percorsi relativi sono risolti rispetto al file che contiene l'import
- La profondità massima di annidamento degli import è 5 livelli

Questo evita duplicazioni e consente a ogni package di includere solo gli standard rilevanti.

5.3 Directory `.claude/rules/`

`.claude/rules/` è un'alternativa al CLAUDE.md monolitico, usata per organizzare le regole per argomento:

```
.claude/rules/
testing.md -- convenzioni di test
api-conventions.md -- convenzioni API
deployment.md -- regole di deployment
react-patterns.md -- pattern React
```

Caratteristica chiave: YAML frontmatter con `paths` per caricamento condizionale:

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

Per i file API, utilizzare async/await con gestione esplicita degli errori.
Ogni endpoint deve restituire un wrapper di risposta standard.
```

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

I test devono usare blocchi describe/it.
Usare factory di dati invece di valori hardcoded.
Non mockare il database—utilizzare un database di test.
```

Come funziona:

- Una regola viene caricata solo quando Claude Code modifica un file che corrisponde al pattern `paths`
- Questo riduce il contesto e il consumo di token: le regole non rilevanti non vengono caricate
- I `Glob` pattern permettono di applicare convenzioni per tipo di file indipendentemente dalla posizione (utile per test distribuiti nel codebase)

Quando usare `.claude/rules/` con `paths` vs CLAUDE.md a livello directory:

- `.claude/rules/` con `paths` — quando le convenzioni si applicano a file distribuiti in più directory (test, migrazioni)
- CLAUDE.md a livello directory — quando le regole sono specifiche di una directory e non servono altrove

5.4 Comandi Slash Personalizzati e Skills

> **Nota:** nella versione attuale di Claude Code, i comandi personalizzati (`.claude/commands/`) sono stati unificati con le skills (`.claude/skills/`). Entrambi i formati creano comandi `/name` . La guida d'esame fa riferimento a `.claude/commands/` , che è ancora supportato.

I comandi slash sono template di prompt riutilizzabili invocati tramite `/name` :

Formato `.claude/commands/` (legacy, ancora supportato):

```
.claude/commands/
review.md -- /review -- code review standard
test-gen.md -- /test-gen -- generazione test
```

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

```
.claude/skills/
review/SKILL.md -- /review -- con configurazione via frontmatter
test-gen/SKILL.md
```

Comandi di progetto (`.claude/commands/` o `.claude/skills/`):

- Salvati nel VCS e disponibili a tutti dopo il clone del repository
- Garantiscono workflow coerenti a livello di team

Comandi utente (`~/.claude/commands/` o `~/.claude/skills/`):

- Comandi personali non condivisi tramite VCS
- Utilizzati per workflow individuali

5.5 Skills — `.claude/skills/`

Le Skills sono comandi avanzati configurati tramite frontmatter in `SKILL.md`:

```yaml
---
context: fork
allowed-tools: ["Read", "Grep", "Glob"]
argument-hint: "Percorso della directory da analizzare"
---

Analizza la struttura del codice nella directory specificata.
Fornisci un report su dipendenze e pattern architetturali.
```

Parametri del frontmatter:

Parametro	Descrizione
`context: fork`	Esegue la skill in un subagent isolato. L'output dettagliato non sporca la sessione principale
`allowed-tools`	Limita gli strumenti disponibili (sicurezza—ad esempio la skill non può eliminare file se non autorizzata)
`argument-hint`	Suggerimento che richiede un argomento quando la skill viene invocata senza parametri

Quando usare una skill vs CLAUDE.md:

- **Skill** — invocazione su richiesta per task specifici (review, analisi, generazione)
- **CLAUDE.md** — standard e convenzioni sempre attivi e caricati automaticamente

Skill personali (`~/.claude/skills/`):

- Permettono varianti personali con nomi diversi per evitare impatti sui collaboratori

5.6 Planning Mode vs Esecuzione Diretta

Planning mode:

- Il modello si limita a esplorare e pianificare; non esegue modifiche
- Utilizza Read, Grep, Glob per analizzare il codebase
- Produce un piano di implementazione che deve essere approvato dall'utente
- Esplorazione sicura senza effetti collaterali

Quando usare il planning mode:

- Modifiche estese (decine di file)
- Più approcci possibili (es. microservizi: definizione dei confini)
- Decisioni architetturali (framework, struttura del progetto)
- Codebase sconosciuti (necessità di comprensione prima delle modifiche)
- Migrazioni di librerie che coinvolgono 45+ file

Quando usare l'esecuzione diretta:

- Fix su singolo file con stack trace chiaro
- Aggiunta di una singola validazione
- Modifiche ben definite e non ambigue

Approccio combinato:

1. Planning mode per investigazione e progettazione
2. Approvazione del piano da parte dell'utente
3. Esecuzione diretta per implementare il piano approvato

Explore subagent — subagent specializzato per l'esplorazione del codebase:

- Isola output verbose dal contesto principale
- Restituisce solo un riassunto
- Previene l'esaurimento della context window in task multi-fase

5.7 Comando `/compact`

`/compact` è un comando integrato per la compressione del contesto:

- Riassume la cronologia precedente per liberare la context window
- Utilizzato in sessioni lunghe quando il contesto si riempie di output verboso di strumenti
- Rischio: valori numerici esatti, date e dettagli specifici possono andare persi durante la sintesi

5.8 Comando `/memory`

`/memory` è un comando integrato per la gestione della memoria tra sessioni:

- Apre il file `CLAUDE.md` per la modifica, permettendo di salvare note, preferenze e contesto
- Le informazioni persistono tra sessioni e vengono caricate automaticamente all'avvio
- Utile per memorizzare convenzioni di progetto, preferenze utente, comandi frequenti e stato del lavoro corrente
- Alternativa al dover ripetere le stesse istruzioni in ogni sessione

5.9 Claude Code CLI per CI/CD

Flag `-p` (o `--print`):

```
claude -p "Analizza questa pull request per problemi di sicurezza"
```

- Modalità non interattiva: elabora il prompt, stampa su stdout ed esce
- Non richiede input dell'utente
- È l'unico modo corretto per eseguire Claude in pipeline CI/CD

Output strutturato per CI:

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

- `--output-format json` — output in formato JSON
- `--json-schema` — validazione dell'output rispetto a uno schema
- Il risultato può essere parsato per generare automaticamente commenti inline sulla PR

Isolamento del contesto tra sessioni: La stessa sessione di Claude che ha generato il codice è spesso meno efficace nel revisarlo (il modello conserva il proprio contesto di ragionamento ed è meno incline a criticare le proprie decisioni). Per la review usare un'istanza indipendente.

Prevenzione dei commenti duplicati: Quando si riesegue la review dopo nuovi commit, includere i risultati precedenti nel contesto e istruire Claude a segnalare solo problemi nuovi o non risolti.

5.10 `fork_session` e Gestione delle Sessioni

`--resume <session-name>` riprende una sessione nominata:

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

- Continua una conversazione precedente mantenendo il contesto salvato

- Utile per indagini lunghe distribuite su più sessioni
- Rischio: se i file sono cambiati rispetto alla sessione precedente, i risultati degli strumenti possono essere obsoleti

`fork_session` crea un ramo indipendente a partire da un contesto condiviso:

```
Analisi codebase
         |
   fork_session
    /          \
Approccio A:    Approccio B:
Redux           Context API
```

- Entrambi i fork ereditano il contesto fino al punto di diramazione
- Successivamente divergono in modo indipendente
- Utile per confrontare approcci o testare strategie alternative

Quando avviare una nuova sessione invece di riprendere una precedente:

- I risultati degli strumenti sono potenzialmente obsoleti (file modificati)
- È passato molto tempo e il contesto è degradato
- È preferibile ripartire con un riassunto esplicito ("Ecco ciò che abbiamo trovato finora: …") invece di continuare una sessione basata su dati non aggiornati

---

Capitolo 6: Prompt Engineering — Tecniche Avanzate

> Documentazione: Prompt Engineering | Anthropic Cookbook

6.1 Few-shot Prompting

Il few-shot prompting consiste nell'inserire 2–4 esempi input/output all'interno di un prompt per dimostrare il comportamento atteso.

Perché il few-shot è più efficace delle descrizioni testuali:

- Un'istruzione vaga come "sii più preciso" può essere interpretata in molti modi
- Un esempio elimina l'ambiguità mostrando formato e logica decisionale attesi
- Il modello generalizza il pattern a nuovi casi (non si limita a ripetere gli esempi)

Tipologie di esempi few-shot e quando usarli:

1. **Esempi per scenari ambigui:**

```
Richiesta: "Il mio ordine è rotto"
Azione: chiamare get_customer -> lookup_order -> verificare stato
Motivazione: “rotto” può indicare un prodotto danneggiato; servono dettagli
sull’ordine

Richiesta: "Mi serve un manager"
Azione: chiamare immediatamente escalate_to_human
Motivazione: il cliente richiede esplicitamente un umano. Non tentare risoluzione
autonoma
```

1. **Esempi per formattazione dell’output:**

```
Esempio di finding:
{
"location": "src/auth/login.ts:42",
"issue": "SQL injection nel parametro username",
"severity": "critical",
"suggested_fix": "Usare query parametrizzata"
}
```

1. **Esempi per distinguere codice accettabile vs problematico:**

```
// Accettabile (non segnalare):
const items = data.filter(x => x.active);

// Problema (segnalare):
const items = data.filter(x => x.active == true); // usare strict equality ===
```

1. **Esempi per estrazione da formati documentali diversi:**

```
Documento con citazioni inline:
"Come mostrato nello studio (Smith, 2023), il tasso è del 42%."
-> {"value": "42%", "source": "Smith, 2023", "type": "inline_citation"}

Documento con bibliografia:
"Il tasso è del 42%. [1]"
-> {"value": "42%", "source": "reference_1", "type": "bibliography"}
```

1. **Esempi per misurazioni informali:**

```
Testo: "circa due manciate di riso"
-> {"amount": "~100g", "original_text": "due manciate", "precision": "approximate"}

Testo: "un pizzico di sale"
-> {"amount": "~1g", "original_text": "un pizzico", "precision": "approximate"}
```

Il few-shot è particolarmente efficace per estrarre unità di misura informali e non standardizzate, troppo variabili per istruzioni puramente regole-based.

Regole di normalizzazione nei prompt: Quando si usano JSON schema rigorosi per output strutturati, è utile aggiungere regole di normalizzazione nel prompt:

```
Normalizzazione:

Date: sempre ISO 8601 (YYYY-MM-DD); "ieri" -> convertire in data assoluta
Valute: importo numerico + codice valuta; "cinque euro" -> {"amount": 5, "currency":
"EUR"}
Percentuali: frazione decimale; "metà" -> 0.5
```

Questo riduce gli errori semantici, in cui il JSON è sintatticamente valido ma i valori risultano incoerenti.

6.2 Criteri Espliciti vs Istruzioni Vaghe

Sbagliato (vago):

```
Controlla la correttezza dei commenti nel codice.
Sii conservativo — riporta solo problemi ad alta confidenza.
```

Corretto (criteri espliciti):

```
Segnala un commento come problematico SOLO se:

1. Il commento descrive un comportamento che CONTRADDICE il comportamento reale del
codice
2. Il commento fa riferimento a una funzione o variabile inesistente
3. Un commento TODO/FIXME fa riferimento a un bug già risolto nel codice

NON segnalare:

- Commenti solo leggermente obsoleti a livello stilistico
- Piccole imprecisioni di formulazione
- Commenti mancanti (categoria separata)
```

Definizione delle severità con esempi:

```
CRITICAL: Fallimento runtime per l'utente
  Esempio: NullPointerException durante l'elaborazione di un pagamento

HIGH: Vulnerabilità di sicurezza
  Esempio: SQL injection, XSS, mancanza di controlli di autorizzazione

MEDIUM: Bug logico senza impatto immediato
  Esempio: ordinamento errato, off-by-one error

LOW: Qualità del codice
  Esempio: duplicazione, algoritmo non ottimale per piccoli dataset
```

6.3 Prompt Chaining

Il prompt chaining scompone un task complesso in una sequenza di passaggi focalizzati:

```
Step 1: Analizzare auth.ts (solo problemi locali)
-> Output: lista di problemi in auth.ts

Step 2: Analizzare database.ts (solo problemi locali)
-> Output: lista di problemi in database.ts

Step 3: Passaggio di integrazione (dipendenze cross-file)
-> Output: problemi a livello di modulo e interfacce tra componenti
```

Perché è importante:

- Evita la **diluizione dell'attenzione**: quando il modello riceve troppi file insieme, può perdere bug in alcuni file e fornire analisi superficiali su altri
- Garantisce qualità di analisi coerente per ogni file
- Permette di separare l'analisi locale da quella delle interazioni tra file

Quando usare prompt chaining vs decomposizione dinamica:

- **Prompt chaining** — task prevedibili e ripetibili (code review, migrazioni di file)
- **Decomposizione dinamica** — investigazioni aperte dove i sotto-task emergono solo durante l'esecuzione

6.4 Pattern "Interview"

Prima di implementare una soluzione, Claude pone domande di chiarimento:

```
Claude: "Prima di implementare il caching per l'API, alcune domande:

Quale strategia di invalidazione preferisci—TTL o event-based?
È accettabile avere dati stantii quando la cache non è disponibile?
Il caching deve essere per utente o globale?
Qual è il volume di dati atteso da memorizzare?"
```

Quando è utile:

- Domini non familiari (fintech, healthcare, sistemi legali)
- Task con implicazioni non ovvie (strategie di caching, failure mode)
- Più approcci validi dove la scelta dipende dal contesto

6.5 Validazione e Retry con Feedback

Quando i dati estratti falliscono la validazione:

```
Step 1: Estrazione dati dal documento
Step 2: Validazione (Pydantic, JSON Schema, regole di business)
Step 3: Se c'è un errore—retry con contesto:

Documento originale
Estrazione precedente (errata)
Errore specifico: "Field 'total' = 150, ma sum(line_items) = 145. Ricontrollare i valori."
```

Quando il retry è efficace:

- Errori di formato (date nel formato errato)
- Errori strutturali (campo posizionato nel punto sbagliato)
- Incoerenze aritmetiche (il modello può ricalcolare)

Quando il retry NON è efficace:

- L'informazione non è presente nel documento sorgente
- Il contesto necessario è esterno (dato in un altro documento non fornito)

Pydantic come strumento di validazione: Pydantic è una libreria Python per la validazione basata su schema. Per l'esame, i punti chiave sono:

- **Validazione strutturale:** tipi, campi obbligatori e vincoli enum verificati nel codice dopo aver ricevuto JSON da Claude
- **Validazione semantica:** validator custom per regole di business (somma degli elementi = totale; start_date < end_date)
- **Loop validate–retry:** in caso di errore Pydantic, si costruisce un messaggio di errore e si ripete la richiesta a Claude con il contesto dell'errore
- **Generazione JSON Schema:** i modelli Pydantic possono generare JSON Schema per `tool_use`, creando una singola fonte di verità

6.6 Self-correction

Pattern per rilevare contraddizioni interne:

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

Il modello estrae sia il valore dichiarato sia quello calcolato—se differiscono, `conflict_detected` consente di gestire esplicitamente la discrepanza.

Capitolo 7: Message Batches API

> Documentazione: Message Batches

7.1 Panoramica

La Message Batches API consente di inviare batch di richieste per elaborazione asincrona:

Attributo	Valore
Risparmio	**50%** rispetto alle chiamate sincrone
Finestra di elaborazione	Fino a **24 ore** (nessuna garanzia di latenza)
Multi-turn tool calling	**Non supportato** (una richiesta = una risposta)
Correlazione	Campo `custom_id` per collegare richiesta e risposta

7.2 Quando usare Batch API vs API sincrona

Task	API	Motivazione
Controllo PR pre-merge	**Sincrona**	Lo sviluppatore è in attesa; 24 ore non accettabili
Report tecnico notturno	**Batch**	Il risultato serve al mattino; utile il risparmio del 50%
Audit di sicurezza settimanale	**Batch**	Non urgente; ottimizzazione dei costi
Code review interattiva	**Sincrona**	Necessaria risposta immediata

Elaborazione 10.000 documenti	**Batch**	Processing massivo; forte beneficio economico

7.3 Utilizzo di `custom_id`

```
{
  "custom_id": "doc-invoice-2024-001",
  "params": {
    "model": "claude-sonnet-4-6",
    "max_tokens": 1024,
    "messages": [{"role": "user", "content": "Estrai i dati da: ..."}]
  }
}
```

`custom_id` consente di:

- Collegare il risultato al documento originale
- In caso di errore, reinviare solo i documenti falliti
- Evitare di rielaborare documenti già processati con successo

7.4 Gestione dei fallimenti nei batch

1. Invio di un batch di 100 documenti
2. 95 elaborati con successo; 5 falliti (context limit exceeded)
3. Identificazione dei fallimenti tramite `custom_id`
4. Modifica della strategia (es. suddivisione dei documenti lunghi in chunk)
5. Reinvio solo dei 5 documenti falliti

7.5 Pianificazione SLA

Se è necessario ottenere un risultato entro 30 ore e la Batch API può impiegare fino a 24 ore:

- Finestra di invio: 30 - 24 = **6 ore**
- I batch devono essere inviati non oltre 24 ore prima della scadenza
- Per invii frequenti, suddividere in finestre da 4 ore

Capitolo 8: Strategie di Decomposizione dei Task

8.1 Pipeline Fisse (Prompt Chaining)

Ogni step è definito in anticipo:

```
Documento -> Estrazione metadata -> Estrazione dati -> Validazione -> Enrichment ->
Output finale
```

Quando usarle:

- La struttura del task è prevedibile (le review seguono sempre lo stesso template)
- Tutti i passaggi sono noti a priori
- Serve stabilità e riproducibilità

8.2 Decomposizione Dinamica Adattiva

I sotto-task vengono generati in base ai risultati intermedi:

```
"Aggiungere test a un codebase legacy"
-> Prima: mappare la struttura (Glob, Grep)
-> Trovati: 3 moduli senza test, 2 con copertura parziale
-> Priorità: iniziare dal modulo pagamenti (alto rischio)
-> Durante il lavoro: scoperta dipendenza da API esterna
-> Adattamento: aggiungere mock per l'API prima di scrivere i test
```

Quando usarla:

- Task investigativi aperti
- Quando lo scope completo non è noto a priori
- Quando ogni step dipende dai risultati del precedente

8.3 Multi-pass Code Review

Per pull request con 10+ file:

```
Pass 1 (per file): Analisi auth.ts -> problemi locali
Pass 1 (per file): Analisi database.ts -> problemi locali
Pass 1 (per file): Analisi routes.ts -> problemi locali
...
Pass 2 (integrazione): analisi relazioni tra file
-> problemi cross-file: tipi incoerenti, dipendenze circolari
```

Perché un singolo pass su 14 file è problematico:

- Diluizione dell'attenzione: analisi approfondita su alcuni file, superficiale su altri
- Incoerenza nelle valutazioni: pattern segnalati in un file ma ignorati in un altro
- Bug mancati: errori evidenti non rilevati per overload cognitivo

---

Capitolo 9: Escalation e Human-in-the-Loop

9.1 Quando effettuare escalation a un umano

Trigger di escalation (regole esplicite):

Situazione	Azione
Il cliente chiede esplicitamente “voglio un manager”	Escalation immediata; non tentare risoluzione
La policy non copre la richiesta	Escalation (es. price matching con competitor non previsto)
L’agente non riesce a progredire	Escalation dopo un numero ragionevole di tentativi
Operazioni finanziarie sopra una soglia	Escalation (meglio se applicata via hook, non prompt)
Più corrispondenze nella ricerca cliente	Richiedere ulteriori identificativi; non fare supposizioni

Non sono trigger affidabili:

Metodo non affidabile	Perché fallisce
Analisi del sentiment	Lo stato emotivo non correlato alla complessità del caso
Confidence score del modello (1–10)	Il modello può essere sicuro ma errato
Classificatori automatici	Overengineering e necessità di dati di training

9.2 Pattern di escalation

Escalation immediata:

```
Cliente: "Voglio parlare con un manager"
Agente: [chiama immediatamente escalate_to_human]
NON: "Posso aiutarti con il tuo problema..."
```

Escalation dopo tentativo di risoluzione:

```
Cliente: "Il frigorifero si è rotto due giorni dopo l’acquisto"
Agente: [verifica ordine, propone sostituzione in garanzia]
Se il cliente non è soddisfatto -> escalation
```

Escalation sfumata (acknowledge → soluzione → escalation su insistenza):

```
Cliente: "Sono molto insoddisfatto della qualità!"
Agente: [riconosce il problema] "Capisco la tua frustrazione."
[propone soluzione] "Posso offrire sostituzione o rimborso."
Cliente: "No, voglio parlare con qualcuno!"
Agente: [insistenza del cliente -> escalation immediata]
```

Principio chiave: riconoscere l'emozione prima, proporre una soluzione concreta, ed effettuare escalation solo se il cliente ribadisce la richiesta di un umano. Non si deve escalare alla prima espressione di insoddisfazione.

Escalation per gap di policy:

```
Cliente: "Il competitor X costa il 30% in meno — applicate uno sconto"
Policy: copre solo aggiustamenti prezzo sul proprio sito
Agente: [escalation — la policy non copre il price matching con competitor]
```

9.3 Protocolli Strutturati di Handoff

In caso di escalation, l'agente deve trasferire a un umano un riepilogo strutturato:

```
{
  "customer_id": "CUST-12345",
  "customer_name": "Ivan Petrov",
  "issue_summary": "Richiesta di rimborso per articolo danneggiato",
  "order_id": "ORD-67890",
  "root_cause": "Articolo arrivato danneggiato; foto allegate",
  "actions_taken": [
    "Verificato cliente via get_customer",
    "Confermato ordine via lookup_order",
    "Offerta sostituzione standard — cliente insiste per rimborso"
  ],
  "refund_amount": "$89.99",
  "recommended_action": "Approva rimborso completo",
  "escalation_reason": "Cliente ha richiesto esplicitamente un manager"
}
```

L'operatore umano non ha accesso all'intera conversazione: vede esclusivamente questo riepilogo. Per questo motivo deve essere completo e auto-contenuto.

9.4 Calibrazione della Confidenza e Supervisione Umana

Per sistemi di estrazione dati:

1. **Confidence score a livello di campo:** il modello assegna un punteggio di confidenza per ogni campo estratto
2. **Calibrazione:** utilizzo di set di validazione etichettati per tarare le soglie

3. **Routing:**
    - Alta confidenza + accuratezza stabile → processamento automatico
    - Bassa confidenza o fonti ambigue → revisione umana

Campionamento random stratificato:

- Anche per estrazioni ad alta confidenza, effettuare audit periodici su campioni
- Un'accuratezza media del 97% può nascondere errori del 40% su specifici tipi di documento
- Analizzare l'accuratezza per tipo di documento e per campo, non solo in aggregato

---

Capitolo 10: Gestione degli errori nei sistemi multi-agent

10.1 Categorie di errore

Categoria	Esempi	Retryable	Azione dell'agente
Transiente	Timeout, 503, errori di rete	Sì	Retry con exponential backoff
Validazione	Formato input non valido, campo obbligatorio mancante	No (correggere input)	Modificare la richiesta e ritentare
Business	Violazione policy, superamento soglia	No	Spiegare all'utente e proporre alternativa
Permessi	Accesso negato	No	Escalation

10.2 Anti-pattern nella gestione errori

Anti-pattern	Problema	Approccio corretto
Stato generico "search unavailable"	Il coordinator non sa come recuperare	Restituire tipo errore, query, risultati parziali, alternative
Soppressione silenziosa (empty = successo)	Il coordinator interpreta assenza come successo	Distinguere chiaramente "nessun risultato" da "errore di ricerca"
Abort dell'intero workflow per un singolo errore	Perdita di risultati parziali	Continuare con risultati parziali e annotare i gap
Retry infiniti in un subagent	Latency e spreco risorse	Retry locale limitato (1–2 tentativi), poi propagazione al coordinator

10.3 Errore strutturato in un subagent

```
{
  "status": "partial_failure",
  "failure_type": "timeout",
  "attempted_query": "AI impact on music industry 2024",
  "partial_results": [
    {"title": "AI Music Generation Report", "url": "...", "relevance": 0.8}
  ],
  "alternative_approaches": [
    "Provare una query più specifica: 'AI music composition tools'",
    "Usare una fonte dati alternativa"
  ],
  "coverage_impact": "Non coperto: impatto AI sulla produzione musicale"
}
```

Questo permette al coordinator di decidere in modo informato:

- Ritentare con una query modificata?
- Usare i risultati parziali?
- Delegare a un altro subagent?
- Continuare ignorando la sezione e annotando il gap?

10.4 Annotazioni di copertura nella sintesi finale

```
## Report: Impatto dell'AI nelle industrie creative

### Arte visiva (COPERTURA COMPLETA)
[research results]

### Musica (COPERTURA PARZIALE — timeout del search agent)
[partial results]
⚠️ Nota: la copertura di questa sezione è limitata a causa di un timeout del search agent.

### Letteratura (COPERTURA COMPLETA)
[research results]
```

Capitolo 11: Gestione del Contesto nei Sistemi in Produzione

11.1 Estrazione dei fatti in un blocco separato

Invece di affidarsi alla cronologia della conversazione (che degrada durante la summarization), è necessario estrarre i fatti chiave in un blocco strutturato:

```
=== FATTI DEL CASO (aggiornati ogni volta che emerge un nuovo fatto) ===
Customer ID: CUST-12345
Order ID: ORD-67890
Data ordine: 2025-01-15
Importo ordine: $89.99
Problema: Prodotto danneggiato alla consegna
Richiesta cliente: Rimborso completo
Stato: In attesa di approvazione manager
```

Questo blocco deve essere incluso in ogni prompt, indipendentemente dalla compressione della history.

11.2 Riduzione dei risultati dei tool (trimming)

Se `lookup_order` restituisce 40+ campi ma ne servono solo 5 per il task corrente:

```python
# Hook PostToolUse: mantenere solo i campi rilevanti
@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"]
    }
```

Questo riduce il consumo di contesto e il rumore informativo.

11.3 Input con consapevolezza della posizione

Posizionare le informazioni critiche tenendo conto del fenomeno “lost-in-the-middle”:

```
[RISULTATI CHIAVE — in alto]
Trovate 3 vulnerabilità critiche...

[DETTAGLIO RISULTATI — parte centrale]
=== File auth.ts ===
...
=== File database.ts ===
...

[AZIONI RICHIESTE — in fondo]
Priorità: correggere le vulnerabilità in auth.ts prima del merge.
```

11.4 Scratchpad Files

In indagini lunghe, l'agente può scrivere i risultati chiave in un file scratchpad:

```
investigation-scratchpad.md
Risultati chiave
PaymentProcessor in src/payments/processor.ts eredita da BaseProcessor
refund() è chiamata da 3 punti: OrderController, AdminPanel, CronJob
L'API esterna PaymentGateway ha un rate limit di 100 req/min
Migration #47 ha introdotto refund_reason (NOT NULL) — 2024-12-01
```

Quando il contesto si degrada (o in una nuova sessione), l'agente può consultare lo scratchpad invece di rieseguire l'intero processo di discovery.

11.5 Delegazione a subagent per protezione del contesto

```
Main agent: "Analizza le dipendenze del modulo payments"
-> Subagent (Explore): legge 15 file, traccia import
-> Output: "Payments dipende da AuthService, OrderModel e dall'API esterna
PaymentGateway"

Il main agent mantiene una sola riga nel contesto invece di 15 file.
```

Separazione dei livelli di contesto: Nei sistemi multi-agent, ogni subagent opera con un budget di contesto limitato: riceve solo le informazioni necessarie al task. Il coordinator agisce come livello separato di contesto: aggrega gli output dei subagent, mantiene lo stato globale e distribuisce il contesto. Questo evita il "context leakage", dove un agente consuma la context window con informazioni non rilevanti per gli altri.

Budget di contesto vincolato per i subagent:

- Inviare contesto minimo: task specifico + dati necessari
- Istruire il subagent a restituire risultati strutturati, non dump di dati grezzi

- Usare `allowedTools` per limitare gli strumenti disponibili—meno strumenti significa meno distrazioni e minore consumo di contesto

11.6 Persistenza strutturata dello stato (recupero da crash)

Ogni agente esporta il proprio stato in una posizione definita:

```
{
  "status": "completed",
  "queries_executed": ["AI music 2024", "AI music composition"],
  "results_count": 12,
  "key_findings": [...],
  "coverage": ["music composition", "music production"],
  "gaps": ["music distribution", "music licensing"]
}
```

Il coordinator carica un manifest in fase di ripresa:

```
{
  "web-search": "completed",
  "doc-analysis": "in_progress",
  "synthesis": "not_started"
}
```

Capitolo 12: Preservare la Provenienza

12.1 Problema della perdita di attribuzione

Quando si sintetizzano risultati provenienti da fonti multiple, il legame tra “affermazione → fonte” può andare perso:

```
Sbagliato: "Il mercato della musica AI è stimato a 3,2 miliardi di dollari."
(Nessuna fonte, nessun anno)

Corretto:
{
"claim": "Il mercato della musica AI è stimato a 3,2 miliardi di dollari.",
"source_url": "https://example.com/report",
"source_name": "Global AI Music Report 2024",
"publication_date": "2024-06-15",
"confidence": 0.9
}
```

12.2 Gestione di dati in conflitto

Quando due fonti forniscono valori diversi:

```
{
  "claim": "Quota di musica generata da AI sulle piattaforme di streaming",
  "values": [
    {
      "value": "12%",
      "source": "Spotify Annual Report 2024",
      "date": "2024-03",
      "methodology": "Classificazione automatica"
    },
    {
      "value": "8%",
      "source": "Music Industry Association Survey",
      "date": "2024-07",
      "methodology": "Survey su 500 etichette discografiche"
    }
  ],
  "conflict_detected": true,
  "possible_explanation": "Differenze di metodologia e periodo temporale"
}
```

Non scegliere arbitrariamente un singolo valore. Conservare entrambe le misurazioni con attribuzione e lasc

12.3 Includere le date per una corretta interpretazione

Senza le date, differenze temporali possono essere interpretate erroneamente come contraddizioni:

```
Sbagliato: "La fonte A dice 10%, la fonte B dice 15%. Contraddizione."

Corretto: "La fonte A (2023) indica 10%, la fonte B (2024) indica 15%. Probabile
crescita del +5% in un anno."
```

12.4 Rendering in base al tipo di contenuto

Non forzare tutto in un unico formato:

- Dati finanziari → tabelle
- Notizie e analisi → testo discorsivo
- Risultati tecnici → liste strutturate
- Serie temporali → ordinamento cronologico

Capitolo 13: Tool integrati di Claude Code

13.1 Riferimento per la selezione dei tool

Task	Tool	Esempio
Trovare file per nome/pattern	**Glob**	`**/*.test.tsx`, `src/components/**/*.ts`
Cercare dentro i file	**Grep**	Nome funzione, messaggio di errore, import
Leggere un file completo	**Read**	Caricare un file per analisi
Creare un nuovo file	**Write**	Generazione file da zero
Modificare un file esistente in modo preciso	**Edit**	Sostituzione di uno snippet tramite match univoco
Eseguire un comando shell	**Bash**	git, npm, test, build

13.2 Strategia di investigazione incrementale

Non leggere tutti i file insieme. Costruire la comprensione in modo progressivo:

```
1. Grep: trovare i punti di ingresso (definizioni di funzione, export)
2. Read: leggere i file individuati
3. Grep: trovare gli utilizzi (import, chiamate)
4. Read: leggere i file consumer
5. Ripetere fino ad avere un quadro completo
```

13.3 Fallback: Read + Write invece di Edit

Quando `Edit` fallisce per match non univoco:

1. Read — caricare l'intero contenuto del file
2. Modificare il contenuto in modo programmatico
3. Write — riscrivere la versione aggiornata

PARTE II: NOTE PER DOMINIO D'ESAME

Dominio 1: Architettura degli agenti e orchestrazione (27%)

1.1 Progettazione di agentic loop per l’esecuzione autonoma dei task

Conoscenze chiave

- Ciclo di vita dell’agent loop: inviare una richiesta a Claude, verificare `stop_reason` (`"tool_use"` vs `"end_turn"`), eseguire i tool, reinserire i risultati per l’iterazione successiva
- I risultati dei tool vengono aggiunti alla history della conversazione, consentendo al modello di decidere l’azione successiva
- Decisione guidata dal modello (Claude seleziona il prossimo tool) vs logiche deterministiche hard-coded

Competenze chiave

- Controllo del flusso: continuare il loop quando `stop_reason = "tool_use"` e terminare su `"end_turn"`
- Aggiunta dei risultati dei tool al contesto tra un’iterazione e la successiva
- Anti-pattern da evitare: parsing del testo dell’assistant per determinare la fine, uso di limiti arbitrari di iterazioni come meccanismo primario di stop

1.2 Orchestrazione di sistemi multi-agent (Coordinator–Subagent)

Conoscenze chiave

- Architettura hub-and-spoke: il coordinator gestisce tutta la comunicazione tra agenti, l’error handling e il routing
- I subagent operano con contesto isolato: non ereditano automaticamente la history del coordinator
- Responsabilità del coordinator: decomposizione del task, delega, aggregazione dei risultati, selezione dinamica dei subagent
- Rischio di decomposizione eccessivamente frammentata da parte del coordinator

Competenze chiave

- Suddividere la copertura di ricerca tra subagent per minimizzare duplicazioni
- Implementare loop di raffinamento iterativo (il coordinator valuta la sintesi e rialloca i task)
- Instradare tutta la comunicazione attraverso il coordinator per garantire osservabilità

1.3 Configurazione delle chiamate ai subagent, context passing e spawning

Conoscenze chiave

- Il tool `Task` permette di spawnare subagent; `allowedTools` del coordinator deve includere `"Task"`
- Il contesto del subagent deve essere passato esplicitamente nel prompt; non esiste inheritance automatica
- Configurazione `AgentDefinition`: descrizioni, system prompt, vincoli sui tool
- Gestione sessioni tramite `fork_session` per esplorare alternative

Competenze chiave

- Includere output completi di agent precedenti nel prompt del subagent
- Usare formati strutturati per separare dati e metadata nel context passing
- Avviare subagent in parallelo tramite più chiamate `Task` in una singola iterazione del coordinator
- Scrivere prompt del coordinator orientati a obiettivi e criteri di qualità, non a istruzioni passo-passo

1.4 Implementazione di workflow multi-step con enforcement e pattern di handoff

Conoscenze chiave

- Differenza tra **enforcement programmatico** (hooks, precondizioni) e guida tramite prompt per l'ordinamento di un workflow
- Quando sono necessarie garanzie deterministiche (es. verifica identità prima di operazioni finanziarie), i prompt non sono sufficienti
- Protocolli strutturati di handoff durante l'escalation (customer ID, motivo, azione raccomandata)

Competenze chiave

- Precondizioni programmatiche che bloccano le chiamate downstream finché gli step precedenti non sono completati (es. bloccare `process_refund` finché `get_customer` non restituisce un ID verificato)
- Scomposizione di richieste multi-aspetto del cliente in elementi separati
- Produzione di summary strutturati durante l'escalation verso un operatore umano

1.5 Hook dell'Agent SDK per intercettazione chiamate tool e normalizzazione dati

Conoscenze chiave

- Pattern di hook (es. `PostToolUse`) per intercettare i risultati dei tool prima che vengano consumati dal modello
- Hook che intercettano chiamate in uscita per imporre regole di compliance (es. blocco rimborsi sopra una soglia)

- Gli hook forniscono **garanzie deterministiche**, mentre le istruzioni nei prompt forniscono **compliance probabilistica**

Competenze chiave

- `PostToolUse` hook per normalizzare formati dati (timestamp Unix, ISO 8601, codici numerici di stato)
- Hook di intercettazione per bloccare azioni non conformi alle policy e reindirizzare verso escalation
- Uso degli hook al posto dei prompt quando le regole di business richiedono enforcement garantito

1.6 Strategie di decomposizione dei task per workflow complessi

Conoscenze chiave

- **Pipeline fisse** (prompt chaining) vs **decomposizione dinamica adattiva** basata su risultati intermedi
- Prompt chaining: sequenze deterministiche (analisi file per file, poi fase di integrazione)
- Piani di investigazione adattivi che generano sotto-task in base ai risultati emersi

Competenze chiave

- Usare prompt chaining per review multi-aspetto prevedibili; usare decomposizione dinamica per investigazioni aperte
- Suddividere code review ampie in analisi per file + passaggio finale di integrazione cross-file
- Decomporre task esplorativi: prima mappare la struttura, poi costruire un piano prioritizzato

1.7 Stato di sessione, resume e fork

Conoscenze chiave

- `--resume <session-name>` per continuare sessioni nominate
- `fork_session` per creare branch indipendenti di investigazione a partire da un contesto condiviso
- Importanza di informare l'agente quando i file sono cambiati durante il resume di una sessione
- Una nuova sessione con un riepilogo strutturato può essere più affidabile rispetto al ripristino di una sessione con risultati obsoleti

Competenze chiave

- Usare `--resume` per continuare sessioni di investigazione nominate
- Usare `fork_session` per confrontare approcci in parallelo
- Scegliere tra resume (contesto ancora valido) e nuova sessione (risultati obsoleti o degradati)

Dominio 2: Progettazione degli Strumenti e Integrazione MCP (18%)

2.1 Progettare Interfacce di Strumenti con Descrizioni Chiare

Conoscenze chiave

- Le descrizioni degli strumenti sono il **meccanismo principale** che un LLM utilizza per selezionare gli strumenti; descrizioni minimali portano a una selezione poco affidabile.
- L'importanza di includere formati di input, esempi di query, casi limite ed eventuali limiti di applicabilità.
- Descrizioni ambigue o sovrapposte causano instradamenti errati.
- La formulazione del prompt di sistema può creare associazioni indesiderate con determinati strumenti.

Competenze chiave

- Scrivere descrizioni che distinguano chiaramente ogni strumento da alternative simili.
- Rinominare gli strumenti per eliminare sovrapposizioni funzionali (ad esempio, `analyze_content` → `extract_web_results`).
- Suddividere strumenti generici in strumenti specializzati con contratti di input/output ben definiti.

2.2 Implementare Risposte di Errore Strutturate per gli Strumenti MCP

Conoscenze chiave

- Il flag `isError` nelle risposte degli strumenti MCP.
- La differenza tra **errori transitori** (timeout), **errori di validazione** (input non valido), **errori di business** (violazioni delle policy) ed **errori di accesso/autorizzazione**.
- Errori generici ("Operazione non riuscita") impediscono di prendere decisioni corrette sul recupero dell'errore.
- La differenza tra errori recuperabili tramite nuovo tentativo (retryable) e non recuperabili.

Competenze chiave

- Restituire metadati strutturati come `errorCategory` (transient/validation/permission), `isRetryable` e un messaggio leggibile dall'utente.
- Utilizzare `retryable: false` per violazioni delle regole di business, accompagnandolo con spiegazioni chiare rivolte all'utente.
- Gestire il recupero locale all'interno dei sottoagenti in caso di errori transitori; propagare solo gli errori che non possono essere risolti autonomamente.
- Distinguere i problemi di accesso (che influenzano la decisione di effettuare un nuovo tentativo) dai risultati validamente vuoti (nessuna corrispondenza trovata).

2.3 Assegnare gli Strumenti agli Agenti e Configurare `tool_choice`

Conoscenze chiave

- Un numero eccessivo di strumenti per agente (ad esempio 18 invece di 4–5) **riduce** l'affidabilità della selezione degli strumenti.
- Gli agenti che dispongono di strumenti al di fuori della propria specializzazione tendono a utilizzarli in modo improprio.
- Accesso limitato agli strumenti (scoped tool access): solo strumenti pertinenti al ruolo, più un insieme ristretto di utility condivise tra ruoli.
- `tool_choice`: `"auto"`, `"any"` e selezione forzata di uno strumento (`{"type": "tool", "name": "..."}`).

Competenze chiave

- Limitare il set di strumenti di ogni sottoagente a quelli rilevanti per il suo ruolo.
- Sostituire strumenti generici con alternative più vincolate (ad esempio, `fetch_url` → `load_document`).
- Utilizzare `tool_choice: "any"` per garantire una chiamata a uno strumento invece di ottenere una risposta testuale.
- Forzare l'uso di uno strumento specifico per garantire un determinato ordine di esecuzione.

2.4 Integrare Server MCP in Claude Code e nei Flussi di Lavoro degli Agenti

Conoscenze chiave

- Ambito dei server MCP: progetto (`.mcp.json`) per i team, utente (`~/.claude.json`) per esperimenti personali.
- Sostituzione delle variabili d'ambiente in `.mcp.json` (ad esempio `${GITHUB_TOKEN}`) per la gestione dei segreti e delle credenziali.
- Gli strumenti provenienti da tutti i server MCP connessi vengono rilevati al momento della connessione e sono disponibili simultaneamente.
- Le risorse MCP come "cataloghi di contenuti" (riepiloghi di attività, schemi di database) riducono la necessità di chiamate esplorative agli strumenti.

Competenze chiave

- Configurare server MCP condivisi nel file `.mcp.json` del progetto utilizzando token basati su variabili d'ambiente.
- Mantenere i server personali o sperimentali nel file `~/.claude.json`.
- Preferire server MCP della community rispetto a server personalizzati per le integrazioni standard.

2.5 Selezionare e Utilizzare gli Strumenti Integrati (Read, Write, Edit, Bash, Grep, Glob)

Conoscenze chiave

- **Grep**: ricerca all'interno del contenuto dei file (nomi di funzioni, messaggi di errore, importazioni).
- **Glob**: ricerca di file tramite pattern di nomi o estensioni.
- **Read/Write**: operazioni sull'intero file; **Edit**: modifiche mirate tramite corrispondenze di testo univoche.
- Se Edit fallisce a causa di corrispondenze non univoche, utilizzare Read + Write come alternativa.

Competenze chiave

- Utilizzare Grep per la ricerca nei contenuti e Glob per individuare file tramite pattern.
- Costruire la comprensione del codice in modo incrementale: usare prima Grep per individuare i punti di ingresso, poi Read per seguire il flusso di esecuzione.
- Tracciare l'utilizzo delle funzioni attraverso moduli wrapper.

Dominio 3: Configurazione di Claude Code e Flussi di Lavoro (20%)

3.1 Configurare CLAUDE.md con Gerarchia, Ambito e Organizzazione Modulare

Conoscenze chiave

- Gerarchia di CLAUDE.md: livello utente (`~/.claude/CLAUDE.md`), livello progetto (`.claude/CLAUDE.md` oppure `CLAUDE.md` nella root del progetto) e livello directory (file `CLAUDE.md` nelle sottodirectory).
- Le impostazioni a livello utente si applicano soltanto a un singolo utente e non vengono condivise tramite il sistema di controllo versione (VCS).
- Sintassi `@path` per fare riferimento a file esterni (ad esempio `@./standards/coding-style.md`) e modularizzare il contenuto di CLAUDE.md.
- La directory `.claude/rules/` per organizzare regole specifiche per argomento invece di utilizzare un unico CLAUDE.md monolitico.

Competenze chiave

- Diagnosticare problemi di gerarchia (ad esempio, un nuovo membro del team non riceve determinate istruzioni perché sono definite a livello utente anziché a livello progetto).
- Utilizzare `@path` (ad esempio `@./standards/testing.md`) per includere selettivamente gli standard appropriati nel file CLAUDE.md di ciascun pacchetto.
- Suddividere un CLAUDE.md di grandi dimensioni in più file all'interno di `.claude/rules/` (ad esempio `testing.md` , `api-conventions.md` , `deployment.md`).

3.2 Creare e Configurare Comandi Slash Personalizzati e Skill

Conoscenze chiave

- **Comandi di progetto** nella directory `.claude/commands/` (condivisi tramite VCS) rispetto ai **comandi utente** in `~/.claude/commands/`.
- Skill archiviate in `.claude/skills/` con frontmatter in `SKILL.md`: `context: fork`, `allowed-tools`, `argument-hint`.
- `context: fork` esegue la skill in un contesto isolato di sottoagente, evitando che influenzi o "inquini" la sessione principale.
- Varianti personali delle skill possono essere conservate in `~/.claude/skills/` con nomi differenti.

Competenze chiave

- Salvare i comandi slash di progetto nella directory `.claude/commands/` affinché siano disponibili a tutto il team.
- Utilizzare `context: fork` per isolare skill che producono output molto verboso.
- Utilizzare `allowed-tools` per limitare gli strumenti che una skill può utilizzare.
- Utilizzare `argument-hint` per richiedere agli sviluppatori i parametri necessari.

3.3 Utilizzare Regole Specifiche per Percorso per il Caricamento Condizionale delle Convenzioni

Conoscenze chiave

- I file presenti in `.claude/rules/` possono includere un frontmatter YAML con il campo `paths` per attivare regole in base a pattern glob.
- Le regole limitate a specifici percorsi vengono caricate **solo** quando si modificano file corrispondenti, riducendo il consumo di contesto e token.
- Le regole basate su pattern glob possono essere preferibili rispetto a CLAUDE.md a livello di directory quando le convenzioni si applicano a molte directory diverse (ad esempio ai test).

Competenze chiave

- Creare file in `.claude/rules/` con `paths: ["terraform/**/*"]` per caricare le regole solo quando si lavora su file corrispondenti.
- Utilizzare pattern glob (ad esempio `**/*.test.tsx`) per applicare convenzioni in base al tipo di file indipendentemente dalla loro posizione.
- Preferire regole specifiche per percorso rispetto a CLAUDE.md a livello di directory quando le convenzioni si estendono a tutto il codebase.

3.4 Decidere Quando Utilizzare la Modalità di Pianificazione rispetto all'Esecuzione Diretta

Conoscenze chiave

- **Modalità di pianificazione (Planning mode)**: adatta a compiti complessi che comportano modifiche estese, più approcci possibili e decisioni architetturali.
- **Esecuzione diretta (Direct execution)**: adatta a modifiche semplici e ben comprese (ad esempio l'aggiunta di una singola validazione).
- La modalità di pianificazione consente di esplorare in sicurezza il codebase prima di apportare modifiche.
- Il sottoagente Explore isola l'output dettagliato dell'attività di esplorazione.

Competenze chiave

- Utilizzare la modalità di pianificazione per attività con implicazioni architetturali (microservizi, migrazioni che coinvolgono oltre 45 file).
- Utilizzare l'esecuzione diretta per correzioni supportate da uno stack trace chiaro e che coinvolgono un solo file.
- Utilizzare il sottoagente Explore per evitare l'esaurimento della finestra di contesto nelle attività suddivise in più fasi.
- Combinare i due approcci: pianificare la fase di analisi ed esplorazione, quindi eseguire l'implementazione.

3.5 Perfezionamento Iterativo per il Miglioramento Progressivo

Conoscenze chiave

- Esempi concreti di input e output rappresentano il modo più efficace per comunicare le aspettative.
- **Iterazione guidata dai test (Test-driven iteration)**: scrivere prima i test e poi iterare in base agli errori e ai fallimenti riscontrati.
- Il modello dell'"intervista" (interview pattern): Claude pone domande per far emergere considerazioni progettuali non immediatamente evidenti.
- Quando presentare tutti i problemi in un unico messaggio (se interdipendenti) rispetto a presentarli in modo sequenziale (se indipendenti).

Competenze chiave

- Fornire 2–3 esempi concreti di input/output per chiarire i requisiti di una trasformazione.
- Costruire set di test che includano comportamento atteso, casi limite e requisiti prestazionali prima dell'implementazione.
- Utilizzare il modello dell'intervista per far emergere aspetti progettuali rilevanti (invalidazione della cache, modalità di errore, ecc.).
- Fornire casi di test concreti con input di esempio e output attesi per i casi limite.

3.6 Integrare Claude Code nelle Pipeline CI/CD

Conoscenze chiave

- Il flag `-p` (oppure `--print`) per l'esecuzione in modalità non interattiva all'interno di pipeline automatizzate.
- `--output-format json` e `--json-schema` per ottenere output strutturati nei processi CI.
- CLAUDE.md fornisce il contesto del progetto (standard di testing, criteri di revisione) alle esecuzioni di Claude Code avviate dalla CI.
- **Isolamento del contesto di sessione**: la stessa sessione che ha generato il codice è generalmente meno efficace nel revisionarlo rispetto a un'istanza indipendente.

Competenze chiave

- Eseguire Claude Code in CI utilizzando `-p` per evitare blocchi dovuti alla richiesta di input interattivi.
- Utilizzare `--output-format json` insieme a `--json-schema` per produrre risultati strutturati (ad esempio commenti inline nelle pull request).
- Includere i risultati delle revisioni precedenti quando si riesegue l'analisi dopo nuovi commit (segnalando solo i problemi nuovi o non ancora risolti).
- Documentare in CLAUDE.md gli standard di testing e i fixture disponibili per migliorare la qualità della generazione dei test.
- Includere nel contesto i file di test esistenti quando si generano nuovi test, per evitare duplicazioni e mantenere uno stile coerente.

Dominio 4: Prompt Engineering e Output Strutturato (20%)

4.1 Progettare Prompt con Criteri Espliciti per Migliorare l'Accuratezza

Conoscenze chiave

- I criteri espliciti sono più efficaci delle istruzioni vaghe (ad esempio, “segnala i commenti solo quando contraddicono il codice” invece di “verifica l'accuratezza dei commenti”).
- Indicazioni generiche come “sii più conservativo” funzionano peggio rispetto a criteri concreti e categorizzati.
- L'effetto dei falsi positivi sulla fiducia degli sviluppatori: tassi elevati di falsi positivi in alcune categorie compromettono la fiducia anche nelle categorie in cui le segnalazioni sono accurate.

Competenze chiave

- Definire criteri di revisione chiari: cosa segnalare (bug, vulnerabilità di sicurezza) e cosa ignorare (piccoli problemi di stile).
- Disattivare temporaneamente le categorie che presentano un alto tasso di falsi positivi.

- Definire criteri di gravità espliciti corredati da esempi di codice per ciascun livello.

4.2 Utilizzare il Few-shot Prompting per Migliorare la Coerenza dell'Output

Conoscenze chiave

- Gli esempi few-shot rappresentano il metodo più efficace per ottenere output coerenti, formattati correttamente e immediatamente utilizzabili.
- Il few-shot può mostrare come gestire casi ambigui (selezione degli strumenti, lacune nella copertura dei test).
- Il few-shot aiuta il modello a generalizzare nuovi schemi invece di limitarsi a ripetere comportamenti predefiniti.
- Il few-shot può ridurre le allucinazioni nei compiti di estrazione delle informazioni.

Competenze chiave

- Fornire da 2 a 4 esempi mirati per scenari ambigui, includendo anche la motivazione delle scelte effettuate.
- Includere esempi few-shot che mostrino chiaramente il formato dell'output (posizione, problema, gravità, correzione suggerita).
- Fornire esempi che distinguano pattern di codice accettabili da problemi reali.
- Fornire esempi di estrazione corretta da documenti con strutture differenti.

4.3 Applicare Output Strutturati con `tool_use` e Schemi JSON

Conoscenze chiave

- L'utilizzo di `tool_use` con schemi JSON è il metodo più affidabile per garantire output conformi allo schema ed eliminare errori di sintassi JSON.
- Con `tool_choice: "auto"` il modello può restituire testo libero; con `"any"` deve necessariamente chiamare uno strumento; la selezione forzata impone l'uso di uno strumento specifico.
- Schemi JSON rigorosi eliminano gli errori sintattici, ma non quelli semantici (totali incoerenti, valori inseriti nel campo sbagliato).
- Progettazione degli schemi: campi obbligatori e facoltativi; enum con valori come `"other"` accompagnati da un campo descrittivo per garantire l'estensibilità.

Competenze chiave

- Definire strumenti di estrazione basati su schemi JSON e analizzare i dati restituiti dai risultati di `tool_use`.
- Utilizzare `tool_choice: "any"` per garantire un output strutturato quando sono disponibili più schemi.
- Forzare la chiamata di uno strumento specifico: `tool_choice: {"type": "tool", "name": "extract_metadata"}`.

- Rendere i campi facoltativi o nullable quando la fonte potrebbe non contenere l'informazione, evitando di inventare dati mancanti.
- Utilizzare valori enum come `"unclear"` e `"other"` accompagnati da campi descrittivi per una categorizzazione estendibile.

4.4 Implementare Validazione, Retry e Cicli di Feedback per Migliorare la Qualità dell'Estrazione

Conoscenze chiave

- Retry con feedback sugli errori (retry-with-error-feedback): includere nel prompt di retry errori di validazione concreti per guidare la correzione.
- I tentativi ripetuti sono inefficaci quando l'informazione richiesta semplicemente non è presente nella fonte.
- Progettazione dei cicli di feedback: tracciare il pattern che ha generato una segnalazione (`detected_pattern`).
- Errori semantici (totali non coerenti) rispetto a errori sintattici (già gestiti tramite `tool_use`).

Competenze chiave

- Creare prompt di follow-up che includano il documento originale, un'estrazione errata e specifici errori di validazione.
- Riconoscere quando un nuovo tentativo sarà inefficace (ad esempio quando le informazioni richieste sono disponibili solo in un documento esterno).
- Includere campi `detected_pattern` nelle segnalazioni per analizzare i falsi positivi.
- Progettare meccanismi di autocorrezione estraendo sia `calculated_total` sia `stated_total` per individuare eventuali discrepanze.

4.5 Progettare Strategie Efficienti di Elaborazione Batch

Conoscenze chiave

- API Message Batches: riduzione dei costi del 50%, finestra di elaborazione fino a 24 ore e assenza di garanzie SLA sulla latenza.
- L'elaborazione batch è adatta a attività non bloccanti (report notturni, audit) e non è adatta a attività bloccanti (controlli pre-merge).
- L'API Batch non supporta chiamate a strumenti multi-turn all'interno della stessa richiesta.
- I campi `custom_id` consentono di correlare richieste e risposte all'interno dei batch.

Competenze chiave

- Utilizzare l'API sincrona per controlli bloccanti e l'API Batch per carichi di lavoro notturni o settimanali.
- Pianificare la frequenza di invio dei batch in base ai requisiti SLA (ad esempio finestre di 4 ore per garantire una copertura operativa di 30 ore con un tempo di elaborazione massimo di 24 ore).

- Gestire i fallimenti reinviando soltanto i documenti che hanno generato errori (identificati tramite `custom_id`).
- Perfezionare i prompt utilizzando un campione rappresentativo prima di avviare elaborazioni su larga scala.

4.6 Progettare Architetture di Revisione Multi-Istanza e Multi-Passaggio

Conoscenze chiave

- Limiti dell'autorevisione (self-review): il modello mantiene il proprio contesto di ragionamento ed è meno incline a mettere in discussione le proprie decisioni.
- Le istanze di revisione indipendenti (prive del contesto di generazione) sono più efficaci nell'individuare problemi sottili.
- Revisione multi-passaggio (multi-pass review): analisi locale per singolo file seguita da una fase di integrazione tra file per evitare la dispersione dell'attenzione.

Competenze chiave

- Utilizzare una seconda istanza indipendente di Claude per revisionare le modifiche senza il contesto della generazione originale.
- Suddividere le revisioni che coinvolgono più file in passaggi per singolo file, seguiti da passaggi di integrazione per analizzare i flussi di dati tra file.
- Utilizzare passaggi di verifica con autovalutazione del livello di confidenza per instradare le revisioni in modo calibrato e coerente.

Dominio 5: Gestione del Contesto e Affidabilità (15%)

5.1 Gestire il Contesto della Conversazione per Preservare le Informazioni Critiche

Conoscenze chiave

- Rischi della sintesi progressiva (progressive summarization): valori numerici, percentuali e date tendono a essere condensati in riepiloghi vaghi.
- Effetto "lost-in-the-middle": i modelli elaborano in modo affidabile l'inizio e la fine di input molto lunghi, ma possono trascurare informazioni presenti nella parte centrale.
- Gli output degli strumenti possono accumularsi nel contesto in modo sproporzionato rispetto alla loro rilevanza (ad esempio oltre 40 campi quando ne servono solo 5).
- L'importanza di inviare l'intera cronologia della conversazione nelle richieste API successive.

Competenze chiave

- Estrarre i fatti transazionali in un blocco persistente di "case facts" separato dalla cronologia sintetizzata.
- Ridurre gli output verbosi degli strumenti mantenendo solo i campi rilevanti.
- Posizionare le informazioni chiave all'inizio dei dati aggregati utilizzando intestazioni di sezione esplicite.
- Richiedere ai sottoagenti di includere metadati (date, fonti) negli output strutturati.

5.2 Progettare Modelli di Escalation Efficaci e Risolvere le Ambiguità

Conoscenze chiave

- Trigger appropriati per l'escalation: richiesta esplicita di un operatore umano, lacune o eccezioni nelle policy, impossibilità di progredire nella risoluzione del problema.
- Escalation immediata (richiesta esplicita di un umano) rispetto a tentativi di risoluzione autonoma (quando rientrano nell'ambito dell'agente).
- L'analisi del sentiment e le autovalutazioni del livello di confidenza del modello sono indicatori poco affidabili della complessità di un caso.
- In presenza di più corrispondenze possibili per un cliente, è necessario richiedere identificatori aggiuntivi anziché affidarsi a supposizioni euristiche.

Competenze chiave

- Definire criteri di escalation espliciti nel prompt di sistema, corredati da esempi few-shot.
- Eseguire immediatamente le richieste esplicite di assistenza umana senza effettuare ulteriori indagini.
- Attivare l'escalation quando la policy è ambigua o non fornisce indicazioni per una richiesta specifica.
- Richiedere identificatori aggiuntivi quando i risultati degli strumenti contengono più corrispondenze.

5.3 Implementare Strategie di Propagazione degli Errori nei Sistemi Multiagente

Conoscenze chiave

- Un contesto di errore strutturato (tipo di errore, query eseguita, risultati parziali, alternative possibili) consente al coordinatore di adottare strategie di recupero più intelligenti.
- È importante distinguere i problemi di accesso (ad esempio timeout che richiedono una decisione sul retry) dai risultati validamente vuoti (nessuna corrispondenza trovata).
- Stati di errore generici ("ricerca non disponibile") nascondono informazioni preziose al coordinatore.
- Sopprimere silenziosamente gli errori o interrompere l'intero workflow a causa di un singolo fallimento sono entrambi anti-pattern.

Competenze chiave

- Restituire un contesto di errore strutturato che includa: tipo di errore, operazione tentata, risultati parziali e possibili alternative.
- Distinguere chiaramente i problemi di accesso dai risultati validamente vuoti.
- Effettuare il recupero locale nei sottoagenti per errori transitori; propagare soltanto gli errori non recuperabili insieme agli eventuali risultati parziali.
- Annotare il livello di copertura nella sintesi finale, indicando cosa è supportato da evidenze solide e dove permangono lacune informative.

5.4 Gestire il Contesto in Modo Efficiente Durante l'Analisi di Codebase di Grandi Dimensioni

Conoscenze chiave

- Degrado del contesto nelle sessioni lunghe: il modello inizia a produrre risposte meno stabili e a fare riferimento a "pattern tipici" invece che a classi o componenti specifici.
- I file scratchpad consentono di preservare le scoperte e le informazioni chiave oltre i limiti del contesto disponibile.
- La delega ai sottoagenti permette di isolare l'output dettagliato delle attività di esplorazione.
- La persistenza strutturata dello stato consente il recupero del lavoro in caso di interruzioni o crash.

Competenze chiave

- Creare sottoagenti dedicati a domande specifiche mantenendo il coordinamento generale nell'agente principale.
- Utilizzare file scratchpad per memorizzare le scoperte principali e farvi riferimento successivamente.
- Riassumere le informazioni chiave prima di avviare sottoagenti per le fasi successive dell'analisi.
- Utilizzare `/compact` per ridurre il consumo di contesto durante indagini prolungate.

5.5 Progettare Workflow con Supervisione Umana e Calibrazione della Confidenza

Conoscenze chiave

- Le metriche aggregate (ad esempio il 97% di accuratezza complessiva) possono nascondere prestazioni scarse su specifici tipi di documenti o campi.
- Il campionamento casuale stratificato (stratified random sampling) consente di misurare i tassi di errore anche nelle estrazioni ad alta confidenza.
- Calibrazione della confidenza a livello di campo tramite set di validazione etichettati.
- Prima di automatizzare un processo, è necessario validarne l'accuratezza per tipologia di documento e segmento di dati.

Competenze chiave

- Implementare il campionamento casuale stratificato per individuare nuovi pattern di errore.

- Analizzare l'accuratezza per tipologia di documento e per campo, verificando che le prestazioni rimangano stabili.
- Generare punteggi di confidenza a livello di singolo campo e calibrare le soglie di revisione utilizzando dati etichettati.
- Instradare verso revisione umana le estrazioni a bassa confidenza o provenienti da fonti ambigue.

5.6 Preservare la Provenienza delle Informazioni e Gestire l'Incertezza nella Sintesi da Più Fonti

Conoscenze chiave

- L'attribuzione delle informazioni viene persa durante la sintesi se non vengono mantenute le associazioni "affermazione → fonte".
- Le associazioni strutturate tra affermazioni e fonti devono essere preservate durante l'aggregazione.
- In presenza di statistiche contrastanti, è necessario annotare il conflitto con le relative attribuzioni invece di scegliere arbitrariamente un valore.
- Includere le date di pubblicazione o raccolta dei dati evita di interpretare erroneamente differenze temporali come contraddizioni.

Competenze chiave

- Richiedere ai sottoagenti di produrre associazioni "affermazione → fonte" (URL, nome del documento, citazioni).
- Strutturare i report separando chiaramente i risultati consolidati da quelli controversi o contestati.
- Conservare valori contrastanti con annotazioni esplicative e inoltrarli al coordinatore per la riconciliazione.
- Includere le date di pubblicazione per consentire una corretta interpretazione temporale delle informazioni.
- Presentare i contenuti in base alla loro tipologia: dati finanziari in tabelle, notizie in forma discorsiva, risultati tecnici come elenchi strutturati.

Esempi di Domande d'Esame con Spiegazioni

Domanda 1 (Scenario: Agente di Assistenza Clienti)

Situazione: I dati mostrano che nel 12% dei casi l'agente salta la chiamata a `get_customer` e utilizza `lookup_order` basandosi soltanto sul nome del cliente, causando rimborsi errati.

Quale modifica è più efficace?

- A) Aggiungere una precondizione a livello di codice che blocchi `lookup_order` e `process_refund` finché non viene ottenuto un ID tramite `get_customer` **[CORRETTA]**
- B) Migliorare il prompt di sistema
- C) Aggiungere esempi few-shot

- D) Implementare un classificatore di instradamento (routing classifier)

Perché A: Quando una logica di business critica richiede una specifica sequenza di utilizzo degli strumenti, il software può fornire **garanzie deterministiche** che gli approcci basati sui prompt (B, C) non possono offrire. L'opzione D riguarda la disponibilità o l'instradamento delle richieste, non l'ordine di esecuzione degli strumenti.

Domanda 2 (Scenario: Agente di Assistenza Clienti)

Situazione: L'agente utilizza spesso `get_customer` invece di `lookup_order` per domande relative agli ordini. Le descrizioni degli strumenti sono minime e molto simili tra loro.

Qual è il primo intervento da effettuare?

- A) Aggiungere esempi few-shot
- B) Ampliare la descrizione di ciascuno strumento includendo formati di input, esempi e limiti di utilizzo **[CORRETTA]**
- C) Aggiungere un livello di routing
- D) Unire i due strumenti

Perché B: Le descrizioni degli strumenti rappresentano il principale meccanismo con cui il modello decide quale strumento utilizzare. Questa soluzione richiede poco sforzo e produce il maggiore impatto. L'opzione A aggiunge token senza affrontare la causa principale del problema. L'opzione C rappresenta un eccesso di complessità. L'opzione D richiede più lavoro di quanto sia giustificato dalla situazione.

Domanda 3 (Scenario: Agente di Assistenza Clienti)

Situazione: L'agente riesce a risolvere soltanto il 55% dei problemi, mentre l'obiettivo è l'80%. Escala casi semplici e cerca di gestire autonomamente eccezioni di policy complesse.

Come si può migliorare la calibrazione delle decisioni?

- A) Aggiungere criteri di escalation espliciti accompagnati da esempi few-shot **[CORRETTA]**
- B) Utilizzare un'autovalutazione della confidenza (da 1 a 10) con escalation automatica
- C) Utilizzare un classificatore separato addestrato su dati storici
- D) Utilizzare l'analisi del sentiment

Perché A: Questa soluzione affronta direttamente la causa del problema: confini decisionali poco chiari. L'opzione B è poco affidabile (il modello può essere molto sicuro pur essendo in errore). L'opzione C rappresenta un eccesso di complessità. L'opzione D affronta un problema diverso (lo stato emotivo non corrisponde alla complessità del caso).

Domanda 4 (Scenario: Generazione di Codice con Claude Code)

Situazione: Hai bisogno di un comando personalizzato `/review` per la revisione standard del codice, disponibile a tutto il team quando clona il repository.

Dove dovresti creare il file del comando?

- A) `.claude/commands/` nel repository del progetto **[CORRETTA]**
- B) `~/.claude/commands/`
- C) `CLAUDE.md` nella root
- D) `.claude/config.json`

Perché A: I comandi di progetto salvati in `.claude/commands/` sono inclusi nel controllo versione e diventano automaticamente disponibili per tutti. B è per i comandi personali. C serve per le istruzioni, non per definire comandi. D non esiste.

Domanda 5 (Scenario: Generazione di Codice con Claude Code)

Situazione: Devi ristrutturare un monolite in microservizi (decine di file, decisioni sui confini tra servizi).

Quale approccio dovresti usare?

- A) Modalità di pianificazione: esplorare il codebase, comprendere le dipendenze, progettare un approccio **[CORRETTA]**
- B) Esecuzione diretta in modo incrementale
- C) Esecuzione diretta con istruzioni iniziali dettagliate
- D) Esecuzione diretta e passaggio alla pianificazione quando diventa difficile

Perché A: La modalità di pianificazione è pensata per modifiche estese, più approcci possibili e decisioni architetturali. B rischia di causare rilavorazioni costose. C presume che la struttura del progetto sia già nota. D è un approccio reattivo.

Domanda 6 (Scenario: Generazione di Codice con Claude Code)

Situazione: Un codebase ha convenzioni diverse a seconda delle aree (React, API, database). I test si trovano accanto al codice relativo. Vuoi che le convenzioni vengano applicate automaticamente.

Quale approccio dovresti usare?

- A) File in `.claude/rules/` con frontmatter YAML e pattern glob **[CORRETTA]**
- B) Inserire tutto nel file CLAUDE.md della root
- C) Skill in `.claude/skills/`
- D) File CLAUDE.md in ogni directory

Perché A: `.claude/rules/` con pattern glob (ad esempio `**/*.test.tsx`) consente di applicare automaticamente le convenzioni in base ai percorsi dei file: una soluzione ideale per test distribuiti in tutto

il codebase. B si affida all'inferenza del modello. C è manuale/su richiesta. D funziona male quando i file rilevanti si trovano in molte directory.

Domanda 7 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Il sistema ricerca “impatto dell'AI sulle industrie creative”, ma i report coprono soltanto le arti visive. Il coordinatore ha scomposto l'argomento in: “AI nell'arte digitale”, “AI nel design grafico”, “AI nella fotografia”.

Qual è la causa?

- A) L'agente di sintesi non rileva le lacune
- B) Il coordinatore ha scomposto il compito in modo troppo ristretto **[CORRETTA]**
- C) L'agente di ricerca web non cerca in modo abbastanza approfondito
- D) L'agente di analisi documentale filtra le fonti non visive

Perché B: I log mostrano che il coordinatore ha scomposto “industrie creative” solo in sottoargomenti visivi, tralasciando completamente musica, letteratura e cinema. I sottoagenti hanno eseguito correttamente il loro compito: il problema riguarda ciò che è stato assegnato loro.

Domanda 8 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Un sottoagente di ricerca web va in timeout mentre sta effettuando ricerche su un argomento complesso. Devi progettare il modo in cui le informazioni sugli errori vengono restituite al coordinatore.

Quale approccio alla propagazione degli errori consente il recupero più intelligente?

- A) Restituire al coordinatore un contesto di errore strutturato: tipo di errore, query eseguita, risultati parziali e possibili alternative **[CORRETTA]**
- B) Implementare retry automatici con backoff esponenziale all'interno del sottoagente e poi restituire uno stato generico "ricerca non disponibile"
- C) Gestire il timeout all'interno del sottoagente e restituire un insieme di risultati vuoto contrassegnato come successo
- D) Propagare l'eccezione di timeout a un gestore di livello superiore che interrompe l'intero workflow

Perché A: Un contesto di errore strutturato fornisce al coordinatore tutte le informazioni necessarie per decidere se riprovare con una query modificata, utilizzare un approccio alternativo o proseguire con i risultati parziali disponibili. L'opzione B nasconde informazioni preziose dietro uno stato generico. L'opzione C maschera un fallimento come se fosse un successo. L'opzione D interrompe inutilmente l'intero workflow.

Domanda 9 (Scenario: Sistema di Ricerca Multiagente)

Situazione: L'agente di sintesi deve spesso verificare affermazioni specifiche durante la fusione dei risultati. Attualmente, quando è necessaria una verifica, l'agente di sintesi restituisce il controllo al coordinatore, che richiama l'agente di ricerca web e poi riesegue la sintesi con i nuovi risultati. Questo aggiunge 2–3 passaggi aggiuntivi per attività e aumenta la latenza del 40%. L'analisi mostra che l'85% di queste verifiche consiste in semplici controlli fattuali (date, nomi, statistiche), mentre il 15% richiede indagini più approfondite.

Come puoi ridurre il sovraccarico mantenendo l'affidabilità?

- A) Fornire all'agente di sintesi uno strumento limitato `verify_fact` per le verifiche semplici e continuare a instradare le verifiche complesse tramite il coordinatore **[CORRETTA]**
- B) Accumulare tutte le esigenze di verifica in un batch e restituirle al coordinatore alla fine
- C) Concedere all'agente di sintesi accesso completo a tutti gli strumenti di ricerca web
- D) Memorizzare preventivamente ulteriore contesto attorno a ogni fonte

Perché A: Questo approccio applica il principio del privilegio minimo (principle of least privilege): l'agente di sintesi riceve esattamente ciò che serve per l'85% dei casi più comuni (semplici verifiche fattuali), mantenendo però il percorso mediato dal coordinatore per le indagini più complesse. L'opzione B introduce dipendenze bloccanti (fasi successive della sintesi potrebbero dipendere da verifiche precedenti). L'opzione C rompe la separazione delle responsabilità. L'opzione D si basa su una cache speculativa che non può prevedere in modo affidabile le necessità future.

Domanda 10 (Scenario: Claude Code in una Pipeline CI)

Situazione: Una pipeline esegue il comando `claude "Analyze this pull request for security issues"`, ma il processo rimane bloccato in attesa di input interattivo.

Qual è l'approccio corretto?

- A) Utilizzare il flag `-p`: `claude -p "Analyze this pull request for security issues"` **[CORRETTA]**
- B) Impostare `CLAUDE_HEADLESS=true`
- C) Reindirizzare lo standard input da `/dev/null`
- D) Utilizzare `--batch`

Perché A: `-p` (oppure `--print`) è il metodo documentato per eseguire Claude Code in modalità non interattiva. Elabora il prompt, stampa il risultato sullo standard output e termina l'esecuzione. Le altre opzioni sono funzionalità inesistenti oppure semplici workaround del sistema operativo Unix.

Domanda 11 (Scenario: Claude Code in una Pipeline CI)

Situazione: Il team vuole ridurre i costi API per l'analisi automatizzata. Claude attualmente serve due workflow in tempo reale: (1) un controllo pre-merge bloccante che deve essere completato prima che gli

sviluppatori possano fare il merge di una PR, e (2) un report sul debito tecnico generato durante la notte per essere revisionato al mattino. Un manager propone di spostare entrambi sull'API Message Batches per risparmiare il 50%.

Come dovresti valutare questa proposta?

- A) Utilizzare l'elaborazione batch solo per i report sul debito tecnico; mantenere le chiamate in tempo reale per i controlli pre-merge **[CORRETTA]**
- B) Spostare entrambi i workflow sull'elaborazione batch e fare polling fino al completamento
- C) Mantenere le chiamate in tempo reale per entrambi per evitare problemi di ordinamento nei risultati batch
- D) Spostare entrambi sull'elaborazione batch con fallback al tempo reale se un batch impiega troppo tempo

Perché A: L'API Message Batches consente di risparmiare il 50%, ma il tempo di elaborazione può arrivare fino a 24 ore e non offre garanzie SLA sulla latenza. Questo la rende inadatta ai controlli pre-merge bloccanti, nei quali gli sviluppatori attendono un risultato prima di procedere, ma ideale per carichi di lavoro notturni come i report sul debito tecnico.

Domanda 12 (Scenario: Revisione di Codice Multi-file)

Situazione: Una pull request modifica 14 file in un modulo di tracciamento dell'inventario. Una revisione a passaggio singolo di tutti i file produce risultati incoerenti: commenti dettagliati per alcuni file ma superficiali per altri, bug evidenti non rilevati e feedback contraddittori (un pattern viene segnalato come problematico in un file ma approvato in codice identico in un altro file).

Come dovresti ristrutturare la revisione?

- A) Suddividere in passaggi mirati: analizzare ogni file singolarmente per problemi locali, poi eseguire un passaggio di integrazione separato per i flussi di dati tra file **[CORRETTA]**
- B) Richiedere agli sviluppatori di suddividere le PR grandi in invii da 3–4 file
- C) Passare a un modello di livello superiore con una finestra di contesto più ampia per revisionare tutti e 14 i file in un unico passaggio
- D) Eseguire tre revisioni indipendenti dell'intera PR e segnalare solo i problemi individuati in almeno due esecuzioni

Perché A: I passaggi mirati affrontano direttamente la causa principale: la dispersione dell'attenzione quando vengono elaborati molti file contemporaneamente. L'analisi per singolo file garantisce una profondità coerente, mentre un passaggio di integrazione separato permette di individuare problemi tra file. B sposta il carico sugli sviluppatori senza migliorare il sistema. C è un fraintendimento: una finestra di contesto più ampia non risolve la qualità dell'attenzione. D rischia di sopprimere bug reali richiedendo consenso tra rilevazioni incoerenti.

Test di Esercitazione

> 60 domande distribuite su 4 scenari. Il formato e la difficoltà corrispondono a quelli dell'esame reale.
>
> In alternativa, puoi esercitarti con queste domande in un file HTML simile a un esame: Test Pratico (IT)

Scenario: Sistema di Ricerca Multiagente

Domanda 1 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Un agente di analisi documentale scopre che due fonti credibili contengono statistiche direttamente contraddittorie per una metrica chiave: un report governativo indica una crescita del 40%, mentre un'analisi di settore indica il 12%. Entrambe le fonti sembrano credibili e la discrepanza potrebbe influenzare in modo significativo le conclusioni della ricerca. Come dovrebbe gestire questa situazione l'agente di analisi documentale nel modo più efficace?

Quale approccio è più efficace?

- A) Applicare euristiche di credibilità per scegliere il numero più probabilmente corretto, completare l'analisi con quel valore e aggiungere una nota a piè di pagina che menzioni la discrepanza.
- B) Includere entrambi i numeri nell'output dell'analisi senza indicarli come contrastanti, lasciando che l'agente di sintesi decida quale usare in base al contesto più ampio.
- C) Interrompere l'analisi ed effettuare immediatamente un'escalation al coordinatore, chiedendogli di decidere quale fonte sia più autorevole prima di proseguire.
- D) Completare l'analisi con entrambi i numeri, annotare esplicitamente il conflitto con attribuzione alle fonti e lasciare che il coordinatore decida come riconciliare i dati prima di passarli alla sintesi. **[CORRETTA]**

Perché D: Questo approccio preserva la separazione delle responsabilità: l'agente di analisi completa il proprio lavoro principale senza bloccarsi, conserva entrambi i valori contrastanti con attribuzione chiara e passa correttamente la riconciliazione al coordinatore, che dispone di un contesto più ampio.

Domanda 2 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Gli agenti di ricerca web e di analisi documentale hanno completato i rispettivi compiti e restituito i risultati al coordinatore. Qual è il passo successivo per creare un report di ricerca integrato?

Quale passo successivo è più appropriato?

- A) Ogni agente invia i propri risultati direttamente all'agente di scrittura del report, bypassando il coordinatore.

- B) L'agente di analisi documentale richiede i risultati della ricerca web e li fonde internamente.
- C) Il coordinatore passa entrambi gli insiemi di risultati all'agente di sintesi per un'integrazione unificata. **[CORRETTA]**
- D) Il coordinatore concatena gli output grezzi di entrambi gli agenti e li restituisce come risultato finale.

Perché C: In un'architettura coordinatore-sottoagenti, il coordinatore inoltra entrambi gli insiemi di risultati all'agente di sintesi per un'integrazione centralizzata, mantenendo il controllo e garantendo una fusione di qualità elevata.

Domanda 3 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Un sottoagente di analisi documentale fallisce spesso durante l'elaborazione di file PDF: alcuni presentano sezioni corrotte che generano eccezioni di parsing, altri sono protetti da password e talvolta la libreria di parsing si blocca su file di grandi dimensioni. Attualmente, qualsiasi eccezione termina immediatamente il sottoagente e restituisce un errore al coordinatore, che deve decidere se riprovare, saltare il file o far fallire l'intero compito. Questo causa un coinvolgimento eccessivo del coordinatore nella gestione ordinaria degli errori. Quale miglioramento architetturale è più efficace?

Quale miglioramento è più efficace?

- A) Creare un agente dedicato alla gestione degli errori che monitori tutti i fallimenti tramite una coda condivisa e decida le azioni di recupero, inviando comandi di riavvio direttamente ai sottoagenti.
- B) Configurare il sottoagente affinché restituisca sempre risultati parziali con stato di successo, incorporando i dettagli dell'errore nei metadati; il coordinatore tratterà tutte le risposte come riuscite.
- C) Fare in modo che il coordinatore validi tutti i documenti prima di inviarli al sottoagente, rifiutando quelli che potrebbero causare fallimenti.
- D) Implementare il recupero locale nel sottoagente per gli errori transitori ed effettuare l'escalation al coordinatore solo per gli errori che non può risolvere, includendo i passaggi tentati e i risultati parziali. **[CORRETTA]**

Perché D: Gli errori vanno gestiti al livello più basso capace di risolverli. Il recupero locale riduce il carico sul coordinatore, continuando però a effettuare l'escalation dei problemi davvero non recuperabili con il contesto completo e i progressi parziali ottenuti.

Domanda 4 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Dopo aver eseguito il sistema su “impatto dell'AI sulle industrie creative”, osservi che ogni sottoagente completa correttamente il proprio compito: l'agente di ricerca web trova articoli pertinenti, l'agente di analisi documentale li riassume correttamente e l'agente di sintesi produce un testo coerente. Tuttavia, i report finali coprono solo le arti visive e ignorano completamente musica, letteratura e cinema. Nei log del coordinatore, vedi che il tema è stato scomposto in tre sottoattività: “AI nell'arte digitale”, “AI nel design grafico” e “AI nella fotografia”. Qual è la causa principale più probabile?

Qual è la causa principale più probabile?

- A) L'agente di sintesi non dispone di istruzioni per rilevare lacune di copertura.

- B) L'agente di analisi documentale filtra le fonti non visive a causa di criteri di rilevanza troppo rigidi.
- C) La scomposizione del compito da parte del coordinatore è troppo ristretta e assegna ai sottoagenti attività che non coprono tutte le aree rilevanti. **[CORRETTA]**
- D) Le query dell'agente di ricerca web sono insufficienti e dovrebbero essere ampliate per coprire più settori.

Perché C: Il coordinatore ha scomposto un tema ampio esclusivamente in sottoattività legate alle arti visive, tralasciando del tutto musica, letteratura e cinema. Poiché i sottoagenti hanno eseguito correttamente gli incarichi ricevuti, la causa evidente è una scomposizione iniziale troppo ristretta.

Domanda 5 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Il sottoagente di ricerca web restituisce risultati solo per 3 delle 5 categorie di fonti richieste: siti dei competitor e report di settore hanno successo, mentre archivi di notizie e feed social vanno in timeout. Il sottoagente di analisi documentale elabora correttamente tutti i documenti forniti. Il sottoagente di sintesi deve produrre un riepilogo a partire da input upstream di qualità mista. Quale strategia di propagazione degli errori è più efficace?

Quale strategia di propagazione degli errori è più efficace?

- A) Continuare la sintesi utilizzando solo le fonti disponibili e produrre un output senza menzionare quali dati non erano disponibili.
- B) Il sottoagente di sintesi restituisce un errore al coordinatore, attivando un retry completo o il fallimento dell'attività a causa dei dati incompleti.
- C) Il sottoagente di sintesi chiede al coordinatore di riprovare le fonti andate in timeout con un timeout più lungo prima di iniziare la sintesi.
- D) Strutturare l'output di sintesi con annotazioni di copertura che indichino quali conclusioni sono ben supportate e dove esistono lacune dovute a fonti non disponibili. **[CORRETTA]**

Perché D: Le annotazioni di copertura consentono una degradazione controllata e trasparente: preservano il valore del lavoro completato, ma propagano l'incertezza in modo esplicito, permettendo decisioni informate sul livello di confidenza.

Domanda 6 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Il sottoagente di analisi documentale incontra un file PDF corrotto che non riesce ad analizzare. Durante la progettazione della gestione degli errori del sistema, qual è il modo più efficace per gestire questo fallimento?

Quale approccio è più efficace?

- A) Restituire un errore con contesto all'agente coordinatore, permettendogli di decidere come procedere. **[CORRETTA]**
- B) Saltare silenziosamente il documento corrotto e continuare a elaborare i file rimanenti per evitare di interrompere il workflow.

- C) Riprovare automaticamente ad analizzare il documento tre volte con backoff esponenziale prima di segnalare il fallimento.
- D) Lanciare un'eccezione che termina l'intero workflow di ricerca.

Perché A: Restituire un errore con contesto al coordinatore è l'approccio più efficace perché gli consente di prendere una decisione informata: saltare il file, provare un metodo di parsing alternativo o informare l'utente, mantenendo al tempo stesso visibilità sul fallimento.

Domanda 7 (Scenario: Sistema di Ricerca Multiagente)

Situazione: I log di produzione mostrano un pattern persistente: richieste come “analizza il report trimestrale caricato” vengono instradate all'agente di ricerca web nel 45% dei casi invece che all'agente di analisi documentale. Esaminando le definizioni degli strumenti, scopri che l'agente di ricerca web ha uno strumento `analyze_content` descritto come “analizza contenuti ed estrae informazioni chiave”, mentre l'agente di analisi documentale ha uno strumento `analyze_document` descritto come “analizza documenti ed estrae informazioni chiave”. Come dovresti risolvere il problema di instradamento errato?

Come dovresti risolvere il problema di instradamento errato?

- A) Aggiungere un classificatore di pre-routing che rilevi se l'utente si riferisce a file caricati o a contenuti web prima che il coordinatore decida la delega.
- B) Rinominare lo strumento di ricerca web in `extract_web_results` e aggiornare la sua descrizione in “elabora e restituisce informazioni recuperate da ricerche web e URL”. **[CORRETTA]**
- C) Aggiungere esempi few-shot al prompt del coordinatore che mostrino l'instradamento corretto: “L'utente carica un report trimestrale → agente di analisi documentale” e “L'utente chiede informazioni su una pagina web → agente di ricerca web”.
- D) Ampliare la descrizione dello strumento di analisi documentale con esempi d'uso come “Usare per PDF, documenti Word e fogli di calcolo caricati”, lasciando invariato lo strumento di ricerca web.

Perché B: Rinominare lo strumento di ricerca web in `extract_web_results` e aggiornare la descrizione per fare riferimento esplicito a ricerche web e URL rimuove direttamente la causa principale del problema, eliminando la sovrapposizione semantica tra i nomi e le descrizioni dei due strumenti. In questo modo lo scopo di ciascuno strumento diventa univoco e il coordinatore può distinguere in modo affidabile l'analisi documentale dalla ricerca web.

Domanda 8 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Un collega propone che l'agente di analisi documentale invii i propri risultati direttamente all'agente di sintesi, bypassando il coordinatore. Qual è il principale vantaggio di mantenere il coordinatore come hub centrale per tutte le comunicazioni tra sottoagenti?

Qual è il principale vantaggio di mantenere il coordinatore come hub centrale?

- A) Il coordinatore può osservare tutte le interazioni, gestire gli errori in modo uniforme e decidere quali informazioni debba ricevere ciascun sottoagente. **[CORRETTA]**

- B) Il coordinatore raggruppa più richieste ai sottoagenti in batch, riducendo il numero totale di chiamate API e la latenza complessiva.
- C) Il passaggio attraverso il coordinatore abilita una logica di retry automatica che le chiamate dirette tra agenti non possono supportare.
- D) I sottoagenti usano memoria isolata e la comunicazione diretta richiederebbe una serializzazione complessa che solo il coordinatore può eseguire.

Perché A: Il pattern con coordinatore offre visibilità centralizzata su tutte le interazioni, una gestione uniforme degli errori in tutto il sistema e un controllo preciso su quali informazioni riceve ciascun sottoagente: questi sono i principali vantaggi di una topologia di comunicazione a stella.

Domanda 9 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Il sottoagente di ricerca web va in timeout mentre sta effettuando ricerche su un argomento complesso. Devi progettare il modo in cui le informazioni su questo fallimento vengono restituite al coordinatore. Quale approccio alla propagazione degli errori consente il recupero più intelligente?

Quale approccio alla propagazione degli errori consente il recupero più intelligente?

- A) Restituire al coordinatore un contesto di errore strutturato che includa il tipo di fallimento, la query eseguita, eventuali risultati parziali e possibili approcci alternativi. **[CORRETTA]**
- B) Gestire il timeout all'interno del sottoagente e restituire un insieme di risultati vuoto contrassegnato come riuscito.
- C) Implementare retry automatici con backoff esponenziale all'interno del sottoagente, restituendo solo uno stato generico “ricerca non disponibile” dopo aver esaurito i tentativi.
- D) Propagare l'eccezione di timeout direttamente al gestore di livello superiore, terminando l'intero workflow di ricerca.

Perché A: Restituire un contesto di errore strutturato — includendo tipo di fallimento, query eseguita, risultati parziali e approcci alternativi — fornisce al coordinatore tutto ciò che serve per prendere decisioni di recupero intelligenti, ad esempio riprovare con una query modificata o proseguire con i risultati parziali. Questo preserva il massimo contesto possibile per decisioni informate a livello di coordinamento.

Domanda 10 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Nella progettazione del sistema hai dato all'agente di analisi documentale accesso a uno strumento generico `fetch_url` affinché potesse scaricare documenti tramite URL. I log di produzione mostrano che questo agente ora scarica spesso pagine di risultati dei motori di ricerca per svolgere ricerche web ad hoc — comportamento che dovrebbe essere instradato all'agente di ricerca web — causando risultati incoerenti. Quale correzione è più efficace?

Quale correzione è più efficace?

- A) Sostituire `fetch_url` con uno strumento `load_document` che validi che gli URL puntino a formati di documento. **[CORRETTA]**

- B) Rimuovere `fetch_url` dall'agente di analisi documentale e instradare tutto il recupero degli URL tramite il coordinatore verso l'agente di ricerca web.
- C) Implementare un filtro che blocchi le chiamate `fetch_url` verso domini di motori di ricerca noti, consentendo però altri URL.
- D) Aggiungere istruzioni al prompt dell'agente di analisi documentale specificando che `fetch_url` deve essere usato solo per scaricare URL di documenti, non per effettuare ricerche.

Perché A: Sostituire uno strumento generico con uno strumento specifico per documenti, capace di validare gli URL rispetto ai formati documentali, risolve la causa principale vincolando la capacità a livello di interfaccia. Questo segue il principio del privilegio minimo, rendendo impossibile il comportamento di ricerca indesiderato invece di limitarsi a scoraggiarlo.

Domanda 11 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Durante la ricerca su un argomento ampio, osservi che l'agente di ricerca web e l'agente di analisi documentale indagano gli stessi sottoargomenti, producendo una sostanziale duplicazione nei loro output. L'uso dei token quasi raddoppia senza un aumento proporzionale dell'ampiezza o della profondità della ricerca. Qual è il modo più efficace per risolvere il problema?

Qual è il modo più efficace per risolvere il problema?

- A) Consentire a entrambi gli agenti di completare il lavoro in parallelo, poi fare in modo che il coordinatore deduplichi i risultati sovrapposti prima di passarli all'agente di sintesi.
- B) Il coordinatore partiziona esplicitamente lo spazio di ricerca prima della delega, assegnando a ciascun agente sottoargomenti o tipi di fonti distinti. **[CORRETTA]**
- C) Implementare un meccanismo di stato condiviso in cui gli agenti registrano la propria area di interesse corrente, così che gli altri agenti possano evitare dinamicamente le duplicazioni durante l'esecuzione.
- D) Passare a un'esecuzione sequenziale in cui l'analisi documentale viene eseguita solo dopo il completamento della ricerca web, utilizzando i risultati della ricerca web come contesto per evitare duplicazioni.

Perché B: Fare in modo che il coordinatore partizioni esplicitamente lo spazio di ricerca prima della delega è l'approccio più efficace perché affronta la causa principale — confini del compito poco chiari — prima che il lavoro inizi. Questo preserva il parallelismo evitando al tempo stesso lavoro duplicato e spreco di token.

Domanda 12 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Durante la ricerca, il sottoagente di ricerca web interroga tre categorie di fonti con esiti diversi: i database accademici restituiscono 15 paper pertinenti, i report di settore restituiscono “0 risultati” e i database dei brevetti restituiscono “Timeout di connessione”. Nella progettazione della propagazione degli errori al coordinatore, quale approccio consente le migliori decisioni di recupero?

Quale approccio consente le migliori decisioni di recupero?

- A) Aggregare i risultati in un'unica metrica percentuale di successo (ad esempio “67% di copertura delle fonti”), con log dettagliati disponibili su richiesta.
- B) Segnalare sia “timeout” sia “0 risultati” come fallimenti che richiedono l'intervento del coordinatore.
- C) Riprovare internamente gli errori transitori e segnalare solo gli errori persistenti.
- D) Distinguere i problemi di accesso (timeout), che richiedono una decisione sul retry, dai risultati validamente vuoti (“0 risultati”), che rappresentano query eseguite con successo. **[CORRETTA]**

Perché D: Un timeout (problema di accesso) e “0 risultati” (risultato validamente vuoto) sono esiti semanticamente diversi e richiedono risposte diverse. Distinguerli consente al coordinatore di riprovare con il database dei brevetti, accettando invece lo “0 risultati” dei report di settore come un'informazione valida.

Domanda 13 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Il monitoraggio in produzione mostra una qualità di sintesi incoerente. Quando i risultati aggregati raggiungono circa 75K token, l'agente di sintesi cita in modo affidabile informazioni dai primi 15K token (titoli e snippet della ricerca web) e dagli ultimi 10K token (conclusioni dell'analisi documentale), ma spesso ignora risultati critici nei 50K token centrali, anche quando rispondono direttamente alla domanda di ricerca. Come dovresti ristrutturare l'input aggregato?

Come dovresti ristrutturare l'input aggregato?

- A) Riassumere tutti gli output dei sottoagenti sotto i 20K token prima dell'aggregazione, per mantenere il contenuto entro un intervallo elaborabile in modo affidabile dal modello.
- B) Trasmettere progressivamente i risultati dei sottoagenti all'agente di sintesi, elaborando prima fino in fondo i risultati della ricerca web e poi aggiungendo quelli dell'analisi documentale.
- C) Inserire un riepilogo dei risultati chiave all'inizio dell'input aggregato e organizzare i risultati dettagliati con intestazioni di sezione esplicite per facilitarne la navigazione. **[CORRETTA]**
- D) Implementare una rotazione che alterni quale output dei sottoagenti appare per primo tra diverse attività di ricerca, così da garantire a entrambe le fonti una posizione iniziale privilegiata nel tempo.

Perché C: Inserire un riepilogo dei risultati chiave all'inizio sfrutta l'effetto di primacy, collocando le informazioni critiche nella posizione elaborata con maggiore affidabilità. Aggiungere intestazioni di sezione esplicite aiuta il modello a navigare e prestare attenzione anche al contenuto centrale dell'input, mitigando direttamente il fenomeno “lost in the middle”.

Domanda 14 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Nei test, l'output combinato dell'agente di ricerca web (85K token, inclusi contenuti delle pagine) e dell'agente di analisi documentale (70K token, incluse catene di ragionamento) raggiunge 155K token, ma l'agente di sintesi ottiene le migliori prestazioni con input inferiori a 50K token. Quale soluzione è più efficace?

Quale soluzione è più efficace?

- A) Modificare gli agenti upstream affinché restituiscano dati strutturati (fatti chiave, citazioni, punteggi di rilevanza) invece di contenuti verbosi e ragionamenti. **[CORRETTA]**
- B) Aggiungere un agente intermedio di sintesi che condensi i risultati prima di passarli alla sintesi finale.
- C) Far elaborare i risultati all'agente di sintesi in batch sequenziali, mantenendo lo stato tra una chiamata e l'altra.
- D) Archiviare i risultati in un database vettoriale e fornire all'agente di sintesi strumenti di ricerca da interrogare durante il lavoro.

Perché A: Modificare gli agenti upstream affinché restituiscano dati strutturati risolve la causa principale, riducendo il volume di token alla fonte e preservando le informazioni essenziali. Evita di passare contenuti di pagina voluminosi e tracce di ragionamento che aumentano i token senza migliorare la fase di sintesi.

Domanda 15 (Scenario: Sistema di Ricerca Multiagente)

Situazione: Nei test, osservi che l'agente di sintesi deve spesso verificare affermazioni specifiche durante l'integrazione dei risultati. Attualmente, quando è necessaria una verifica, l'agente di sintesi restituisce il controllo al coordinatore, che chiama l'agente di ricerca web e poi richiama la sintesi con i risultati ottenuti. Questo aggiunge 2–3 cicli extra per attività e aumenta la latenza del 40%. La tua valutazione mostra che l'85% di queste verifiche consiste in semplici controlli fattuali (date, nomi, statistiche) e il 15% richiede ricerche più approfondite. Quale approccio riduce più efficacemente il sovraccarico preservando l'affidabilità del sistema?

Quale approccio è più efficace?

- A) Dare all'agente di sintesi accesso a tutti gli strumenti di ricerca web, così che possa gestire direttamente qualsiasi necessità di verifica senza cicli attraverso il coordinatore.
- B) Fare in modo che l'agente di sintesi accumuli tutte le esigenze di verifica e le restituisca in batch al coordinatore alla fine, così che il coordinatore possa inviarle tutte insieme all'agente di ricerca web.
- C) Fare in modo che l'agente di ricerca web memorizzi preventivamente contesto extra attorno a ogni fonte durante la ricerca iniziale, anticipando le possibili necessità di verifica della sintesi.
- D) Fornire all'agente di sintesi uno strumento `verify_fact` ad ambito limitato per i controlli semplici, continuando a instradare le verifiche complesse tramite il coordinatore verso l'agente di ricerca web. **[CORRETTA]**

Perché D: Uno strumento di verifica fattuale ad ambito limitato permette all'agente di sintesi di gestire direttamente l'85% dei controlli semplici, eliminando la maggior parte dei cicli aggiuntivi, ma mantiene il percorso di delega tramite coordinatore per il 15% delle verifiche complesse. Questo applica il principio del privilegio minimo riducendo in modo significativo la latenza.

Scenario: Claude Code per la Continuous Integration

Domanda 16 (Scenario: Claude Code per la Continuous Integration)

Situazione: La tua pipeline CI esegue la CLI di Claude Code in modalità `--print`, utilizzando CLAUDE.md per fornire il contesto del progetto per la revisione del codice, e gli sviluppatori trovano generalmente le revisioni sostanziali. Tuttavia, segnalano che integrare i risultati nel workflow è difficile: Claude produce paragrafi narrativi che devono essere copiati manualmente nei commenti della PR. Il team vuole pubblicare automaticamente ogni segnalazione come commento inline separato nel punto rilevante del codice, il che richiede dati strutturati con percorso del file, numero di riga, livello di gravità e correzione suggerita. Quale approccio è più efficace?

Quale approccio è più efficace?

- A) Aggiungere una sezione “Formato di Output per la Revisione” a CLAUDE.md con esempi di segnalazioni strutturate, così che Claude apprenda il formato atteso dal contesto del progetto.
- B) Utilizzare i flag CLI `--output-format json` e `--json-schema` per imporre segnalazioni strutturate, poi analizzare l'output per pubblicare commenti inline tramite l'API di GitHub. **[CORRETTA]**
- C) Includere istruzioni esplicite di formattazione nel prompt di revisione, richiedendo che ogni segnalazione segua un template analizzabile come `[FILE:path] [LINE:n] [SEVERITY:level] ...`.
- D) Mantenere il formato narrativo della revisione, ma aggiungere un passaggio di sintesi che usi Claude per generare un riepilogo JSON strutturato delle segnalazioni.

Perché B: L'uso di `--output-format json` insieme a `--json-schema` impone l'output strutturato a livello di CLI, garantendo JSON ben formato con i campi richiesti (percorso del file, numero di riga, gravità, correzione suggerita), che può essere analizzato in modo affidabile e pubblicato come commenti inline nelle PR tramite l'API di GitHub. Sfrutta funzionalità integrate della CLI progettate specificamente per l'output strutturato.

Domanda 17 (Scenario: Claude Code per la Continuous Integration)

Situazione: Il tuo team usa Claude Code per generare suggerimenti di codice, ma noti un pattern: problemi non ovvi — ottimizzazioni prestazionali che rompono casi limite, pulizie del codice che modificano inaspettatamente il comportamento — vengono individuati solo quando un altro membro del team revisiona la PR. Il ragionamento di Claude durante la generazione mostra che ha considerato questi casi, ma ha concluso che il proprio approccio fosse corretto. Quale approccio affronta direttamente la causa principale di questo limite dell'autoverifica?

Quale approccio affronta direttamente la causa principale?

- A) Eseguire una seconda istanza indipendente di Claude Code per revisionare le modifiche senza accesso al ragionamento del generatore. **[CORRETTA]**
- B) Abilitare la modalità di ragionamento esteso nella fase di generazione per consentire una deliberazione più approfondita prima di produrre suggerimenti.
- C) Aggiungere istruzioni esplicite di autorevisione al prompt di generazione, chiedendo a Claude di criticare i propri suggerimenti prima di finalizzare l'output.

- D) Includere nel contesto del prompt i file di test completi e la documentazione, così che Claude comprenda meglio il comportamento atteso durante la generazione.

Perché A: Una seconda istanza indipendente di Claude Code, senza accesso al ragionamento del generatore, affronta direttamente la causa principale evitando il bias di conferma. Questa prospettiva "a occhi nuovi" rispecchia la revisione tra pari umana, in cui un altro revisore individua problemi che l'autore ha razionalizzato.

Domanda 18 (Scenario: Claude Code per la Continuous Integration)

Situazione: Il tuo componente di revisione del codice è iterativo: Claude analizza il file modificato, poi può richiedere file correlati (importazioni, classi base, test) tramite chiamate a strumenti per comprendere il contesto prima di fornire il feedback finale. La tua applicazione definisce uno strumento che consente a Claude di richiedere il contenuto dei file; Claude chiama lo strumento, riceve i risultati e continua l'analisi. Stai valutando l'elaborazione batch per ridurre i costi API. Qual è la principale limitazione tecnica da considerare per questo workflow?

Qual è la principale limitazione tecnica?

- A) L'elaborazione batch non include ID di correlazione per associare gli output alle richieste di input.
- B) Il modello asincrono non può eseguire strumenti durante la richiesta e restituire i risultati a Claude per continuare l'analisi. **[CORRETTA]**
- C) L'API Batch non supporta definizioni di strumenti nei parametri della richiesta.
- D) La latenza dell'elaborazione batch, fino a 24 ore, è troppo lenta per il feedback sulle pull request, anche se il workflow funzionerebbe altrimenti.

Perché B: Un modello asincrono "fire-and-forget" come quello della Batch API non dispone di un meccanismo per intercettare una chiamata a uno strumento durante una richiesta, eseguire lo strumento e restituire i risultati a Claude affinché possa continuare l'analisi. Questo è fondamentalmente incompatibile con workflow iterativi basati su tool calling che richiedono più cicli richiesta/risposta con strumenti all'interno di una singola interazione logica.

Domanda 19 (Scenario: Claude Code per la Continuous Integration)

Situazione: Il tuo sistema CI/CD esegue tre analisi basate su Claude: (1) controlli rapidi di stile su ogni PR, che bloccano il merge fino al completamento, (2) audit di sicurezza settimanali completi dell'intero codebase e (3) generazione notturna di casi di test per i moduli modificati di recente. L'API Message Batches offre un risparmio del 50%, ma l'elaborazione può richiedere fino a 24 ore. Vuoi ottimizzare i costi API mantenendo un'esperienza accettabile per gli sviluppatori. Quale combinazione associa correttamente ogni attività all'approccio API?

Quale combinazione è corretta?

- A) Utilizzare l'API Message Batches per tutte e tre le attività per massimizzare il risparmio del 50%, configurando la pipeline per fare polling fino al completamento del batch.
- B) Utilizzare chiamate sincrone per i controlli di stile sulle PR; utilizzare l'API Message Batches per gli audit di sicurezza settimanali e la generazione notturna di test. **[CORRETTA]**
- C) Utilizzare chiamate sincrone per tutte e tre le attività per avere tempi di risposta coerenti, affidandosi al prompt caching per ridurre i costi nei diversi workload.
- D) Utilizzare chiamate sincrone per i controlli di stile sulle PR e la generazione notturna di test; utilizzare l'API Message Batches solo per gli audit di sicurezza settimanali.

Perché B: I controlli di stile sulle PR bloccano gli sviluppatori e richiedono risposte immediate tramite chiamate sincrone, mentre gli audit di sicurezza settimanali e la generazione notturna di test sono attività pianificate con scadenze flessibili, quindi possono tollerare una finestra batch fino a 24 ore e sfruttare il risparmio del 50%.

Domanda 20 (Scenario: Claude Code per la Continuous Integration)

Situazione: Le tue revisioni automatizzate individuano problemi reali, ma gli sviluppatori segnalano che il feedback non è utilizzabile concretamente. Le segnalazioni includono frasi come “logica complessa di instradamento dei ticket” o “potenziale puntatore nullo” senza specificare esattamente cosa modificare. Quando aggiungi istruzioni dettagliate come “includi sempre suggerimenti di correzione concreti”, il modello produce comunque output incoerenti: a volte dettagliati, a volte vaghi. Quale tecnica di prompting produce nel modo più affidabile feedback costantemente utilizzabile?

Quale tecnica di prompting è più affidabile?

- A) Raffinare ulteriormente le istruzioni con requisiti più espliciti per ogni parte del formato del feedback (posizione, problema, gravità, correzione proposta).
- B) Espandere la finestra di contesto per includere più codice circostante, così che il modello abbia abbastanza informazioni per proporre correzioni concrete.
- C) Implementare un approccio in due passaggi, in cui un prompt identifica i problemi e un secondo genera le correzioni, consentendo specializzazione.
- D) Aggiungere 3–4 esempi few-shot che mostrino l'esatto formato richiesto: problema individuato, posizione nel codice, suggerimento di correzione concreto. **[CORRETTA]**

Perché D: Gli esempi few-shot sono la tecnica più efficace per ottenere un formato di output coerente quando le sole istruzioni producono risultati variabili. Fornire 3–4 esempi che mostrano la struttura esatta desiderata (problema, posizione, correzione concreta) offre al modello un pattern concreto da seguire, più affidabile delle istruzioni astratte.

Domanda 21 (Scenario: Claude Code per la Continuous Integration)

Situazione: La tua pipeline CI include due modalità di revisione del codice basate su Claude: un hook pre-merge-commit che blocca il merge della PR fino al completamento, e una “deep analysis” che viene eseguita durante la notte, fa polling fino al completamento del batch e pubblica suggerimenti dettagliati sulla PR. Vuoi ridurre i costi API usando l'API Message Batches, che offre un risparmio del 50% ma richiede polling e può richiedere fino a 24 ore. Quale modalità dovrebbe usare l'elaborazione batch?

Quale modalità dovrebbe usare l'elaborazione batch?

- A) Solo l'hook pre-merge-commit.
- B) Solo la deep analysis. **[CORRETTA]**
- C) Entrambe le modalità.
- D) Nessuna delle due modalità.

Perché B: La deep analysis è una candidata ideale per l'elaborazione batch perché viene già eseguita durante la notte, tollera ritardi e usa un modello basato su polling prima di pubblicare i risultati: caratteristiche che corrispondono all'architettura asincrona e basata su polling dell'API Message Batches, consentendo al tempo stesso di ottenere il risparmio del 50%.

Domanda 22 (Scenario: Claude Code per la Continuous Integration)

Situazione: La tua revisione automatizzata analizza commenti e docstring. Il prompt attuale istruisce Claude a “verificare che i commenti siano accurati e aggiornati”. Le segnalazioni spesso evidenziano pattern accettabili (marcatori TODO, descrizioni semplici) mentre non rilevano commenti che descrivono comportamenti che il codice non implementa più. Quale modifica affronta la causa principale di questa analisi incoerente?

Quale modifica affronta la causa principale?

- A) Includere dati `git blame` affinché Claude possa identificare i commenti che precedono modifiche recenti del codice.
- B) Aggiungere esempi few-shot di commenti fuorvianti per aiutare il modello a riconoscere pattern simili nel codebase.
- C) Filtrare i pattern di commenti TODO, FIXME e descrittivi prima dell'analisi per ridurre il rumore.
- D) Specificare criteri espliciti: segnalare i commenti solo quando il comportamento che descrivono contraddice il comportamento effettivo del codice. **[CORRETTA]**

Perché D: I criteri espliciti — segnalare i commenti solo quando il comportamento dichiarato contraddice quello effettivamente implementato dal codice — affrontano direttamente la causa principale sostituendo un'istruzione vaga con una definizione precisa di ciò che costituisce un problema. Questo riduce i falsi positivi sui pattern accettabili e diminuisce il rischio di ignorare commenti realmente fuorvianti.

Domanda 23 (Scenario: Claude Code per la Continuous Integration)

Situazione: Il tuo sistema di revisione automatizzata del codice mostra valutazioni di gravità incoerenti: problemi simili, come il rischio di puntatori nulli, vengono classificati come “critici” in alcune PR e solo come “medi” in altre. I sondaggi tra gli sviluppatori mostrano una crescente perdita di fiducia: molti iniziano a ignorare le segnalazioni senza leggerle perché “la metà è sbagliata”. Le categorie con molti falsi positivi erodono la fiducia anche nelle categorie accurate. Quale approccio ripristina al meglio la fiducia degli sviluppatori migliorando al contempo il sistema?

Quale approccio ripristina al meglio la fiducia degli sviluppatori?

- A) Disattivare temporaneamente le categorie con molti falsi positivi (stile, nomenclatura, documentazione) e mantenere solo le categorie ad alta precisione mentre si migliorano i prompt. **[CORRETTA]**
- B) Mantenere tutte le categorie attive ma visualizzare un punteggio di confidenza per ogni segnalazione, lasciando agli sviluppatori la decisione su cosa investigare.
- C) Mantenere tutte le categorie attive e aggiungere esempi few-shot per migliorare l'accuratezza di ciascuna categoria nelle settimane successive.
- D) Applicare una riduzione uniforme della severità in tutte le categorie per abbassare il tasso complessivo di falsi positivi.

Perché A: Disattivare temporaneamente le categorie con molti falsi positivi interrompe immediatamente l'erosione della fiducia eliminando le segnalazioni rumorose che portano gli sviluppatori a ignorare tutto il feedback. Nel frattempo, si mantiene il valore delle categorie ad alta precisione come sicurezza e correttezza, creando lo spazio necessario per migliorare i prompt delle categorie problematiche prima di riattivarle.

Domanda 24 (Scenario: Claude Code per la Continuous Integration)

Situazione: La tua revisione automatizzata genera suggerimenti di casi di test per ogni PR. Durante la revisione di una PR che aggiunge il tracciamento del completamento dei corsi, Claude suggerisce 10 casi di test, ma il feedback degli sviluppatori mostra che 6 di essi duplicano scenari già coperti dalla suite di test esistente. Quale modifica riduce più efficacemente i suggerimenti duplicati?

Quale modifica è più efficace?

- A) Includere il file di test esistente nel contesto affinché Claude possa determinare quali scenari sono già coperti. **[CORRETTA]**
- B) Ridurre il numero di suggerimenti richiesti da 10 a 5, assumendo che Claude dia priorità ai casi più preziosi.
- C) Aggiungere istruzioni che indirizzino Claude a concentrarsi esclusivamente sui casi limite e sulle condizioni di errore anziché sui percorsi di successo.
- D) Implementare un post-processing che filtri i suggerimenti le cui descrizioni corrispondono ai nomi dei test esistenti tramite sovrapposizione di parole chiave.

Perché A: Includere il file di test esistente risolve la causa principale della duplicazione: Claude può evitare di suggerire scenari già coperti solo se conosce quali test esistono già. Questo gli fornisce le informazioni necessarie per proporre test realmente nuovi e utili.

Domanda 25 (Scenario: Claude Code per la Continuous Integration)

Situazione: Dopo che una revisione automatizzata iniziale ha individuato 12 problemi, uno sviluppatore pubblica nuovi commit per correggerli. Rieseguendo la revisione vengono segnalati 8 problemi, ma gli sviluppatori riferiscono che 5 di essi duplicano commenti precedenti relativi a codice già corretto nei nuovi commit. Qual è il modo più efficace per eliminare questo feedback ridondante mantenendo al contempo la completezza della revisione?

Qual è il modo più efficace per eliminare il feedback ridondante?

- A) Eseguire la revisione solo quando la PR viene creata e nello stato finale pre-merge, saltando i commit intermedi.
- B) Aggiungere un filtro di post-processing che rimuova le segnalazioni corrispondenti a quelle precedenti in base ai percorsi dei file e alle descrizioni dei problemi prima di pubblicare i commenti.
- C) Limitare l'ambito della revisione ai file modificati nell'ultimo push, escludendo i file dei commit precedenti.
- D) Includere nel contesto le segnalazioni delle revisioni precedenti e istruire Claude a riportare solo problemi nuovi o ancora irrisolti. **[CORRETTA]**

Perché D: Includere nel contesto i risultati delle revisioni precedenti permette a Claude di distinguere i nuovi problemi da quelli già corretti nei commit recenti. Questo mantiene la completezza della revisione sfruttando il ragionamento del modello per evitare feedback ridondanti su codice già sistemato.

Domanda 26 (Scenario: Claude Code per la Continuous Integration)

Situazione: Lo script della tua pipeline esegue `claude "Analyze this pull request for security issues"`, ma il job rimane bloccato indefinitamente. I log mostrano che Claude Code è in attesa di input interattivo. Qual è l'approccio corretto per eseguire Claude Code in una pipeline automatizzata?

Qual è l'approccio corretto?

- A) Aggiungere il flag `--batch`: `claude --batch "Analyze this pull request for security issues"`.
- B) Aggiungere il flag `-p`: `claude -p "Analyze this pull request for security issues"`. **[CORRETTA]**
- C) Reindirizzare lo standard input da `/dev/null`: `claude "Analyze this pull request for security issues" < /dev/null`.
- D) Impostare la variabile d'ambiente `CLAUDE_HEADLESS=true` prima di eseguire il comando.

Perché B: Il flag `-p` (oppure `--print`) è il metodo documentato per eseguire Claude Code in modalità non interattiva. Elabora il prompt, stampa il risultato sullo standard output e termina senza attendere input dall'utente, rendendolo ideale per pipeline CI/CD.

Domanda 27 (Scenario: Claude Code per la Continuous Integration)

Situazione: Una pull request modifica 14 file in un modulo di tracciamento dell'inventario. Una revisione a passaggio singolo che analizza tutti i file insieme produce risultati incoerenti: feedback dettagliati su alcuni file ma commenti superficiali su altri, bug evidenti non rilevati e feedback contraddittori (un pattern viene segnalato in un file ma codice identico viene approvato in un altro file della stessa PR). Come dovresti ristrutturare la revisione?

Come dovresti ristrutturare la revisione?

- A) Eseguire tre revisioni indipendenti dell'intera PR e segnalare solo i problemi che compaiono in almeno due delle tre esecuzioni.
- B) Suddividere la revisione in passaggi mirati: revisionare ogni file singolarmente per i problemi locali, quindi eseguire un passaggio separato orientato all'integrazione per esaminare i flussi di dati tra file. **[CORRETTA]**
- C) Richiedere agli sviluppatori di suddividere le PR più grandi in invii più piccoli da 3–4 file prima di eseguire la revisione automatizzata.
- D) Passare a un modello più grande con una finestra di contesto più ampia affinché possa prestare sufficiente attenzione a tutti i 14 file in un unico passaggio.

Perché B: I passaggi focalizzati per singolo file affrontano la causa principale — la dispersione dell'attenzione — garantendo una profondità di analisi coerente e un rilevamento affidabile dei problemi locali. Un passaggio separato orientato all'integrazione copre poi le problematiche tra file, come dipendenze e interazioni nei flussi di dati.

Domanda 28 (Scenario: Claude Code per la Continuous Integration)

Situazione: La tua revisione automatizzata del codice produce in media 15 segnalazioni per pull request e gli sviluppatori riportano un tasso di falsi positivi del 40%. Il collo di bottiglia è il tempo di investigazione: gli sviluppatori devono aprire ogni segnalazione per leggere il ragionamento di Claude prima di decidere se correggerla o ignorarla. Il tuo CLAUDE.md contiene già regole complete sui pattern accettabili e gli stakeholder hanno rifiutato qualsiasi approccio che filtri le segnalazioni prima che gli sviluppatori le vedano. Quale modifica affronta meglio il problema del tempo di investigazione?

Quale modifica affronta meglio il problema del tempo di investigazione?

- A) Richiedere a Claude di includere direttamente in ogni segnalazione il proprio ragionamento e una stima di confidenza. **[CORRETTA]**

- B) Aggiungere un post-processore che analizzi i pattern delle segnalazioni e sopprima automaticamente quelle che corrispondono a firme storiche di falsi positivi.
- C) Categorizzare le segnalazioni come “problemi bloccanti” oppure “suggerimenti”, applicando requisiti di revisione differenti in base al livello.
- D) Configurare Claude affinché mostri solo le segnalazioni ad alta confidenza, filtrando quelle incerte prima che gli sviluppatori le vedano.

Perché A: Includere direttamente in ogni segnalazione il ragionamento e il livello di confidenza riduce il tempo di investigazione perché consente agli sviluppatori di effettuare rapidamente il triage senza dover aprire ogni singolo elemento. Inoltre rispetta il vincolo di “nessun filtraggio”, poiché tutte le segnalazioni restano visibili, accelerando comunque il processo decisionale degli sviluppatori.

Domanda 29 (Scenario: Claude Code per la Continuous Integration)

Situazione: L'analisi del tuo sistema di revisione automatizzata del codice mostra grandi differenze nei tassi di falsi positivi a seconda della categoria di segnalazione: le segnalazioni di sicurezza/correttezza hanno l'8% di falsi positivi, quelle sulle prestazioni il 18%, quelle su stile/nomenclatura il 52% e quelle sulla documentazione il 48%. I sondaggi tra gli sviluppatori mostrano una crescente perdita di fiducia: molti iniziano a ignorare le segnalazioni senza leggerle perché “la metà è sbagliata”. Le categorie con molti falsi positivi erodono la fiducia anche nelle categorie accurate. Quale approccio ripristina al meglio la fiducia degli sviluppatori migliorando al contempo il sistema?

Quale approccio ripristina al meglio la fiducia degli sviluppatori?

- A) Disattivare temporaneamente le categorie con molti falsi positivi (stile, nomenclatura, documentazione) e mantenere solo le categorie ad alta precisione mentre si migliorano i prompt. **[CORRETTA]**
- B) Mantenere tutte le categorie attive ma visualizzare un punteggio di confidenza per ogni segnalazione, lasciando agli sviluppatori la decisione su cosa investigare.
- C) Mantenere tutte le categorie attive e aggiungere esempi few-shot per migliorare l'accuratezza di ciascuna categoria nelle settimane successive.
- D) Applicare una riduzione uniforme della severità in tutte le categorie per abbassare il tasso complessivo di falsi positivi.

Perché A: Disattivare temporaneamente le categorie con molti falsi positivi interrompe immediatamente l'erosione della fiducia eliminando le segnalazioni rumorose che portano gli sviluppatori a ignorare tutto il feedback. Nel frattempo, si mantiene il valore delle categorie ad alta precisione come sicurezza e correttezza, creando lo spazio necessario per migliorare i prompt delle categorie problematiche prima di riattivarle.

Domanda 30 (Scenario: Claude Code per la Continuous Integration)

Situazione: Il tuo team vuole ridurre i costi API per l'analisi automatizzata. Attualmente, le chiamate sincrone a Claude supportano due workflow: (1) un controllo pre-merge bloccante che deve essere completato prima che gli sviluppatori possano eseguire il merge, e (2) un report sul debito tecnico generato durante la notte e revisionato la mattina successiva. Il tuo manager propone di spostare entrambi sull'API Message Batches per risparmiare il 50%. Come dovresti valutare questa proposta?

Come dovresti valutare questa proposta?

- A) Spostare entrambi i workflow sull'elaborazione batch con fallback a chiamate sincrone se i batch impiegano troppo tempo.
- B) Spostare entrambi i workflow sull'elaborazione batch utilizzando il polling dello stato per verificare il completamento.
- C) Utilizzare l'elaborazione batch solo per i report sul debito tecnico; mantenere le chiamate sincrone per i controlli pre-merge. **[CORRETTA]**
- D) Mantenere le chiamate sincrone per entrambi i workflow per evitare problemi legati all'ordinamento dei risultati batch.

Perché C: L'elaborazione tramite API Message Batches può richiedere fino a 24 ore e non offre alcuna garanzia SLA sulla latenza. Questo è accettabile per report sul debito tecnico eseguiti durante la notte, ma inaccettabile per controlli pre-merge bloccanti nei quali gli sviluppatori attendono il risultato prima di procedere. Questa soluzione assegna correttamente ogni workflow al tipo di API più adatto in base ai requisiti di latenza.

Scenario: Generazione di Codice con Claude Code

Domanda 31 (Scenario: Generazione di Codice con Claude Code)

Situazione: Hai chiesto a Claude Code di implementare una funzione che trasforma le risposte API in un formato interno normalizzato. Dopo due iterazioni, la struttura dell'output continua a non corrispondere alle aspettative: alcuni campi sono annidati in modo diverso e i timestamp sono formattati in modo errato. Hai descritto i requisiti in forma discorsiva, ma Claude li interpreta ogni volta in modo diverso.

Quale approccio è più efficace per la prossima iterazione?

- A) Scrivere uno schema JSON che descriva la struttura di output attesa e validare l'output di Claude rispetto a esso dopo ogni iterazione.
- B) Fornire 2–3 esempi concreti di input-output che mostrino la trasformazione attesa per risposte API rappresentative. **[CORRETTA]**
- C) Riscrivere i requisiti con maggiore precisione tecnica, specificando mapping esatti dei campi, regole di annidamento e stringhe di formato per i timestamp.

- D) Chiedere a Claude di spiegare la sua attuale comprensione dei requisiti per identificare dove divergono le interpretazioni.

Perché B: Gli esempi concreti di input-output eliminano l'ambiguità delle descrizioni discorsive mostrando a Claude il risultato esatto della trasformazione attesa. Questo affronta direttamente la causa principale — l'interpretazione errata dei requisiti testuali — fornendo pattern non ambigui per l'annidamento dei campi e la formattazione dei timestamp.

Domanda 32 (Scenario: Generazione di Codice con Claude Code)

Situazione: Devi aggiungere Slack come nuovo canale di notifica. Il codebase esistente presenta pattern chiari e consolidati per i canali email, SMS e push. Tuttavia, l'API di Slack offre approcci di integrazione fondamentalmente diversi: incoming webhook (semplici, unidirezionali), bot token (supportano conferma di consegna e controllo programmatico) oppure Slack App (eventi bidirezionali, richiede approvazione del workspace). Il tuo compito dice “aggiungi supporto a Slack” senza specificare il metodo di integrazione né richiedere funzionalità avanzate come il tracciamento della consegna.

Come dovresti affrontare questo compito?

- A) Iniziare in modalità di esecuzione diretta usando incoming webhook per seguire il pattern di notifica unidirezionale esistente.
- B) Passare alla modalità di pianificazione per esplorare le opzioni di integrazione e le implicazioni architetturali, poi presentare una raccomandazione prima dell'implementazione. **[CORRETTA]**
- C) Iniziare in modalità di esecuzione diretta creando lo scheletro di una classe per il canale Slack usando i pattern esistenti, rimandando la decisione sul metodo di integrazione.
- D) Iniziare in modalità di esecuzione diretta usando un approccio con bot token per assicurare la possibilità di conferma della consegna.

Perché B: L'integrazione con Slack presenta più approcci validi con implicazioni architetturali molto diverse, e i requisiti sono ambigui. La modalità di pianificazione consente di valutare i compromessi tra webhook, bot token e Slack App, e di allinearsi su un approccio prima dell'implementazione.

Domanda 33 (Scenario: Generazione di Codice con Claude Code)

Situazione: Il tuo file CLAUDE.md è cresciuto fino a superare le 400 righe e contiene standard di codifica, convenzioni di testing, una checklist dettagliata per la revisione delle PR, istruzioni di deployment e procedure di migrazione del database. Vuoi che Claude segua sempre gli standard di codifica e le convenzioni di testing, ma applichi le indicazioni su revisione PR, deploy e migrazioni solo quando svolge quei compiti.

Quale approccio di ristrutturazione è più efficace?

- A) Spostare tutte le indicazioni in file Skills separati organizzati per tipo di workflow, lasciando in CLAUDE.md solo una breve descrizione del progetto.

- B) Mantenere tutto in CLAUDE.md ma usare la sintassi `@import` per organizzare il contenuto in file separati per categoria.
- C) Suddividere CLAUDE.md in file dentro `.claude/rules/` con pattern glob vincolati ai percorsi, così che ogni regola venga caricata solo per i tipi di file rilevanti.
- D) Mantenere gli standard universali in CLAUDE.md e creare Skill per le indicazioni specifiche di workflow (revisione PR, deploy, migrazioni) con parole chiave di attivazione. **[CORRETTA]**

Perché D: Il contenuto di CLAUDE.md viene caricato in ogni sessione, garantendo che gli standard di codifica e le convenzioni di testing siano sempre applicati, mentre le Skill vengono invocate su richiesta quando Claude rileva parole chiave di attivazione: una soluzione ideale per indicazioni specifiche di workflow come revisione PR, deployment e migrazioni.

Domanda 34 (Scenario: Generazione di Codice con Claude Code)

Situazione: Ti viene assegnato il compito di ristrutturare l'applicazione monolitica del tuo team in microservizi. Questo comporta modifiche su decine di file e richiede decisioni sui confini dei servizi e sulle dipendenze tra moduli.

Quale approccio dovresti scegliere?

- A) Passare alla modalità di pianificazione per esplorare il codebase, comprendere le dipendenze e progettare l'approccio di implementazione prima di apportare modifiche. **[CORRETTA]**
- B) Iniziare in modalità di esecuzione diretta e passare alla pianificazione solo dopo aver incontrato complessità impreviste durante l'implementazione.
- C) Iniziare in modalità di esecuzione diretta e apportare modifiche incrementali, lasciando che l'implementazione riveli naturalmente i confini dei servizi.
- D) Usare l'esecuzione diretta con istruzioni iniziali dettagliate che specifichino la struttura di ciascun servizio.

Perché A: La modalità di pianificazione è la strategia corretta per ristrutturazioni architetturali complesse come la suddivisione di un monolite: consente un'esplorazione sicura e decisioni informate sui confini prima di impegnarsi in modifiche potenzialmente costose su molti file.

Domanda 35 (Scenario: Generazione di Codice con Claude Code)

Situazione: Il tuo team ha creato una skill `/analyze-codebase` che esegue un'analisi approfondita del codice: scansione delle dipendenze, conteggio della copertura dei test e metriche di qualità del codice. Dopo aver eseguito il comando, i membri del team segnalano che Claude diventa meno reattivo nella sessione e perde il contesto del compito originale.

Come risolvi il problema nel modo più efficace mantenendo tutte le capacità di analisi?

- A) Aggiungere `context: fork` nel frontmatter della skill per eseguire l'analisi in un contesto isolato di sottoagente. **[CORRETTA]**
- B) Aggiungere `model: haiku` nel frontmatter per usare un modello più veloce ed economico per l'analisi.
- C) Suddividere la skill in tre skill più piccole, ciascuna con un output più ridotto.
- D) Aggiungere istruzioni alla skill per comprimere tutti i risultati in un breve riepilogo prima di mostrarli.

Perché A: `context: fork` esegue l'analisi in un contesto isolato di sottoagente, così l'output esteso non inquina la finestra di contesto della sessione principale e Claude non perde traccia del compito originale. Questo preserva tutte le capacità di analisi mantenendo la sessione principale reattiva.

Domanda 36 (Scenario: Generazione di Codice con Claude Code)

Situazione: Il tuo team usa una skill `/commit` in `.claude/skills/commit/SKILL.md`. Uno sviluppatore vuole personalizzarla per il proprio workflow personale (formato diverso dei messaggi di commit, controlli extra) senza influenzare i compagni di team.

Cosa consigli?

- A) Creare una versione personale in `~/.claude/skills/` con un nome diverso, ad esempio `/my-commit`. **[CORRETTA]**
- B) Aggiungere logica condizionale basata sul nome utente nel frontmatter della skill di progetto.
- C) Creare una versione personale in `~/.claude/skills/commit/SKILL.md` con lo stesso nome.
- D) Impostare `override: true` nel frontmatter della skill personale per darle priorità rispetto alla versione di progetto.

Perché A: Le skill personali hanno la precedenza sulle skill di progetto con lo stesso nome, quindi riutilizzare il nome `commit` nasconderebbe silenziosamente la skill del team solo per questo sviluppatore, che smetterebbe di ricevere gli aggiornamenti ogni volta che il team migliora `/commit` e dovrebbe ricordarsi di star eseguendo una skill diversa sotto lo stesso comando. Chiamare la variante personale `/my-commit` evita del tutto questo conflitto: lo sviluppatore continua a usare il `/commit` mantenuto dal team e ottiene una skill separata, chiaramente identificata, per il proprio workflow personale, senza rischio di confondere le due o perdere gli aggiornamenti del team.

Domanda 37 (Scenario: Generazione di Codice con Claude Code)

Situazione: Il tuo team usa Claude Code da mesi. Di recente, tre sviluppatori riferiscono che Claude segue l'indicazione “includi sempre una gestione completa degli errori”, ma un quarto sviluppatore appena entrato dice che Claude non la segue. Tutti e quattro lavorano nello stesso repository e hanno il codice aggiornato.

Qual è la causa più probabile e la relativa soluzione?

- A) L'indicazione si trova nei file `~/.claude/CLAUDE.md` a livello utente degli sviluppatori originali, non nel file `.claude/CLAUDE.md` a livello progetto. Spostare l'istruzione nel file a livello progetto affinché tutti i membri del team la ricevano. **[CORRETTA]**
- B) Il file `~/.claude/CLAUDE.md` del nuovo sviluppatore contiene istruzioni in conflitto che sovrascrivono le impostazioni del progetto; dovrebbe eliminare la sezione in conflitto.
- C) Claude Code apprende nel tempo le preferenze specifiche dell'utente; il nuovo sviluppatore deve ripetere il requisito finché Claude non lo “ricorda”.
- D) Claude Code memorizza CLAUDE.md nella cache dopo la prima lettura; gli sviluppatori originali usano versioni in cache. Tutti dovrebbero svuotare la cache di Claude Code.

Perché A: Se l'indicazione è stata aggiunta solo alle configurazioni a livello utente degli sviluppatori originali e non al file `.claude/CLAUDE.md` a livello progetto, i nuovi membri del team non la riceveranno. Spostarla nella configurazione a livello progetto garantisce che tutti i membri attuali e futuri ricevano automaticamente l'indicazione.

Domanda 38 (Scenario: Generazione di Codice con Claude Code)

Situazione: Scopri che includere 2–3 esempi completi di implementazione di endpoint come contesto migliora significativamente la coerenza quando si generano nuovi endpoint API. Tuttavia, questo contesto è utile solo quando si creano nuovi endpoint, non durante debugging, revisione del codice o altri lavori nella directory API.

Quale approccio di configurazione è più efficace?

- A) Aggiungere esempi di endpoint e documentazione sui pattern al CLAUDE.md del progetto, così che siano sempre disponibili.
- B) Fare riferimento manualmente agli esempi di endpoint in ogni richiesta di generazione copiando il codice nel prompt.
- C) Configurare regole specifiche per percorso in `.claude/rules/api/` che includano esempi di endpoint e si attivino quando si lavora nella directory API.
- D) Creare una skill che faccia riferimento agli esempi di endpoint e contenga istruzioni per seguire i pattern, invocata su richiesta tramite comando slash. **[CORRETTA]**

Perché D: Una skill invocata su richiesta carica il contesto degli esempi solo quando si generano nuovi endpoint, non durante attività non correlate come debugging o revisione. Questo mantiene pulito il contesto principale, preservando al tempo stesso una generazione di alta qualità quando necessario.

Domanda 39 (Scenario: Generazione di Codice con Claude Code)

Situazione: Il tuo team ha creato una skill `/migration` che genera file di migrazione del database. Riceve il nome della migrazione tramite `$ARGUMENTS`. In produzione osservi tre problemi: (1) gli sviluppatori spesso eseguono la skill senza argomenti, causando file con nomi poco chiari, (2) la skill a

volte usa dettagli dello schema del database provenienti da conversazioni precedenti non correlate, e (3) uno sviluppatore ha eseguito accidentalmente una pulizia distruttiva dei test quando la skill aveva accesso ampio agli strumenti.

Quale approccio di configurazione risolve tutti e tre i problemi?

- A) Usare parametri posizionali `$1` e `$2` invece di `$ARGUMENTS` per imporre input specifici, includere riferimenti espliciti ai file dello schema tramite sintassi `@` per controllare il contesto e aggiungere una descrizione nel frontmatter che avvisi sulle operazioni distruttive.
- B) Aggiungere `argument-hint` nel frontmatter per richiedere i parametri necessari, usare `context: fork` per isolare l'esecuzione e limitare `allowed-tools` alle operazioni di scrittura su file. **[CORRETTA]**
- C) Suddividere la skill in `/migration-create` e `/migration-apply`, aggiungere istruzioni di validazione per richiedere il nome della migrazione se manca e usare ambiti `allowed-tools` diversi per ciascuna.
- D) Aggiungere istruzioni di validazione nel file SKILL.md per assicurarsi che `$ARGUMENTS` sia un nome valido, aggiungere prompt per ignorare il contesto delle conversazioni precedenti ed elencare le operazioni proibite da evitare.

Perché B: Questo approccio usa tre funzionalità di configurazione distinte per affrontare ciascun problema: `argument-hint` migliora l'inserimento degli argomenti e riduce i casi in cui mancano, `context: fork` evita perdite di contesto da conversazioni precedenti e `allowed-tools` limita la skill a operazioni sicure di scrittura su file, prevenendo azioni distruttive.

Domanda 40 (Scenario: Generazione di Codice con Claude Code)

Situazione: Il tuo codebase contiene aree con convenzioni di codifica differenti: i componenti React usano uno stile funzionale con hook, gli handler API usano async/await con una specifica gestione degli errori e i modelli di database seguono il pattern repository. I file di test sono distribuiti in tutto il codebase accanto al codice testato (ad esempio `Button.test.tsx` accanto a `Button.tsx`) e vuoi che tutti i test seguano le stesse convenzioni indipendentemente dalla loro posizione.

Qual è il metodo più supportato per assicurare che Claude applichi automaticamente le convenzioni corrette quando genera codice?

- A) Inserire tutte le convenzioni nel CLAUDE.md della root sotto intestazioni dedicate a ciascuna area e affidarsi a Claude perché inferisca quale sezione applicare.
- B) Creare skill in `.claude/skills/` per ciascun tipo di codice, incorporando le convenzioni in ogni SKILL.md.
- C) Inserire un file CLAUDE.md separato in ogni sottodirectory contenente le convenzioni per quell'area.
- D) Creare file di regole sotto `.claude/rules/` con frontmatter YAML che specifichi pattern glob per applicare condizionalmente le convenzioni in base ai percorsi dei file. **[CORRETTA]**

Perché D: I file in `.claude/rules/` con frontmatter YAML e pattern glob (ad esempio `**/*.test.tsx`, `src/api/**/*.ts`) permettono di applicare convenzioni in modo deterministico in base al percorso dei

file, indipendentemente dalla struttura delle directory. Questo è l'approccio più supportato per pattern trasversali come i file di test distribuiti nel codebase.

Domanda 41 (Scenario: Generazione di Codice con Claude Code)

Situazione: Vuoi creare un comando slash personalizzato `/review` che esegua la checklist standard di revisione del codice del tuo team. Deve essere disponibile per ogni sviluppatore quando clona o aggiorna il repository.

Dove dovresti creare il file del comando?

- A) In `~/.claude/commands/` nella home directory di ciascuno sviluppatore.
- B) Nel repository del progetto, dentro `.claude/commands/`. **[CORRETTA]**
- C) In `.claude/config.json` come array di comandi.
- D) Nel file CLAUDE.md della root del progetto.

Perché B: Inserire i comandi slash personalizzati dentro `.claude/commands/` nel repository del progetto garantisce che siano inclusi nel controllo versione e automaticamente disponibili per ogni sviluppatore che clona o aggiorna il repository. Questa è la posizione prevista per i comandi personalizzati a livello di progetto in Claude Code.

Domanda 42 (Scenario: Generazione di Codice con Claude Code)

Situazione: Il CLAUDE.md del tuo team è cresciuto oltre 500 righe, mescolando convenzioni TypeScript, linee guida per i test, pattern API e procedure di deployment. Gli sviluppatori trovano difficile individuare e aggiornare le sezioni corrette.

Quale approccio supporta Claude Code per organizzare le istruzioni a livello progetto in moduli tematici mirati?

- A) Definire un file `.claude/config.yaml` che associ pattern di file a sezioni specifiche dentro CLAUDE.md.
- B) Creare file Markdown separati in `.claude/rules/`, ciascuno dedicato a un argomento (ad esempio `testing.md`, `api-conventions.md`). **[CORRETTA]**
- C) Suddividere le istruzioni in file README.md nelle sottodirectory rilevanti, che Claude carica automaticamente come istruzioni.
- D) Creare più file chiamati CLAUDE.md a diversi livelli dell'albero delle directory, ciascuno capace di sovrascrivere le istruzioni del livello superiore.

Perché B: Claude Code supporta una directory `.claude/rules/` in cui è possibile creare file Markdown separati per indicazioni tematiche (ad esempio `testing.md`, `api-conventions.md`), permettendo ai team di organizzare insiemi estesi di istruzioni in moduli mirati e più facili da mantenere.

Domanda 43 (Scenario: Generazione di Codice con Claude Code)

Situazione: Crei una skill personalizzata `/explore-alternatives` che il tuo team usa per fare brainstorming e valutare diversi approcci implementativi prima di sceglierne uno. Gli sviluppatori riferiscono che, dopo aver eseguito la skill, le risposte successive di Claude sono influenzate dalla discussione sulle alternative: a volte cita approcci scartati o mantiene contesto esplorativo che interferisce con l'implementazione effettiva.

Come dovresti configurare questa skill nel modo più efficace?

- A) Usare il prefisso `!` nella skill per eseguire la logica di esplorazione come sottoprocesso bash.
- B) Aggiungere `context: fork` nel frontmatter della skill. **[CORRETTA]**
- C) Suddividere la skill in due skill — `/explore-start` e `/explore-end` — per segnare i confini in cui il contesto esplorativo dovrebbe essere eliminato.
- D) Creare la skill in `~/.claude/skills/` invece che in `.claude/skills/`.

Perché B: `context: fork` esegue la skill in un contesto isolato di sottoagente, così le discussioni esplorative non inquinano la cronologia della conversazione principale. Questo impedisce che approcci scartati e contesto di brainstorming influenzino il lavoro di implementazione successivo.

Domanda 44 (Scenario: Generazione di Codice con Claude Code)

Situazione: Il tuo team vuole aggiungere un server MCP GitHub per cercare pull request e controllare lo stato della CI tramite Claude Code. Ognuno dei sei sviluppatori possiede il proprio token personale di accesso a GitHub. Vuoi strumenti coerenti per tutto il team senza salvare credenziali nel controllo versione.

Quale approccio di configurazione è più efficace?

- A) Fare in modo che ogni sviluppatore aggiunga il server a livello utente tramite `claude mcp add --scope user`.
- B) Creare un wrapper per il server MCP che legga i token da un file `.env` e faccia da proxy alle chiamate dell'API GitHub, quindi aggiungere il wrapper al file `.mcp.json` del progetto.
- C) Aggiungere il server al file `.mcp.json` del progetto utilizzando la sostituzione tramite variabili d'ambiente (`${GITHUB_TOKEN}`) per l'autenticazione e documentare la variabile richiesta nel README del progetto. **[CORRETTA]**
- D) Configurare il server a livello progetto con un token segnaposto e poi chiedere agli sviluppatori di sovrascriverlo nella propria configurazione locale.

Perché C: Un file `.mcp.json` di progetto che utilizza la sostituzione tramite variabili d'ambiente rappresenta l'approccio idiomatico: fornisce un'unica fonte di configurazione MCP sotto controllo versione, permettendo però a ciascuno sviluppatore di fornire le proprie credenziali tramite variabili d'ambiente. Documentare la variabile necessaria semplifica inoltre l'onboarding senza esporre segreti nel repository.

Domanda 45 (Scenario: Generazione di Codice con Claude Code)

Situazione: Stai aggiungendo wrapper per la gestione degli errori attorno alle chiamate a API esterne in un codebase composto da 120 file. Il lavoro è suddiviso in tre fasi: (1) individuare tutti i punti in cui vengono effettuate le chiamate e i pattern utilizzati, (2) progettare collaborativamente l'approccio di gestione degli errori e (3) implementare i wrapper in modo coerente. Nella Fase 1, Claude genera un output molto esteso che elenca centinaia di punti di chiamata con il relativo contesto, riempiendo rapidamente la finestra di contesto prima ancora che la fase di scoperta sia completata.

Quale approccio è più efficace per completare il lavoro mantenendo la coerenza dell'implementazione?

- A) Utilizzare un sottoagente Explore per la Fase 1, così da isolare l'output verboso della scoperta e restituire solo un riepilogo, quindi proseguire con le Fasi 2 e 3 nella conversazione principale. **[CORRETTA]**
- B) Eseguire tutte le fasi nella conversazione principale, utilizzando periodicamente `/compact` per ridurre l'utilizzo del contesto mentre si procede tra i file.
- C) Passare alla modalità headless con `--continue`, fornendo riepiloghi espliciti del contesto tra le chiamate batch per mantenere la continuità.
- D) Definire il pattern di gestione degli errori in CLAUDE.md e poi elaborare i file in batch attraverso più sessioni, affidandosi al file di memoria condivisa per mantenere la coerenza.

Perché A: Un sottoagente Explore isola l'output dettagliato e voluminoso della fase di scoperta in un contesto separato e restituisce alla conversazione principale soltanto un riepilogo conciso. In questo modo si preserva la finestra di contesto principale per le fasi di progettazione collaborativa e implementazione coerente, nelle quali mantenere il contesto è molto più importante.

Scenario: Agente di Assistenza Clienti

Domanda 46 (Scenario: Agente di Assistenza Clienti)

Situazione: Durante i test, noti che l'agente chiama spesso `get_customer` quando gli utenti chiedono informazioni sullo stato di un ordine, anche se `lookup_order` sarebbe più appropriato. Quale aspetto dovresti verificare per primo per affrontare questo problema?

Quale aspetto dovresti verificare per primo?

- A) Implementare un classificatore di pre-elaborazione che rilevi le richieste relative agli ordini e le instradi direttamente verso `lookup_order`.
- B) Ridurre il numero di strumenti disponibili per l'agente per semplificare la scelta.
- C) Aggiungere esempi few-shot al prompt di sistema che coprano tutti i possibili pattern di richieste sugli ordini per migliorare la selezione degli strumenti.

- D) Verificare che le descrizioni degli strumenti distinguano chiaramente lo scopo di ciascuno. **[CORRETTA]**

Perché D: Le descrizioni degli strumenti rappresentano l'input principale che il modello utilizza per decidere quale strumento chiamare. Quando un agente seleziona costantemente lo strumento sbagliato, il primo passo diagnostico consiste nel verificare che le descrizioni separino chiaramente lo scopo e i limiti di utilizzo di ciascuno strumento.

Domanda 47 (Scenario: Agente di Assistenza Clienti)

Situazione: Il tuo agente gestisce richieste con un singolo problema con il 94% di accuratezza (ad esempio: “Ho bisogno di un rimborso per l'ordine #1234”). Tuttavia, quando i clienti includono più problemi nello stesso messaggio (ad esempio: “Ho bisogno di un rimborso per l'ordine #1234 e voglio anche aggiornare l'indirizzo di spedizione dell'ordine #5678”), l'accuratezza nella selezione degli strumenti scende al 58%. L'agente di solito risolve solo uno dei problemi oppure mescola i parametri tra le richieste. Quale approccio migliora più efficacemente l'affidabilità per le richieste con più problemi?

Quale approccio è più efficace?

- A) Implementare un livello di pre-elaborazione che utilizzi una chiamata a un modello separato per scomporre i messaggi con più problemi in richieste distinte, gestirle indipendentemente e poi unire i risultati.
- B) Combinare gli strumenti correlati in un numero inferiore di strumenti universali.
- C) Aggiungere esempi few-shot al prompt che mostrino il corretto ragionamento e la corretta sequenza di utilizzo degli strumenti per richieste con più problemi. **[CORRETTA]**
- D) Implementare una validazione delle risposte che rilevi risposte incomplete e riprompti automaticamente l'agente affinché risolva i problemi trascurati.

Perché C: Gli esempi few-shot che dimostrano il corretto ragionamento e la corretta sequenza degli strumenti per richieste con più problemi sono la soluzione più efficace perché l'agente già gestisce bene i singoli problemi. Ciò di cui ha bisogno è una guida sul pattern da seguire per scomporre, instradare e mantenere separati i parametri relativi a più richieste nello stesso messaggio.

Domanda 48 (Scenario: Agente di Assistenza Clienti)

Situazione: I log di produzione mostrano che, per richieste semplici come “rimborso per l'ordine #1234”, il tuo agente risolve il problema in 3–4 chiamate agli strumenti con un tasso di successo del 91%. Tuttavia, per richieste complesse come “Mi è stato addebitato due volte, il mio sconto non è stato applicato e voglio annullare l'ordine”, l'agente effettua in media più di 12 chiamate agli strumenti con un tasso di successo del 54%, spesso investigando i problemi in sequenza e recuperando dati cliente ridondanti per ciascuno di essi. Quale modifica migliora più efficacemente la gestione delle richieste complesse?

Quale modifica è più efficace?

- A) Aggiungere checkpoint di verifica espliciti tra le fasi, richiedendo all'agente di registrare i progressi dopo aver risolto ogni problema prima di passare al successivo.

- B) Ridurre il numero di strumenti combinando `get_customer`, `lookup_order` e gli strumenti relativi alla fatturazione in un unico strumento `investigate_issue`.
- C) Scomporre la richiesta in problemi separati, quindi analizzare ciascuno in parallelo utilizzando un contesto cliente condiviso prima di sintetizzare una risoluzione finale. **[CORRETTA]**
- D) Aggiungere esempi few-shot al prompt di sistema che mostrino sequenze ideali di chiamate agli strumenti per diversi scenari complessi di fatturazione.

Perché C: Scomporre la richiesta in problemi distinti e analizzarli in parallelo utilizzando un contesto cliente condiviso risolve entrambi i problemi principali: elimina il recupero ridondante dei dati riutilizzando il contesto condiviso tra le varie problematiche e riduce il numero totale di cicli di chiamate agli strumenti grazie all'investigazione parallela prima della sintesi di una singola risoluzione finale.

Domanda 49 (Scenario: Agente di Assistenza Clienti)

Situazione: Il tuo agente raggiunge un tasso di risoluzione al primo contatto del 55%, ben al di sotto dell'obiettivo dell'80%. I log mostrano che effettua escalation per casi semplici (sostituzioni standard di prodotti danneggiati con prova fotografica) mentre cerca di gestire autonomamente situazioni complesse che richiedono eccezioni alle policy. Qual è il modo più efficace per migliorare la calibrazione delle escalation?

Qual è il modo più efficace per migliorare la calibrazione delle escalation?

- A) Richiedere all'agente di auto-valutare la propria confidenza su una scala da 1 a 10 prima di ogni risposta e instradare automaticamente a un operatore umano quando la confidenza scende sotto una determinata soglia.
- B) Implementare un classificatore separato addestrato su ticket storici per prevedere quali richieste necessitano di escalation prima che l'agente principale inizi l'elaborazione.
- C) Aggiungere criteri di escalation espliciti al prompt di sistema, accompagnati da esempi few-shot che mostrino quando effettuare un'escalation e quando risolvere autonomamente. **[CORRETTA]**
- D) Implementare un'analisi del sentiment per determinare il livello di frustrazione del cliente ed effettuare automaticamente un'escalation oltre una soglia di sentiment negativo.

Perché C: Criteri di escalation espliciti accompagnati da esempi few-shot affrontano direttamente la causa principale: confini decisionali poco chiari tra casi semplici e complessi. È l'intervento iniziale più proporzionato ed efficace perché insegna all'agente quando effettuare un'escalation e quando risolvere autonomamente, senza richiedere infrastrutture aggiuntive.

Domanda 50 (Scenario: Agente di Assistenza Clienti)

Situazione: Dopo aver chiamato `get_customer` e `lookup_order`, l'agente dispone di tutti i dati disponibili nei sistemi ma si trova ancora in una situazione di incertezza. Quale scenario rappresenta il motivo più giustificato per chiamare `escalate_to_human`?

Quale situazione giustifica maggiormente un'escalation?

- A) Un cliente vuole annullare un ordine spedito ieri e in consegna domani. L'agente dovrebbe effettuare un'escalation perché il cliente potrebbe cambiare idea dopo aver ricevuto il pacco.
- B) Un cliente sostiene di non aver ricevuto un ordine, ma il tracking mostra che è stato consegnato e firmato al suo indirizzo tre giorni prima. L'agente dovrebbe effettuare un'escalation perché presentare prove contraddittorie potrebbe danneggiare il rapporto con il cliente.
- C) Un cliente richiede l'allineamento del prezzo a quello di un concorrente. Le policy consentono adeguamenti di prezzo per ribassi sul proprio sito entro 14 giorni, ma non dicono nulla riguardo ai prezzi dei concorrenti. L'agente dovrebbe effettuare un'escalation per interpretazione della policy. **[CORRETTA]**
- D) Un messaggio del cliente contiene sia una domanda sulla fatturazione sia una richiesta di reso prodotto. L'agente dovrebbe effettuare un'escalation affinché un operatore umano possa coordinare entrambe le problematiche in un'unica interazione.

Perché C: Questo è un vero vuoto normativo nelle policy aziendali: le regole coprono i ribassi di prezzo sul proprio sito ma non affrontano il caso del price matching con i concorrenti. L'agente non deve inventare una policy e dovrebbe effettuare un'escalation per consentire a un umano di decidere come interpretare o estendere le regole esistenti.

Domanda 51 (Scenario: Agente di Assistenza Clienti)

Situazione: I log di produzione mostrano che nel 12% dei casi il tuo agente salta `get_customer` e chiama direttamente `lookup_order` utilizzando soltanto il nome fornito dal cliente, causando talvolta identificazioni errate degli account e rimborsi non corretti. Quale modifica risolve più efficacemente questo problema di affidabilità?

Quale modifica è più efficace?

- A) Aggiungere esempi few-shot che mostrino come l'agente debba sempre chiamare `get_customer` per primo, anche quando i clienti forniscono spontaneamente dettagli relativi all'ordine.
- B) Implementare un classificatore di instradamento che analizzi ogni richiesta e abiliti solo un sottoinsieme di strumenti appropriati per quel tipo di richiesta.
- C) Aggiungere una precondizione programmatica che blocchi `lookup_order` e `process_refund` finché `get_customer` non restituisce un identificatore cliente verificato. **[CORRETTA]**
- D) Rafforzare il prompt di sistema specificando che la verifica del cliente tramite `get_customer` è obbligatoria prima di qualsiasi operazione sugli ordini.

Perché C: Una precondizione programmatica fornisce una garanzia deterministica che la sequenza richiesta venga rispettata. È l'approccio più efficace perché elimina completamente la possibilità di saltare la fase di verifica, indipendentemente dal comportamento dell'LLM.

Domanda 52 (Scenario: Agente di Assistenza Clienti)

Situazione: Le metriche di produzione mostrano che, nella gestione di contestazioni di fatturazione complesse o resi relativi a più ordini, i punteggi di soddisfazione dei clienti sono inferiori del 15% rispetto ai casi semplici, anche quando la soluzione è tecnicamente corretta. L'analisi delle cause mostra che

l'agente fornisce soluzioni accurate ma spiega il ragionamento in modo incoerente: talvolta omette dettagli rilevanti delle policy, altre volte trascura informazioni sulle tempistiche o sui passaggi successivi. Le lacune specifiche variano da caso a caso. Vuoi migliorare la qualità delle soluzioni senza aggiungere supervisione umana. Quale approccio è più efficace?

Quale approccio è più efficace?

- A) Aggiungere una fase di auto-critica in cui l'agente valuta una bozza di risposta verificandone la completezza — assicurandosi che risolva il problema del cliente, includa il contesto rilevante e anticipi possibili domande successive. **[CORRETTA]**
- B) Aggiungere una fase di conferma in cui l'agente chiede “Questo risolve completamente il suo problema?” prima di chiudere il caso, consentendo al cliente di richiedere informazioni aggiuntive se necessario.
- C) Passare dal modello Haiku al modello Sonnet per i casi complessi, instradando le richieste in base a una metrica di complessità definita.
- D) Implementare esempi few-shot nel prompt di sistema che mostrino spiegazioni complete per cinque tipologie comuni di casi complessi, dimostrando come includere contesto delle policy, tempistiche e passaggi successivi.

Perché A: Una fase di auto-critica (pattern evaluator-optimizer) affronta direttamente il problema dell'incompletezza delle spiegazioni, costringendo l'agente a valutare la propria bozza rispetto a criteri concreti — come contesto delle policy, tempistiche e passaggi successivi — prima di presentarla al cliente. Questo permette di individuare lacune specifiche del caso senza richiedere supervisione umana.

Domanda 53 (Scenario: Agente di Assistenza Clienti)

Situazione: Le metriche di produzione mostrano che il tuo agente effettua in media più di 4 cicli API per ogni risoluzione. L'analisi rivela che Claude richiede spesso `get_customer` e `lookup_order` in turni sequenziali separati, anche quando entrambi sarebbero necessari fin dall'inizio. Qual è il modo più efficace per ridurre il numero di cicli?

Qual è il modo più efficace per ridurre i cicli?

- A) Implementare un'esecuzione speculativa che chiami automaticamente in parallelo gli strumenti probabilmente necessari insieme a qualsiasi strumento richiesto e restituisca tutti i risultati indipendentemente da ciò che era stato richiesto.
- B) Aumentare `max_tokens` per dare a Claude più spazio per pianificare e combinare naturalmente le richieste agli strumenti.
- C) Creare strumenti compositi come `get_customer_with_orders` che raggruppino le combinazioni di ricerca più comuni in un'unica chiamata.
- D) Istruire Claude nel prompt a raggruppare le richieste agli strumenti in un unico turno e a restituire tutti i risultati insieme prima della successiva chiamata API. **[CORRETTA]**

Perché D: Istruire Claude a raggruppare le richieste correlate agli strumenti in un unico turno sfrutta la sua capacità nativa di richiedere più strumenti contemporaneamente. Questo corregge direttamente il pattern di chiamate sequenziali con modifiche architetturali minime.

Domanda 54 (Scenario: Agente di Assistenza Clienti)

Situazione: I log di produzione mostrano un pattern ricorrente: i clienti fanno riferimento a importi specifici (ad esempio “lo sconto del 15% che ho menzionato”), ma l'agente risponde con valori errati. L'indagine mostra che questi dettagli erano stati menzionati oltre 20 turni prima e sono stati condensati in riassunti vaghi come “si è discusso di prezzi promozionali”. Quale soluzione è più efficace?

Qual è la soluzione più efficace?

- A) Aumentare la soglia di riassunto dal 70% all'85%, in modo che le conversazioni abbiano più spazio prima che venga attivata la sintesi.
- B) Memorizzare l'intera cronologia della conversazione in uno storage esterno e implementare un sistema di recupero quando l'agente rileva riferimenti come “come ho già detto”.
- C) Estrarre i fatti transazionali (importi, date, numeri d'ordine) in un blocco persistente di “fatti del caso” incluso in ogni prompt al di fuori della cronologia riassunta. **[CORRETTA]**
- D) Modificare il prompt di sintesi affinché preservi esplicitamente tutti i numeri, le percentuali, le date e le aspettative espresse dal cliente in forma letterale.

Perché C: La sintesi comporta inevitabilmente la perdita di dettagli precisi. Estrarre i fatti transazionali in un blocco strutturato di “fatti del caso”, separato dalla cronologia riassunta, preserva le informazioni critiche e le rende disponibili in modo affidabile in ogni prompt, indipendentemente dal numero di turni già sintetizzati.

Domanda 55 (Scenario: Agente di Assistenza Clienti)

Situazione: Il tuo strumento `get_customer` restituisce tutte le corrispondenze quando la ricerca viene effettuata per nome. Attualmente, quando ci sono più risultati, Claude sceglie il cliente con l'ordine più recente, ma i dati di produzione mostrano che questa scelta identifica l'account sbagliato nel 15% dei casi ambigui. Come dovresti affrontare il problema?

Come dovresti affrontare il problema?

- A) Implementare un sistema di punteggio di confidenza che agisca autonomamente sopra l'85% di confidenza e richieda chiarimenti al di sotto di tale soglia.
- B) Istruire Claude a richiedere un identificatore aggiuntivo (email, numero di telefono o numero d'ordine) quando `get_customer` restituisce più corrispondenze, prima di eseguire qualsiasi azione specifica per il cliente. **[CORRETTA]**
- C) Modificare `get_customer` affinché restituisca solo la corrispondenza più probabile in base a un algoritmo di ranking, eliminando l'ambiguità.
- D) Aggiungere esempi few-shot al prompt che mostrino il corretto ragionamento e la corretta sequenza degli strumenti per i casi di corrispondenza ambigua.

Perché B: Chiedere all'utente un identificatore aggiuntivo è il modo più affidabile per risolvere l'ambiguità, perché l'utente possiede informazioni definitive sulla propria identità. Un turno conversazionale aggiuntivo è un prezzo minimo da pagare per eliminare un tasso di errore del 15% causato dalla selezione dell'account sbagliato.

Domanda 56 (Scenario: Agente di Assistenza Clienti)

Situazione: I log di produzione mostrano un pattern costante: quando i clienti includono la parola “account” nel messaggio (ad esempio “Voglio controllare il mio account per un ordine effettuato ieri”), l'agente chiama `get_customer` per primo nel 78% dei casi. Quando richieste simili vengono formulate senza la parola “account” (ad esempio “Voglio controllare un ordine effettuato ieri”), l'agente chiama `lookup_order` per primo nel 93% dei casi. Le descrizioni degli strumenti sono chiare e prive di ambiguità. Qual è la causa principale più probabile di questa discrepanza?

Qual è la causa principale più probabile?

- A) Il prompt di sistema contiene istruzioni sensibili alle parole chiave che orientano il comportamento in base a termini come “account”, creando pattern indesiderati nella selezione degli strumenti. **[CORRETTA]**
- B) L'addestramento di base del modello crea associazioni tra la terminologia “account” e le operazioni relative ai clienti, sovrascrivendo le descrizioni degli strumenti.
- C) Il modello necessita di più dati di addestramento sui messaggi multi-concetto e dovrebbe essere sottoposto a fine-tuning con esempi che contengano sia terminologia relativa agli account sia agli ordini.
- D) Le descrizioni degli strumenti necessitano di ulteriori esempi negativi che specifichino quando NON utilizzare ciascuno strumento per evitare questa confusione indotta dalle parole chiave.

Perché A: Il pattern sistematico guidato da parole chiave (78% contro 93%) indica fortemente la presenza di una logica di instradamento esplicita nel prompt di sistema che reagisce alla parola “account” e indirizza l'agente verso strumenti legati ai clienti. Poiché le descrizioni degli strumenti sono già chiare, la discrepanza punta a istruzioni a livello di prompt che generano un orientamento comportamentale indesiderato.

Domanda 57 (Scenario: Agente di Assistenza Clienti)

Situazione: I log di produzione mostrano che l'agente chiama spesso `get_customer` quando gli utenti chiedono informazioni sugli ordini (ad esempio: “controlla il mio ordine #12345”) invece di chiamare `lookup_order`. Entrambi gli strumenti hanno descrizioni minime (“Ottiene informazioni sul cliente” / “Ottiene dettagli dell'ordine”) e accettano formati di identificatore apparentemente simili. Qual è il primo passo più efficace per migliorare l'affidabilità della selezione degli strumenti?

Qual è il primo passo più efficace?

- A) Implementare un livello di instradamento che analizzi l'input dell'utente prima di ogni turno e preselezioni lo strumento corretto in base alle parole chiave e ai pattern degli identificatori rilevati.
- B) Combinare entrambi gli strumenti in un unico `lookup_entity` che accetti qualsiasi identificatore e decida internamente quale backend interrogare.
- C) Aggiungere esempi few-shot al prompt di sistema che dimostrino i corretti pattern di selezione degli strumenti, con 5–8 esempi che instradano le richieste relative agli ordini verso `lookup_order`.
- D) Ampliare la descrizione di ciascuno strumento includendo formati di input, esempi di richieste, casi limite e confini di utilizzo che spieghino quando usarlo rispetto a strumenti simili. **[CORRETTA]**

Perché D: Ampliare le descrizioni degli strumenti con formati di input, esempi di richieste, casi limite e confini di utilizzo risolve direttamente la causa principale: descrizioni troppo minimali che non forniscono all'LLM informazioni sufficienti per distinguere strumenti simili. È un intervento a basso costo e ad alto impatto che migliora il principale meccanismo utilizzato dall'LLM per la selezione degli strumenti.

Domanda 58 (Scenario: Agente di Assistenza Clienti)

Situazione: Stai implementando il ciclo dell'agente per il tuo sistema di assistenza clienti. Dopo ogni chiamata all'API di Claude, devi decidere se continuare il ciclo (eseguire gli strumenti richiesti e richiamare Claude) oppure interromperlo (presentare la risposta finale al cliente). Da cosa dipende questa decisione?

Da cosa dipende questa decisione?

- A) Controllare il campo `stop_reason` nella risposta di Claude: continuare se è `tool_use` e fermarsi se è `end_turn`. **[CORRETTA]**
- B) Analizzare il testo di Claude alla ricerca di frasi come “Ho finito” o “Posso aiutarla in altro?”: questi segnali in linguaggio naturale indicano il completamento del compito.
- C) Impostare un numero massimo di iterazioni (ad esempio 10 chiamate) e fermarsi quando viene raggiunto, indipendentemente dal fatto che Claude indichi di dover fare altro lavoro.
- D) Verificare se la risposta contiene contenuto testuale dell'assistente: se Claude ha generato un testo esplicativo, il ciclo dovrebbe terminare.

Perché A: `stop_reason` è il segnale strutturato esplicito che Claude fornisce per controllare il ciclo. `tool_use` indica che Claude desidera eseguire uno strumento e ricevere i risultati, mentre `end_turn` indica che la risposta è stata completata e il ciclo può terminare.

Domanda 59 (Scenario: Agente di Assistenza Clienti)

Situazione: I log di produzione mostrano che l'agente interpreta in modo errato gli output dei tuoi strumenti MCP: timestamp Unix provenienti da `get_customer`, date ISO 8601 provenienti da `lookup_order` e codici di stato numerici (1 = in attesa, 2 = spedito). Alcuni strumenti provengono da server MCP di terze parti che non puoi modificare. Quale approccio alla normalizzazione dei formati dati è il più manutenibile?

Quale approccio è il più manutenibile?

- A) Utilizzare un hook PostToolUse per intercettare gli output degli strumenti e applicare trasformazioni di formattazione prima che l'agente li elabori. **[CORRETTA]**
- B) Modificare gli strumenti sotto il tuo controllo affinché restituiscano formati leggibili dall'uomo e creare wrapper per quelli di terze parti.
- C) Creare uno strumento `normalize_data` che l'agente chiami dopo ogni recupero di dati per trasformare i valori.
- D) Aggiungere al prompt di sistema una documentazione dettagliata che spieghi le convenzioni di formato dati di ciascuno strumento.

Perché A: Un hook PostToolUse fornisce un punto centralizzato e deterministico per intercettare e normalizzare tutti gli output degli strumenti — inclusi quelli provenienti da server MCP di terze parti — prima che vengano elaborati dall'agente. È più manutenibile perché le trasformazioni sono implementate nel codice e applicate uniformemente, invece di affidarsi all'interpretazione dell'LLM.

Domanda 60 (Scenario: Agente di Assistenza Clienti)

Situazione: I log di produzione mostrano che l'agente sceglie talvolta `get_customer` quando `lookup_order` sarebbe più appropriato, soprattutto per richieste ambigue come “Ho bisogno di aiuto con il mio acquisto recente”. Decidi di aggiungere esempi few-shot al prompt di sistema per migliorare la selezione degli strumenti. Quale approccio affronta il problema nel modo più efficace?

Quale approccio è più efficace?

- A) Aggiungere nelle descrizioni degli strumenti indicazioni esplicite del tipo “usa quando” e “non usare quando” che coprano i casi ambigui.
- B) Aggiungere esempi raggruppati per strumento: prima tutti gli scenari relativi a `get_customer`, poi tutti quelli relativi a `lookup_order`.
- C) Aggiungere 4–6 esempi mirati a scenari ambigui, ciascuno con una spiegazione del motivo per cui uno strumento è stato scelto rispetto ad alternative plausibili. **[CORRETTA]**
- D) Aggiungere 10–15 esempi di richieste chiare e non ambigue che dimostrino la corretta scelta dello strumento per gli scenari tipici di ciascuno strumento.

Perché C: Concentrarsi sugli scenari ambigui in cui si verificano gli errori, fornendo una spiegazione esplicita del motivo per cui uno strumento è preferibile rispetto alle alternative, insegna al modello il processo decisionale comparativo necessario per gestire i casi limite. Questo è più efficace di esempi generici o regole puramente dichiarative.

Domanda 61 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Il tuo strumento `remove_team_member` utilizza un parametro `dry_run: boolean` per visualizzare in anteprima gli impatti prima dell'esecuzione. Il monitoraggio in produzione mostra che l'agente aggira il passaggio di anteprima chiamando direttamente lo strumento con `dry_run=false`. Devi garantire che ogni rimozione sia preceduta da un'anteprima che l'utente confermi esplicitamente.

Qual è l'approccio più affidabile?

- A) Aggiungere una validazione lato server che consenta `dry_run=false` solo se una chiamata con `dry_run=true` e parametri identici è stata effettuata negli ultimi 60 secondi.
- B) Contrassegnare lo strumento come richiedente conferma e configurare il livello di orchestrazione affinché chieda l'approvazione dell'utente prima di inoltrare qualsiasi chiamata a strumenti contrassegnati.
- C) Aggiungere istruzioni dettagliate ed esempi few-shot alla descrizione dello strumento, richiedendo all'agente di chiamarlo sempre prima con `dry_run=true` e attendere la conferma dell'utente prima di

richiamarlo.

- D) Sostituirlo con due strumenti: `preview_remove_member`, che restituisce i dettagli dell'impatto e un token di conferma monouso, ed `execute_remove_member`, che richiede quel token, vincolando l'esecuzione all'anteprima. **[CORRETTA]**

Perché D: L'approccio basato su due strumenti con vincolo tramite token rende architetturalmente impossibile eseguire l'azione senza una precedente anteprima: lo strumento di esecuzione richiede infatti un token che solo lo strumento di anteprima può generare. È l'unico approccio che applica il vincolo a livello di codice invece di affidarsi al rispetto delle istruzioni da parte dell'LLM (C), a euristiche temporali (A) o all'infrastruttura di orchestrazione (B).

Domanda 62 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Il monitoraggio in produzione mostra che il tuo strumento `search_catalog` fallisce nel 12% dei casi: l'8% sono timeout di rete che si risolvono con un nuovo tentativo, mentre il 4% sono errori di sintassi nelle query che non si risolvono mai, indipendentemente dai tentativi. Attualmente entrambi i tipi di errore vengono restituiti nello stesso modo, causando tentativi inutili.

Come dovresti modificare la gestione degli errori dello strumento?

- A) Aggiungere esempi few-shot al prompt di sistema che mostrino come distinguere gli errori di rete dagli errori di sintassi.
- B) Applicare una logica di retry con backoff esponenziale a tutti gli errori in modo uniforme.
- C) Implementare all'interno dello strumento un retry automatico con backoff per i timeout di rete; restituire immediatamente gli errori di sintassi con dettagli sulla validazione dei parametri. **[CORRETTA]**
- D) Restituire tutti gli errori con un flag booleano `retryable` e dettagli sul tipo di errore.

Perché C: Gestire i retry direttamente a livello dello strumento per gli errori transitori è il corretto livello di astrazione: lo strumento conosce con certezza il tipo di errore e può implementare una logica di retry deterministica senza affidarsi all'agente per interpretare un flag (D) o seguire istruzioni nel prompt (A). Applicare un backoff uniforme (B) spreca tempo sugli errori di sintassi che non avranno mai successo.

Domanda 63 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Nel corso di più turni di conversazione su una strategia di investimento, un utente ha dichiarato prima: “Ho una tolleranza al rischio molto bassa” e successivamente: “Voglio massimizzare i miei rendimenti”. Ora chiede: “In cosa dovrei investire?”

Quale approccio garantisce meglio che la raccomandazione sia allineata alla reale priorità dell'utente?

- A) Evidenziare la contraddizione e chiedere all'utente di chiarire quale aspetto sia più importante. **[CORRETTA]**
- B) Fornire raccomandazioni separate per entrambi gli scenari.
- C) Procedere sulla base della preferenza espressa più recentemente.
- D) Consigliare un portafoglio bilanciato senza affrontare il conflitto.

Perché A: Quando le preferenze dell'utente sono direttamente in contraddizione, evidenziare il conflitto e chiedere chiarimenti è l'unico modo per garantire che la raccomandazione sia realmente allineata alle sue intenzioni. Qualsiasi altra soluzione implica un'ipotesi che potrebbe essere errata: massimizzare i rendimenti e avere una bassa tolleranza al rischio sono obiettivi fondamentalmente incompatibili che richiedono una decisione umana.

Domanda 64 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Gli utenti affinano le preferenze musicali per le playlist nel corso di più turni di conversazione. Due messaggi dopo che un utente ha detto “Adoro il jazz”, Claude chiede: “Quali generi musicali ti piacciono?”

Qual è la causa più probabile?

- A) Claude richiede una connessione a un database vettoriale per mantenere la memoria della conversazione.
- B) È stata superata la finestra di contesto del modello.
- C) L'API di Claude richiede un parametro `session_id`.
- D) La tua applicazione non sta includendo i messaggi precedenti nell'array `messages`. **[CORRETTA]**

Perché D: Claude non possiede memoria lato server: ogni chiamata API è stateless. Se la cronologia completa della conversazione non viene inclusa nell'array `messages` di ogni richiesta, Claude non ha alcuna conoscenza dei turni precedenti. I database vettoriali (A) e il parametro `session_id` (C) non fanno parte dell'architettura di Claude; inoltre un overflow della finestra di contesto (B) è impossibile in uno scambio di soli due messaggi.

Domanda 65 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Dopo una sessione di cucina durata 40 minuti, la conversazione raggiunge 78.000 token. La cronologia include allergie alimentari, ridimensionamento delle ricette, chiarimenti su termini culinari e discussioni generali. Devi ridurre il numero di token preservando le informazioni importanti.

Quale approccio offre il miglior equilibrio tra conservazione delle informazioni e riduzione dei token?

- A) Riassumere l'intera cronologia della conversazione.
- B) Conservare solo gli ultimi 20.000 token.

- C) Estrarre i dati strutturati critici (allergie, quantità, preferenze), riassumere le discussioni generali e mantenere verbatim gli scambi più recenti. **[CORRETTA]**
- D) Archiviare l'intera conversazione esternamente e recuperare le parti rilevanti tramite ricerca semantica.

Perché C: L'approccio ibrido preserva le informazioni di maggior valore al costo più basso. I fatti critici come allergie e quantità delle ricette vengono estratti in un blocco strutturato compatto (evitando la perdita di precisione tipica della sintesi), le discussioni generali vengono riassunte e gli scambi più recenti vengono mantenuti integralmente per preservare la coerenza conversazionale. Le opzioni A e B rischiano di perdere informazioni dietetiche fondamentali; la D rappresenta una soluzione eccessivamente complessa per una singola sessione di cucina.

Domanda 66 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Gli utenti segnalano che, durante conversazioni prolungate, l'assistente perde traccia degli argomenti e delle preferenze menzionati in precedenza. L'implementazione attuale conserva solo le ultime 25 coppie di messaggi.

Qual è la soluzione più efficace?

- A) Approccio ibrido: riassumere i messaggi più vecchi mantenendo quelli recenti in forma integrale. **[CORRETTA]**
- B) Ricerca per similarità vettoriale sull'intera cronologia della conversazione.
- C) Aumentare la finestra a 50 coppie di messaggi.
- D) Riassumere i messaggi eliminati a ogni turno e anteporre il riassunto cumulativo.

Perché A: L'approccio ibrido affronta entrambe le dimensioni del problema: mantiene il contesto recente in forma esatta (fondamentale per la coerenza conversazionale) e conserva una rappresentazione compressa delle preferenze e informazioni più vecchie (evitando che vadano completamente perse quando i messaggi vengono rimossi). Aumentare la finestra (C) rimanda semplicemente lo stesso problema. La ricerca vettoriale (B) potrebbe non recuperare contesti importanti che non sono semanticamente simili alla richiesta corrente. La sintesi completa a ogni turno (D) introduce overhead e accumula errori di riassunto.

Domanda 67 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Gli utenti segnalano che la latenza aumenta e i costi crescono quando le conversazioni superano i 50 turni.

Qual è la causa principale?

- A) L'intera cronologia della conversazione viene inclusa in ogni richiesta API. **[CORRETTA]**
- B) Il modello genera risposte progressivamente più lunghe.

- C) Le operazioni sul database rallentano man mano che la cronologia cresce.
- D) Il modello costruisce un profilo interno dell'utente che richiede più elaborazione.

Perché A: L'API di Claude è completamente stateless: ogni richiesta deve includere l'intera cronologia della conversazione nell'array `messages`. Con la crescita della conversazione, ogni richiesta trasporta più token, aumentando direttamente sia la latenza di elaborazione sia il costo. Il modello non mantiene alcuno stato interno tra le chiamate (quindi D è falsa) e la lunghezza delle risposte non è intrinsecamente legata alla lunghezza della conversazione (B).

Domanda 68 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Dopo tre mesi di sessioni settimanali, la cronologia della conversazione è cresciuta fino a 85.000 token. Quando un utente chiede: “A quale conclusione siamo arrivati riguardo al tema dell'isolamento?”, l'assistente fornisce risposte generiche invece di fare riferimento alle discussioni precedenti.

Qual è l'approccio più efficace?

- A) Troncamento tramite finestra mobile (*rolling window*).
- B) Riassunto progressivo che catturi le conclusioni principali.
- C) Embedding semantici con recupero degli scambi rilevanti. **[CORRETTA]**
- D) Aggiungere tag XML strutturati per contrassegnare le conclusioni delle discussioni.

Perché C: La ricerca semantica sulla cronologia della conversazione è l'unico approccio che scala a tre mesi di discussioni mantenendo la capacità di recuperare su richiesta scambi specifici e pertinenti. La finestra mobile (A) eliminerebbe gran parte della cronologia. Il riassunto progressivo (B) comprime le discussioni in astrazioni che perdono le conclusioni specifiche richieste dagli utenti. I tag XML (D) richiederebbero una ristrutturazione di tutti i contenuti passati e non risolvono il problema del recupero a questa scala.

Domanda 69 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Durante i test QA, Claude segue le linee guida del prompt di sistema per i primi 10–15 turni, ma successivamente le risposte iniziano a deviare. La conversazione è ancora entro i limiti di token consentiti.

Qual è la soluzione migliore?

- A) Spostare le linee guida comportamentali nel primo messaggio dell'utente.
- B) Avviare una nuova conversazione dopo 20 turni.
- C) Inserire messaggi con ruolo utente che rafforzino le linee guida nei punti di interruzione della conversazione. **[CORRETTA]**
- D) Utilizzare una validazione post-risposta per rigenerare le risposte non conformi.

Perché C: L'iniezione periodica di promemoria comportamentali contrasta direttamente la deriva delle istruzioni (*instruction drift*) ristabilendo regolarmente i vincoli man mano che la cronologia della conversazione cresce. Spostare le linee guida nel primo messaggio dell'utente (A) ne riduce l'autorità. Avviare una nuova conversazione (B) distrugge il contesto. La validazione post-risposta (D) è correttiva anziché preventiva e aggiunge una latenza significativa.

Domanda 70 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Il tuo tutor IA utilizza un prompt di sistema di 2.800 token che definisce la metodologia didattica e le regole di adattamento. Dopo 12 turni, l'assistente inizia a ignorare i livelli di competenza dello studente.

Qual è la soluzione più efficace?

- A) Inserire promemoria ogni 4–5 turni.
- B) Sostituire le regole verbose con esempi few-shot che dimostrino l'adattamento al livello di competenza. **[CORRETTA]**
- C) Posizionare le regole critiche alla fine del prompt di sistema.
- D) Valutare le risposte e rigenerarle se il livello di difficoltà non corrisponde.

Perché B: Un prompt di sistema di 2.800 token composto principalmente da regole dichiarative è vulnerabile alla deriva perché richiede al modello di ragionare continuamente su istruzioni astratte. Sostituire tali regole con esempi few-shot concreti che mostrano come adattare correttamente la spiegazione al livello di competenza fornisce pattern comportamentali chiari da imitare. Questo approccio viene seguito più affidabilmente nel corso di molti turni rispetto alle istruzioni astratte. L'iniezione di promemoria (A) allevia il sintomo ma non la causa; il posizionamento finale (C) aiuta solo inizialmente; la rigenerazione (D) è costosa e correttiva.

Domanda 71 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Il tuo assistente deve mantenere un tono entusiasta, spiegare il proprio ragionamento e porre domande di chiarimento. Dove dovrebbero essere definite queste linee guida comportamentali?

Dove dovrebbero essere definite queste linee guida comportamentali?

- A) Anteposte a ogni messaggio dell'utente.
- B) Nel prompt di sistema. **[CORRETTA]**
- C) Nel primo messaggio dell'assistente.
- D) Nelle variabili d'ambiente.

Perché B: Il prompt di sistema è progettato specificamente per contenere vincoli comportamentali persistenti e linee guida che devono applicarsi durante l'intera conversazione. Anteporre istruzioni a ogni messaggio utente (A) introduce un overhead ridondante. Il primo messaggio dell'assistente (C) è

inaffidabile perché il modello può discostarsi dalle proprie affermazioni precedenti. Le variabili d'ambiente (D) non influenzano il comportamento del modello.

Domanda 72 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Gli utenti segnalano aperture di risposta ripetitive come "Certamente!" e "Sarò felice di aiutarti!".

Qual è l'approccio più efficace?

- A) Anteporre un messaggio parziale dell'assistente con l'inizio diretto della risposta. **[CORRETTA]**
- B) Ridurre il valore della temperatura.
- C) Applicare un post-processing alle risposte per rimuovere i saluti.
- D) Aggiungere istruzioni nel prompt di sistema per evitare quelle frasi.

Perché A: Il prefill della risposta dell'assistente con l'inizio di una risposta diretta impedisce la generazione dei consueti saluti già a livello di generazione: il modello continua dal testo precompilato invece di creare nuove formule di apertura. Le istruzioni nel prompt di sistema (D) possono aiutare, ma sono meno affidabili perché il modello potrebbe comunque produrre varianti simili. Il post-processing (C) è una soluzione fragile. La temperatura (B) controlla la casualità, non specifici pattern linguistici.

Domanda 73 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Un webhook notifica al tuo sistema che il pacco di un utente è stato spedito mentre l'utente sta attivamente chattando. Vuoi che l'assistente integri questa informazione in modo naturale nella risposta successiva.

Qual è l'approccio migliore?

- A) Aggiungere lo stato della spedizione al prompt di sistema.
- B) Inviare immediatamente un messaggio utente sintetico.
- C) Forzare l'assistente a chiamare uno strumento di stato a ogni turno.
- D) Anteporre l'aggiornamento sullo stato della spedizione al successivo messaggio dell'utente. **[CORRETTA]**

Perché D: Anteporre l'aggiornamento sullo stato della spedizione al messaggio successivo dell'utente inserisce il contesto in tempo reale in un punto naturale della conversazione senza interromperne il flusso. Modificare il prompt di sistema (A) richiede la ricostruzione della sessione o comporta complessità architetturali. Un messaggio utente sintetico (B) può interrompere il dialogo naturale e creare confusione sull'origine dell'informazione. Forzare una chiamata a uno strumento a ogni turno (C) è inefficiente quando gli eventi sono rari.

Domanda 74 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Gli utenti inviano frequentemente richieste come "Prenota una sala per la festa". L'assistente pone più di quattro domande di chiarimento, causando un tasso di abbandono del 35%.

Quale approccio migliora meglio il compromesso tra precisione e fluidità?

- A) Procedere utilizzando valori predefiniti nascosti.
- B) Porre tutte le domande di chiarimento in un unico messaggio composto.
- C) Dichiarare esplicitamente le ipotesi assunte e procedere, invitando l'utente a correggerle se necessario. **[CORRETTA]**
- D) Utilizzare un modulo strutturato di raccolta dati.

Perché C: Dichiarare esplicitamente le ipotesi e procedere fornisce immediatamente un risultato utile all'utente, mantenendo al contempo la possibilità di correggere eventuali supposizioni errate. I valori predefiniti nascosti (A) impediscono all'utente di sapere cosa è stato assunto. Un elenco di domande (B) richiede comunque uno sforzo iniziale significativo. Un modulo strutturato (D) aggiunge ulteriore attrito, andando contro l'obiettivo di ridurre l'abbandono.

Domanda 75 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Il tuo assistente utilizza un prompt di sistema basato sulla figura di un consulente professionista. Nei primi turni segue correttamente le regole, ma intorno al settimo turno inizia a fornire consigli generici. La conversazione è lunga solo 2.500 token.

Qual è la causa più probabile?

- A) I prompt di sistema stabiliscono il comportamento solo all'inizio della conversazione.
- B) L'attenzione del modello si indebolisce con l'accumularsi dei turni.
- C) Le risposte accumulate dell'assistente diluiscono l'influenza del prompt di sistema. **[CORRETTA]**
- D) Il prompt di sistema viene inviato una sola volta.

Perché C: Man mano che le risposte dell'assistente si accumulano nella cronologia della conversazione, la proporzione di testo che riflette i vincoli comportamentali del prompt di sistema diminuisce rispetto al crescente volume di contenuto generato dall'assistente stesso. Il modello tende progressivamente a imitare i propri output precedenti invece delle istruzioni iniziali, amplificando la deriva comportamentale anche con conversazioni relativamente brevi. Il prompt di sistema viene incluso in ogni chiamata API (quindi D, da sola, è falsa) e il degrado dell'attenzione del modello (B) non è un fattore rilevante a soli 2.500 token.

Domanda 76 (Scenario: Pattern Architetturali per l'IA Conversazionale)

Situazione: Gli utenti formulano richieste vaghe come "Puoi aiutarmi con il report?". L'assistente risponde ponendo molte domande (quale report? che tipo di aiuto? qual è la scadenza?), causando un tasso di abbandono del 40%.

Qual è la soluzione migliore?

- A) Formulare ipotesi ragionevoli, dichiararle esplicitamente e offrire la possibilità di modificarle. **[CORRETTA]**
- B) Classificare l'ambiguità utilizzando un modello più piccolo prima di rispondere.
- C) Utilizzare interpretazioni predefinite senza dichiarare le ipotesi effettuate.
- D) Limitare l'assistente a una sola domanda di chiarimento per turno.

Perché A: Procedere sulla base di ipotesi ragionevoli dichiarate esplicitamente elimina completamente il continuo scambio di domande e risposte, mantenendo al tempo stesso l'utente informato e in controllo della conversazione. Interpretazioni predefinite ma non dichiarate (C) possono confondere l'utente quando la risposta non corrisponde alle sue reali intenzioni. Limitare le domande a una per turno (D) richiede comunque diversi scambi. Utilizzare un modello più piccolo per classificare l'ambiguità (B) aggiunge latenza e complessità infrastrutturale senza risolvere il vero problema dell'esperienza utente.

Esercizi Pratici

Esercizio 1: Agente Multi-strumento con Logica di Escalation

Obiettivo: Progettare un ciclo agente con integrazione degli strumenti, gestione strutturata degli errori ed escalation.

Passaggi:

1. Definire 3–4 strumenti MCP con descrizioni dettagliate (includere due strumenti simili per testare la selezione degli strumenti).
2. Implementare un ciclo agente che controlli `stop_reason` (`"tool_use"` / `"end_turn"`).
3. Aggiungere risposte di errore strutturate: `errorCategory`, `isRetryable`, descrizione.
4. Implementare un hook intercettore che blocchi le operazioni sopra una certa soglia e le instradi verso l'escalation.
5. Testare con richieste multi-aspetto.

Domini: 1 (Architettura degli agenti), 2 (Strumenti e MCP), 5 (Contesto e affidabilità)

Esercizio 2: Configurare Claude Code per lo Sviluppo in Team

Obiettivo: Configurare CLAUDE.md, comandi personalizzati, regole specifiche per percorso e server MCP.

Passaggi:

1. Creare un CLAUDE.md a livello progetto con standard universali.
2. Creare file in `.claude/rules/` con frontmatter YAML per diverse aree del codice (`paths: ["src/api/**/*"]`, `paths: ["**/*.test.*"]`).
3. Creare una skill di progetto in `.claude/skills/` con `context: fork` e `allowed-tools`.
4. Configurare un server MCP in `.mcp.json` usando variabili d'ambiente e un override personale in `~/.claude.json`.
5. Testare la modalità di pianificazione rispetto all'esecuzione diretta su attività di diversa complessità.

Domini: 3 (Configurazione di Claude Code), 2 (Strumenti e MCP)

Esercizio 3: Pipeline di Estrazione di Dati Strutturati

Obiettivo: Schemi JSON, `tool_use` per output strutturato, cicli di validazione/retry, elaborazione batch.

Passaggi:

1. Definire uno strumento di estrazione con schema JSON (campi obbligatori/facoltativi, enum con `"other"`, campi nullable).
2. Costruire un ciclo di validazione: in caso di errore, riprovare includendo il documento, l'estrazione errata e lo specifico errore di validazione.
3. Aggiungere esempi few-shot per documenti con strutture diverse.
4. Utilizzare l'elaborazione batch tramite API Message Batches: 100 documenti, gestendo i fallimenti tramite `custom_id`.
5. Instradare verso revisione umana: punteggi di confidenza a livello di campo, analisi per tipo di documento.

Domini: 4 (Prompt engineering), 5 (Contesto e affidabilità)

Esercizio 4: Progettare e Debuggare una Pipeline di Ricerca Multiagente

Obiettivo: Orchestrazione dei sottoagenti, passaggio del contesto, propagazione degli errori, sintesi con tracciamento delle fonti.

Passaggi:

1. Un coordinatore con 2+ sottoagenti (`allowedTools` include `"Task"`, il contesto viene passato esplicitamente nei prompt).
2. Eseguire i sottoagenti in parallelo tramite più chiamate `Task` in una singola risposta.

3. Richiedere output strutturato dai sottoagenti: affermazione, citazione, URL della fonte, data di pubblicazione.
4. Simulare un timeout di un sottoagente: restituire al coordinatore un contesto di errore strutturato e continuare con risultati parziali.
5. Testare con dati conflittuali: preservare entrambi i valori con attribuzione; separare risultati confermati e risultati contestati.

Domini: 1 (Architettura degli agenti), 2 (Strumenti e MCP), 5 (Contesto e affidabilità)

Appendice: Tecnologie e Concetti

Tecnologia	Aspetti principali
Claude Agent SDK	AgentDefinition, cicli degli agenti, `stop_reason`, hook (PostToolUse), creazione di sottoagenti tramite Task, `allowedTools`
Model Context Protocol (MCP)	Server MCP, strumenti, risorse, `isError`, descrizioni degli strumenti, `.mcp.json`, variabili d'ambiente
Claude Code	Gerarchia di CLAUDE.md, `.claude/rules/` con pattern glob, `.claude/commands/`, `.claude/skills/` con SKILL.md, modalità di pianificazione (*planning mode*), `/compact`, `--resume`, `fork_session`
Claude Code CLI	`-p` / `--print` per la modalità non interattiva, `--output-format json`, `--json-schema`
Claude API	`tool_use` con schemi JSON, `tool_choice` ("auto"/"any"/forzato), `stop_reason`, `max_tokens`, prompt di sistema
Message Batches API	Risparmio del 50%, finestra di elaborazione fino a 24 ore, `custom_id`, nessun supporto per chiamate multi-turno agli strumenti
JSON Schema	Campi obbligatori vs facoltativi, campi nullable, tipi enum, `"other"` + campo dettaglio, modalità strict
Pydantic	Validazione degli schemi, errori semantici, cicli di validazione/retry
Strumenti integrati	Read, Write, Edit, Bash, Grep, Glob — scopo e criteri di selezione
Few-shot prompting	Esempi mirati per situazioni ambigue, generalizzazione a nuovi pattern
Prompt chaining	Scomposizione sequenziale in passaggi focalizzati
Finestra di contesto	Budget di token, sintesi progressiva, fenomeno del "lost in the middle", file scratchpad

Gestione delle sessioni	Resume, `fork_session`, sessioni nominate, isolamento del contesto
Calibrazione della confidenza	Punteggi a livello di campo, calibrazione su dataset etichettati, campionamento stratificato

---

Argomenti Fuori Programma

I seguenti argomenti correlati **NON** saranno presenti all'esame:

- Fine-tuning dei modelli Claude o addestramento di modelli personalizzati
- Autenticazione delle API Claude, fatturazione o gestione degli account
- Implementazioni dettagliate in linguaggi di programmazione o framework specifici (oltre quanto necessario per la configurazione di strumenti e schemi)
- Distribuzione o hosting di server MCP (infrastruttura, networking, orchestrazione di container)
- Architettura interna di Claude, processo di addestramento o pesi del modello
- Constitutional AI, RLHF o metodologie di addestramento per la sicurezza
- Modelli di embedding o dettagli implementativi di database vettoriali
- Utilizzo del computer (*computer use*: automazione del browser, interazione desktop)
- Capacità di analisi delle immagini (Vision)
- Streaming API o Server-Sent Events (SSE)
- Rate limiting, quote o calcoli dettagliati dei costi API
- OAuth, rotazione delle API key o dettagli sui protocolli di autenticazione
- Configurazioni specifiche per provider cloud (AWS, GCP, Azure)
- Benchmark prestazionali o metriche di confronto tra modelli
- Dettagli implementativi della prompt caching (oltre al sapere che esiste)
- Algoritmi di conteggio dei token o dettagli della tokenizzazione

---

Raccomandazioni per la Preparazione

1. **Costruisci un agente con il Claude Agent SDK** — implementa un ciclo agente completo con chiamate agli strumenti, gestione degli errori e gestione delle sessioni. Esercitati con i sottoagenti e con il passaggio esplicito del contesto.
2. **Configura Claude Code per un progetto reale** — utilizza la gerarchia di CLAUDE.md, regole specifiche per percorso in `.claude/rules/`, skill con `context: fork` e `allowed-tools`, e integrazione di server MCP.
3. **Progetta e testa strumenti MCP** — scrivi descrizioni che distinguano chiaramente strumenti simili, restituiscano errori strutturati con categorie e flag di retry e testali con richieste utente ambigue.

4. **Costruisci una pipeline di estrazione dati** — utilizza `tool_use` con schemi JSON, cicli di validazione/retry, campi opzionali/nullable e l'elaborazione batch tramite la Message Batches API.
5. **Esercitati con il prompt engineering** — aggiungi esempi few-shot per scenari ambigui, criteri di revisione espliciti e architetture multi-pass per revisioni di codice su larga scala.
6. **Studia i pattern di gestione del contesto** — estrai i fatti importanti da output verbosi, utilizza file scratchpad e delega l'attività esplorativa ai sottoagenti per gestire i limiti di contesto.
7. **Comprendi escalation e supervisione umana (*human-in-the-loop*)** — quando effettuare un'escalation (lacune nelle policy, richiesta esplicita di un operatore umano, impossibilità di progredire) e come progettare flussi di instradamento basati sulla confidenza.
8. **Sostieni un esame di prova** prima di affrontare quello reale. Utilizza gli stessi scenari e lo stesso formato dell'esame ufficiale.