Certification Claude Certified Architect — Foundations

Guide d'étude (basé sur le guide officiel de l'examen)

Introduction

La certification **Claude Certified Architect — Foundations** atteste qu'un spécialiste sait prendre des décisions de compromis pertinentes lors de la mise en œuvre de solutions concrètes basées sur Claude. L'examen évalue les connaissances fondamentales de Claude Code, du Claude Agent SDK, de la Claude API et du Model Context Protocol (MCP) — les technologies clés pour construire des applications de production avec Claude.

Les questions de l'examen reposent sur des scénarios industriels réalistes : construction de systèmes agentiques pour le support client, conception de pipelines de recherche multi-agents, intégration de Claude Code dans la CI/CD, création d'outils de productivité pour les développeurs et extraction de données structurées à partir de documents non structurés.

Candidat cible

Le candidat idéal est un **architecte de solutions** qui conçoit et livre des applications de production avec Claude. Vous devez avoir au moins 6 mois d'expérience pratique avec :

- **Claude Agent SDK** — orchestration multi-agents, délégation à des sous-agents, intégration d'outils, hooks de cycle de vie
- **Claude Code** — CLAUDE.md, serveurs MCP, Agent Skills, mode planification
- **Model Context Protocol (MCP)** — outils et ressources pour l'intégration backend
- **Ingénierie de prompts** — schémas JSON, exemples few-shot, modèles d'extraction de données
- **Fenêtres de contexte** — travail avec de longs documents, passage de contexte entre plusieurs agents
- **Pipelines CI/CD** — revue de code automatisée, génération de tests
- **Escalade et fiabilité** — gestion des erreurs, intervention humaine (human-in-the-loop)

Format de l'examen

Paramètre	Valeur
Type de question	Choix multiple (1 bonne réponse sur 4)
Notation	Échelle de 100 à 1000, note de réussite **720**

Pénalité pour les suppositions	Aucune (répondez à toutes les questions !)
Scénarios	4 sur 8 possibles (sélection aléatoire)

Contenu de l'examen : 5 domaines

Domaine	Pondération
1. Architecture et orchestration d'agents	**27 %**
2. Conception d'outils et intégration MCP	**18 %**
3. Configuration et workflows de Claude Code	**20 %**
4. Ingénierie de prompts et sortie structurée	**20 %**
5. Gestion du contexte et fiabilité	**15 %**

Scénarios de l'examen

Scénario 1 : Agent de support client

Vous construisez un agent pour gérer les retours, les litiges de facturation et les problèmes de compte à l'aide du Claude Agent SDK. L'agent utilise des outils MCP (`get_customer` , `lookup_order` , `process_refund` , `escalate_to_human`). L'objectif est une résolution au premier contact de plus de 80 % avec une escalade appropriée.

Scénario 2 : Génération de code avec Claude Code

Vous utilisez Claude Code pour accélérer le développement : génération de code, refactorisation, débogage, documentation. Vous devez l'intégrer avec des commandes slash personnalisées et la configuration CLAUDE.md, et comprendre quand utiliser le mode planification.

Scénario 3 : Système de recherche multi-agents

Un agent coordinateur délègue des tâches à des sous-agents spécialisés : recherche web, analyse de documents, synthèse et génération de rapports. Le système doit produire des rapports complets avec citations.

Scénario 4 : Outils de productivité pour développeurs

L'agent aide les ingénieurs à explorer des bases de code inconnues, à générer du code passe-partout (boilerplate) et à automatiser les tâches routinières. Des outils intégrés (Read, Write, Bash, Grep, Glob) et des serveurs MCP sont utilisés.

Scénario 5 : Claude Code pour l'intégration continue

Intégrer Claude Code dans un pipeline CI/CD pour des revues de code automatisées, la génération de tests et le retour sur les pull requests. Les prompts doivent être conçus pour minimiser les faux positifs.

Scénario 6 : Extraction de données structurées

Le système extrait des informations de documents non structurés, valide la sortie avec des schémas JSON et maintient une grande précision. Il doit gérer correctement les cas limites.

Scénario 7 : Modèles d'architecture d'IA conversationnelle

Vous concevez des systèmes conversationnels multi-tours couvrant la gestion de la fenêtre de contexte, la persistance des instructions au fil des tours, les stratégies de mémoire, la conception d'outils pour une exécution sûre, et la gestion d'entrées utilisateur ambiguës ou contradictoires.

Scénario 8 : Outils d'IA agentique *(contenu manquant — aidez-nous à le compléter !)*

Ce scénario a été signalé par des candidats à l'examen, mais n'est pas encore couvert dans ce guide. Si vous avez rencontré des questions de ce scénario lors du véritable examen, veuillez les partager dans les GitHub Issues afin que nous puissions ajouter une couverture complète. Votre contribution aidera tous ceux qui préparent l'examen.

Documentation officielle

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

Claude Code — CLAUDE.md and Memory	https://code.claude.com/docs/en/memory
Claude Code — Skills (incl. slash commands)	https://code.claude.com/docs/en/skills
Claude Code — Hooks	https://code.claude.com/docs/en/hooks
Claude Code — Sub-agents	https://code.claude.com/docs/en/sub-agents
Claude Code — MCP Integration	https://code.claude.com/docs/en/mcp
Claude Code — GitHub Actions CI/CD	https://code.claude.com/docs/en/github-actions
Claude Code — GitLab CI/CD	https://code.claude.com/docs/en/gitlab-ci-cd
Claude Code — Headless (mode non interactif)	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 (exemples de code)	https://github.com/anthropics/anthropic-cookbook

---

PARTIE I : FONDEMENTS THÉORIQUES

Cette partie couvre toute la théorie nécessaire pour réussir l'examen. Le matériel est organisé par technologies et concepts plutôt que par domaines d'examen — cela vous aide à construire une compréhension plus profonde de chaque sujet.

Chapitre 1 : Claude API — Fondamentaux de l'interaction avec le modèle

> Documentation : Messages API | Prompt Engineering

1.1 Structure d'une requête API

La Claude API suit un modèle requête–réponse. Chaque requête à la Claude Messages API comprend :

```
{
  "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"}
}
```

Champs clés :

- `model` — sélection du modèle (`claude-opus-4-6`, `claude-sonnet-4-6`, `claude-haiku-4-5`)
- `max_tokens` — nombre maximal de tokens dans la réponse
- `system` — le prompt système (définit le comportement du modèle)
- `messages` — historique de la conversation (**vous devez envoyer l'historique complet** pour maintenir la cohérence)
- `tools` — définitions des outils disponibles
- `tool_choice` — stratégie de sélection des outils

1.2 Rôles des messages

Le tableau `messages` utilise deux rôles conversationnels ainsi qu'un rôle d'instruction :

- `user` — messages de l'utilisateur, y compris les résultats d'outils (envoyés sous forme de bloc de contenu `tool_result` au sein d'un message de rôle `user`, et non comme un rôle `tool` distinct)
- `assistant` — réponses du modèle (incluses lors de l'envoi de l'historique), y compris les demandes d'utilisation d'outils (blocs de contenu `tool_use`)
- `system` — peut être défini via le champ `system` de premier niveau (s'applique dès le premier tour) ou de façon intercalée dans `messages` sous la forme `{"role": "system", ...}` (s'applique à partir de ce point, sous réserve de règles de placement précises — voir ci-dessous)

Les résultats d'outils ne sont pas envoyés comme un message de rôle `"tool"`. Ils sont envoyés comme un message de rôle `user` dont le contenu inclut un bloc de contenu `tool_result` :

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

`system` peut aussi apparaître directement comme rôle dans le tableau `messages`, pas seulement via le paramètre `system` de premier niveau. Cela permet d'ajouter des instructions en cours de conversation sans invalider le préfixe mis en cache du champ `system` de premier niveau. Ce mécanisme obéit à des règles de placement précises :

- Il doit immédiatement suivre un tour `user` (y compris un tour contenant des blocs `tool_result`) ou un tour `assistant` se terminant par une utilisation d'outil côté serveur.
- Il doit précéder un tour `assistant` ou terminer le tableau.
- Il ne peut pas se situer entre un bloc `tool_use` et son `tool_result` — cela renvoie une erreur 400.
- Les messages `system` ultérieurs (y compris ceux intercalés en cours de conversation) priment sur les précédents et sur le champ `system` de premier niveau pour les tours suivants.

Crucialement important : à chaque requête API, vous devez envoyer **l'historique complet de la conversation**. Le modèle ne conserve pas d'état entre les requêtes — chaque appel est indépendant.

1.3 Le champ `stop_reason` dans la réponse

La réponse de la Claude API inclut `stop_reason`, qui indique pourquoi le modèle a cessé de générer :

Valeur	Description	Action
`"end_turn"`	Le modèle a terminé sa réponse	Afficher le résultat à l'utilisateur
`"tool_use"`	Le modèle veut appeler un outil	Exécuter l'outil et renvoyer le résultat
`"max_tokens"`	Limite de tokens atteinte	La réponse est tronquée ; vous devrez peut-être augmenter la limite
`"stop_sequence"`	Une séquence d'arrêt a été rencontrée	Gérer selon la logique de votre application

Pour les systèmes agentiques, `"tool_use"` et `"end_turn"` sont les plus importants — ils contrôlent la boucle de l'agent.

1.4 Prompt système

Le prompt système est une instruction spéciale qui définit le contexte et les règles de comportement. Il :

- Ne fait pas partie du tableau `messages` ; il est transmis séparément dans le champ `system`
- A la priorité sur les messages de l'utilisateur
- Est chargé une fois et s'applique tout au long de la conversation
- Sert à définir le rôle, les contraintes et le format de sortie

Important pour l'examen : la formulation du prompt système peut créer des associations involontaires avec les outils. Par exemple, une instruction comme « toujours vérifier le client » peut amener le modèle à surutiliser `get_customer`, même lorsque c'est inutile.

1.5 Fenêtre de contexte

La fenêtre de contexte est la quantité totale de texte (en tokens) que le modèle peut traiter en une fois. Elle comprend :

- Le prompt système
- L'historique complet des messages
- Les définitions d'outils
- Les résultats d'outils

Problèmes clés de la fenêtre de contexte :

1. **Effet « perdu au milieu » (lost-in-the-middle) :** les modèles traitent de manière fiable les informations au début et à la fin d'une longue entrée, mais peuvent manquer des détails au milieu. Atténuation : placez les informations clés près du début ou de la fin.
2. **Accumulation des résultats d'outils :** chaque appel d'outil ajoute une sortie au contexte. Si un outil renvoie plus de 40 champs mais que seulement 5 importent, alors la majeure partie du contexte est gaspillée.
3. **Résumé progressif :** lors de la compression de l'historique, les valeurs numériques, les pourcentages et les dates sont souvent perdus et deviennent vagues (« environ », « à peu près », « quelques »).

Chapitre 2 : Outils et `tool_use`

> Documentation : Tool Use

2.1 Qu'est-ce que `tool_use`

`tool_use` est un mécanisme qui permet à Claude d'appeler des fonctions externes. Le modèle n'exécute pas le code directement — il génère une requête structurée d'appel d'outil ; votre code l'exécute et renvoie le résultat.

2.2 Définition d'un outil

Chaque outil est défini à l'aide d'un schéma JSON :

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

Aspects crucialement importants d'une description d'outil :

1. **La description est le principal mécanisme de sélection.** Un LLM choisit les outils en fonction de leurs descriptions. Des descriptions minimales (« Récupère les informations client ») entraînent des erreurs lorsque les outils se chevauchent.
2. **Incluez dans la description :**
   - Ce que l'outil fait et renvoie
   - Les formats d'entrée et des exemples de valeurs
   - Les cas limites et les contraintes
   - Quand utiliser cet outil par rapport à des alternatives similaires
3. **Évitez** les descriptions identiques ou qui se chevauchent entre les outils. Si `analyze_content` et `analyze_document` ont des descriptions presque identiques, le modèle les confondra.
4. **Outils intégrés vs outils MCP :** les agents peuvent préférer les outils intégrés (Read, Grep) aux outils MCP de fonctionnalité similaire. Pour éviter cela, renforcez les descriptions des outils MCP — mettez en avant des avantages concrets, des données uniques ou un contexte que les outils intégrés ne peuvent pas fournir.

2.3 Le paramètre `tool_choice`

`tool_choice` contrôle la façon dont le modèle sélectionne les outils :

Valeur	Comportement	Quand l'utiliser
`{"type": "auto"}`	Le modèle décide d'appeler un outil ou de répondre en texte	Par défaut dans la plupart des cas

`{"type": "any"}`	Le modèle **doit** appeler un outil quelconque	Lorsque vous avez besoin d'une sortie structurée garantie
`{"type": "tool", "name": "extract_metadata"}`	Le modèle **doit** appeler un outil précis	Lorsque vous avez besoin d'une première étape forcée / d'un ordre d'exécution

Scénarios importants :

- `tool_choice: "any"` + plusieurs outils d'extraction → le modèle choisit le meilleur, mais vous obtenez tout de même une sortie structurée
- Sélection forcée → lorsque vous devez garantir une première action précise (p. ex. `extract_metadata` avant l'enrichissement)

2.4 Schémas JSON pour la sortie structurée

Utiliser `tool_use` avec des schémas JSON est le moyen **le plus fiable** d'obtenir une sortie structurée de Claude. Cela :

- Garantit un JSON syntaxiquement valide (pas d'accolades manquantes, pas de virgules superflues)
- Impose la structure requise (les champs requis sont présents)
- Ne garantit **pas** l'exactitude sémantique (les valeurs peuvent quand même être erronées)

Conception de schéma — principes clés :

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

Règles de conception de schéma :

1. **Requis vs optionnel :** marquez les champs comme requis uniquement si l'information est toujours disponible. Les champs requis poussent le modèle à inventer des valeurs lorsque les données manquent.
2. **Champs nullable :** utilisez `"type": ["string", "null"]` pour les informations qui peuvent être absentes. Le modèle peut renvoyer `null` au lieu d'halluciner.
3. **Enums avec `"other"` :** ajoutez `"other"` + une chaîne de détail pour éviter de perdre des données en dehors de vos catégories prédéfinies.
4. **Enum `"unclear"` :** pour les cas où le modèle ne peut pas choisir une catégorie avec confiance — un `"unclear"` honnête vaut mieux qu'une catégorie erronée.

2.5 Erreurs syntaxiques vs sémantiques

Type d'erreur	Exemple	Atténuation
Syntaxique	JSON invalide, mauvais type de champ	`tool_use` avec un schéma JSON (élimine)
Sémantique	Les totaux ne correspondent pas, valeur dans le mauvais champ, hallucination	Contrôles de validation, nouvelle tentative avec feedback, auto-correction

Chapitre 3 : Claude Agent SDK — Construire des systèmes agentiques

> Documentation : Agent SDK | Hooks | Subagents | Sessions

3.1 Qu'est-ce qu'une boucle agentique

La boucle agentique est le motif central de l'exécution autonome de tâches. Le modèle ne se contente pas de répondre — il réalise une séquence d'actions :

```
1. Envoyer une requête à Claude avec des outils
2. Recevoir une réponse
3. Vérifier stop_reason :
   - "tool_use" -> exécuter l'outil, ajouter le résultat à l'historique, revenir à
l'étape 1
   - "end_turn" -> la tâche est terminée, afficher le résultat à l'utilisateur
4. Répéter jusqu'à l'achèvement
```

C'est une approche pilotée par le modèle : Claude décide quel outil appeler ensuite en fonction du contexte et des résultats d'outils précédents. Cela diffère des arbres de décision codés en dur où la séquence d'actions est fixe.

Anti-modèles (à éviter) :

- Analyser le texte de l'assistant pour détecter l'achèvement (« Tâche terminée »)
- Utiliser une limite d'itérations arbitraire (p. ex. `max_iterations=5`) comme condition d'arrêt principale
- Vérifier si l'assistant a produit du contenu textuel comme signal d'achèvement

Approche correcte : le seul signal d'achèvement fiable est `stop_reason == "end_turn"`.

3.2 Configuration d'`AgentDefinition`

`AgentDefinition` est l'objet de configuration de l'agent dans le Claude Agent SDK :

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

Paramètres clés :

- `name` / `description` — identification et description de l'agent
- `system_prompt` — prompt système avec instructions
- `allowed_tools` — liste des outils autorisés (principe du moindre privilège)

3.3 Hub-and-spoke : coordinateur et sous-agents

Une architecture multi-agents est généralement construite selon une topologie en étoile (hub-and-spoke) :

```
        Coordinateur
       /     |      \
  Sous-agent1  Sous-agent2  Sous-agent3
   (recherche)   (analyse)   (synthèse)
```

Le coordinateur est responsable de :

- Décomposer la tâche en sous-tâches
- Décider quels sous-agents sont nécessaires (sélection dynamique)
- Déléguer le travail aux sous-agents
- Agréger et valider les résultats
- Gérer les erreurs et les nouvelles tentatives
- Communiquer les résultats à l'utilisateur

Principe critique : les sous-agents ont un contexte isolé.

- Les sous-agents n'héritent **pas** automatiquement de l'historique de conversation du coordinateur
- Tout le contexte requis doit être **transmis explicitement** dans le prompt du sous-agent
- Les sous-agents ne partagent pas de mémoire entre les appels
- Toute communication passe par le coordinateur (pour l'observabilité et le contrôle des erreurs)

3.4 L'outil `Task` pour engendrer des sous-agents

Les sous-agents sont engendrés via l'outil `Task` :

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

Le passage explicite du contexte est obligatoire :

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

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

Engendrement parallèle : un coordinateur peut appeler plusieurs `Task` dans une seule réponse — les sous-agents s'exécutent en parallèle :

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

3.5 Les hooks dans l'Agent SDK

Les hooks permettent l'interception et la transformation à des points précis du cycle de vie de l'agent.

PostToolUse intercepte un résultat d'outil avant qu'il ne soit fourni au modèle :

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

Le hook d'interception des appels sortants bloque les actions qui violent une politique :

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

Différence clé : hooks vs instructions de prompt

Attribut	Hooks	Instructions de prompt
Garantie	**Déterministe** (100 %)	**Probabiliste** (>90 %, pas 100 %)
Quand l'utiliser	Règles métier critiques, opérations financières, conformité	Préférences générales, recommandations, formatage

Exemple	Bloquer les remboursements > 500 $	« Essaie de résoudre avant d'escalader »

Règle : lorsqu'un échec a des conséquences financières, légales ou de sécurité — utilisez des hooks, pas des prompts.

Chapitre 4 : Model Context Protocol (MCP)

> Documentation : MCP | Tools | Resources | Servers

4.1 Qu'est-ce que MCP

Le Model Context Protocol (MCP) est un protocole ouvert pour connecter des systèmes externes à Claude. MCP définit trois types principaux de ressources :

1. **Tools (outils)** — fonctions que l'agent peut appeler pour effectuer des actions (opérations CRUD, appels d'API, exécution de commandes)
2. **Resources (ressources)** — données que l'agent peut lire pour le contexte (documentation, schémas de base de données, catalogues de contenu)
3. **Prompts** — modèles de prompts prédéfinis pour les tâches courantes

4.2 Serveurs MCP

Un serveur MCP est un processus qui implémente le protocole MCP et fournit des outils/ressources. Lorsque vous vous connectez à un serveur MCP :

- Tous les outils sont découverts automatiquement
- Les outils de tous les serveurs connectés sont disponibles à la fois
- Les descriptions des outils déterminent comment le modèle les utilisera

4.3 Configuration des serveurs MCP

Configuration de projet (`.mcp.json`) — pour une utilisation en équipe :

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

Points clés :

- `.mcp.json` est stocké à la racine du projet et géré dans le contrôle de version
- Les variables d'environnement (`${GITHUB_TOKEN}`) sont utilisées pour les secrets — les tokens eux-mêmes ne sont pas commités
- Disponible pour tous les contributeurs du projet

Configuration utilisateur (`~/.claude.json`) — pour les serveurs personnels/expérimentaux :

- Stockée dans le répertoire personnel de l'utilisateur
- Non partagée via le contrôle de version
- Adaptée aux expériences personnelles et aux tests

Choisir les serveurs :

- Pour les intégrations standard (Jira, GitHub, Slack), préférez les serveurs MCP communautaires existants
- Ne construisez vos propres serveurs que pour des workflows uniques, spécifiques à votre équipe

4.4 Le drapeau `isError` dans MCP

Lorsqu'un outil MCP rencontre une erreur, il utilise `isError: true` dans la réponse. Cela signale à l'agent que l'appel a échoué.

Erreur structurée (bonne pratique) :

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

Erreur générique (anti-modèle) :

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

Une erreur générique ne donne à l'agent aucune information pour la prise de décision — doit-il réessayer, modifier la requête ou escalader ?

4.5 Ressources MCP

Les ressources sont des données qu'un agent peut demander pour obtenir du contexte sans entreprendre d'actions :

- Catalogues de contenu (p. ex. une liste de toutes les tâches du projet, navigation hiérarchique)
- Schémas de base de données (compréhension de la structure des données)
- Documentation (références d'API, guides internes)
- Résumés de tickets/tâches

Avantage des ressources : l'agent n'a pas besoin d'appels d'outils exploratoires pour comprendre quelles données existent. Une ressource fournit immédiatement une « carte ».

Chapitre 5 : Claude Code — Configuration et workflows

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

5.1 La hiérarchie CLAUDE.md

CLAUDE.md est le ou les fichiers d'instructions pour Claude Code. Il existe une hiérarchie à trois niveaux :

```
1. Niveau utilisateur : ~/.claude/CLAUDE.md
   - Ne s'applique qu'à cet utilisateur
   - PAS partagé via le VCS
   - Préférences personnelles et style de travail

2. Niveau projet : .claude/CLAUDE.md ou un CLAUDE.md racine
   - S'applique à tous les contributeurs du projet
   - Géré via le VCS
   - Normes de codage, normes de test, décisions architecturales

3. Niveau répertoire : CLAUDE.md dans les sous-répertoires
   - S'applique lors du travail sur les fichiers de ce répertoire
   - Conventions propres à cette partie de la base de code
```

Erreur courante : un nouveau membre de l'équipe ne reçoit pas les instructions du projet parce qu'elles ont été placées dans `~/.claude/CLAUDE.md` (niveau utilisateur) au lieu de `.claude/CLAUDE.md` (niveau projet).

5.2 Syntaxe `@path` (imports de fichiers)

CLAUDE.md peut référencer des fichiers externes à l'aide de `@path`, rendant la configuration modulaire :

```
# Project CLAUDE.md

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

Règles pour `@path` :

- Utilisez `@` immédiatement avant le chemin du fichier (sans espace)
- Les chemins relatifs et absolus sont pris en charge
- Les chemins relatifs sont résolus par rapport au fichier qui contient l'import
- La profondeur maximale d'imbrication des imports est de 5

Cela évite la duplication et permet à chaque package d'inclure uniquement les normes pertinentes.

5.3 Le répertoire `.claude/rules/`

`.claude/rules/` est une alternative à un CLAUDE.md monolithique, utilisée pour organiser les règles par thème :

```
.claude/rules/
  testing.md          -- conventions de test
  api-conventions.md  -- conventions d'API
  deployment.md       -- règles de déploiement
  react-patterns.md   -- patterns React
```

Fonctionnalité clé : frontmatter YAML avec `paths` pour le chargement conditionnel :

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

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

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

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

Comment cela fonctionne :

- Une règle est chargée **uniquement** lorsque Claude Code édite un fichier correspondant au motif `paths`
- Cela économise du contexte et des tokens — les règles non pertinentes ne sont pas chargées
- Les motifs glob permettent d'appliquer des conventions par type de fichier, quel que soit leur emplacement (idéal pour les tests dispersés dans la base de code)

Quand utiliser `.claude/rules/` avec `paths` vs CLAUDE.md au niveau répertoire :

- `.claude/rules/` avec `paths` — lorsque les conventions s'appliquent à des fichiers répartis dans de nombreux répertoires (tests, migrations)
- CLAUDE.md au niveau répertoire — lorsque les conventions sont liées à un répertoire précis et ne sont pas nécessaires ailleurs

5.4 Commandes slash personnalisées et Skills

> **Remarque :** dans la version actuelle de Claude Code, les commandes personnalisées (`.claude/commands/`) sont unifiées avec les skills (`.claude/skills/`). Les deux formats créent des commandes `/name`. Le guide de l'examen fait référence à `.claude/commands/` — ce format est toujours pris en charge.

Les commandes slash sont des modèles de prompts réutilisables invoqués via `/name` :

Format `.claude/commands/` (hérité, pris en charge) :

```
.claude/commands/
  review.md        -- /review -- revue de code standard
  test-gen.md      -- /test-gen -- génération de tests
```

Format `.claude/skills/` (actuel) :

```
.claude/skills/
  review/SKILL.md  -- /review -- avec configuration frontmatter
  test-gen/SKILL.md
```

Commandes de projet (`.claude/commands/` ou `.claude/skills/`) :

- Stockées dans le VCS et disponibles pour tous lors du clonage du dépôt
- Garantissent des workflows cohérents au sein de l'équipe

Commandes utilisateur (`~/.claude/commands/` ou `~/.claude/skills/`) :

- Commandes personnelles non partagées via le VCS
- Pour les workflows individuels

5.5 Skills — `.claude/skills/`

Les skills sont des commandes avancées configurées via le frontmatter de SKILL.md :

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

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

Paramètres du frontmatter :

Paramètre	Description
`context: fork`	Exécute le skill dans un sous-agent isolé. La sortie verbeuse ne pollue pas la session principale
`allowed-tools`	Restreint les outils disponibles (sécurité — p. ex. le skill ne peut pas supprimer de fichiers s'il n'y est pas autorisé)
`argument-hint`	Indice qui demande un argument lorsqu'il est invoqué sans paramètres

Quand utiliser un skill vs CLAUDE.md :

- **Skill** — invocation à la demande pour une tâche précise (revue, analyse, génération)

- **CLAUDE.md** — normes et conventions générales toujours chargées

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

- Créez des variantes personnelles sous des noms différents pour ne pas affecter vos coéquipiers

5.6 Mode planification vs exécution directe

Mode planification :

- Le modèle investigue et planifie uniquement ; il ne fait pas de modifications
- Utilise Read, Grep, Glob pour explorer la base de code
- Produit un plan d'implémentation que l'utilisateur approuve
- Exploration sûre, sans effets de bord

Quand utiliser le mode planification :

- Changements importants (des dizaines de fichiers)
- Plusieurs approches plausibles (microservices : comment définir les frontières ?)
- Décisions architecturales (quel framework ? quelle structure ?)
- Base de code inconnue (vous devez comprendre avant de modifier)
- Migrations de bibliothèques affectant plus de 45 fichiers

Quand utiliser l'exécution directe :

- Corrections sur un seul fichier avec une trace de pile claire
- Ajout d'un seul contrôle de validation
- Changements bien compris et sans ambiguïté

Approche combinée :

1. Mode planification pour l'investigation et la conception
2. L'utilisateur approuve le plan
3. Exécution directe pour implémenter le plan approuvé

Sous-agent Explore — un sous-agent spécialisé pour explorer la base de code :

- Isole la sortie verbeuse du contexte principal
- Ne renvoie qu'un résumé
- Évite l'épuisement de la fenêtre de contexte dans les tâches multi-phases

5.7 La commande `/compact`

`/compact` est une commande intégrée pour compresser le contexte :

- Résume l'historique antérieur pour libérer la fenêtre de contexte
- Utilisée dans les longues sessions d'investigation lorsque le contexte se remplit de sorties d'outils verbeuses

- Risque : les valeurs numériques exactes, les dates et les détails spécifiques peuvent être perdus lors du résumé

5.8 La commande `/memory`

`/memory` est une commande intégrée pour gérer la mémoire entre les sessions :

- Ouvre le fichier `CLAUDE.md` pour édition, vous permettant d'enregistrer des notes, des préférences et du contexte
- Les informations persistent entre les sessions et sont automatiquement chargées au démarrage
- Utile pour stocker les conventions du projet, les préférences de l'utilisateur, les commandes fréquemment utilisées et le contexte de travail actuel
- Alternative au fait de réexpliquer les mêmes instructions à chaque session

5.9 Claude Code CLI pour la CI/CD

Le drapeau `-p` (ou `--print`) :

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

- Mode non interactif : traite le prompt, l'affiche sur stdout, puis quitte
- N'attend pas d'entrée de l'utilisateur
- La seule façon correcte d'exécuter Claude dans les pipelines CI/CD

Sortie structurée pour la CI :

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

- `--output-format json` — sortie en JSON
- `--json-schema` — valide la sortie par rapport à un schéma
- Le résultat peut être analysé pour publier automatiquement des commentaires inline sur la PR

Isolation du contexte de session : La même session Claude qui a généré le code est souvent moins efficace pour le réviser (le modèle conserve son contexte de raisonnement et est moins enclin à remettre en question ses propres décisions). Utilisez une instance indépendante pour la revue.

Éviter les commentaires en double : Lors d'une nouvelle revue après de nouveaux commits, incluez les résultats de revue antérieurs dans le contexte et demandez à Claude de ne signaler que les problèmes nouveaux ou non résolus.

5.10 `fork_session` et gestion des sessions

`--resume <session-name>` reprend une session nommée :

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

- Poursuit une conversation antérieure avec le contexte sauvegardé
- Utile pour les longues investigations sur plusieurs sessions
- Risque : si des fichiers ont changé depuis la session précédente, les résultats d'outils peuvent être obsolètes

`fork_session` crée une branche indépendante à partir d'un contexte partagé :

```
Investigation de la base de code
         |
    fork_session
    /           \
Approche A :      Approche B :
Redux            Context API
```

- Les deux forks héritent du contexte jusqu'au point de branchement
- Ensuite, ils divergent indépendamment
- Utile pour comparer des approches ou tester des stratégies

Quand démarrer une nouvelle session au lieu de reprendre :

- Les résultats d'outils sont obsolètes (les fichiers ont changé)
- Beaucoup de temps s'est écoulé et le contexte s'est dégradé
- Il vaut mieux redémarrer avec « Voici un bref résumé de ce que nous avons trouvé : ... » que de reprendre avec d'anciennes données d'outils

---

Chapitre 6 : Ingénierie de prompts — Techniques avancées

> Documentation : Prompt Engineering | Anthropic Cookbook

6.1 Prompting few-shot

Le prompting few-shot consiste à inclure 2 à 4 exemples entrée/sortie dans un prompt pour démontrer le comportement attendu.

Pourquoi le few-shot est plus efficace que les descriptions textuelles :

- Une instruction vague comme « sois plus précis » peut être interprétée de nombreuses façons
- Un exemple montre sans ambiguïté le format attendu et la logique de décision
- Le modèle généralise le motif à de nouveaux cas (il ne se contente pas de répéter les exemples)

Types d'exemples few-shot et quand les utiliser :

1. **Exemples pour les scénarios ambigus :**

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

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

2. **Exemples pour le formatage de la sortie :**

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

3. **Exemples pour distinguer le code acceptable du code problématique :**

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

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

4. **Exemples d'extraction à partir de différents formats de documents :**

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

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

5. **Exemples pour les mesures informelles :**

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

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

Le few-shot est particulièrement efficace pour extraire des unités de mesure informelles et non standard, trop variées pour des instructions purement basées sur des règles.

Règles de normalisation de format dans les prompts : Lorsque vous utilisez des schémas JSON stricts pour une sortie structurée, ajoutez des règles de normalisation dans le prompt :

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

Cela évite les erreurs sémantiques où le JSON est syntaxiquement valide mais les valeurs sont incohérentes.

6.2 Critères explicites vs instructions vagues

Mauvais (vague) :

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

Bon (critères explicites) :

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

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

Définissez des critères de gravité avec des exemples :

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

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

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

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

6.3 Chaînage de prompts (prompt chaining)

Le chaînage de prompts décompose une tâche complexe en une séquence d'étapes ciblées :

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

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

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

Pourquoi c'est important :

- Évite la **dilution de l'attention** — lorsque le modèle reçoit trop de fichiers à la fois, il peut manquer des bugs dans certains fichiers tout en fournissant des commentaires superficiels sur d'autres
- Garantit une qualité d'analyse cohérente par fichier
- Permet d'analyser séparément les interactions entre fichiers

Quand utiliser le chaînage de prompts vs la décomposition dynamique :

- **Chaînage de prompts** — tâches prévisibles et reproductibles (revue de code, migrations de fichiers)
- **Décomposition dynamique** — investigations ouvertes où les sous-tâches ne deviennent claires que pendant l'exécution

6.4 Le modèle « entretien » (Interview)

Avant d'implémenter une solution, Claude pose des questions de clarification :

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

Quand c'est utile :

- Domaine inconnu (fintech, santé, systèmes juridiques)
- Tâches aux implications non évidentes (stratégies de cache, modes de défaillance)
- Plusieurs approches viables où le meilleur choix dépend du contexte

6.5 Validation et nouvelle tentative avec feedback (retry-with-feedback)

Lorsque des données extraites échouent à la validation :

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

Quand la nouvelle tentative sera efficace :

- Erreurs de format (date dans le mauvais format)
- Erreurs de structure (un champ placé au mauvais endroit)
- Incohérences arithmétiques (le modèle peut revérifier)

Quand la nouvelle tentative N'AIDERA PAS :

- L'information est absente du document source
- Le contexte requis est externe (les données sont dans un autre document non fourni)

Pydantic comme outil de validation : Pydantic est une bibliothèque Python pour la validation de données basée sur un schéma. Pour l'examen, les points clés sont :

- **Validation structurelle :** types, obligation, contraintes d'enum vérifiés dans le code après réception du JSON de Claude
- **Validation sémantique :** des validateurs personnalisés imposent la logique métier (la somme des éléments est égale au total ; start_date < end_date)
- **Boucles validation–retry :** en cas d'échec de validation Pydantic, construisez un message d'erreur et re-promptez Claude avec le contexte de l'erreur
- **Génération de JSON Schema :** les modèles Pydantic peuvent générer un JSON Schema pour `tool_use`, fournissant une source unique de vérité

6.6 Auto-correction

Un motif pour détecter les contradictions internes :

```
{
  "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}
  ]
}
```

Le modèle extrait à la fois la valeur déclarée et une valeur calculée — si elles diffèrent, `conflict_detected` vous permet de gérer l'écart.

Chapitre 7 : Message Batches API

> Documentation : Message Batches

7.1 Vue d'ensemble

La Message Batches API vous permet de soumettre des lots de requêtes pour un traitement asynchrone :

Attribut	Valeur
Économies	**50 %** par rapport aux appels synchrones
Fenêtre de traitement	Jusqu'à **24 heures** (aucune garantie de SLA de latence)
Appels d'outils multi-tours	**Non pris en charge** (une requête = une réponse)
Corrélation	Champ `custom_id` pour relier requête et réponse

7.2 Quand utiliser la Batch API vs l'API synchrone

Tâche	API	Pourquoi
Vérification de PR avant fusion	**Synchrone**	Le développeur attend ; 24 heures sont inacceptables
Rapport de dette technique de nuit	**Batch**	Le résultat est nécessaire le matin ; 50 % d'économies
Audit de sécurité hebdomadaire	**Batch**	Non urgent ; 50 % d'économies
Revue de code interactive	**Synchrone**	Réponse immédiate requise
Traitement de 10 000 documents	**Batch**	Traitement en masse ; les économies sont importantes

7.3 Utilisation de `custom_id`

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

`custom_id` vous permet de :

- Relier le résultat au document d'origine

- En cas d'échec, ne re-soumettre que les documents échoués
- Éviter de retraiter les documents réussis

7.4 Gestion des échecs dans les lots

1. Soumettre un lot de 100 documents
2. 95 réussissent ; 5 échouent (limite de contexte dépassée)
3. Identifier les échecs par `custom_id`
4. Modifier la stratégie (p. ex. découper les longs documents en morceaux)
5. Re-soumettre uniquement les 5 documents échoués

7.5 Planification du SLA

Si vous avez besoin d'un résultat dans 30 heures et que la Batch API peut prendre jusqu'à 24 heures :

- Fenêtre de soumission : 30 - 24 = **6 heures**
- Les lots doivent être soumis au plus tard 24 heures avant l'échéance
- Pour des soumissions fréquentes, découpez en fenêtres de 4 heures

---

Chapitre 8 : Stratégies de décomposition des tâches

8.1 Pipelines fixes (chaînage de prompts)

Chaque étape est définie à l'avance :

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

Quand l'utiliser :

- La structure de la tâche est prévisible (les revues suivent toujours le même modèle)
- Toutes les étapes sont connues d'avance
- Vous avez besoin de stabilité et de reproductibilité

8.2 Décomposition adaptative dynamique

Les sous-tâches sont générées en fonction des résultats intermédiaires :

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

Quand l'utiliser :

- Tâches d'investigation ouvertes
- Lorsque la portée complète est inconnue d'avance
- Lorsque chaque étape dépend des résultats de la précédente

8.3 Revue de code multi-passes

Pour les pull requests de plus de 10 fichiers :

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

Pourquoi une seule passe sur 14 fichiers est mauvaise :

- Dilution de l'attention : analyse approfondie pour certains fichiers, superficielle pour d'autres
- Commentaires incohérents : un motif est signalé dans un fichier mais approuvé dans un autre
- Bugs manqués : des erreurs évidentes sont ignorées en raison de la surcharge cognitive

---

Chapitre 9 : Escalade et intervention humaine (Human-in-the-Loop)

9.1 Quand escalader vers un humain

Déclencheurs d'escalade (règles claires) :

Situation	Action
Le client demande explicitement « passez-moi un responsable »	Escalader immédiatement ; ne pas tenter de résoudre
La politique ne couvre pas la demande	Escalader (p. ex. alignement sur le prix d'un concurrent lorsque la politique est muette)
L'agent ne peut pas progresser	Escalader après un nombre raisonnable de tentatives

Opération financière au-dessus d'un seuil	Escalader (de préférence imposé via un hook, pas un prompt)
Plusieurs correspondances lors de la recherche d'un client	Demander des identifiants supplémentaires ; ne pas deviner

Ce qui N'EST PAS un déclencheur fiable :

Méthode non fiable	Pourquoi elle échoue
Analyse de sentiment	L'humeur du client ne corrèle pas avec la complexité du cas
Confiance auto-évaluée par le modèle (1–10)	Le modèle peut se tromper avec assurance ; le calibrage est médiocre
Un classificateur automatique	Sur-ingénierie ; peut nécessiter des données d'entraînement que vous n'avez pas

9.2 Modèles d'escalade

Escalade immédiate :

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

Escalade après une tentative de résolution :

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

Escalade nuancée (reconnaître → résoudre → escalader si réitération) :

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

Principe clé : reconnaissez d'abord l'émotion, puis proposez une solution concrète, et n'escaladez que si le client réitère son souhait de parler à un humain. N'escaladez pas à la première expression d'insatisfaction (ce n'est pas la même chose que demander un responsable).

Escalade pour une lacune de politique :

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

9.3 Protocoles de transfert structurés (handoff)

Lors de l'escalade, l'agent doit transmettre un résumé structuré à un humain :

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

L'opérateur humain n'a pas accès à la transcription complète de la conversation — il ne voit que ce résumé. Il doit donc être complet et autonome.

9.4 Calibrage de la confiance et supervision humaine

Pour les systèmes d'extraction de données :

1. **Scores de confiance au niveau du champ :** le modèle produit un score de confiance par champ extrait
2. **Calibrage :** utilisez des ensembles de validation étiquetés pour ajuster les seuils
3. **Routage :**
   - Confiance élevée + précision stable -> traitement automatisé
   - Confiance faible ou sources ambiguës -> revue humaine

Échantillonnage aléatoire stratifié :

- Même pour les extractions à confiance élevée, auditez régulièrement un échantillon
- Une précision globale de 97 % peut masquer 40 % d'erreurs pour un type de document particulier
- Analysez la précision par type de document et par champ, pas seulement globalement

---

Chapitre 10 : Gestion des erreurs dans les systèmes multi-agents

10.1 Catégories d'erreurs

Catégorie	Exemples	Réessayable	Action de l'agent
Transitoire	Timeout, 503, panne réseau	Oui	Réessayer avec backoff exponentiel
Validation	Format d'entrée invalide, champ requis manquant	Non (corriger l'entrée)	Modifier la requête et réessayer
Métier	Violation de politique, seuil dépassé	Non	Expliquer à l'utilisateur ; proposer une alternative
Permission	Accès refusé	Non	Escalader

10.2 Anti-modèles de gestion des erreurs

Anti-modèle	Problème	Approche correcte
Statut générique « recherche indisponible »	Le coordinateur ne peut pas décider comment récupérer	Renvoyer le type d'erreur, la requête, les résultats partiels, les alternatives
Suppression silencieuse (résultat vide = succès)	Le coordinateur pense qu'il n'y avait pas de correspondance, mais c'était un échec	Distinguer « aucun résultat » de « échec de recherche »
Abandonner tout le workflow sur un seul échec	Vous perdez tous les résultats partiels	Continuer avec les résultats partiels ; annoter les lacunes
Nouvelles tentatives infinies dans un sous-agent	Latence et gaspillage de ressources	Récupération locale (1–2 tentatives), puis propager au coordinateur

10.3 Une erreur structurée de sous-agent

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

Cela fournit au coordinateur les informations nécessaires pour décider :

- Réessayer avec une requête modifiée ?
- Utiliser les résultats partiels ?
- Déléguer à un autre sous-agent ?
- Continuer sans cette section et annoter la lacune ?

10.4 Annotations de couverture dans la synthèse finale

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

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

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

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

Chapitre 11 : Gestion du contexte dans les systèmes de production

11.1 Extraire les faits dans un bloc séparé

Au lieu de vous fier à l'historique de la conversation (qui se dégrade lors du résumé), extrayez les faits clés dans un bloc structuré :

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

Incluez ce bloc dans chaque prompt, indépendamment de la façon dont l'historique est résumé.

11.2 Réduction des résultats d'outils

Si `lookup_order` renvoie plus de 40 champs mais que vous n'en avez besoin que de 5 pour la tâche en cours :

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

Cela préserve le contexte et réduit le bruit.

11.3 Entrée tenant compte de la position

Placez les informations critiques en tenant compte de l'effet « perdu au milieu » :

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

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

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

11.4 Fichiers bloc-notes (scratchpad)

Dans les longues investigations, l'agent peut écrire les conclusions clés dans un fichier bloc-notes :

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

Lorsque le contexte se dégrade (ou dans une nouvelle session), l'agent peut consulter le bloc-notes au lieu de relancer la découverte.

11.5 Déléguer aux sous-agents pour protéger le contexte

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

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

Couche de contexte séparée : Dans les systèmes multi-agents, chaque sous-agent opère dans un budget de contexte limité — il ne reçoit que les informations nécessaires à sa tâche. Le coordinateur agit comme une couche de contexte distincte : il agrège les sorties des sous-agents, stocke l'état global et alloue le contexte. Cela évite la « fuite de contexte », où un agent consomme la fenêtre avec des informations non pertinentes pour les autres.

Budgets de contexte contraints pour les sous-agents :

- Envoyez un contexte minimal : une tâche précise + les données nécessaires
- Demandez au sous-agent de renvoyer des résultats structurés, pas des décharges de données brutes
- Utilisez `allowedTools` pour limiter l'ensemble d'outils du sous-agent — moins d'outils signifie moins de distractions et un coût de contexte plus faible

11.6 Persistance d'état structurée (pour la reprise après panne)

Chaque agent exporte son état vers un emplacement connu :

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

Le coordinateur charge un manifeste lors de la reprise :

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

Chapitre 12 : Préserver la provenance

12.1 Le problème de la perte d'attribution

Lors du résumé de résultats provenant de plusieurs sources, le lien « affirmation → source » peut être perdu :

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

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

12.2 Gestion des données contradictoires

Lorsque deux sources fournissent des valeurs différentes :

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

Ne choisissez pas arbitrairement une valeur. Préservez les deux avec leur attribution et laissez le coordinateur décider.

12.3 Inclure les dates pour une interprétation correcte

Sans dates, les différences temporelles peuvent être interprétées à tort comme des contradictions :

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

12.4 Rendu par type de contenu

Ne forcez pas tout dans un seul format :

- Données financières -> tableaux
- Actualités et analyses -> prose
- Conclusions techniques -> listes structurées
- Séries temporelles -> ordre chronologique

Chapitre 13 : Outils intégrés de Claude Code

13.1 Référence de sélection d'outils

Tâche	Outil	Exemple
Trouver des fichiers par nom/motif	**Glob**	`**/*.test.tsx`, `src/components/**/*.ts`
Rechercher dans les fichiers	**Grep**	Nom de fonction, message d'erreur, import
Lire un fichier en entier	**Read**	Charger un fichier pour analyse
Écrire un nouveau fichier	**Write**	Créer un fichier de zéro
Éditer précisément un fichier existant	**Edit**	Remplacer un extrait précis via une correspondance de texte unique
Exécuter une commande shell	**Bash**	git, npm, exécuter des tests, build

13.2 Stratégie d'investigation incrémentale

Ne lisez pas tous les fichiers à la fois. Construisez votre compréhension de manière incrémentale :

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

13.3 Repli : Read + Write au lieu d'Edit

Lorsque Edit échoue en raison d'une correspondance de texte non unique :

1. Read — charger le contenu complet du fichier
2. Modifier le contenu par programmation
3. Write — écrire la version mise à jour

PARTIE II : NOTES SUR LES DOMAINES DE L'EXAMEN

Domaine 1 : Architecture et orchestration d'agents (27 %)

1.1 Concevoir des boucles agentiques pour l'exécution autonome de tâches

Connaissances clés :

- Cycle de vie de la boucle d'agent : envoyer une requête à Claude, vérifier `stop_reason` (`"tool_use"` vs `"end_turn"`), exécuter les outils, renvoyer les résultats pour l'itération suivante
- Les résultats d'outils sont ajoutés à l'historique de la conversation pour que le modèle puisse décider de l'action suivante
- Prise de décision pilotée par le modèle (Claude choisit l'outil suivant) vs arbres de décision codés en dur

Compétences clés :

- Contrôle de flux : poursuivre la boucle lorsque `stop_reason = "tool_use"` et s'arrêter sur `"end_turn"`
- Ajout des résultats d'outils au contexte entre les itérations
- Anti-modèles à éviter : analyser le texte de l'assistant pour détecter l'achèvement, utiliser des limites d'itérations arbitraires comme mécanisme d'arrêt principal

1.2 Orchestrer des systèmes multi-agents (coordinateur–sous-agent)

Connaissances clés :

- Architecture en étoile (hub-and-spoke) : le coordinateur possède toute la communication inter-agents, la gestion des erreurs et le routage
- Les sous-agents opèrent avec un contexte isolé — ils n'héritent pas automatiquement de l'historique du coordinateur
- Responsabilités du coordinateur : décomposition des tâches, délégation, agrégation des résultats, sélection dynamique des sous-agents
- Risque d'une décomposition trop étroite par le coordinateur

Compétences clés :

- Répartir la couverture de recherche entre les sous-agents pour minimiser la duplication
- Implémenter des boucles de raffinement itératif (le coordinateur évalue la synthèse et redistribue les tâches)
- Faire passer toute la communication par le coordinateur pour l'observabilité

1.3 Configurer les appels de sous-agents, le passage de contexte et l'engendrement

Connaissances clés :

- L'outil `Task` engendre des sous-agents ; les `allowedTools` du coordinateur doivent inclure `"Task"`
- Le contexte du sous-agent doit être inclus explicitement dans le prompt ; les sous-agents n'héritent pas du contexte parent
- Configuration d'`AgentDefinition` : descriptions, prompts système, contraintes d'outils
- Gestion des sessions via `fork_session` pour explorer des alternatives

Compétences clés :

- Inclure les sorties complètes des agents précédents dans le prompt du sous-agent
- Utiliser des formats structurés pour séparer les données des métadonnées lors du passage du contexte
- Engendrer des sous-agents parallèles via plusieurs appels `Task` dans un seul tour du coordinateur
- Rédiger les prompts du coordinateur en termes d'objectifs et de critères de qualité plutôt que d'instructions étape par étape

1.4 Implémenter des workflows multi-étapes avec des modèles d'application (enforcement) et de transfert (handoff)

Connaissances clés :

- La différence entre **l'application programmatique** (hooks, préconditions) et **le guidage par prompt** pour ordonner un workflow
- Lorsque vous avez besoin de garanties déterministes (p. ex. vérification d'identité avant des opérations financières), les prompts seuls sont insuffisants
- Protocoles de transfert structurés lors de l'escalade (ID client, motif, action recommandée)

Compétences clés :

- Préconditions programmatiques qui bloquent les appels en aval jusqu'à ce que les étapes antérieures soient terminées (p. ex. bloquer `process_refund` jusqu'à ce que `get_customer` renvoie un ID vérifié)
- Décomposer les demandes clients multi-aspects en éléments distincts
- Produire des résumés structurés lors de l'escalade vers un humain

1.5 Hooks de l'Agent SDK pour intercepter les appels d'outils et normaliser les données

Connaissances clés :

- Modèles de hooks (p. ex. `PostToolUse`) pour intercepter les résultats d'outils avant que le modèle ne les consomme

- Hooks qui interceptent les appels sortants pour appliquer des règles de conformité (p. ex. bloquer les remboursements au-dessus d'un seuil)
- Les hooks fournissent des **garanties déterministes** vs les instructions de prompt qui fournissent une **conformité probabiliste**

Compétences clés :

- Hooks `PostToolUse` pour normaliser les formats de données (timestamps Unix, ISO 8601, codes de statut numériques)
- Hooks d'interception pour bloquer les actions violant une politique avec redirection vers l'escalade
- Choisir des hooks plutôt que des prompts lorsque les règles métier exigent une conformité garantie

1.6 Stratégies de décomposition des tâches pour les workflows complexes

Connaissances clés :

- **Pipelines fixes** (chaînage de prompts) vs **décomposition adaptative dynamique** basée sur les résultats intermédiaires
- Chaînage de prompts : étapes séquentielles (analyser chaque fichier séparément, puis effectuer une passe d'intégration)
- Plans d'investigation adaptatifs qui génèrent des sous-tâches en fonction de ce qui a été découvert

Compétences clés :

- Utiliser le chaînage de prompts pour les revues multi-aspects prévisibles ; utiliser la décomposition dynamique pour les investigations ouvertes
- Découper les grandes revues de code en analyse par fichier plus une passe d'intégration inter-fichiers séparée
- Décomposer les tâches ouvertes : d'abord cartographier la structure, puis construire un plan priorisé

1.7 État de session, reprise et fork

Connaissances clés :

- `--resume <session-name>` pour poursuivre des sessions nommées
- `fork_session` pour créer des branches d'investigation indépendantes à partir d'un contexte partagé
- L'importance d'informer l'agent des changements de fichiers lors de la reprise de sessions
- Une nouvelle session avec un résumé structuré peut être plus fiable que la reprise avec des résultats obsolètes

Compétences clés :

- Utiliser `--resume` pour poursuivre des sessions d'investigation nommées
- Utiliser `fork_session` pour comparer des approches en parallèle

- Choisir entre reprendre (contexte toujours à jour) ou démarrer une nouvelle session (résultats obsolètes)

Domaine 2 : Conception d'outils et intégration MCP (18 %)

2.1 Concevoir des interfaces d'outils avec des descriptions claires

Connaissances clés :

- Les descriptions d'outils sont le **mécanisme principal** qu'un LLM utilise pour sélectionner les outils ; des descriptions minimales conduisent à une sélection peu fiable
- L'importance d'inclure les formats d'entrée, des exemples de requêtes, les cas limites et les limites d'applicabilité
- Des descriptions ambiguës ou qui se chevauchent provoquent un mauvais routage
- La formulation du prompt système peut créer des associations involontaires avec les outils

Compétences clés :

- Rédiger des descriptions qui distinguent clairement chaque outil d'alternatives similaires
- Renommer les outils pour éliminer le chevauchement fonctionnel (p. ex. `analyze_content` -> `extract_web_results`)
- Diviser les outils polyvalents en outils spécialisés avec des contrats d'entrée/sortie clairs

2.2 Implémenter des réponses d'erreur structurées pour les outils MCP

Connaissances clés :

- Le drapeau `isError` dans les réponses des outils MCP
- La différence entre **erreurs transitoires** (timeouts), **erreurs de validation** (mauvaise entrée), **erreurs métier** (violations de politique) et **erreurs d'accès/permission**
- Les erreurs génériques (« Operation failed ») empêchent les bonnes décisions de récupération
- La différence entre erreurs réessayables et non réessayables

Compétences clés :

- Renvoyer des métadonnées structurées telles que `errorCategory` (transient/validation/permission), `isRetryable` et un message lisible par un humain
- Utiliser `retryable: false` pour les violations de règles métier avec des explications claires destinées à l'utilisateur

- Effectuer une récupération locale dans les sous-agents pour les échecs transitoires ; ne propager que les erreurs qu'ils ne peuvent pas résoudre
- Distinguer les échecs d'accès (décision de réessai) des résultats vides valides (aucune correspondance)

2.3 Allouer les outils entre les agents et configurer `tool_choice`

Connaissances clés :

- Trop d'outils par agent (p. ex. 18 au lieu de 4–5) **réduit** la fiabilité de la sélection d'outils
- Les agents disposant d'outils en dehors de leur spécialisation ont tendance à mal les utiliser
- Accès aux outils limité au périmètre : uniquement les outils pertinents pour le rôle plus un ensemble limité d'utilitaires inter-rôles
- `tool_choice` : `"auto"`, `"any"` et sélection forcée d'outil (`{"type": "tool", "name": "..."}`)

Compétences clés :

- Restreindre l'ensemble d'outils de chaque sous-agent à ce qui est pertinent pour son rôle
- Remplacer les outils généraux par des alternatives contraintes (p. ex. `fetch_url` -> `load_document`)
- Utiliser `tool_choice: "any"` pour garantir un appel d'outil au lieu d'une réponse en texte
- Forcer un outil précis pour garantir l'ordre d'exécution

2.4 Intégrer les serveurs MCP dans Claude Code et les workflows d'agents

Connaissances clés :

- Portée des serveurs MCP : projet (`.mcp.json`) pour les équipes vs utilisateur (`~/.claude.json`) pour les expériences
- Substitution de variables d'environnement dans `.mcp.json` (p. ex. `${GITHUB_TOKEN}`) pour la gestion des secrets
- Les outils de tous les serveurs MCP connectés sont découverts à la connexion et disponibles simultanément
- Les ressources MCP comme « catalogues de contenu » (résumés de tâches, schémas de base de données) pour réduire les appels d'outils exploratoires

Compétences clés :

- Configurer les serveurs MCP partagés dans le `.mcp.json` du projet avec des tokens basés sur des variables d'environnement
- Garder les serveurs personnels/expérimentaux dans `~/.claude.json`
- Préférer les serveurs MCP communautaires aux serveurs personnalisés pour les intégrations standard

2.5 Sélectionner et appliquer les outils intégrés (Read, Write, Edit, Bash, Grep, Glob)

Connaissances clés :

- **Grep** : rechercher dans le contenu des fichiers (noms de fonctions, messages d'erreur, imports)
- **Glob** : trouver des fichiers par motifs de nom/extension
- **Read/Write** : opérations sur fichier entier ; **Edit** : changements précis via des correspondances de texte uniques
- Si Edit échoue en raison de correspondances non uniques, se replier sur Read + Write

Compétences clés :

- Utiliser Grep pour la recherche de contenu et Glob pour la découverte de fichiers par motifs
- Construire la compréhension de manière incrémentale : Grep les points d'entrée, puis Read pour tracer les flux
- Tracer l'utilisation des fonctions à travers les modules wrappers

---

Domaine 3 : Configuration et workflows de Claude Code (20 %)

3.1 Configurer CLAUDE.md avec hiérarchie, portée et organisation modulaire

Connaissances clés :

- Hiérarchie de CLAUDE.md : utilisateur (`~/.claude/CLAUDE.md`), projet (`.claude/CLAUDE.md` ou `CLAUDE.md` racine) et niveau répertoire (CLAUDE.md dans les sous-répertoires)
- Les paramètres au niveau utilisateur ne s'appliquent qu'à un seul utilisateur et ne sont pas partagés via le VCS
- Syntaxe `@path` pour référencer des fichiers externes (p. ex. `@./standards/coding-style.md`) afin de modulariser CLAUDE.md
- Le répertoire `.claude/rules/` pour des fichiers de règles ciblés par thème au lieu d'un CLAUDE.md monolithique

Compétences clés :

- Diagnostiquer les problèmes de hiérarchie (un nouveau membre de l'équipe manque les instructions parce qu'elles sont au niveau utilisateur au lieu du niveau projet)
- Utiliser `@path` (p. ex. `@./standards/testing.md`) pour inclure sélectivement des normes dans le CLAUDE.md de chaque package
- Diviser un grand CLAUDE.md en plusieurs fichiers `.claude/rules/` (testing.md, api-conventions.md, deployment.md)

3.2 Créer et configurer des commandes slash personnalisées et des Skills

Connaissances clés :

- **Commandes de projet** dans `.claude/commands/` (partagées via le VCS) vs **commandes utilisateur** dans `~/.claude/commands/`
- Skills dans `.claude/skills/` avec frontmatter `SKILL.md` : `context: fork`, `allowed-tools`, `argument-hint`
- `context: fork` exécute le skill dans un contexte de sous-agent isolé afin qu'il ne pollue pas la session principale
- Les variantes personnelles de skills peuvent vivre dans `~/.claude/skills/` sous des noms différents

Compétences clés :

- Stocker les commandes slash de projet dans `.claude/commands/` pour que toute l'équipe en bénéficie
- Utiliser `context: fork` pour isoler les skills à sortie verbeuse
- Utiliser `allowed-tools` pour restreindre les outils qu'un skill peut utiliser
- Utiliser `argument-hint` pour inviter les développeurs à fournir les paramètres requis

3.3 Utiliser des règles spécifiques au chemin pour le chargement conditionnel de conventions

Connaissances clés :

- Les fichiers `.claude/rules/` peuvent inclure un frontmatter YAML `paths` pour activer les règles selon des motifs glob
- Les règles à portée de chemin se chargent **uniquement** lors de l'édition de fichiers correspondants, économisant contexte et tokens
- Les règles à base de glob peuvent être préférables au CLAUDE.md de niveau répertoire lorsque les conventions s'appliquent à de nombreux répertoires (p. ex. les tests)

Compétences clés :

- Créer des fichiers `.claude/rules/` avec `paths: ["terraform/**/*"]` pour ne charger que lors du travail sur les fichiers correspondants
- Utiliser des motifs glob (`**/*.test.tsx`) pour appliquer les conventions par type de fichier, quel que soit l'emplacement
- Préférer les règles spécifiques au chemin au CLAUDE.md de niveau répertoire lorsque les conventions s'étendent à toute la base de code

3.4 Décider quand utiliser le mode planification vs l'exécution directe

Connaissances clés :

- **Mode planification** : pour les tâches complexes avec des changements importants, plusieurs approches viables et des décisions architecturales
- **Exécution directe** : pour les changements simples et bien compris (p. ex. ajouter une seule validation)
- Le mode planification permet une exploration sûre de la base de code avant d'apporter des modifications
- Le sous-agent Explore isole la sortie verbeuse de découverte

Compétences clés :

- Utiliser le mode planification pour les tâches aux conséquences architecturales (microservices, migrations touchant plus de 45 fichiers)
- Utiliser l'exécution directe pour les corrections avec une trace de pile claire et un seul fichier
- Utiliser le sous-agent Explore pour éviter l'épuisement de la fenêtre de contexte dans les tâches multi-phases
- Combiner les approches : planifier pour la découverte, puis exécuter pour l'implémentation

3.5 Raffinement itératif pour une amélioration progressive

Connaissances clés :

- Des exemples concrets entrée/sortie sont le moyen le plus efficace de communiquer les attentes
- **Itération guidée par les tests** : écrire les tests d'abord, puis itérer en fonction des échecs
- Le modèle « entretien » : Claude pose des questions pour faire émerger des considérations de conception non évidentes
- Quand fournir tous les problèmes en un seul message (interdépendants) vs séquentiellement (indépendants)

Compétences clés :

- Fournir 2 à 3 exemples concrets entrée/sortie pour clarifier les exigences de transformation
- Construire des ensembles de tests avec comportement attendu, cas limites et exigences de performance avant l'implémentation
- Utiliser le modèle d'entretien pour faire émerger les aspects de conception (invalidation de cache, modes de défaillance)
- Fournir des cas de test concrets avec des entrées d'exemple et des sorties attendues pour les cas limites

3.6 Intégrer Claude Code dans les pipelines CI/CD

Connaissances clés :

- Le drapeau `-p` (ou `--print`) pour le mode non interactif dans les pipelines automatisés
- `--output-format json` et `--json-schema` pour une sortie structurée en CI
- CLAUDE.md fournit le contexte du projet (normes de test, critères de revue) pour Claude Code déclenché en CI
- **Isolation du contexte de session** : la même session qui a généré le code est moins efficace pour le réviser qu'une instance indépendante

Compétences clés :

- Exécuter Claude Code en CI avec `-p` pour éviter de bloquer sur une entrée interactive
- Utiliser `--output-format json` + `--json-schema` pour des résultats structurés (p. ex. commentaires inline sur la PR)
- Inclure les résultats de revue antérieurs lors d'une nouvelle exécution après de nouveaux commits (ne signaler que les problèmes nouveaux/non corrigés)
- Documenter les normes de test et les fixtures disponibles dans CLAUDE.md pour améliorer la qualité de la génération de tests
- Inclure les fichiers de test existants dans le contexte lors de la génération de nouveaux tests pour éviter la duplication et garder un style cohérent

---

Domaine 4 : Ingénierie de prompts et sortie structurée (20 %)

4.1 Concevoir des prompts avec des critères explicites pour améliorer la précision

Connaissances clés :

- Les critères explicites sont plus efficaces que les instructions vagues (p. ex. « signaler les commentaires uniquement lorsqu'ils contredisent le code » vs « vérifier l'exactitude des commentaires »)
- Un guidage générique comme « sois plus conservateur » fonctionne moins bien que des critères catégoriels concrets
- L'effet des faux positifs sur la confiance des développeurs : des taux de faux positifs élevés dans certaines catégories sapent la confiance dans les catégories précises

Compétences clés :

- Définir des critères de revue : ce qu'il faut signaler (bugs, sécurité) vs ce qu'il faut ignorer (style mineur)
- Désactiver temporairement les catégories à taux de faux positifs élevés

- Définir des critères de gravité explicites avec des exemples de code pour chaque niveau

4.2 Utiliser le prompting few-shot pour améliorer la cohérence de la sortie

Connaissances clés :

- Les exemples few-shot sont la méthode la plus efficace pour produire une sortie cohérente, bien formatée et exploitable
- Le few-shot peut démontrer la gestion des cas ambigus (sélection d'outil, lacunes dans la couverture de test)
- Le few-shot aide le modèle à généraliser à de nouveaux motifs plutôt que de simplement répéter les valeurs par défaut
- Le few-shot peut réduire les hallucinations dans les tâches d'extraction

Compétences clés :

- Fournir 2 à 4 exemples ciblés pour les scénarios ambigus avec une justification
- Inclure des exemples few-shot qui démontrent le format de sortie (emplacement, problème, gravité, correction suggérée)
- Fournir des exemples qui distinguent les motifs de code acceptables des vrais problèmes
- Fournir des exemples d'extraction correcte à partir de documents de structures différentes

4.3 Imposer une sortie structurée avec `tool_use` et des schémas JSON

Connaissances clés :

- `tool_use` avec des schémas JSON est le moyen le plus fiable de garantir une sortie conforme au schéma et d'éliminer les erreurs de syntaxe JSON
- Avec `tool_choice: "auto"` le modèle peut renvoyer du texte ; avec `"any"` il doit appeler un outil ; la sélection forcée choisit un outil précis
- Les schémas JSON stricts éliminent les erreurs de syntaxe mais n'empêchent pas les erreurs sémantiques (les totaux ne correspondent pas ; valeurs dans les mauvais champs)
- Conception de schéma : champs requis vs optionnels ; enums avec « other » plus une chaîne de détail pour l'extensibilité

Compétences clés :

- Définir des outils d'extraction avec des schémas JSON et analyser les données depuis les résultats `tool_use`
- Utiliser `tool_choice: "any"` pour garantir une sortie structurée lorsque plusieurs schémas existent
- Forcer un appel d'outil précis : `tool_choice: {"type": "tool", "name": "extract_metadata"}`
- Rendre les champs optionnels/nullable lorsque la source peut ne pas contenir d'information, afin d'éviter d'inventer des valeurs

- Utiliser des valeurs d'enum comme `"unclear"` et `"other"` plus des champs de détail pour une catégorisation extensible

4.4 Implémenter validation, nouvelles tentatives et boucles de feedback pour la qualité de l'extraction

Connaissances clés :

- Nouvelle tentative avec feedback d'erreur : inclure des erreurs de validation concrètes dans le prompt de réessai pour guider les corrections
- Les nouvelles tentatives sont inefficaces lorsque l'information est tout simplement absente de la source
- Conception de la boucle de feedback : suivre le motif qui a déclenché une détection (`detected_pattern`)
- Erreurs sémantiques (les totaux ne se réconcilient pas) vs erreurs de syntaxe (traitées par `tool_use`)

Compétences clés :

- Prompts de suivi avec le document original, une extraction incorrecte et des erreurs de validation précises
- Identifier quand une nouvelle tentative sera inefficace (l'info requise est uniquement dans un document externe)
- Inclure des champs `detected_pattern` dans les détections pour analyser les faux positifs
- Concevoir l'auto-correction en extrayant à la fois `calculated_total` et `stated_total` pour détecter les écarts

4.5 Concevoir des stratégies de traitement par lots efficaces

Connaissances clés :

- Message Batches API : 50 % d'économies, fenêtre de traitement jusqu'à 24 heures, aucune garantie de SLA de latence
- Le traitement par lots convient aux tâches non bloquantes (rapports de nuit, audits) et non aux tâches bloquantes (vérifications avant fusion)
- La Batch API ne prend pas en charge les appels d'outils multi-tours au sein d'une seule requête
- Les champs `custom_id` corrèlent requête/réponse au sein des lots

Compétences clés :

- Utiliser l'API synchrone pour les vérifications bloquantes ; utiliser la Batch API pour les charges de travail de nuit/hebdomadaires
- Planifier la cadence de soumission des lots selon les besoins de SLA (p. ex. fenêtres de 4 heures pour une garantie de 30 heures avec un traitement de 24 heures)
- Gérer les échecs en ne re-soumettant que les documents échoués (identifiés par `custom_id`)
- Itérer sur les prompts à l'aide d'un échantillon avant d'exécuter un traitement à grande échelle

4.6 Concevoir des architectures de revue multi-instances et multi-passes

Connaissances clés :

- Limites de l'auto-revue : le modèle conserve son contexte de raisonnement et est moins enclin à remettre en question ses propres décisions
- Les instances de revue indépendantes (sans contexte de génération) détectent mieux les problèmes subtils
- Revue multi-passes : analyse locale par fichier plus une passe d'intégration inter-fichiers pour éviter la dilution de l'attention

Compétences clés :

- Utiliser une deuxième instance Claude indépendante pour réviser les changements sans contexte de génération
- Diviser les revues multi-fichiers en passes par fichier plus des passes d'intégration pour l'analyse des flux de données inter-fichiers
- Utiliser des passes de vérification avec confiance auto-évaluée pour router les revues de manière calibrée

---

Domaine 5 : Gestion du contexte et fiabilité (15 %)

5.1 Gérer le contexte de conversation pour préserver les informations critiques

Connaissances clés :

- Risques du résumé progressif : les valeurs numériques, les pourcentages et les dates sont condensés en résumés vagues
- Effet « perdu au milieu » : les modèles traitent de manière fiable le début et la fin des longues entrées, mais peuvent manquer des conclusions du milieu
- Les sorties d'outils peuvent s'accumuler dans le contexte de manière disproportionnée par rapport à leur pertinence (plus de 40 champs alors que 5 sont nécessaires)
- L'importance d'envoyer l'historique complet de la conversation dans les requêtes API ultérieures

Compétences clés :

- Extraire les faits transactionnels dans un bloc persistant « case facts » en dehors de l'historique résumé
- Réduire les sorties d'outils verbeuses aux champs pertinents
- Placer les conclusions clés au début des données agrégées avec des en-têtes de section explicites

- Exiger des sous-agents qu'ils incluent des métadonnées (dates, sources) dans les sorties structurées

5.2 Concevoir des modèles d'escalade efficaces et résoudre l'ambiguïté

Connaissances clés :

- Déclencheurs d'escalade appropriés : demande explicite d'un humain, lacunes/exceptions de politique, incapacité à progresser
- Escalade immédiate (demande explicite) vs tentative de résolution (dans le périmètre de l'agent)
- L'analyse de sentiment et les auto-évaluations de confiance du modèle sont des indicateurs peu fiables de la complexité d'un cas
- Plusieurs correspondances client nécessitent de demander des identifiants supplémentaires, pas une supposition heuristique

Compétences clés :

- Critères d'escalade explicites avec exemples few-shot dans le prompt système
- Exécuter immédiatement les demandes explicites d'un humain sans investigation supplémentaire
- Escalader lorsque la politique est ambiguë ou muette pour une demande précise
- Demander des identifiants supplémentaires lorsque les résultats d'outils contiennent plusieurs correspondances

5.3 Implémenter des stratégies de propagation d'erreurs dans les systèmes multi-agents

Connaissances clés :

- Un contexte d'erreur structuré (type d'échec, requête, résultats partiels, alternatives) permet une récupération plus intelligente par le coordinateur
- Distinguer les échecs d'accès (les timeouts nécessitent une décision de réessai) des résultats vides valides (aucune correspondance)
- Les statuts d'erreur génériques (« recherche indisponible ») cachent un contexte précieux au coordinateur
- La suppression silencieuse ou l'abandon de tout le workflow sur un seul échec sont tous deux des anti-modèles

Compétences clés :

- Renvoyer un contexte d'erreur structuré : type d'échec, ce qui a été tenté, résultats partiels, alternatives possibles
- Distinguer les échecs d'accès des résultats vides valides
- Effectuer une récupération locale dans les sous-agents pour les échecs transitoires ; ne propager que les erreurs non récupérables avec les résultats partiels
- Annoter la couverture dans la synthèse : ce qui est bien étayé vs là où subsistent des lacunes

5.4 Gérer efficacement le contexte lors de l'investigation de grandes bases de code

Connaissances clés :

- Dégradation du contexte dans les longues sessions : le modèle commence à produire des réponses instables et à se référer à des « motifs typiques » au lieu de classes spécifiques
- Les fichiers bloc-notes préservent les conclusions clés au-delà des frontières de contexte
- La délégation aux sous-agents isole la sortie verbeuse de découverte
- La persistance d'état structurée permet la reprise après panne

Compétences clés :

- Engendrer des sous-agents pour des questions précises tout en gardant la coordination de haut niveau dans l'agent principal
- Utiliser des fichiers bloc-notes pour stocker les conclusions clés et les référencer plus tard
- Résumer les conclusions clés avant d'engendrer les sous-agents de la phase suivante
- Utiliser `/compact` pour réduire l'utilisation du contexte lors des longues investigations

5.5 Concevoir des workflows avec supervision humaine et calibrage de la confiance

Connaissances clés :

- Les métriques agrégées (p. ex. 97 % de précision globale) peuvent masquer de mauvaises performances sur des types de documents ou des champs spécifiques
- L'échantillonnage aléatoire stratifié mesure les taux d'erreur dans les extractions à confiance élevée
- Calibrage de la confiance au niveau du champ à l'aide d'ensembles de validation étiquetés
- Valider la précision par type de document et par segment de champ avant d'automatiser

Compétences clés :

- Implémenter un échantillonnage aléatoire stratifié pour détecter de nouveaux motifs d'erreur
- Analyser la précision par type de document et par champ pour valider une performance stable
- Produire des scores de confiance au niveau du champ et calibrer les seuils de revue à l'aide de données étiquetées
- Router les extractions à faible confiance ou à source ambiguë vers une revue humaine

5.6 Préserver la provenance et gérer l'incertitude dans la synthèse multi-sources

Connaissances clés :

- L'attribution est perdue lors du résumé si l'on ne préserve pas les correspondances « affirmation → source »

- Les correspondances structurées doivent être préservées lors de l'agrégation
- Gérer les statistiques contradictoires en annotant les conflits avec leur attribution plutôt qu'en choisissant arbitrairement une valeur
- Inclure les dates de publication/collecte pour éviter de lire à tort les différences temporelles comme des contradictions

Compétences clés :

- Exiger des sous-agents qu'ils produisent des correspondances « affirmation → source » (URL, nom du document, citations)
- Structurer les rapports pour séparer les conclusions stables des conclusions contestées
- Préserver les valeurs contradictoires avec des annotations et les transmettre au coordinateur pour réconciliation
- Inclure les dates de publication pour une interprétation temporelle correcte
- Rendre le contenu par type : données financières en tableaux, actualités en prose, conclusions techniques en listes structurées

---

Exemples de questions d'examen avec explications

Question 1 (Scénario : Agent de support client)

Situation : Les données montrent que dans 12 % des cas, l'agent saute `get_customer` et appelle `lookup_order` en utilisant uniquement le nom du client, ce qui entraîne des remboursements incorrects.

Quel changement est le plus efficace ?

- A) Ajouter une précondition programmatique qui bloque `lookup_order` et `process_refund` jusqu'à ce qu'un ID soit obtenu via `get_customer` **[CORRECT]**
- B) Améliorer le prompt système
- C) Ajouter des exemples few-shot
- D) Implémenter un classificateur de routage

Pourquoi A : Lorsque la logique métier critique exige une séquence d'outils précise, le logiciel fournit des **garanties déterministes** que les approches par prompt (B, C) ne peuvent pas offrir. D traite la disponibilité, pas l'ordre des outils.

Question 2 (Scénario : Agent de support client)

Situation : L'agent appelle souvent `get_customer` au lieu de `lookup_order` pour les questions liées aux commandes. Les descriptions des outils sont minimales et similaires.

Quelle est la première étape ?

- A) Exemples few-shot
- B) Étendre la description de chaque outil avec des formats d'entrée, des exemples et des limites **[CORRECT]**
- C) Ajouter une couche de routage
- D) Fusionner les outils

Pourquoi B : Les descriptions d'outils sont le principal mécanisme de sélection du modèle. C'est la correction au moindre effort et au plus fort impact. A ajoute des tokens sans traiter la cause racine. C est de la sur-ingénierie. D demande plus d'effort que justifié.

Question 3 (Scénario : Agent de support client)

Situation : L'agent ne résout que 55 % des problèmes pour un objectif de 80 %. Il escalade les cas simples et tente de gérer de manière autonome des exceptions de politique complexes.

Comment améliorer le calibrage ?

- A) Ajouter des critères d'escalade explicites avec des exemples few-shot **[CORRECT]**
- B) Confiance auto-évaluée (1–10) avec escalade automatique
- C) Un classificateur séparé entraîné sur des données historiques
- D) Analyse de sentiment

Pourquoi A : Cela traite directement la cause racine — des frontières de décision floues. B n'est pas fiable (le modèle peut se tromper avec assurance). C est de la sur-ingénierie. D résout un autre problème (humeur != complexité).

Question 4 (Scénario : Génération de code avec Claude Code)

Situation : Vous avez besoin d'une commande `/review` personnalisée pour la revue de code standard, disponible pour toute l'équipe lorsqu'elle clone le dépôt.

Où créer le fichier de commande ?

- A) `.claude/commands/` dans le dépôt du projet **[CORRECT]**
- B) `~/.claude/commands/`
- C) `CLAUDE.md` racine
- D) `.claude/config.json`

Pourquoi A : Les commandes de projet stockées dans `.claude/commands/` sont versionnées et automatiquement disponibles pour tous. B est pour les commandes personnelles. C est pour les instructions, pas les définitions de commandes. D n'existe pas.

Question 5 (Scénario : Génération de code avec Claude Code)

Situation : Vous devez restructurer un monolithe en microservices (des dizaines de fichiers, décisions sur les frontières de service).

Quelle approche utiliser ?

- A) Mode planification : explorer la base de code, comprendre les dépendances, concevoir une approche **[CORRECT]**
- B) Exécution directe de manière incrémentale
- C) Exécution directe avec des instructions détaillées en amont
- D) Exécution directe et passage à la planification quand ça devient difficile

Pourquoi A : Le mode planification est conçu pour les changements importants, les approches multiples possibles et les décisions architecturales. B risque des reprises coûteuses. C suppose que vous connaissez déjà la structure. D est réactif.

Question 6 (Scénario : Génération de code avec Claude Code)

Situation : Une base de code a des conventions différentes selon les domaines (React, API, base de données). Les tests sont co-localisés avec le code. Vous voulez que les conventions soient appliquées automatiquement.

Quelle approche utiliser ?

- A) Fichiers `.claude/rules/` avec frontmatter YAML et motifs glob **[CORRECT]**
- B) Tout mettre dans le CLAUDE.md racine
- C) Skills dans `.claude/skills/`
- D) CLAUDE.md dans chaque répertoire

Pourquoi A : `.claude/rules/` avec des motifs glob (p. ex. `**/*.test.tsx`) permet l'application automatique des conventions en fonction des chemins de fichiers — idéal pour les tests dispersés dans la base de code. B repose sur l'inférence du modèle. C est manuel/à la demande. D fonctionne mal lorsque les fichiers pertinents sont dans de nombreux répertoires.

Question 7 (Scénario : Système de recherche multi-agents)

Situation : Le système recherche « l'impact de l'IA sur les industries créatives », mais les rapports ne couvrent que les arts visuels. Le coordinateur a décomposé le sujet en : « IA dans l'art numérique », « IA dans le design graphique », « IA dans la photographie ».

Quelle en est la cause ?

- A) L'agent de synthèse ne détecte pas les lacunes
- B) Le coordinateur a décomposé la tâche de manière trop étroite **[CORRECT]**
- C) L'agent de recherche web ne cherche pas assez en profondeur

- D) L'agent d'analyse de documents filtre les sources non visuelles

Pourquoi B : Les logs montrent que le coordinateur n'a décomposé « industries créatives » qu'en sous-thèmes visuels, manquant complètement la musique, la littérature et le cinéma. Les sous-agents ont exécuté correctement — le problème est ce qui leur a été assigné.

Question 8 (Scénario : Système de recherche multi-agents)

Situation : Un sous-agent de recherche web dépasse son délai (timeout) en recherchant un sujet complexe. Vous devez concevoir la façon dont l'information d'erreur est renvoyée au coordinateur.

Quelle approche de propagation d'erreurs permet le mieux une récupération intelligente ?

- A) Renvoyer un contexte d'erreur structuré au coordinateur : type d'échec, requête, résultats partiels et alternatives **[CORRECT]**
- B) Implémenter des nouvelles tentatives automatiques avec backoff exponentiel à l'intérieur du sous-agent, puis renvoyer un statut générique « recherche indisponible »
- C) Capturer le timeout à l'intérieur du sous-agent et renvoyer un ensemble de résultats vide marqué comme succès
- D) Propager l'exception de timeout à un gestionnaire de haut niveau qui termine tout le workflow

Pourquoi A : Un contexte d'erreur structuré donne au coordinateur ce dont il a besoin pour décider de réessayer avec une requête modifiée, d'essayer une approche alternative ou de continuer avec des résultats partiels. B cache le contexte derrière un statut générique. C masque l'échec en succès. D abandonne tout le workflow inutilement.

Question 9 (Scénario : Système de recherche multi-agents)

Situation : L'agent de synthèse a souvent besoin de vérifier des affirmations précises lors de la fusion des résultats. Actuellement, lorsqu'une vérification est nécessaire, l'agent de synthèse rend le contrôle au coordinateur, qui appelle l'agent de recherche web puis relance la synthèse avec les nouveaux résultats. Cela ajoute 2 à 3 allers-retours supplémentaires par tâche et augmente la latence de 40 %. Votre évaluation montre que 85 % de ces vérifications sont de simples vérifications de faits (dates, noms, statistiques), tandis que 15 % nécessitent une investigation plus approfondie.

Comment réduire la surcharge tout en maintenant la fiabilité ?

- A) Donner à l'agent de synthèse un outil `verify_fact` limité pour les vérifications simples, et continuer à router les vérifications complexes via le coordinateur **[CORRECT]**
- B) Accumuler tous les besoins de vérification en un lot et les renvoyer au coordinateur à la fin
- C) Donner à l'agent de synthèse un accès complet à tous les outils de recherche web
- D) Mettre en cache de manière proactive du contexte supplémentaire autour de chaque source

Pourquoi A : Cela applique le principe du moindre privilège : l'agent de synthèse obtient exactement ce dont il a besoin pour le cas courant à 85 % (vérifications simples de faits) tout en préservant le chemin médiatisé par le coordinateur pour les investigations complexes. B introduit des dépendances bloquantes

(les étapes de synthèse ultérieures peuvent dépendre des faits vérifiés antérieurs). C rompt la séparation des responsabilités. D repose sur une mise en cache spéculative qui ne peut pas prédire les besoins de manière fiable.

Question 10 (Scénario : Claude Code pour la CI)

Situation : Un pipeline exécute `claude "Analyze this pull request for security issues"`, mais reste bloqué en attente d'une entrée interactive.

Quelle est l'approche correcte ?

- A) Utiliser le drapeau `-p` : `claude -p "Analyze this pull request for security issues"` **[CORRECT]**
- B) Définir `CLAUDE_HEADLESS=true`
- C) Rediriger stdin depuis `/dev/null`
- D) Utiliser `--batch`

Pourquoi A : `-p` (ou `--print`) est la façon documentée d'exécuter Claude Code en mode non interactif. Il traite le prompt, l'affiche sur stdout et quitte. Les autres options sont soit des fonctionnalités inexistantes, soit des contournements Unix.

Question 11 (Scénario : Claude Code pour la CI)

Situation : L'équipe veut réduire le coût d'API pour l'analyse automatisée. Claude sert actuellement deux workflows en temps réel : (1) une vérification bloquante avant fusion qui doit se terminer avant que les développeurs puissent fusionner une PR, et (2) un rapport de dette technique généré la nuit pour revue le matin. Un manager propose de déplacer les deux vers la Message Batches API pour économiser 50 %.

Comment évaluer cette proposition ?

- A) Utiliser le traitement par lots uniquement pour les rapports de dette technique ; garder les appels en temps réel pour les vérifications avant fusion **[CORRECT]**
- B) Déplacer les deux workflows vers le traitement par lots et interroger l'achèvement (polling)
- C) Garder les appels en temps réel pour les deux afin d'éviter les problèmes d'ordre dans les résultats de lot
- D) Déplacer les deux vers le traitement par lots avec un repli vers le temps réel si un lot prend trop de temps

Pourquoi A : La Message Batches API économise 50 %, mais le temps de traitement peut atteindre 24 heures sans SLA de latence garanti. Cela la rend inadaptée aux vérifications bloquantes avant fusion où les développeurs attendent, mais idéale pour les charges de travail de nuit comme les rapports de dette technique.

Question 12 (Scénario : Revue de code multi-fichiers)

Situation : Une pull request modifie 14 fichiers dans un module de suivi des stocks. Une revue en une seule passe de tous les fichiers produit des résultats incohérents : commentaires détaillés pour certains fichiers mais superficiels pour d'autres, bugs évidents manqués et retours contradictoires (un motif est signalé comme problématique dans un fichier mais approuvé dans un code identique dans un autre fichier).

Comment restructurer la revue ?

- A) Diviser en passes ciblées : analyser chaque fichier individuellement pour les problèmes locaux, puis effectuer une passe d'intégration séparée pour les flux de données inter-fichiers **[CORRECT]**
- B) Exiger des développeurs qu'ils divisent les grandes PR en soumissions de 3 à 4 fichiers
- C) Passer à un modèle de niveau supérieur avec une fenêtre de contexte plus grande pour réviser les 14 fichiers en une seule passe
- D) Exécuter trois passes de revue complètes indépendantes et ne signaler que les problèmes trouvés dans au moins deux exécutions

Pourquoi A : Les passes ciblées traitent directement la cause racine — la dilution de l'attention lors du traitement de nombreux fichiers à la fois. L'analyse par fichier garantit une profondeur cohérente, et une passe d'intégration séparée détecte les problèmes inter-fichiers. B reporte la charge sur les développeurs sans améliorer le système. C est une idée fausse : un contexte plus grand ne corrige pas la qualité de l'attention. D supprime de vrais bugs en exigeant un consensus entre des détections incohérentes.

Test d'entraînement

> 60 questions réparties sur 4 scénarios. Le format et la difficulté correspondent au véritable examen.
>
> Vous pouvez également vous entraîner sur ces questions dans un fichier HTML de type examen : Test pratique (FR)

Scénario : Système de recherche multi-agents

Question 1 (Scénario : Système de recherche multi-agents)

Situation : Un agent d'analyse de documents découvre que deux sources crédibles contiennent des statistiques directement contradictoires pour une métrique clé : un rapport gouvernemental indique une croissance de 40 %, tandis qu'une analyse sectorielle indique 12 %. Les deux sources paraissent crédibles, et l'écart pourrait affecter de manière significative les conclusions de la recherche. Comment l'agent d'analyse de documents doit-il gérer cette situation le plus efficacement ?

Quelle approche est la plus efficace ?

- A) Appliquer des heuristiques de crédibilité pour choisir le nombre le plus probablement correct, terminer l'analyse avec cette valeur et ajouter une note de bas de page mentionnant l'écart.
- B) Inclure les deux nombres dans la sortie d'analyse sans les marquer comme contradictoires, laissant l'agent de synthèse décider lequel utiliser selon un contexte plus large.
- C) Arrêter l'analyse et escalader immédiatement vers le coordinateur, en lui demandant de décider quelle source fait le plus autorité avant de continuer.
- D) Terminer l'analyse avec les deux nombres, annoter explicitement le conflit avec l'attribution des sources, et laisser le coordinateur décider comment réconcilier les données avant de les transmettre à la synthèse. **[CORRECT]**

Pourquoi D : Cette approche préserve la séparation des responsabilités : l'agent d'analyse termine son travail principal sans blocage, préserve les deux valeurs contradictoires avec une attribution claire, et transmet correctement la réconciliation au coordinateur, qui dispose d'un contexte plus large.

Question 2 (Scénario : Système de recherche multi-agents)

Situation : Les agents de recherche web et d'analyse de documents ont terminé leurs tâches et renvoyé les résultats au coordinateur. Quelle est l'étape suivante pour créer un rapport de recherche intégré ?

Quelle étape suivante est la plus appropriée ?

- A) Chaque agent envoie ses résultats directement à l'agent de rédaction de rapport, en contournant le coordinateur.
- B) L'agent d'analyse de documents demande les résultats de la recherche web et les fusionne en interne.
- C) Le coordinateur transmet les deux ensembles de résultats à l'agent de synthèse pour une intégration unifiée. **[CORRECT]**
- D) Le coordinateur concatène les sorties brutes des deux agents et les renvoie comme résultat final.

Pourquoi C : Dans une architecture coordinateur–sous-agent, le coordinateur transmet les deux ensembles de résultats à l'agent de synthèse pour une intégration centralisée, préservant le contrôle et assurant une fusion de haute qualité.

Question 3 (Scénario : Système de recherche multi-agents)

Situation : Un sous-agent d'analyse de documents échoue fréquemment lors du traitement de fichiers PDF : certains ont des sections corrompues qui déclenchent des exceptions d'analyse, d'autres sont protégés par mot de passe, et parfois la bibliothèque d'analyse se bloque sur de gros fichiers. Actuellement, toute exception termine immédiatement le sous-agent et renvoie une erreur au coordinateur, qui doit décider de réessayer, d'ignorer ou de faire échouer toute la tâche. Cela entraîne une implication excessive du coordinateur dans la gestion d'erreurs routinières. Quelle amélioration architecturale est la plus efficace ?

Quelle amélioration est la plus efficace ?

- A) Créer un agent dédié à la gestion des erreurs qui surveille tous les échecs via une file partagée et décide des actions de récupération, en envoyant des commandes de redémarrage directement aux sous-agents.
- B) Configurer le sous-agent pour qu'il renvoie toujours des résultats partiels avec un statut de succès, en intégrant les détails d'erreur dans les métadonnées ; le coordinateur traite toutes les réponses comme réussies.
- C) Faire valider tous les documents par le coordinateur avant de les envoyer au sous-agent, en rejetant les documents susceptibles de provoquer des échecs.
- D) Implémenter une récupération locale dans le sous-agent pour les échecs transitoires et n'escalader vers le coordinateur que les erreurs qu'il ne peut pas résoudre, en incluant les étapes tentées et les résultats partiels. **[CORRECT]**

Pourquoi D : Gérer les erreurs au niveau le plus bas capable de les résoudre. La récupération locale réduit la charge du coordinateur tout en escaladant les problèmes véritablement irrécupérables avec le contexte complet et la progression partielle.

Question 4 (Scénario : Système de recherche multi-agents)

Situation : Après avoir exécuté le système sur « l'impact de l'IA sur les industries créatives », vous observez que chaque sous-agent se termine avec succès : l'agent de recherche web trouve des articles pertinents, l'agent d'analyse de documents les résume correctement, et l'agent de synthèse produit un texte cohérent. Cependant, les rapports finaux ne couvrent que les arts visuels et manquent complètement la musique, la littérature et le cinéma. Dans les logs du coordinateur, vous voyez qu'il a décomposé le sujet en trois sous-tâches : « IA dans l'art numérique », « IA dans le design graphique » et « IA dans la photographie ». Quelle est la cause racine la plus probable ?

Quelle est la cause racine la plus probable ?

- A) L'agent de synthèse manque d'instructions pour détecter les lacunes de couverture.
- B) L'agent d'analyse de documents filtre les sources non visuelles à cause de critères de pertinence trop stricts.
- C) La décomposition de tâche du coordinateur est trop étroite, assignant aux sous-agents un travail qui ne couvre pas tous les domaines pertinents. **[CORRECT]**
- D) Les requêtes de l'agent de recherche web sont insuffisantes et devraient être élargies pour couvrir plus de secteurs.

Pourquoi C : Le coordinateur a décomposé un sujet large uniquement en sous-tâches d'arts visuels, manquant entièrement la musique, la littérature et le cinéma. Puisque les sous-agents ont correctement exécuté leurs assignations, la décomposition trop étroite est la cause racine évidente.

Question 5 (Scénario : Système de recherche multi-agents)

Situation : Le sous-agent de recherche web renvoie des résultats pour seulement 3 des 5 catégories de sources demandées (les sites concurrents et les rapports sectoriels réussissent, mais les archives d'actualités et les flux sociaux dépassent le délai). Le sous-agent d'analyse de documents traite avec

succès tous les documents fournis. Le sous-agent de synthèse doit produire un résumé à partir d'entrées en amont de qualité mixte. Quelle stratégie de propagation d'erreurs est la plus efficace ?

Quelle stratégie de propagation d'erreurs est la plus efficace ?

- A) Poursuivre la synthèse en utilisant uniquement les sources réussies et produire une sortie sans mentionner quelles données étaient indisponibles.
- B) Le sous-agent de synthèse renvoie une erreur au coordinateur, déclenchant une nouvelle tentative complète ou un échec de tâche en raison de données incomplètes.
- C) Le sous-agent de synthèse demande au coordinateur de réessayer les sources en timeout avec un délai plus long avant de démarrer la synthèse.
- D) Structurer la sortie de synthèse avec des annotations de couverture qui indiquent quelles conclusions sont bien étayées et où subsistent des lacunes dues à des sources indisponibles. **[CORRECT]**

Pourquoi D : Les annotations de couverture mettent en œuvre une dégradation gracieuse avec transparence, préservant la valeur du travail accompli tout en propageant l'incertitude pour permettre des décisions éclairées sur la confiance.

Question 6 (Scénario : Système de recherche multi-agents)

Situation : Le sous-agent d'analyse de documents rencontre un fichier PDF corrompu qu'il ne peut pas analyser. Lors de la conception de la gestion d'erreurs du système, quelle est la manière la plus efficace de gérer cet échec ?

Quelle approche est la plus efficace ?

- A) Renvoyer une erreur avec contexte à l'agent coordinateur, lui permettant de décider comment procéder. **[CORRECT]**
- B) Ignorer silencieusement le document corrompu et continuer à traiter les fichiers restants pour éviter d'interrompre le workflow.
- C) Réessayer automatiquement d'analyser le document trois fois avec backoff exponentiel avant de signaler un échec.
- D) Lever une exception qui termine tout le workflow de recherche.

Pourquoi A : Renvoyer une erreur avec contexte au coordinateur est l'approche la plus efficace car elle lui permet de prendre une décision éclairée — ignorer le fichier, essayer une méthode d'analyse alternative ou notifier l'utilisateur — tout en gardant une visibilité sur l'échec.

Question 7 (Scénario : Système de recherche multi-agents)

Situation : Les logs de production montrent un motif persistant : des requêtes comme « analyse le rapport trimestriel téléversé » sont routées vers l'agent de recherche web 45 % du temps au lieu de l'agent d'analyse de documents. En examinant les définitions d'outils, vous constatez que l'agent de recherche web a un outil `analyze_content` décrit comme « analyzes content and extracts key information », tandis

que l'agent d'analyse de documents a un outil `analyze_document` décrit comme « analyzes documents and extracts key information ». Comment corriger le problème de mauvais routage ?

Comment corriger le problème de mauvais routage ?

- A) Ajouter un classificateur de pré-routage qui détecte si l'utilisateur fait référence à des fichiers téléversés ou à du contenu web avant que le coordinateur ne décide de la délégation.
- B) Renommer l'outil de recherche web en `extract_web_results` et mettre à jour sa description en « processes and returns information retrieved from web search and URLs ». **[CORRECT]**
- C) Ajouter des exemples few-shot au prompt du coordinateur montrant le routage correct : « L'utilisateur téléverse un rapport trimestriel → agent d'analyse de documents » et « L'utilisateur pose une question sur une page web → agent de recherche web ».
- D) Étendre la description de l'outil d'analyse de documents avec des exemples d'usage comme « À utiliser pour les PDF téléversés, les docs Word et les feuilles de calcul », en laissant l'outil de recherche web inchangé.

Pourquoi B : Renommer l'outil de recherche web en `extract_web_results` et mettre à jour sa description pour référencer explicitement la recherche web et les URL supprime directement la cause racine en éliminant le chevauchement sémantique entre les deux noms et descriptions d'outils. Cela rend l'objectif de chaque outil sans ambiguïté, permettant au coordinateur de distinguer de manière fiable l'analyse de documents de la recherche web.

Question 8 (Scénario : Système de recherche multi-agents)

Situation : Un collègue propose que l'agent d'analyse de documents envoie ses résultats directement à l'agent de synthèse, en contournant le coordinateur. Quel est le principal avantage de garder le coordinateur comme hub central pour toute communication entre sous-agents ?

Quel est le principal avantage de garder le coordinateur comme hub central ?

- A) Le coordinateur peut observer toutes les interactions, gérer les erreurs de manière uniforme et décider quelles informations chaque sous-agent doit recevoir. **[CORRECT]**
- B) Le coordinateur regroupe plusieurs requêtes vers les sous-agents, réduisant le nombre total d'appels API et la latence globale.
- C) Le routage via le coordinateur permet une logique de nouvelle tentative automatique que les appels directs entre agents ne peuvent pas prendre en charge.
- D) Les sous-agents utilisent une mémoire isolée, et la communication directe nécessiterait une sérialisation complexe que seul le coordinateur peut effectuer.

Pourquoi A : Le modèle de coordinateur fournit une visibilité centralisée sur toutes les interactions, une gestion d'erreurs uniforme dans tout le système et un contrôle fin sur les informations que chaque sous-agent reçoit — ce sont les principaux avantages d'une topologie de communication en étoile.

Question 9 (Scénario : Système de recherche multi-agents)

Situation : Le sous-agent de recherche web dépasse son délai en recherchant un sujet complexe. Vous devez concevoir la façon dont l'information sur cet échec est renvoyée au coordinateur. Quelle approche de propagation d'erreurs permet le mieux une récupération intelligente ?

Quelle approche de propagation d'erreurs permet le mieux une récupération intelligente ?

- A) Renvoyer un contexte d'erreur structuré au coordinateur incluant le type d'échec, la requête exécutée, tout résultat partiel et les approches alternatives potentielles. **[CORRECT]**
- B) Capturer le timeout dans le sous-agent et renvoyer un ensemble de résultats vide marqué comme réussi.
- C) Implémenter des nouvelles tentatives automatiques avec backoff exponentiel dans le sous-agent, ne renvoyant un statut générique « recherche indisponible » qu'après épuisement des tentatives.
- D) Propager l'exception de timeout directement au gestionnaire de haut niveau, terminant tout le workflow de recherche.

Pourquoi A : Renvoyer un contexte d'erreur structuré — incluant le type d'échec, la requête exécutée, les résultats partiels et les approches alternatives — donne au coordinateur tout ce dont il a besoin pour prendre des décisions de récupération intelligentes (p. ex. réessayer avec une requête modifiée ou continuer avec des résultats partiels). Cela préserve un maximum de contexte pour une prise de décision éclairée au niveau de la coordination.

Question 10 (Scénario : Système de recherche multi-agents)

Situation : Dans votre conception, vous avez donné à l'agent d'analyse de documents l'accès à un outil polyvalent `fetch_url` afin qu'il puisse télécharger des documents par URL. Les logs de production montrent que cet agent télécharge maintenant fréquemment des pages de résultats de moteurs de recherche pour effectuer des recherches web ad hoc — un comportement qui devrait passer par l'agent de recherche web — causant des résultats incohérents. Quelle correction est la plus efficace ?

Quelle correction est la plus efficace ?

- A) Remplacer `fetch_url` par un outil `load_document` qui valide que les URL pointent vers des formats de documents. **[CORRECT]**
- B) Retirer `fetch_url` de l'agent d'analyse de documents et router toute récupération d'URL via le coordinateur vers l'agent de recherche web.
- C) Implémenter un filtrage qui bloque les appels `fetch_url` vers les domaines de moteurs de recherche connus tout en autorisant les autres URL.
- D) Ajouter des instructions au prompt de l'agent d'analyse de documents indiquant que `fetch_url` ne doit servir qu'à télécharger des URL de documents, pas à chercher.

Pourquoi A : Remplacer un outil polyvalent par un outil spécifique aux documents qui valide les URL par rapport aux formats de documents corrige la cause racine en contraignant la capacité au niveau de l'interface. Cela suit le principe du moindre privilège, rendant le comportement de recherche indésirable impossible plutôt que simplement déconseillé.

Question 11 (Scénario : Système de recherche multi-agents)

Situation : En recherchant un sujet large, vous observez que l'agent de recherche web et l'agent d'analyse de documents investiguent les mêmes sous-thèmes, entraînant une duplication substantielle dans leurs sorties. L'utilisation de tokens double presque sans augmentation proportionnelle de l'étendue ou de la profondeur de la recherche. Quelle est la manière la plus efficace de traiter cela ?

Quelle est la manière la plus efficace de traiter cela ?

- A) Laisser les deux agents terminer en parallèle, puis faire dédupliquer les résultats qui se chevauchent par le coordinateur avant de les transmettre à l'agent de synthèse.
- B) Le coordinateur partitionne explicitement l'espace de recherche avant de déléguer, en assignant à chaque agent des sous-thèmes ou types de sources distincts. **[CORRECT]**
- C) Implémenter un mécanisme d'état partagé où les agents consignent leur domaine de focus actuel afin que les autres agents puissent éviter dynamiquement la duplication pendant l'exécution.
- D) Passer à une exécution séquentielle où l'analyse de documents ne s'exécute qu'après la fin de la recherche web, en utilisant les résultats de la recherche web comme contexte pour éviter la duplication.

Pourquoi B : Faire partitionner explicitement l'espace de recherche par le coordinateur avant de déléguer est le plus efficace car cela traite la cause racine — des frontières de tâche floues — avant que tout travail ne commence. Cela préserve le parallélisme tout en évitant les efforts dupliqués et les tokens gaspillés.

Question 12 (Scénario : Système de recherche multi-agents)

Situation : Pendant la recherche, le sous-agent de recherche web interroge trois catégories de sources avec des résultats différents : les bases de données académiques renvoient 15 articles pertinents, les rapports sectoriels renvoient « 0 résultat », et les bases de données de brevets renvoient « Connection timeout ». Lors de la conception de la propagation d'erreurs vers le coordinateur, quelle approche permet les meilleures décisions de récupération ?

Quelle approche permet les meilleures décisions de récupération ?

- A) Agréger les résultats en une seule métrique de pourcentage de succès (p. ex. « 67 % de couverture des sources ») avec des logs détaillés disponibles à la demande.
- B) Signaler à la fois « timeout » et « 0 résultat » comme des échecs nécessitant l'intervention du coordinateur.
- C) Réessayer les échecs transitoires en interne et ne signaler que les erreurs persistantes.
- D) Distinguer les échecs d'accès (timeout) qui nécessitent une décision de réessai des résultats vides valides (« 0 résultat ») qui représentent des requêtes réussies. **[CORRECT]**

Pourquoi D : Un timeout (échec d'accès) et « 0 résultat » (résultat vide valide) sont des issues sémantiquement différentes nécessitant des réponses différentes. Les distinguer permet au coordinateur de réessayer la base de données de brevets tout en acceptant le « 0 résultat » des rapports sectoriels comme une conclusion valide et informative.

Question 13 (Scénario : Système de recherche multi-agents)

Situation : La surveillance de production montre une qualité de synthèse incohérente. Lorsque les résultats agrégés font ~75 000 tokens, l'agent de synthèse cite de manière fiable les informations des 15 000 premiers tokens (titres/extraits de recherche web) et des 10 000 derniers tokens (conclusions de l'analyse de documents), mais manque souvent des conclusions critiques dans les 50 000 tokens du milieu — même lorsqu'elles répondent directement à la question de recherche. Comment restructurer l'entrée agrégée ?

Comment restructurer l'entrée agrégée ?

- A) Résumer toutes les sorties des sous-agents à moins de 20 000 tokens avant l'agrégation pour garder le contenu dans la plage de traitement fiable du modèle.
- B) Diffuser les résultats des sous-agents à l'agent de synthèse de manière incrémentale, en traitant d'abord les résultats de recherche web jusqu'à leur terme, puis en ajoutant les résultats d'analyse de documents.
- C) Placer un résumé des conclusions clés au début de l'entrée agrégée et organiser les résultats détaillés avec des en-têtes de section explicites pour faciliter la navigation. **[CORRECT]**
- D) Implémenter une rotation qui alterne quel résultat de sous-agent apparaît en premier selon les tâches de recherche pour que les deux sources obtiennent à terme un positionnement de tête équivalent.

Pourquoi C : Placer un résumé des conclusions clés au début exploite les effets de primauté afin que les informations critiques se trouvent dans la position la plus fiablement traitée. Ajouter des en-têtes de section explicites tout au long aide le modèle à naviguer et à prêter attention au contenu du milieu, atténuant directement le phénomène « perdu au milieu ».

Question 14 (Scénario : Système de recherche multi-agents)

Situation : Lors des tests, la sortie combinée de l'agent de recherche web (85 000 tokens incluant le contenu des pages) et de l'agent d'analyse de documents (70 000 tokens incluant des chaînes de raisonnement) totalise 155 000 tokens, mais l'agent de synthèse est le plus performant avec des entrées de moins de 50 000 tokens. Quelle solution est la plus efficace ?

Quelle solution est la plus efficace ?

- A) Modifier les agents en amont pour qu'ils renvoient des données structurées (faits clés, citations, scores de pertinence) au lieu d'un contenu et d'un raisonnement verbeux. **[CORRECT]**
- B) Ajouter un agent de résumé intermédiaire qui condense les conclusions avant de les transmettre à la synthèse.
- C) Faire traiter par l'agent de synthèse les conclusions en lots séquentiels, en maintenant l'état entre les appels.
- D) Stocker les conclusions dans une base de données vectorielle et donner à l'agent de synthèse des outils de recherche pour l'interroger pendant son travail.

Pourquoi A : Modifier les agents en amont pour qu'ils renvoient des données structurées corrige la cause racine en réduisant le volume de tokens à la source tout en préservant les informations essentielles. Cela

évite de transmettre un contenu de page volumineux et des traces de raisonnement qui gonflent les tokens sans améliorer l'étape de synthèse.

Question 15 (Scénario : Système de recherche multi-agents)

Situation : Lors des tests, vous observez que l'agent de synthèse a souvent besoin de vérifier des affirmations précises lors de la fusion des résultats. Actuellement, lorsqu'une vérification est nécessaire, l'agent de synthèse rend le contrôle au coordinateur, qui appelle l'agent de recherche web puis relance la synthèse avec les résultats. Cela ajoute 2 à 3 boucles supplémentaires par tâche et augmente la latence de 40 %. Votre évaluation montre que 85 % de ces vérifications sont de simples vérifications de faits (dates, noms, statistiques) et 15 % nécessitent une recherche plus approfondie. Quelle approche réduit le plus efficacement la surcharge tout en préservant la fiabilité du système ?

Quelle approche est la plus efficace ?

- A) Donner à l'agent de synthèse l'accès à tous les outils de recherche web afin qu'il puisse gérer tout besoin de vérification directement sans boucles via le coordinateur.
- B) Faire accumuler par l'agent de synthèse tous les besoins de vérification et les renvoyer en lot au coordinateur à la fin, qui les envoie alors tous à la fois à l'agent de recherche web.
- C) Faire mettre en cache de manière proactive par l'agent de recherche web du contexte supplémentaire autour de chaque source pendant la recherche initiale, en prévision du besoin de vérification de la synthèse.
- D) Donner à l'agent de synthèse un outil `verify_fact` à portée limitée pour les vérifications simples, tout en routant les vérifications complexes via le coordinateur vers l'agent de recherche web. **[CORRECT]**

Pourquoi D : Un outil de vérification de faits à portée limitée permet à l'agent de synthèse de gérer 85 % des vérifications simples directement, éliminant la plupart des boucles, tout en préservant le chemin de délégation par le coordinateur pour les 15 % de vérifications complexes. Cela applique le moindre privilège tout en réduisant significativement la latence.

Scénario : Claude Code pour l'intégration continue

Question 16 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre pipeline CI exécute le CLI Claude Code (en mode `--print`) en utilisant CLAUDE.md pour fournir le contexte du projet pour la revue de code, et les développeurs trouvent généralement les revues substantielles. Cependant, ils signalent que l'intégration des conclusions dans le workflow est difficile — Claude produit des paragraphes narratifs qui doivent être copiés manuellement dans les commentaires de PR. L'équipe veut publier automatiquement chaque conclusion comme un commentaire inline distinct au bon endroit dans le code, ce qui nécessite des données structurées avec chemin de fichier, numéro de ligne, niveau de gravité et correction suggérée. Quelle approche est la plus efficace ?

Quelle approche est la plus efficace ?

- A) Ajouter une section « Output Format for Review » à CLAUDE.md avec des exemples de conclusions structurées pour que Claude apprenne le format attendu à partir du contexte du projet.
- B) Utiliser les drapeaux CLI `--output-format json` et `--json-schema` pour imposer des conclusions structurées, puis analyser la sortie pour publier des commentaires inline via l'API GitHub. **[CORRECT]**
- C) Inclure des instructions de formatage explicites dans le prompt de revue exigeant que chaque conclusion suive un modèle analysable comme `[FILE:path] [LINE:n] [SEVERITY:level] ...`.
- D) Garder le format de revue narratif mais ajouter une étape de résumé qui utilise Claude pour générer un résumé JSON structuré des conclusions.

Pourquoi B : Utiliser `--output-format json` avec `--json-schema` impose une sortie structurée au niveau du CLI, garantissant un JSON bien formé avec les champs requis (chemin de fichier, numéro de ligne, gravité, correction suggérée) qui peut être analysé de manière fiable et publié comme commentaire inline sur la PR via l'API GitHub. Cela exploite des capacités CLI intégrées conçues spécifiquement pour la sortie structurée.

Question 17 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre équipe utilise Claude Code pour générer des suggestions de code, mais vous remarquez un motif : les problèmes non évidents — optimisations de performance qui cassent des cas limites, nettoyages qui modifient le comportement de manière inattendue — ne sont détectés que lorsqu'un autre membre de l'équipe révise la PR. Le raisonnement de Claude pendant la génération montre qu'il a considéré ces cas mais a conclu que son approche était correcte. Quelle approche traite directement la cause racine de cette limite d'auto-vérification ?

Quelle approche traite directement la cause racine ?

- A) Exécuter une deuxième instance indépendante de Claude Code pour réviser les changements sans accès au raisonnement du générateur. **[CORRECT]**
- B) Activer le mode de réflexion étendue (extended thinking) pour l'étape de génération afin de permettre une délibération plus approfondie avant de produire des suggestions.
- C) Ajouter des instructions d'auto-revue explicites au prompt de génération demandant à Claude de critiquer ses propres suggestions avant de finaliser la sortie.
- D) Inclure les fichiers de test complets et la documentation dans le contexte du prompt pour que Claude comprenne mieux le comportement attendu pendant la génération.

Pourquoi A : Une deuxième instance Claude Code indépendante sans accès au raisonnement du générateur traite directement la cause racine en évitant le biais de confirmation. Cette perspective « œil neuf » reflète la revue par les pairs chez les humains, où un autre relecteur détecte les problèmes que l'auteur a rationalisés.

Question 18 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre composant de revue de code est itératif : Claude analyse le fichier modifié, puis peut demander des fichiers connexes (imports, classes de base, tests) via des appels d'outils pour comprendre le contexte avant de fournir un retour final. Votre application définit un outil qui permet à Claude de demander le contenu des fichiers ; Claude appelle l'outil, obtient les résultats et poursuit l'analyse. Vous évaluez le traitement par lots pour réduire le coût d'API. Quelle est la principale limite technique à considérer pour le traitement par lots de ce workflow ?

Quelle est la principale limite technique ?

- A) Le traitement par lots n'inclut pas d'identifiants de corrélation pour faire correspondre les sorties aux requêtes d'entrée.
- B) Le modèle asynchrone ne peut pas exécuter des outils en cours de requête et renvoyer les résultats pour que Claude poursuive l'analyse. **[CORRECT]**
- C) La Batch API ne prend pas en charge les définitions d'outils dans les paramètres de requête.
- D) La latence de traitement par lots pouvant atteindre 24 heures est trop lente pour le retour sur les pull requests, bien que le workflow fonctionnerait par ailleurs.

Pourquoi B : Un modèle de Batch API asynchrone « fire-and-forget » n'a aucun mécanisme pour intercepter un appel d'outil pendant une requête, exécuter l'outil et renvoyer les résultats pour que Claude poursuive l'analyse. C'est fondamentalement incompatible avec les workflows d'appels d'outils itératifs qui nécessitent plusieurs tours requête/réponse d'outils au sein d'une seule interaction logique.

Question 19 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre système CI/CD exécute trois analyses basées sur Claude : (1) des vérifications de style rapides sur chaque PR qui bloquent la fusion jusqu'à leur achèvement, (2) des audits de sécurité hebdomadaires complets de toute la base de code, et (3) une génération nocturne de cas de test pour les modules récemment modifiés. La Message Batches API offre 50 % d'économies mais le traitement peut prendre jusqu'à 24 heures. Vous voulez optimiser le coût d'API tout en maintenant une expérience développeur acceptable. Quelle combinaison fait correspondre correctement chaque tâche à une approche d'API ?

Quelle combinaison est correcte ?

- A) Utiliser la Message Batches API pour les trois tâches afin de maximiser les 50 % d'économies, en configurant le pipeline pour interroger l'achèvement des lots.
- B) Utiliser des appels synchrones pour les vérifications de style de PR ; utiliser la Message Batches API pour les audits de sécurité hebdomadaires et la génération nocturne de tests. **[CORRECT]**
- C) Utiliser des appels synchrones pour les trois tâches afin d'obtenir des temps de réponse cohérents, en s'appuyant sur la mise en cache de prompts pour réduire les coûts sur toutes les charges de travail.
- D) Utiliser des appels synchrones pour les vérifications de style de PR et la génération nocturne de tests ; utiliser la Message Batches API uniquement pour les audits de sécurité hebdomadaires.

Pourquoi B : Les vérifications de style de PR bloquent les développeurs et nécessitent des réponses immédiates via des appels synchrones, tandis que les audits de sécurité hebdomadaires et la génération

nocturne de tests sont des tâches planifiées avec des échéances flexibles qui peuvent tolérer une fenêtre de lot pouvant atteindre 24 heures — captant 50 % d'économies pour les deux.

Question 20 (Scénario : Claude Code pour l'intégration continue)

Situation : Vos revues automatisées trouvent de vrais problèmes, mais les développeurs signalent que le retour n'est pas exploitable. Les conclusions incluent des phrases comme « logique complexe de routage de tickets » ou « pointeur null potentiel » sans préciser quoi changer exactement. Lorsque vous ajoutez des instructions détaillées comme « inclure toujours des suggestions de correction concrètes », le modèle produit toujours une sortie incohérente — parfois détaillée, parfois vague. Quelle technique de prompting produit le plus fiablement un retour systématiquement exploitable ?

Quelle technique de prompting est la plus fiable ?

- A) Affiner davantage les instructions avec des exigences plus explicites pour chaque partie du format de retour (emplacement, problème, gravité, correction proposée).
- B) Étendre la fenêtre de contexte pour inclure davantage de code environnant afin que le modèle dispose de suffisamment d'informations pour proposer des corrections concrètes.
- C) Implémenter une approche à deux passes où un prompt identifie les problèmes et un second génère les corrections, permettant la spécialisation.
- D) Ajouter 3 à 4 exemples few-shot montrant le format exact requis : problème identifié, emplacement dans le code, suggestion de correction concrète. **[CORRECT]**

Pourquoi D : Les exemples few-shot sont la technique la plus efficace pour obtenir un format de sortie cohérent lorsque les instructions seules produisent des résultats variables. Fournir 3 à 4 exemples qui montrent la structure exacte souhaitée (problème, emplacement, correction concrète) donne au modèle un motif concret à suivre, ce qui est plus fiable que des instructions abstraites.

Question 21 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre pipeline CI inclut deux modes de revue de code basés sur Claude : un hook de pré-commit-fusion qui bloque la fusion de la PR jusqu'à son achèvement, et une « analyse approfondie » qui s'exécute la nuit, interroge l'achèvement du lot et publie des suggestions détaillées sur la PR. Vous voulez réduire le coût d'API en utilisant la Message Batches API, qui offre 50 % d'économies mais nécessite du polling et peut prendre jusqu'à 24 heures. Quel mode devrait utiliser le traitement par lots ?

Quel mode devrait utiliser le traitement par lots ?

- A) Uniquement le hook de pré-commit-fusion.
- B) Uniquement l'analyse approfondie. **[CORRECT]**
- C) Les deux modes.
- D) Aucun des deux modes.

Pourquoi B : L'analyse approfondie est un candidat idéal pour le traitement par lots car elle s'exécute déjà la nuit, tolère le délai et utilise un modèle de polling avant de publier les résultats — correspondant à

l'architecture asynchrone basée sur le polling de la Message Batches API tout en captant 50 % d'économies.

Question 22 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre revue automatisée analyse les commentaires et les docstrings. Le prompt actuel demande à Claude de « vérifier que les commentaires sont exacts et à jour ». Les conclusions signalent souvent des motifs acceptables (marqueurs TODO, descriptions simples) tout en manquant les commentaires décrivant un comportement que le code n'implémente plus. Quel changement traite la cause racine de cette analyse incohérente ?

Quel changement traite la cause racine ?

- A) Inclure les données `git blame` pour que Claude puisse identifier les commentaires antérieurs aux changements de code récents.
- B) Ajouter des exemples few-shot de commentaires trompeurs pour aider le modèle à reconnaître des motifs similaires dans la base de code.
- C) Filtrer les motifs de commentaires TODO, FIXME et descriptifs avant l'analyse pour réduire le bruit.
- D) Spécifier des critères explicites : signaler les commentaires uniquement lorsque le comportement qu'ils affirment contredit le comportement réel du code. **[CORRECT]**

Pourquoi D : Des critères explicites — signaler les commentaires uniquement lorsque le comportement affirmé contredit le comportement réel du code — traitent directement la cause racine en remplaçant une instruction vague par une définition précise de ce qui constitue un problème. Cela réduit les faux positifs sur les motifs acceptables et les omissions de commentaires véritablement trompeurs.

Question 23 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre système de revue de code automatisée présente des évaluations de gravité incohérentes — des problèmes similaires comme les risques de pointeur null sont notés « critique » dans certaines PR mais seulement « moyen » dans d'autres. Les enquêtes auprès des développeurs montrent une méfiance croissante — beaucoup commencent à rejeter les conclusions sans les lire parce que « la moitié sont fausses ». Les catégories à fort taux de faux positifs érodent la confiance dans les catégories précises. Quelle approche restaure le mieux la confiance des développeurs tout en améliorant le système ?

Quelle approche restaure le mieux la confiance des développeurs ?

- A) Désactiver temporairement les catégories à fort taux de faux positifs (style, nommage, documentation) et ne garder que les catégories de haute précision tout en améliorant les prompts. **[CORRECT]**
- B) Garder toutes les catégories activées mais afficher des scores de confiance avec chaque conclusion pour que les développeurs puissent décider quoi investiguer.
- C) Garder toutes les catégories activées et ajouter des exemples few-shot pour améliorer la précision de chaque catégorie au cours des prochaines semaines.

- D) Appliquer une réduction de sévérité uniforme à toutes les catégories pour faire baisser le taux global de faux positifs.

Pourquoi A : Désactiver temporairement les catégories à fort taux de faux positifs arrête immédiatement l'érosion de la confiance en supprimant les conclusions bruyantes qui poussent les développeurs à tout rejeter, tout en préservant la valeur des catégories de haute précision comme la sécurité et l'exactitude. Cela crée aussi un espace pour améliorer les prompts des catégories problématiques avant de les réactiver.

Question 24 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre revue automatisée génère des suggestions de cas de test pour chaque PR. En révisant une PR qui ajoute le suivi de l'achèvement des cours, Claude suggère 10 cas de test, mais le retour des développeurs montre que 6 dupliquent des scénarios déjà couverts par la suite de tests existante. Quel changement réduit le plus efficacement les suggestions en double ?

Quel changement est le plus efficace ?

- A) Inclure le fichier de test existant dans le contexte pour que Claude puisse déterminer quels scénarios sont déjà couverts. **[CORRECT]**
- B) Réduire le nombre de suggestions demandées de 10 à 5, en supposant que Claude priorise d'abord les cas les plus précieux.
- C) Ajouter des instructions demandant à Claude de se concentrer exclusivement sur les cas limites et les conditions d'erreur plutôt que sur les chemins de réussite.
- D) Implémenter un post-traitement qui filtre les suggestions dont les descriptions correspondent aux noms de tests existants via le chevauchement de mots-clés.

Pourquoi A : Inclure le fichier de test existant corrige la cause racine de la duplication : Claude ne peut éviter de suggérer des scénarios déjà couverts que s'il sait quels tests existent déjà. Cela donne à Claude l'information nécessaire pour proposer des tests véritablement nouveaux et précieux.

Question 25 (Scénario : Claude Code pour l'intégration continue)

Situation : Après qu'une revue automatisée initiale a identifié 12 conclusions, un développeur pousse de nouveaux commits pour traiter les problèmes. Relancer la revue produit 8 conclusions, mais les développeurs signalent que 5 dupliquent des commentaires précédents sur du code déjà corrigé dans les nouveaux commits. Quelle est la manière la plus efficace d'éliminer ce retour redondant tout en maintenant la rigueur ?

Quelle est la manière la plus efficace d'éliminer le retour redondant ?

- A) Exécuter la revue uniquement à la création de la PR et dans l'état final avant fusion, en sautant les commits intermédiaires.
- B) Ajouter un filtre de post-traitement qui supprime les conclusions correspondant aux précédentes par chemins de fichiers et descriptions de problèmes avant de publier les commentaires.

- C) Restreindre la portée de la revue aux fichiers modifiés dans le dernier push, en excluant les fichiers des commits antérieurs.
- D) Inclure les conclusions de revue précédentes dans le contexte et demander à Claude de ne signaler que les problèmes nouveaux ou encore non résolus. **[CORRECT]**

Pourquoi D : Inclure les conclusions de revue antérieures dans le contexte permet à Claude de distinguer les nouveaux problèmes de ceux déjà traités dans les commits récents. Cela préserve la rigueur de la revue tout en utilisant le raisonnement de Claude pour éviter un retour redondant sur du code corrigé.

Question 26 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre script de pipeline exécute `claude "Analyze this pull request for security issues"`, mais le job se bloque indéfiniment. Les logs montrent que Claude Code attend une entrée interactive. Quelle est l'approche correcte pour exécuter Claude Code dans un pipeline automatisé ?

Quelle est l'approche correcte ?

- A) Ajouter un drapeau `--batch` : `claude --batch "Analyze this pull request for security issues"`.
- B) Ajouter le drapeau `-p` : `claude -p "Analyze this pull request for security issues"`. **[CORRECT]**
- C) Rediriger stdin depuis `/dev/null` : `claude "Analyze this pull request for security issues" < /dev/null`.
- D) Définir la variable d'environnement `CLAUDE_HEADLESS=true` avant d'exécuter la commande.

Pourquoi B : Le drapeau `-p` (ou `--print`) est la façon documentée d'exécuter Claude Code de manière non interactive. Il traite le prompt, affiche le résultat sur stdout et quitte sans attendre d'entrée de l'utilisateur — idéal pour les pipelines CI/CD.

Question 27 (Scénario : Claude Code pour l'intégration continue)

Situation : Une pull request modifie 14 fichiers dans un module de suivi des stocks. Une revue en une seule passe qui analyse tous les fichiers ensemble produit des résultats incohérents : retour détaillé sur certains fichiers mais commentaires superficiels sur d'autres, bugs évidents manqués et retours contradictoires (un motif est signalé dans un fichier mais un code identique est approuvé dans un autre fichier de la même PR). Comment restructurer la revue ?

Comment restructurer la revue ?

- A) Exécuter trois passes de revue complètes indépendantes et ne signaler que les problèmes apparaissant dans au moins deux des trois exécutions.
- B) Diviser en passes ciblées : réviser chaque fichier individuellement pour les problèmes locaux, puis effectuer une passe orientée intégration séparée pour examiner les flux de données inter-fichiers. **[CORRECT]**

- C) Exiger des développeurs qu'ils divisent les grandes PR en soumissions plus petites de 3 à 4 fichiers avant d'exécuter la revue automatisée.
- D) Passer à un modèle plus grand avec une fenêtre de contexte plus importante pour qu'il puisse prêter une attention suffisante à tous les 14 fichiers en une seule passe.

Pourquoi B : Les passes ciblées par fichier traitent la cause racine — la dilution de l'attention — en garantissant une profondeur cohérente et une détection fiable des problèmes locaux. Une passe orientée intégration séparée couvre ensuite les préoccupations inter-fichiers comme les interactions de dépendances et de flux de données.

Question 28 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre revue de code automatisée produit en moyenne 15 conclusions par pull request, et les développeurs signalent un taux de faux positifs de 40 %. Le goulot d'étranglement est le temps d'investigation : les développeurs doivent cliquer sur chaque conclusion pour lire la justification de Claude avant de décider de corriger ou de rejeter. Votre CLAUDE.md contient déjà des règles complètes pour les motifs acceptables, et les parties prenantes ont rejeté toute approche qui filtre les conclusions avant que les développeurs ne les voient. Quel changement traite le mieux le temps d'investigation ?

Quel changement traite le mieux le temps d'investigation ?

- A) Exiger de Claude qu'il inclue sa justification et une estimation de confiance directement dans chaque conclusion. **[CORRECT]**
- B) Ajouter un post-processeur qui analyse les motifs de conclusions et supprime automatiquement ceux qui correspondent à des signatures historiques de faux positifs.
- C) Catégoriser les conclusions en « problèmes bloquants » vs « suggestions », avec des exigences de revue différentes par niveau.
- D) Configurer Claude pour n'afficher que les conclusions à haute confiance, en filtrant les signalements incertains avant que les développeurs ne les voient.

Pourquoi A : Inclure la justification et la confiance directement dans chaque conclusion réduit le temps d'investigation en permettant aux développeurs de trier rapidement sans ouvrir chaque conclusion. Cela satisfait la contrainte « pas de filtrage » car toutes les conclusions restent visibles tout en accélérant la prise de décision des développeurs.

Question 29 (Scénario : Claude Code pour l'intégration continue)

Situation : L'analyse de votre revue de code automatisée montre de grandes différences de taux de faux positifs par catégorie de conclusion : les conclusions de sécurité/exactitude ont 8 % de faux positifs, les conclusions de performance 18 %, les conclusions de style/nommage 52 %, et les conclusions de documentation 48 %. Les enquêtes auprès des développeurs montrent une méfiance croissante — beaucoup commencent à rejeter les conclusions sans les lire parce que « la moitié sont fausses ». Les catégories à fort taux de faux positifs érodent la confiance dans les catégories précises. Quelle approche restaure le mieux la confiance des développeurs tout en améliorant le système ?

Quelle approche restaure le mieux la confiance des développeurs ?

- A) Désactiver temporairement les catégories à fort taux de faux positifs (style, nommage, documentation) et ne garder que les catégories de haute précision tout en améliorant les prompts. **[CORRECT]**
- B) Garder toutes les catégories activées mais afficher des scores de confiance avec chaque conclusion pour que les développeurs puissent décider quoi investiguer.
- C) Garder toutes les catégories activées et ajouter des exemples few-shot pour améliorer la précision de chaque catégorie au cours des prochaines semaines.
- D) Appliquer une réduction de sévérité uniforme à toutes les catégories pour faire baisser le taux global de faux positifs.

Pourquoi A : Désactiver temporairement les catégories à fort taux de faux positifs arrête immédiatement l'érosion de la confiance en supprimant les conclusions bruyantes qui poussent les développeurs à tout rejeter, tout en préservant la valeur des catégories de haute précision comme la sécurité et l'exactitude. Cela crée aussi un espace pour améliorer les prompts des catégories problématiques avant de les réactiver.

Question 30 (Scénario : Claude Code pour l'intégration continue)

Situation : Votre équipe veut réduire les coûts d'API pour l'analyse automatisée. Actuellement, des appels Claude synchrones prennent en charge deux workflows : (1) une vérification bloquante avant fusion qui doit se terminer avant que les développeurs puissent fusionner, et (2) un rapport de dette technique généré la nuit pour revue le lendemain matin. Votre manager propose de déplacer les deux vers la Message Batches API pour économiser 50 %. Comment évaluer cette proposition ?

Comment évaluer cette proposition ?

- A) Déplacer les deux vers le traitement par lots avec un repli vers les appels synchrones si les lots prennent trop de temps.
- B) Déplacer les deux workflows vers le traitement par lots avec polling de statut pour vérifier l'achèvement.
- C) Utiliser le traitement par lots uniquement pour les rapports de dette technique ; garder les appels synchrones pour les vérifications avant fusion. **[CORRECT]**
- D) Garder les appels synchrones pour les deux workflows afin d'éviter les problèmes d'ordre des résultats de lot.

Pourquoi C : Le traitement de la Message Batches API peut prendre jusqu'à 24 heures sans SLA de latence, ce qui est acceptable pour les rapports de dette technique de nuit mais inacceptable pour les vérifications bloquantes avant fusion où les développeurs attendent. Cela fait correspondre chaque workflow à la bonne API en fonction des exigences de latence.

Scénario : Génération de code avec Claude Code

Question 31 (Scénario : Génération de code avec Claude Code)

Situation : Vous avez demandé à Claude Code d'implémenter une fonction qui transforme les réponses d'API en un format normalisé interne. Après deux itérations, la structure de sortie ne correspond toujours pas aux attentes — certains champs sont imbriqués différemment et les timestamps sont mal formatés. Vous avez décrit les exigences en prose, mais Claude les interprète différemment à chaque fois.

Quelle approche est la plus efficace pour la prochaine itération ?

- A) Écrire un schéma JSON décrivant la structure de sortie attendue et valider la sortie de Claude par rapport à celui-ci après chaque itération.
- B) Fournir 2 à 3 exemples concrets entrée-sortie montrant la transformation attendue pour des réponses d'API représentatives. **[CORRECT]**
- C) Réécrire les exigences avec plus de précision technique, en spécifiant les mappages de champs exacts, les règles d'imbrication et les chaînes de format de timestamp.
- D) Demander à Claude d'expliquer sa compréhension actuelle des exigences pour identifier où les interprétations divergent.

Pourquoi B : Les exemples concrets entrée-sortie suppriment l'ambiguïté inhérente aux descriptions en prose en montrant à Claude les résultats de transformation exacts attendus. Cela traite directement la cause racine — la mauvaise interprétation des exigences textuelles — en fournissant des motifs sans ambiguïté pour l'imbrication des champs et le formatage des timestamps.

Question 32 (Scénario : Génération de code avec Claude Code)

Situation : Vous devez ajouter Slack comme nouveau canal de notification. La base de code existante a des motifs clairs et établis pour les canaux email, SMS et push. Cependant, l'API de Slack offre des approches d'intégration fondamentalement différentes — webhooks entrants (simples, unidirectionnels), tokens de bot (prennent en charge la confirmation de livraison et le contrôle programmatique) ou Slack Apps (événements bidirectionnels, nécessite l'approbation de l'espace de travail). Votre tâche dit « ajouter le support de Slack » sans spécifier de méthode d'intégration ni exiger de fonctionnalités avancées comme le suivi de livraison.

Comment aborder cette tâche ?

- A) Démarrer en mode d'exécution directe en utilisant des webhooks entrants pour correspondre au motif de notification unidirectionnel existant.
- B) Passer en mode planification pour explorer les options d'intégration et les implications architecturales, puis présenter une recommandation avant l'implémentation. **[CORRECT]**
- C) Démarrer en mode d'exécution directe en échafaudant une classe de canal Slack à partir des motifs existants, en différant la décision de méthode d'intégration.
- D) Démarrer en mode d'exécution directe en utilisant une approche par token de bot pour garantir que la confirmation de livraison est possible.

Pourquoi B : L'intégration de Slack a plusieurs approches valides aux implications architecturales très différentes, et les exigences sont ambiguës. Le mode planification vous permet d'évaluer les compromis entre webhooks, tokens de bot et Slack Apps et de vous aligner sur une approche avant l'implémentation.

Question 33 (Scénario : Génération de code avec Claude Code)

Situation : Votre fichier CLAUDE.md a dépassé 400 lignes contenant des normes de codage, des conventions de test, une liste de contrôle détaillée de revue de PR, des instructions de déploiement et des procédures de migration de base de données. Vous voulez que Claude suive toujours les normes de codage et les conventions de test, mais n'applique le guidage de revue de PR, de déploiement et de migration que lors de ces tâches.

Quelle approche de restructuration est la plus efficace ?

- A) Déplacer tout le guidage dans des fichiers Skills séparés organisés par type de workflow, ne laissant qu'une brève description du projet dans CLAUDE.md.
- B) Garder tout dans CLAUDE.md mais utiliser la syntaxe `@import` pour organiser en fichiers maintenus séparément par catégorie.
- C) Diviser CLAUDE.md en fichiers sous `.claude/rules/` avec des motifs glob liés au chemin afin que chaque règle ne se charge que pour les types de fichiers pertinents.
- D) Garder les normes universelles dans CLAUDE.md et créer des Skills pour le guidage spécifique au workflow (revue de PR, déploiement, migrations) avec des mots-clés déclencheurs. **[CORRECT]**

Pourquoi D : Le contenu de CLAUDE.md se charge à chaque session, garantissant que les normes de codage et les conventions de test s'appliquent toujours, tandis que les Skills sont invoqués à la demande lorsque Claude détecte des mots-clés déclencheurs — idéal pour le guidage spécifique au workflow comme la revue de PR, le déploiement et les migrations.

Question 34 (Scénario : Génération de code avec Claude Code)

Situation : Vous êtes chargé de restructurer l'application monolithique de votre équipe en microservices. Cela implique des changements dans des dizaines de fichiers et nécessite des décisions sur les frontières de service et les dépendances de modules.

Quelle approche choisir ?

- A) Passer en mode planification pour explorer la base de code, comprendre les dépendances et concevoir l'approche d'implémentation avant d'apporter des modifications. **[CORRECT]**
- B) Démarrer en mode d'exécution directe et passer à la planification seulement après avoir rencontré une complexité inattendue pendant l'implémentation.
- C) Démarrer en mode d'exécution directe et apporter des changements incrémentaux, en laissant l'implémentation révéler les frontières de service naturelles.
- D) Utiliser l'exécution directe avec des instructions détaillées en amont qui spécifient la structure de chaque service.

Pourquoi A : Le mode planification est la bonne stratégie pour une restructuration architecturale complexe comme la division d'un monolithe : il permet une exploration sûre et des décisions éclairées sur les frontières avant de s'engager dans des changements potentiellement coûteux sur de nombreux fichiers.

Question 35 (Scénario : Génération de code avec Claude Code)

Situation : Votre équipe a créé un skill `/analyze-codebase` qui effectue une analyse de code approfondie — scan des dépendances, comptage de la couverture de tests et métriques de qualité du code. Après l'exécution de la commande, les membres de l'équipe signalent que Claude devient moins réactif dans la session et perd le contexte de la tâche initiale.

Comment corriger cela le plus efficacement tout en conservant des capacités d'analyse complètes ?

- A) Ajouter `context: fork` dans le frontmatter du skill pour exécuter l'analyse dans un contexte de sous-agent isolé. **[CORRECT]**
- B) Ajouter `model: haiku` dans le frontmatter pour utiliser un modèle plus rapide et moins cher pour l'analyse.
- C) Diviser le skill en trois skills plus petits, chacun produisant moins de sortie.
- D) Ajouter des instructions au skill pour compresser tous les résultats en un court résumé avant de les afficher.

Pourquoi A : `context: fork` exécute l'analyse dans un contexte de sous-agent isolé afin que la grande sortie ne pollue pas la fenêtre de contexte de la session principale et que Claude ne perde pas le fil de la tâche initiale. Cela préserve la capacité d'analyse complète tout en gardant la session principale réactive.

Question 36 (Scénario : Génération de code avec Claude Code)

Situation : Votre équipe utilise un skill `/commit` dans `.claude/skills/commit/SKILL.md`. Un développeur veut le personnaliser pour son workflow personnel (format de message de commit différent, vérifications supplémentaires) sans affecter ses coéquipiers.

Que recommandez-vous ?

- A) Créer une version personnelle sous `~/.claude/skills/` avec un nom différent, p. ex. `/my-commit`. **[CORRECT]**
- B) Ajouter une logique conditionnelle basée sur le nom d'utilisateur dans le frontmatter du skill de projet.
- C) Créer une version personnelle dans `~/.claude/skills/commit/SKILL.md` avec le même nom.
- D) Définir `override: true` dans le frontmatter du skill personnel pour le prioriser sur la version de projet.

Pourquoi A : Les skills personnels ont la priorité sur les skills de projet portant le même nom. Réutiliser le nom `commit` masquerait donc silencieusement le skill de l'équipe pour ce seul développeur — il cesserait de recevoir les mises à jour à chaque amélioration de `/commit` par l'équipe, et devrait se rappeler qu'il exécute un skill différent sous la même commande. Nommer la variante personnelle `/my-commit` évite entièrement ce conflit : le développeur continue d'utiliser le `/commit` maintenu par l'équipe et dispose d'un skill séparé, clairement nommé, pour son workflow personnel, sans risque de confusion ni de mise à jour manquée.

Question 37 (Scénario : Génération de code avec Claude Code)

Situation : Votre équipe utilise Claude Code depuis des mois. Récemment, trois développeurs signalent que Claude suit le guidage « toujours inclure une gestion d'erreurs complète », mais un quatrième développeur qui vient de rejoindre dit que Claude ne le suit pas. Tous les quatre travaillent dans le même dépôt et ont du code à jour.

Quelle est la cause la plus probable et la correction ?

- A) Le guidage se trouve dans les fichiers `~/.claude/CLAUDE.md` au niveau utilisateur des développeurs d'origine, pas dans le `.claude/CLAUDE.md` du projet. Déplacer l'instruction vers le fichier de niveau projet pour que tous les membres de l'équipe la reçoivent. **[CORRECT]**
- B) Le `~/.claude/CLAUDE.md` du nouveau développeur contient des instructions contradictoires qui remplacent les paramètres du projet ; il devrait supprimer la section contradictoire.
- C) Claude Code apprend les préférences par utilisateur au fil du temps ; le nouveau développeur doit répéter l'exigence jusqu'à ce que Claude « s'en souvienne ».
- D) Claude Code met en cache CLAUDE.md après la première lecture ; les développeurs d'origine utilisent les versions en cache. Tout le monde devrait vider le cache de Claude Code.

Pourquoi A : Si le guidage n'a été ajouté qu'aux configurations de niveau utilisateur des développeurs d'origine et non au `.claude/CLAUDE.md` de niveau projet, les nouveaux membres de l'équipe ne le recevront pas. Le déplacer vers la configuration de niveau projet garantit que tous les membres actuels et futurs de l'équipe l'obtiennent automatiquement.

Question 38 (Scénario : Génération de code avec Claude Code)

Situation : Vous constatez qu'inclure 2 à 3 exemples complets d'implémentation de endpoints comme contexte améliore significativement la cohérence lors de la génération de nouveaux endpoints d'API. Cependant, ce contexte n'est utile que lors de la création de nouveaux endpoints — pas lors du débogage, de la revue de code ou d'autres travaux dans le répertoire de l'API.

Quelle approche de configuration est la plus efficace ?

- A) Ajouter les exemples de endpoints et la documentation des motifs au CLAUDE.md du projet afin qu'ils soient toujours disponibles.
- B) Référencer manuellement les exemples de endpoints dans chaque requête de génération en copiant le code dans le prompt.
- C) Configurer des règles spécifiques au chemin dans `.claude/rules/api/` qui incluent les exemples de endpoints et s'activent lors du travail dans le répertoire de l'API.
- D) Créer un skill qui référence les exemples de endpoints et contient des instructions de suivi de motifs, invoqué à la demande via une commande slash. **[CORRECT]**

Pourquoi D : Un skill invoqué à la demande charge le contexte d'exemple uniquement lors de la génération de nouveaux endpoints, pas pendant des tâches non liées comme le débogage ou la revue. Cela garde le contexte principal propre tout en préservant une génération de haute qualité quand c'est nécessaire.

Question 39 (Scénario : Génération de code avec Claude Code)

Situation : Votre équipe a créé un skill `/migration` qui génère des fichiers de migration de base de données. Il prend le nom de migration via `$ARGUMENTS`. En production, vous observez trois problèmes : (1) les développeurs exécutent souvent le skill sans arguments, causant des fichiers mal nommés, (2) le skill utilise parfois des détails de schéma de base de données provenant de conversations antérieures non liées, et (3) un développeur a accidentellement exécuté un nettoyage de test destructif lorsque le skill avait un accès large aux outils.

Quelle approche de configuration corrige les trois problèmes ?

- A) Utiliser les paramètres positionnels `$1` et `$2` au lieu de `$ARGUMENTS` pour imposer des entrées précises, inclure des références explicites aux fichiers de schéma via la syntaxe `@` pour le contrôle du contexte, et ajouter une description de frontmatter avertissant des opérations destructives.
- B) Ajouter `argument-hint` dans le frontmatter pour demander les paramètres requis, utiliser `context: fork` pour isoler l'exécution, et restreindre `allowed-tools` aux opérations d'écriture de fichiers. **[CORRECT]**
- C) Diviser en skills `/migration-create` et `/migration-apply`, ajouter des instructions de validation pour demander le nom de migration s'il manque, et utiliser des portées `allowed-tools` différentes pour chacun.
- D) Ajouter des instructions de validation dans le SKILL.md du skill pour garantir que `$ARGUMENTS` est un nom valide, ajouter des prompts pour ignorer le contexte de conversation antérieur, et lister les opérations interdites à éviter.

Pourquoi B : Cela utilise trois fonctionnalités de configuration distinctes pour traiter chaque problème : `argument-hint` améliore la saisie d'arguments et réduit les arguments manquants, `context: fork` empêche la fuite de contexte des conversations antérieures, et `allowed-tools` contraint le skill à des opérations sûres d'écriture de fichiers, empêchant les actions destructives.

Question 40 (Scénario : Génération de code avec Claude Code)

Situation : Votre base de code contient des zones avec des conventions de codage différentes : les composants React utilisent un style fonctionnel avec hooks, les gestionnaires d'API utilisent async/await avec une gestion d'erreurs spécifique, et les modèles de base de données suivent le pattern repository. Les fichiers de test sont distribués dans la base de code à côté du code testé (p. ex. `Button.test.tsx` à côté de `Button.tsx`), et vous voulez que tous les tests suivent les mêmes conventions quel que soit leur emplacement.

Quelle est la manière la mieux prise en charge pour garantir que Claude applique automatiquement les bonnes conventions lors de la génération de code ?

- A) Mettre toutes les conventions dans le CLAUDE.md racine sous des en-têtes pour chaque zone et compter sur Claude pour inférer quelle section s'applique.
- B) Créer des skills dans `.claude/skills/` pour chaque type de code, en intégrant les conventions dans chaque SKILL.md.
- C) Placer un fichier CLAUDE.md distinct dans chaque sous-répertoire contenant les conventions de cette zone.

- D) Créer des fichiers de règles sous `.claude/rules/` avec un frontmatter YAML spécifiant des motifs glob pour appliquer conditionnellement les conventions selon les chemins de fichiers. **[CORRECT]**

Pourquoi D : Les fichiers `.claude/rules/` avec frontmatter YAML et motifs glob (p. ex. `**/*.test.tsx`, `src/api/**/*.ts`) permettent une application déterministe et basée sur le chemin des conventions, quelle que soit la structure des répertoires. C'est l'approche la mieux prise en charge pour les motifs transversaux comme les fichiers de test distribués.

Question 41 (Scénario : Génération de code avec Claude Code)

Situation : Vous voulez créer une commande slash personnalisée `/review` qui exécute la liste de contrôle de revue de code standard de votre équipe. Elle devrait être disponible pour chaque développeur lorsqu'il clone ou met à jour le dépôt.

Où créer le fichier de commande ?

- A) Dans `~/.claude/commands/` dans le répertoire personnel de chaque développeur.
- B) Dans le dépôt du projet sous `.claude/commands/`. **[CORRECT]**
- C) Dans `.claude/config.json` comme un tableau de commandes.
- D) Dans le CLAUDE.md racine du projet.

Pourquoi B : Placer les commandes slash personnalisées sous `.claude/commands/` à l'intérieur du dépôt du projet garantit qu'elles sont versionnées et automatiquement disponibles pour chaque développeur qui clone ou met à jour le dépôt. C'est l'emplacement prévu pour les commandes personnalisées de niveau projet dans Claude Code.

Question 42 (Scénario : Génération de code avec Claude Code)

Situation : Le CLAUDE.md de votre équipe a dépassé 500 lignes mêlant conventions TypeScript, guidage de test, motifs d'API et procédures de déploiement. Les développeurs ont du mal à localiser et à mettre à jour les bonnes sections.

Quelle approche Claude Code prend-il en charge pour organiser les instructions de niveau projet en modules thématiques ciblés ?

- A) Définir un fichier de mappage `.claude/config.yaml` reliant les motifs de fichiers à des sections précises dans CLAUDE.md.
- B) Créer des fichiers Markdown séparés dans `.claude/rules/`, chacun couvrant un sujet (p. ex. `testing.md`, `api-conventions.md`). **[CORRECT]**
- C) Diviser les instructions en fichiers README.md dans les sous-répertoires pertinents que Claude charge automatiquement comme instructions.
- D) Créer plusieurs fichiers nommés CLAUDE.md à différents niveaux de l'arborescence des répertoires, chacun remplaçant les instructions parentes.

Pourquoi B : Claude Code prend en charge un répertoire `.claude/rules/` où vous pouvez créer des fichiers Markdown séparés pour le guidage thématique (p. ex. `testing.md`, `api-conventions.md`),

permettant aux équipes d'organiser de grands ensembles d'instructions en modules ciblés et maintenables.

Question 43 (Scénario : Génération de code avec Claude Code)

Situation : Vous créez un skill personnalisé `/explore-alternatives` que votre équipe utilise pour réfléchir et évaluer des approches d'implémentation avant d'en choisir une. Les développeurs signalent qu'après l'exécution du skill, les réponses suivantes de Claude sont influencées par la discussion sur les alternatives — référençant parfois des approches rejetées ou conservant un contexte d'exploration qui interfère avec l'implémentation réelle.

Comment configurer ce skill le plus efficacement ?

- A) Utiliser le préfixe `!` dans le skill pour exécuter la logique d'exploration comme un sous-processus bash.
- B) Ajouter `context: fork` dans le frontmatter du skill. **[CORRECT]**
- C) Diviser en deux skills — `/explore-start` et `/explore-end` — pour marquer les frontières où le contexte d'exploration doit être abandonné.
- D) Créer le skill dans `~/.claude/skills/` au lieu de `.claude/skills/`.

Pourquoi B : `context: fork` exécute le skill dans un contexte de sous-agent isolé afin que les discussions d'exploration ne polluent pas l'historique de conversation principal. Cela empêche les approches rejetées et le contexte de brainstorming d'influencer le travail d'implémentation ultérieur.

Question 44 (Scénario : Génération de code avec Claude Code)

Situation : Votre équipe veut ajouter un serveur MCP GitHub pour rechercher des PR et vérifier le statut CI via Claude Code. Chacun des six développeurs a son propre token d'accès personnel GitHub. Vous voulez un outillage cohérent dans toute l'équipe sans committer d'identifiants dans le contrôle de version.

Quelle approche de configuration est la plus efficace ?

- A) Faire ajouter le serveur par chaque développeur au niveau utilisateur via `claude mcp add --scope user`.
- B) Créer un wrapper de serveur MCP qui lit les tokens depuis un fichier `.env` et relaie les appels à l'API GitHub, puis ajouter le wrapper au `.mcp.json` du projet.
- C) Ajouter le serveur au `.mcp.json` du projet en utilisant la substitution de variables d'environnement (`${GITHUB_TOKEN}`) pour l'authentification et documenter la variable d'environnement requise dans le README du projet. **[CORRECT]**
- D) Configurer le serveur au niveau projet avec un token fictif, puis dire aux développeurs de le remplacer dans leur configuration locale.

Pourquoi C : Un `.mcp.json` de projet avec substitution de variables d'environnement est idiomatique : il fournit une source unique de vérité versionnée pour la configuration MCP tout en laissant chaque

développeur fournir ses identifiants via des variables d'environnement. Documenter la variable facilite l'intégration des nouveaux arrivants sans committer de secrets.

Question 45 (Scénario : Génération de code avec Claude Code)

Situation : Vous ajoutez des wrappers de gestion d'erreurs autour des appels d'API externes dans une base de code de 120 fichiers. Le travail comporte trois phases : (1) découvrir tous les sites d'appel et motifs, (2) concevoir collaborativement l'approche de gestion d'erreurs, et (3) implémenter les wrappers de manière cohérente. En phase 1, Claude génère une grande sortie listant des centaines de sites d'appel avec contexte, remplissant rapidement la fenêtre de contexte avant la fin de la découverte.

Quelle approche est la plus efficace pour accomplir la tâche tout en maintenant la cohérence de l'implémentation ?

- A) Utiliser un sous-agent Explore pour la phase 1 afin d'isoler la sortie verbeuse de découverte et de renvoyer un résumé, puis poursuivre les phases 2–3 dans la conversation principale. **[CORRECT]**
- B) Faire toutes les phases dans la conversation principale, en utilisant périodiquement `/compact` pour réduire l'utilisation du contexte au fil des fichiers.
- C) Passer en mode headless avec `--continue`, en transmettant des résumés de contexte explicites entre les appels par lots pour maintenir la continuité.
- D) Définir le motif de gestion d'erreurs dans CLAUDE.md, puis traiter les fichiers par lots sur plusieurs sessions en s'appuyant sur le fichier de mémoire partagé pour la cohérence.

Pourquoi A : Un sous-agent Explore isole la sortie verbeuse de découverte dans un contexte séparé et ne renvoie qu'un résumé concis à la conversation principale. Cela préserve la fenêtre de contexte principale pour les phases de conception collaborative et d'implémentation cohérente, où le contexte conservé est le plus précieux.

Scénario : Agent de support client

Question 46 (Scénario : Agent de support client)

Situation : Pendant les tests, vous remarquez que l'agent appelle souvent `get_customer` lorsque les utilisateurs posent des questions sur le statut des commandes, même si `lookup_order` serait plus approprié. Que devez-vous vérifier en premier pour traiter ce problème ?

Que devez-vous vérifier en premier ?

- A) Implémenter un classificateur de prétraitement pour détecter les requêtes liées aux commandes et les router directement vers `lookup_order`.
- B) Réduire le nombre d'outils disponibles pour l'agent afin de simplifier le choix.
- C) Ajouter des exemples few-shot au prompt système couvrant tous les motifs possibles de requêtes de commande pour améliorer la sélection d'outils.

- D) Vérifier les descriptions des outils pour s'assurer qu'elles différencient clairement l'objectif de chaque outil. **[CORRECT]**

Pourquoi D : Les descriptions d'outils sont l'entrée principale que le modèle utilise pour décider quel outil appeler. Lorsqu'un agent choisit systématiquement le mauvais outil, la première étape de diagnostic est de vérifier que les descriptions d'outils séparent clairement l'objectif et les limites d'usage de chaque outil.

Question 47 (Scénario : Agent de support client)

Situation : Votre agent gère les demandes à problème unique avec 94 % de précision (p. ex. « j'ai besoin d'un remboursement pour la commande #1234 »). Mais lorsque les clients incluent plusieurs problèmes dans un message (p. ex. « j'ai besoin d'un remboursement pour la commande #1234 et je veux aussi mettre à jour l'adresse de livraison de la commande #5678 »), la précision de sélection d'outils chute à 58 %. L'agent ne résout généralement qu'un problème ou mélange les paramètres entre les requêtes. Quelle approche améliore le plus efficacement la fiabilité pour les demandes à problèmes multiples ?

Quelle approche est la plus efficace ?

- A) Implémenter une couche de prétraitement qui utilise un appel de modèle séparé pour décomposer les messages à problèmes multiples en requêtes distinctes, traiter chacune indépendamment et fusionner les résultats.
- B) Combiner les outils connexes en moins d'outils universels.
- C) Ajouter des exemples few-shot au prompt démontrant le raisonnement correct et la séquence d'outils pour les demandes à problèmes multiples. **[CORRECT]**
- D) Implémenter une validation de réponse qui détecte les réponses incomplètes et re-prompte automatiquement l'agent pour résoudre les problèmes manqués.

Pourquoi C : Les exemples few-shot qui démontrent le raisonnement correct et la séquence d'outils pour les demandes à problèmes multiples sont les plus efficaces car l'agent performe déjà bien sur les problèmes uniques — ce dont il a besoin, c'est d'un guidage sur le motif de décomposition et de routage de plusieurs problèmes tout en gardant les paramètres séparés.

Question 48 (Scénario : Agent de support client)

Situation : Les logs de production montrent que pour les demandes simples comme « remboursement pour la commande #1234 », votre agent résout le problème en 3 à 4 appels d'outils avec 91 % de succès. Mais pour les demandes complexes comme « j'ai été facturé deux fois, ma remise ne s'est pas appliquée, et je veux annuler », l'agent fait en moyenne plus de 12 appels d'outils avec seulement 54 % de succès — investiguant souvent les problèmes séquentiellement et récupérant des données client redondantes pour chacun. Quel changement améliore le plus efficacement la gestion des demandes complexes ?

Quel changement est le plus efficace ?

- A) Ajouter des points de vérification explicites entre les étapes, exigeant que l'agent enregistre la progression après avoir résolu chaque problème avant de passer au suivant.

- B) Réduire le nombre d'outils en combinant `get_customer`, `lookup_order` et les outils liés à la facturation en un seul outil `investigate_issue`.
- C) Décomposer la demande en problèmes distincts, puis investiguer chacun en parallèle en utilisant un contexte client partagé avant de synthétiser une résolution finale. **[CORRECT]**
- D) Ajouter des exemples few-shot au prompt système démontrant des séquences d'appels d'outils idéales pour divers scénarios de facturation à multiples facettes.

Pourquoi C : Décomposer en problèmes distincts et investiguer en parallèle avec un contexte client partagé corrige les deux problèmes clés : cela élimine la récupération de données redondante en réutilisant le contexte partagé entre les problèmes et réduit le nombre total de boucles d'appels d'outils en parallélisant l'investigation avant de synthétiser une résolution unique.

Question 49 (Scénario : Agent de support client)

Situation : Votre agent atteint 55 % de résolution au premier contact, bien en dessous de l'objectif de 80 %. Les logs montrent qu'il escalade des cas simples (remplacements standard pour biens endommagés avec preuve photo) tout en tentant de gérer de manière autonome des situations complexes nécessitant des exceptions de politique. Quelle est la manière la plus efficace d'améliorer le calibrage de l'escalade ?

Quelle est la manière la plus efficace d'améliorer le calibrage de l'escalade ?

- A) Exiger que l'agent auto-évalue sa confiance sur une échelle de 1 à 10 avant chaque réponse et route automatiquement vers les humains lorsque la confiance tombe sous un seuil.
- B) Déployer un modèle classificateur séparé entraîné sur les tickets historiques pour prédire quelles demandes nécessitent une escalade avant que l'agent principal ne commence le traitement.
- C) Ajouter des critères d'escalade explicites au prompt système avec des exemples few-shot montrant quand escalader vs résoudre de manière autonome. **[CORRECT]**
- D) Implémenter une analyse de sentiment pour déterminer le niveau de frustration du client et escalader automatiquement au-delà d'un seuil de sentiment négatif.

Pourquoi C : Des critères d'escalade explicites avec des exemples few-shot traitent directement la cause racine — des frontières de décision floues entre cas simples et complexes. C'est la première intervention la plus proportionnée et efficace, qui enseigne à l'agent quand escalader et quand résoudre de manière autonome sans infrastructure supplémentaire.

Question 50 (Scénario : Agent de support client)

Situation : Après avoir appelé `get_customer` et `lookup_order`, l'agent dispose de toutes les données système disponibles mais fait toujours face à de l'incertitude. Quelle situation est le déclencheur le plus justifié pour appeler `escalate_to_human` ?

Quelle situation est la plus justifiée pour une escalade ?

- A) Un client veut annuler une commande expédiée hier et arrivant demain. L'agent devrait escalader car le client pourrait changer d'avis après réception du colis.
- B) Un client affirme ne pas avoir reçu une commande, mais le suivi indique qu'elle a été livrée et signée à son adresse il y a trois jours. L'agent devrait escalader car présenter des preuves contradictoires pourrait nuire à la relation client.
- C) Un client demande un alignement sur le prix d'un concurrent. Vos politiques autorisent les ajustements de prix pour les baisses de prix sur votre propre site dans les 14 jours, mais ne disent rien des prix des concurrents. L'agent devrait escalader pour interprétation de politique. **[CORRECT]**
- D) Un message client contient à la fois une question de facturation et un retour de produit. L'agent devrait escalader pour qu'un humain coordonne les deux problèmes en une seule interaction.

Pourquoi C : Il s'agit d'une véritable lacune de politique : les règles de l'entreprise couvrent les baisses de prix sur votre propre site mais ne traitent pas de l'alignement sur les prix des concurrents. L'agent ne doit pas inventer de politique et devrait escalader pour un jugement humain sur la façon d'interpréter ou d'étendre les règles existantes.

Question 51 (Scénario : Agent de support client)

Situation : Les logs de production montrent que dans 12 % des cas, votre agent saute `get_customer` et appelle `lookup_order` directement en utilisant uniquement le nom fourni par le client, conduisant parfois à des comptes mal identifiés et à des remboursements incorrects. Quel changement corrige le plus efficacement ce problème de fiabilité ?

Quel changement est le plus efficace ?

- A) Ajouter des exemples few-shot montrant que l'agent appelle toujours `get_customer` en premier, même lorsque les clients fournissent volontairement des détails de commande.
- B) Implémenter un classificateur de routage qui analyse chaque demande et n'active qu'un sous-ensemble d'outils appropriés à ce type de demande.
- C) Ajouter une précondition programmatique qui bloque `lookup_order` et `process_refund` jusqu'à ce que `get_customer` renvoie un identifiant client vérifié. **[CORRECT]**
- D) Renforcer le prompt système en indiquant que la vérification du client via `get_customer` est obligatoire avant toute opération de commande.

Pourquoi C : Une précondition programmatique fournit une garantie déterministe que la séquence requise est suivie. C'est l'approche la plus efficace car elle élimine la possibilité de sauter la vérification, quel que soit le comportement du LLM.

Question 52 (Scénario : Agent de support client)

Situation : Les métriques de production montrent que lors de la résolution de litiges de facturation complexes ou de retours multi-commandes, les scores de satisfaction client sont 15 % plus bas que pour les cas simples — même lorsque la résolution est techniquement correcte. L'analyse de cause racine montre que l'agent fournit des solutions exactes mais explique la justification de manière incohérente : omettant parfois des détails de politique pertinents, manquant parfois des informations de calendrier ou les

étapes suivantes. Les lacunes de contexte spécifiques varient d'un cas à l'autre. Vous voulez améliorer la qualité des solutions sans ajouter de supervision humaine. Quelle approche est la plus efficace ?

Quelle approche est la plus efficace ?

- A) Ajouter une étape d'auto-critique où l'agent évalue un brouillon de réponse pour sa complétude — en s'assurant qu'il résout le problème du client, inclut le contexte pertinent et anticipe les questions de suivi. **[CORRECT]**
- B) Ajouter une étape de confirmation où l'agent demande « Cela résout-il complètement votre problème ? » avant de clore, permettant aux clients de demander des informations supplémentaires si nécessaire.
- C) Mettre à niveau le modèle de Haiku vers Sonnet pour les cas complexes, en routant selon une métrique de complexité définie.
- D) Implémenter des exemples few-shot dans le prompt système montrant des explications complètes pour cinq types courants de cas complexes, démontrant comment inclure le contexte de politique, les calendriers et les étapes suivantes.

Pourquoi A : Une étape d'auto-critique (le pattern évaluateur-optimiseur) traite directement la complétude incohérente des explications en forçant l'agent à évaluer son propre brouillon par rapport à des critères concrets — comme le contexte de politique, les calendriers et les étapes suivantes — avant de le présenter. Cela détecte les lacunes spécifiques au cas sans supervision humaine.

Question 53 (Scénario : Agent de support client)

Situation : Les métriques de production montrent que votre agent fait en moyenne plus de 4 boucles d'API par résolution. L'analyse révèle que Claude demande souvent `get_customer` et `lookup_order` dans des tours séquentiels distincts même lorsque les deux sont nécessaires initialement. Quelle est la manière la plus efficace de réduire le nombre de boucles ?

Quelle est la manière la plus efficace de réduire les boucles ?

- A) Implémenter une exécution spéculative qui appelle automatiquement les outils probablement nécessaires en parallèle de tout outil demandé et renvoie tous les résultats quel que soit ce qui a été demandé.
- B) Augmenter `max_tokens` pour donner à Claude plus de marge pour planifier et combiner naturellement les requêtes d'outils.
- C) Créer des outils composites comme `get_customer_with_orders` qui regroupent les combinaisons de recherche courantes en appels uniques.
- D) Demander à Claude dans le prompt de regrouper les requêtes d'outils en un seul tour et de renvoyer tous les résultats ensemble avant l'appel d'API suivant. **[CORRECT]**

Pourquoi D : Demander à Claude de regrouper les requêtes d'outils connexes en un seul tour exploite sa capacité native à demander plusieurs outils à la fois. Cela corrige directement le motif d'appels séquentiels avec un changement architectural minimal.

Question 54 (Scénario : Agent de support client)

Situation : Les logs de production montrent un motif : les clients font référence à des montants précis (p. ex. « la remise de 15 % que j'ai mentionnée »), mais l'agent répond avec des valeurs incorrectes. L'investigation montre que ces détails ont été mentionnés il y a plus de 20 tours et condensés en résumés vagues comme « la tarification promotionnelle a été discutée ». Quelle correction est la plus efficace ?

Quelle correction est la plus efficace ?

- A) Augmenter le seuil de résumé de 70 % à 85 % pour que les conversations aient plus de marge avant le déclenchement du résumé.
- B) Stocker l'historique complet de la conversation dans un stockage externe et implémenter une récupération lorsque l'agent détecte des références comme « comme je l'ai mentionné ».
- C) Extraire les faits transactionnels (montants, dates, numéros de commande) dans un bloc persistant « case facts » inclus dans chaque prompt en dehors de l'historique résumé. **[CORRECT]**
- D) Réviser le prompt de résumé pour préserver explicitement tous les nombres, pourcentages, dates et attentes déclarées par le client mot pour mot.

Pourquoi C : Le résumé perd intrinsèquement les détails précis. Extraire les faits transactionnels dans un bloc structuré « case facts » en dehors de l'historique résumé préserve les informations critiques afin qu'elles soient disponibles de manière fiable dans chaque prompt, quel que soit le nombre de tours résumés.

Question 55 (Scénario : Agent de support client)

Situation : Votre outil `get_customer` renvoie toutes les correspondances lors d'une recherche par nom. Actuellement, lorsqu'il y a plusieurs résultats, Claude choisit le client avec la commande la plus récente, mais les données de production montrent que cela sélectionne le mauvais compte 15 % du temps pour les correspondances ambiguës. Comment traiter cela ?

Comment traiter cela ?

- A) Implémenter un système de scoring de confiance qui agit de manière autonome au-dessus de 85 % de confiance et demande des clarifications en dessous du seuil.
- B) Demander à Claude de demander un identifiant supplémentaire (email, téléphone ou numéro de commande) lorsque `get_customer` renvoie plusieurs correspondances, avant d'entreprendre toute action spécifique au client. **[CORRECT]**
- C) Modifier `get_customer` pour ne renvoyer qu'une seule correspondance la plus probable basée sur un algorithme de classement, éliminant l'ambiguïté.
- D) Ajouter des exemples few-shot au prompt démontrant le raisonnement correct et la séquence d'outils pour les correspondances ambiguës.

Pourquoi B : Demander à l'utilisateur un identifiant supplémentaire est la manière la plus fiable de résoudre l'ambiguïté car l'utilisateur a une connaissance définitive de son identité. Un tour de conversation supplémentaire est un petit prix à payer pour éliminer un taux d'erreur de 15 % causé par le choix du mauvais compte.

Question 56 (Scénario : Agent de support client)

Situation : Les logs de production montrent un motif constant : lorsque les clients incluent le mot « compte » dans leur message (p. ex. « je veux vérifier mon compte pour une commande que j'ai passée hier »), l'agent appelle `get_customer` en premier 78 % du temps. Lorsque les clients formulent des demandes similaires sans « compte » (p. ex. « je veux vérifier une commande que j'ai passée hier »), il appelle `lookup_order` en premier 93 % du temps. Les descriptions d'outils sont claires et sans ambiguïté. Quelle est la cause racine la plus probable de cet écart ?

Quelle est la cause racine la plus probable ?

- A) Le prompt système contient des instructions sensibles aux mots-clés qui orientent le comportement en fonction de termes comme « compte », créant des motifs de sélection d'outils involontaires. **[CORRECT]**
- B) L'entraînement de base du modèle crée des associations entre la terminologie « compte » et les opérations liées au client qui remplacent les descriptions d'outils.
- C) Le modèle a besoin de plus de données d'entraînement sur les messages multi-concepts et devrait être fine-tuné sur des exemples contenant à la fois la terminologie de compte et de commande.
- D) Les descriptions d'outils ont besoin d'exemples négatifs supplémentaires spécifiant quand NE PAS utiliser chaque outil pour empêcher cette confusion induite par les mots-clés.

Pourquoi A : Le motif systématique piloté par mot-clé (78 % vs 93 %) indique fortement une logique de routage explicite dans le prompt système réagissant au mot « compte » et orientant l'agent vers les outils liés au client. Puisque les descriptions d'outils sont déjà claires, l'écart pointe vers des instructions de niveau prompt créant une orientation comportementale involontaire.

Question 57 (Scénario : Agent de support client)

Situation : Les logs de production montrent que l'agent appelle souvent `get_customer` lorsque les utilisateurs posent des questions sur les commandes (p. ex. « vérifie ma commande #12345 ») au lieu d'appeler `lookup_order`. Les deux outils ont des descriptions minimales (« Gets customer information » / « Gets order details ») et acceptent des formats d'identifiants d'apparence similaire. Quelle est la première étape la plus efficace pour améliorer la fiabilité de la sélection d'outils ?

Quelle est la première étape la plus efficace ?

- A) Implémenter une couche de routage qui analyse l'entrée utilisateur avant chaque tour et présélectionne le bon outil en fonction des mots-clés détectés et des motifs d'ID.
- B) Combiner les deux outils en un seul `lookup_entity` qui accepte tout identifiant et décide en interne quel backend interroger.
- C) Ajouter des exemples few-shot au prompt système démontrant les motifs corrects de sélection d'outils, avec 5 à 8 exemples routant les requêtes liées aux commandes vers `lookup_order`.
- D) Étendre la description de chaque outil pour inclure les formats d'entrée, des exemples de requêtes, les cas limites et les limites expliquant quand l'utiliser par rapport à des outils similaires. **[CORRECT]**

Pourquoi D : Étendre les descriptions d'outils avec les formats d'entrée, des exemples de requêtes, les cas limites et des limites claires corrige directement la cause racine — des descriptions minimales qui ne

donnent pas au LLM assez d'informations pour distinguer des outils similaires. C'est une première étape à faible effort et fort impact qui améliore le mécanisme principal que le LLM utilise pour la sélection d'outils.

Question 58 (Scénario : Agent de support client)

Situation : Vous implémentez la boucle d'agent pour votre agent de support. Après chaque appel à la Claude API, vous devez décider de poursuivre la boucle (exécuter les outils demandés et rappeler Claude) ou de vous arrêter (présenter la réponse finale au client). Qu'est-ce qui détermine cette décision ?

Qu'est-ce qui détermine cette décision ?

- A) Vérifier le champ `stop_reason` dans la réponse de Claude — continuer s'il est `tool_use` et s'arrêter s'il est `end_turn`. **[CORRECT]**
- B) Analyser le texte de Claude à la recherche de phrases comme « J'ai terminé » ou « Puis-je vous aider avec autre chose ? » — les signaux en langage naturel indiquent l'achèvement de la tâche.
- C) Définir un nombre maximal d'itérations (p. ex. 10 appels) et s'arrêter une fois atteint, indépendamment de ce que Claude indique sur la nécessité de plus de travail.
- D) Vérifier si la réponse contient du contenu textuel de l'assistant — si Claude a généré du texte explicatif, la boucle devrait se terminer.

Pourquoi A : `stop_reason` est le signal structuré explicite de Claude pour le contrôle de boucle : `tool_use` indique que Claude veut exécuter un outil et recevoir les résultats en retour, tandis que `end_turn` indique que Claude a terminé sa réponse et que la boucle devrait se terminer.

Question 59 (Scénario : Agent de support client)

Situation : Les logs de production montrent que l'agent interprète mal les sorties de vos outils MCP : timestamps Unix de `get_customer`, dates ISO 8601 de `lookup_order`, et codes de statut numériques (1=pending, 2=shipped). Certains outils sont des serveurs MCP tiers que vous ne pouvez pas modifier. Quelle approche de normalisation des formats de données est la plus maintenable ?

Quelle approche est la plus maintenable ?

- A) Utiliser un hook PostToolUse pour intercepter les sorties d'outils et appliquer des transformations de formatage avant que l'agent ne les traite. **[CORRECT]**
- B) Modifier les outils que vous contrôlez pour qu'ils renvoient des formats lisibles par l'humain et créer des wrappers pour les outils tiers.
- C) Créer un outil `normalize_data` que l'agent appelle après chaque récupération de données pour transformer les valeurs.
- D) Ajouter une documentation de format détaillée au prompt système expliquant les conventions de données de chaque outil.

Pourquoi A : Un hook PostToolUse fournit un point centralisé et déterministe pour intercepter et normaliser toutes les sorties d'outils — y compris les données de serveurs MCP tiers — avant que l'agent

ne les traite. C'est plus maintenable car les transformations vivent dans le code et s'appliquent uniformément, plutôt que de s'appuyer sur l'interprétation du LLM.

Question 60 (Scénario : Agent de support client)

Situation : Les logs de production montrent que l'agent choisit parfois `get_customer` lorsque `lookup_order` serait plus approprié, surtout pour des requêtes ambiguës comme « j'ai besoin d'aide pour mon achat récent ». Vous décidez d'ajouter des exemples few-shot au prompt système pour améliorer la sélection d'outils. Quelle approche traite le plus efficacement le problème ?

Quelle approche est la plus efficace ?

- A) Ajouter un guidage explicite « à utiliser quand » et « à ne pas utiliser quand » dans chaque description d'outil couvrant les cas ambigus.
- B) Ajouter des exemples groupés par outil — tous les scénarios `get_customer` ensemble, puis tous les scénarios `lookup_order`.
- C) Ajouter 4 à 6 exemples ciblés sur les scénarios ambigus, chacun avec une justification expliquant pourquoi un outil a été choisi plutôt que des alternatives plausibles. **[CORRECT]**
- D) Ajouter 10 à 15 exemples de demandes claires et sans ambiguïté démontrant le choix d'outil correct pour les scénarios typiques de chaque outil.

Pourquoi C : Cibler les exemples few-shot sur les scénarios ambigus précis où les erreurs se produisent, avec une justification explicite expliquant pourquoi un outil est préférable aux alternatives, enseigne au modèle le processus de décision comparatif nécessaire aux cas limites. C'est plus efficace que des exemples génériques ou des règles déclaratives.

Question 61 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Votre outil `remove_team_member` utilise un paramètre `dry_run: boolean` pour prévisualiser les impacts avant l'exécution. La surveillance de production montre que l'agent contourne l'étape de prévisualisation en appelant directement avec `dry_run=false`. Vous devez garantir que chaque suppression est précédée d'une prévisualisation que l'utilisateur confirme explicitement.

Quelle est l'approche la plus fiable ?

- A) Ajouter une validation côté serveur qui n'autorise `dry_run=false` que lorsqu'un appel `dry_run=true` avec des paramètres identiques a eu lieu au cours des 60 dernières secondes.
- B) Annoter l'outil comme nécessitant une confirmation et configurer la couche d'orchestration pour demander l'approbation de l'utilisateur avant de transférer tout appel vers les outils annotés.
- C) Ajouter des instructions détaillées et des exemples few-shot à la description de l'outil exigeant que l'agent appelle toujours d'abord avec `dry_run=true` et attende la confirmation de l'utilisateur avant d'appeler à nouveau.
- D) Remplacer par deux outils : `preview_remove_member` renvoie les détails d'impact et un token de confirmation à usage unique ; `execute_remove_member` exige ce token, liant l'exécution à la

prévisualisation. **[CORRECT]**

Pourquoi D : L'approche à deux outils avec liaison par token rend architecturalement impossible l'exécution sans prévisualisation préalable — l'outil d'exécution exige littéralement un token que seul l'outil de prévisualisation peut générer. C'est la seule approche qui impose la contrainte au niveau du code plutôt que de s'appuyer sur la conformité du LLM aux instructions (C), des heuristiques de temps (A) ou une infrastructure d'orchestration (B).

Question 62 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : La surveillance de production montre que votre outil `search_catalog` échoue 12 % du temps : 8 % sont des timeouts réseau qui réussissent lorsqu'on réessaie, et 4 % sont des erreurs de syntaxe de requête qui ne réussissent jamais, quel que soit le nombre de tentatives. Actuellement, les deux types d'erreurs sont renvoyés à l'identique, provoquant des tentatives gaspillées.

Comment modifier la gestion d'erreurs de l'outil ?

- A) Ajouter des exemples few-shot à votre prompt système démontrant comment distinguer les erreurs réseau des erreurs de syntaxe.
- B) Appliquer une logique de nouvelle tentative avec backoff exponentiel à toutes les erreurs de manière uniforme.
- C) Implémenter une nouvelle tentative automatique avec backoff pour les timeouts réseau à l'intérieur de l'outil ; renvoyer immédiatement les erreurs de syntaxe avec des détails de validation des paramètres. **[CORRECT]**
- D) Renvoyer toutes les erreurs avec un drapeau booléen `retryable` et des détails sur le type d'erreur.

Pourquoi C : Gérer les nouvelles tentatives au niveau de l'outil pour les erreurs transitoires est la bonne frontière d'abstraction — l'outil a une connaissance définitive du type d'erreur et peut implémenter une logique de nouvelle tentative déterministe sans s'appuyer sur l'agent pour interpréter un drapeau (D) ou suivre des instructions de niveau prompt (A). Un backoff uniforme (B) gaspille du temps sur des erreurs de syntaxe qui ne réussiront jamais.

Question 63 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Au fil de plusieurs tours discutant de stratégie d'investissement, un utilisateur a déclaré « j'ai une très faible tolérance au risque » puis plus tard « je veux maximiser mes rendements ». Il demande maintenant : « Dans quoi devrais-je investir ? »

Quelle approche garantit le mieux que la recommandation s'aligne sur la véritable priorité de l'utilisateur ?

- A) Faire ressortir la contradiction et demander à l'utilisateur de clarifier ce qui importe le plus. **[CORRECT]**
- B) Fournir des recommandations distinctes pour les deux scénarios.
- C) Procéder avec la préférence la plus récemment déclarée.
- D) Recommander un portefeuille équilibré sans traiter le conflit.

Pourquoi A : Lorsque les préférences de l'utilisateur se contredisent directement, faire ressortir le conflit et demander une clarification est la seule façon de garantir que la recommandation s'aligne sur sa véritable intention. Toute autre approche implique de faire une supposition qui peut être erronée — maximiser les rendements et une faible tolérance au risque sont des objectifs fondamentalement incompatibles qui nécessitent une décision humaine.

Question 64 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Les utilisateurs affinent leurs préférences de playlist au fil de plusieurs tours de conversation. Deux messages après qu'un utilisateur a dit « j'adore le jazz », Claude demande « Quels genres aimez-vous ? »

Quelle est la cause la plus probable ?

- A) Claude nécessite une connexion à une base de données vectorielle pour maintenir la mémoire de conversation.
- B) La fenêtre de contexte du modèle a été dépassée.
- C) La Claude API nécessite un paramètre `session_id`.
- D) Votre application n'inclut pas les messages antérieurs dans le tableau `messages`. **[CORRECT]**

Pourquoi D : Claude n'a pas de mémoire côté serveur — chaque appel d'API est sans état (stateless). Sans inclure l'historique complet de la conversation dans le tableau `messages` de chaque requête, Claude n'a aucune connaissance des tours antérieurs. Les bases de données vectorielles (A) et `session_id` (C) ne font pas partie de l'architecture de Claude ; un débordement de la fenêtre de contexte (B) est impossible pour des échanges de deux messages.

Question 65 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Après une session de cuisine de 40 minutes, la conversation atteint 78 000 tokens. L'historique comprend des allergies, le calcul de proportions de recettes, des termes culinaires clarifiés et une discussion générale. Vous devez réduire les tokens tout en préservant les informations importantes.

Quelle approche équilibre le mieux la préservation et la réduction des tokens ?

- A) Résumer tout l'historique de la conversation.
- B) Garder uniquement les 20 000 tokens les plus récents.

- C) Extraire les données structurées critiques (allergies, quantités, préférences), résumer la discussion générale et garder les échanges récents mot pour mot. **[CORRECT]**
- D) Stocker la conversation complète en externe et récupérer les parties pertinentes via une recherche sémantique.

Pourquoi C : L'approche hybride préserve l'information de plus grande valeur au coût le plus bas. Les faits critiques comme les allergies et les quantités de recette sont extraits dans un bloc structuré compact (évitant la perte de précision qui se produit lors du résumé), la discussion générale est résumée, et les échanges récents sont gardés mot pour mot pour la cohérence conversationnelle. Les options A et B risquent de perdre des informations alimentaires critiques ; D est une surenchère architecturale pour une seule session de cuisine.

Question 66 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Les utilisateurs signalent que lors de conversations prolongées, l'assistant perd le fil des sujets et préférences antérieurs. Votre implémentation actuelle ne garde que les 25 dernières paires de messages.

Quelle est la solution la plus efficace ?

- A) Approche hybride : résumer les messages plus anciens tout en gardant les récents mot pour mot. **[CORRECT]**
- B) Recherche par similarité vectorielle sur l'historique complet de la conversation.
- C) Augmenter la fenêtre à 50 paires de messages.
- D) Résumer les messages abandonnés à chaque tour et préfixer le résumé courant.

Pourquoi A : L'approche hybride traite les deux dimensions du problème : conserver le contexte récent exact (critique pour la cohérence conversationnelle) tout en maintenant une représentation compressée des préférences antérieures (évitant la perte totale lorsque les paires sont abandonnées). Augmenter la fenêtre (C) ne fait que retarder le même problème. La recherche vectorielle (B) peut manquer un contexte important qui n'est pas sémantiquement similaire à la requête actuelle. Le résumé complet à chaque tour (D) ajoute de la surcharge et accumule les erreurs de résumé.

Question 67 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Les utilisateurs signalent que la latence augmente et que les coûts s'élèvent lorsque les conversations dépassent 50 tours.

Quelle est la cause principale ?

- A) L'historique complet de la conversation est inclus à chaque requête API. **[CORRECT]**
- B) Le modèle génère des réponses progressivement plus longues.
- C) Les opérations de base de données ralentissent à mesure que l'historique grandit.

- D) Le modèle construit un profil utilisateur interne nécessitant plus de traitement.

Pourquoi A : L'API de Claude est entièrement sans état — chaque requête doit inclure l'historique complet de la conversation dans le tableau `messages`. À mesure que les conversations grandissent, chaque requête transporte plus de tokens, ce qui augmente directement la latence de traitement et le coût. Le modèle ne maintient aucun état interne entre les appels (D est faux), et la longueur de la réponse n'est pas intrinsèquement liée à la longueur de la conversation (B).

Question 68 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Après trois mois de sessions hebdomadaires, l'historique de conversation atteint 85 000 tokens. Lorsqu'un utilisateur demande « Qu'avons-nous conclu sur le thème de l'isolement ? », l'assistant donne des réponses génériques au lieu de faire référence aux discussions précédentes.

Quelle est l'approche la plus efficace ?

- A) Troncature par fenêtre glissante.
- B) Résumé progressif capturant les conclusions clés.
- C) Embeddings sémantiques avec récupération des échanges pertinents. **[CORRECT]**
- D) Ajouter des balises XML structurées marquant les conclusions de discussion.

Pourquoi C : La recherche sémantique sur l'historique de conversation est la seule approche qui passe à l'échelle de trois mois de discussion tout en pouvant faire ressortir des échanges pertinents précis à la demande. La fenêtre glissante (A) abandonnerait la majeure partie de l'historique. Le résumé progressif (B) compresse les discussions en abstractions qui perdent les conclusions précises que les utilisateurs demandent. Les balises XML (D) nécessitent de restructurer tout le contenu passé et ne résolvent pas le problème de récupération à cette échelle.

Question 69 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Lors des tests QA, Claude suit les directives du prompt système pendant les 10 à 15 premiers tours, mais les réponses ultérieures dévient. La conversation est toujours dans les limites de tokens.

Quelle est la meilleure solution ?

- A) Déplacer les directives comportementales dans le premier message utilisateur.
- B) Démarrer une nouvelle conversation après 20 tours.
- C) Insérer des messages de rôle utilisateur renforçant les directives aux points de rupture de la conversation. **[CORRECT]**
- D) Utiliser une validation post-réponse pour régénérer les réponses non conformes.

Pourquoi C : L'injection périodique de rappels comportementaux combat directement la dérive d'instructions en rétablissant les contraintes à intervalles réguliers à mesure que l'historique de conversation s'accumule. Déplacer les directives vers le premier message utilisateur (A) réduit leur

autorité. Démarrer une nouvelle conversation (B) détruit le contexte. La validation post-réponse (D) est corrective plutôt que préventive et ajoute une latence significative.

Question 70 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Votre tuteur IA a un prompt système de 2 800 tokens définissant la méthodologie d'enseignement et les règles d'adaptation. Après 12 tours, l'assistant commence à ignorer les niveaux de compétence.

Quelle est la correction la plus efficace ?

- A) Injecter des rappels tous les 4 à 5 tours.
- B) Remplacer les règles verbeuses par des exemples few-shot démontrant l'adaptation au niveau de compétence. **[CORRECT]**
- C) Placer les règles critiques à la fin du prompt système.
- D) Évaluer les réponses et régénérer si le niveau de difficulté ne correspond pas.

Pourquoi B : Un prompt système de 2 800 tokens avec des règles déclaratives est vulnérable à la dérive car les règles abstraites obligent le modèle à raisonner à leur sujet à chaque tour. Remplacer les règles verbeuses par des exemples few-shot concrets qui démontrent l'adaptation correcte au niveau de compétence donne au modèle des motifs comportementaux clairs à imiter — c'est plus fiablement suivi sur de nombreux tours que des instructions abstraites. L'injection de rappels (A) aide mais traite les symptômes ; le placement en fin (C) aide initialement mais pas avec la dérive au niveau des tours ; la régénération (D) est coûteuse et corrective.

Question 71 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Votre assistant doit maintenir un ton enthousiaste, expliquer son raisonnement et poser des questions de clarification. Où ces directives comportementales devraient-elles être définies ?

Où ces directives comportementales devraient-elles être définies ?

- A) Préfixées à chaque message utilisateur.
- B) Dans le prompt système. **[CORRECT]**
- C) Dans le premier message de l'assistant.
- D) Dans des variables d'environnement.

Pourquoi B : Le prompt système est spécifiquement conçu pour les contraintes comportementales persistantes et les directives qui s'appliquent tout au long de la conversation. Le préfixer à chaque message utilisateur (A) est une surcharge redondante. Le premier message de l'assistant (C) n'est pas fiable car le modèle peut dévier de ses propres déclarations antérieures. Les variables d'environnement (D) n'ont aucun effet sur le comportement du modèle.

Question 72 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Les utilisateurs signalent des ouvertures de réponse répétitives comme « Bien sûr ! » et « Je serais ravi de vous aider ! »

Quelle est l'approche la plus efficace ?

- A) Ajouter un message d'assistant partiel avec une ouverture de réponse directe. **[CORRECT]**
- B) Baisser le réglage de température.
- C) Post-traiter les réponses pour supprimer les salutations.
- D) Ajouter des instructions au prompt système pour éviter ces phrases.

Pourquoi A : Préremplir la réponse de l'assistant avec le début d'une réponse directe empêche les motifs de salutation au niveau de la génération — le modèle continue à partir du préremplissage plutôt que de générer de nouvelles phrases d'ouverture. Les instructions du prompt système (D) peuvent aider mais sont moins fiables car le modèle peut encore produire des variantes. Le post-traitement (C) est un contournement fragile. La température (B) contrôle l'aléatoire, pas des motifs de phrases précis.

Question 73 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Un webhook notifie votre système que le colis d'un utilisateur a été expédié pendant que l'utilisateur discute activement. Vous voulez que l'assistant intègre cela naturellement dans la réponse suivante.

Quelle est la meilleure approche ?

- A) Ajouter le statut d'expédition au prompt système.
- B) Envoyer un message utilisateur synthétique immédiat.
- C) Forcer l'assistant à appeler un outil de statut à chaque tour.
- D) Ajouter la mise à jour de statut comme préfixe au message utilisateur suivant. **[CORRECT]**

Pourquoi D : Préfixer la mise à jour de statut au message utilisateur suivant injecte un contexte en temps réel à une frontière de conversation naturelle sans perturber le flux. Modifier le prompt système (A) nécessite de reconstruire la session ou est architecturalement lourd. Un message utilisateur synthétique (B) peut casser le flux de dialogue naturel et brouiller l'attribution. Forcer un appel d'outil à chaque tour (C) est gaspilleur lorsque les événements sont rares.

Question 74 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Les utilisateurs envoient fréquemment des demandes comme « Réserve un lieu pour la fête. » L'assistant pose plus de 4 questions de clarification, causant 35 % d'abandon.

Quelle approche améliore le mieux le compromis ?

- A) Procéder avec des valeurs par défaut cachées.
- B) Poser toutes les questions de clarification dans un seul message composé.
- C) Énoncer les hypothèses explicitement et procéder tout en invitant aux corrections. **[CORRECT]**
- D) Utiliser un formulaire d'admission structuré.

Pourquoi C : Énoncer les hypothèses explicitement et procéder donne à l'utilisateur une réponse immédiate et utile tout en préservant sa capacité à corriger les hypothèses erronées. Les valeurs par défaut cachées (A) laissent l'utilisateur ignorant de ce qui a été supposé. Une liste de questions composée (B) exige toujours un effort initial de l'utilisateur. Un formulaire structuré (D) ajoute plus de friction, pas moins — contredisant l'objectif de réduire l'abandon.

Question 75 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Votre assistant utilise un prompt système avec une persona de prestataire. Les premiers tours suivent les règles, mais au tour 7, l'assistant donne des conseils génériques. La longueur de la conversation n'est que de 2 500 tokens.

Quelle est la cause la plus probable ?

- A) Les prompts système n'établissent que le comportement initial.
- B) L'attention du modèle faiblit à mesure que les tours s'accumulent.
- C) Les réponses accumulées de l'assistant diluent l'influence du prompt système. **[CORRECT]**
- D) Le prompt système n'est envoyé qu'une seule fois.

Pourquoi C : À mesure que les réponses de l'assistant s'accumulent dans l'historique de conversation, la proportion de texte reflétant les contraintes comportementales du prompt système diminue par rapport au corps croissant de contenu généré par l'assistant. Le modèle s'aligne de plus en plus sur ses propres sorties antérieures plutôt que sur le prompt système, aggravant la dérive même à de faibles longueurs de tokens. Le prompt système est inclus dans chaque appel d'API (D est faux en tant qu'explication isolée), et la dégradation de l'attention du modèle (B) n'opère pas à 2 500 tokens.

Question 76 (Scénario : Modèles d'architecture d'IA conversationnelle)

Situation : Les utilisateurs posent des demandes vagues comme « Peux-tu m'aider avec le rapport ? » L'assistant répond en posant plusieurs questions (quel rapport ? quelle aide ? échéance ?), causant 40 % d'abandon.

Quelle est la meilleure solution ?

- A) Faire des hypothèses raisonnables, les énoncer explicitement et proposer d'ajuster. **[CORRECT]**
- B) Classer l'ambiguïté avec un modèle plus petit avant de répondre.

- C) Utiliser des interprétations prédéfinies sans énoncer les hypothèses.
- D) Limiter l'assistant à une question de clarification par tour.

Pourquoi A : Procéder avec des hypothèses raisonnables énoncées élimine entièrement les allers-retours tout en gardant l'utilisateur informé et aux commandes. Les interprétations silencieuses prédéfinies (C) laissent les utilisateurs confus lorsque la réponse ne correspond pas à leur intention. Une limite d'une seule question (D) nécessite toujours des tours d'allers-retours. Un modèle de classification plus petit (B) ajoute de la latence et de la complexité d'infrastructure sans résoudre le problème UX fondamental.

Exercices pratiques

Exercice 1 : Agent multi-outils avec logique d'escalade

Objectif : Concevoir une boucle d'agent avec intégration d'outils, gestion d'erreurs structurée et escalade.

Étapes :

1. Définir 3 à 4 outils MCP avec des descriptions détaillées (inclure deux outils similaires pour tester la sélection d'outils)
2. Implémenter une boucle d'agent vérifiant `stop_reason` (`"tool_use"` / `"end_turn"`)
3. Ajouter des réponses d'erreur structurées : `errorCategory`, `isRetryable`, description
4. Implémenter un hook d'interception qui bloque les opérations au-dessus d'un seuil et route vers l'escalade
5. Tester avec des demandes à multiples facettes

Domaines : 1 (Architecture d'agents), 2 (Outils et MCP), 5 (Contexte et fiabilité)

Exercice 2 : Configurer Claude Code pour le développement en équipe

Objectif : Configurer CLAUDE.md, des commandes personnalisées, des règles spécifiques au chemin et des serveurs MCP.

Étapes :

1. Créer un CLAUDE.md de niveau projet avec des normes universelles
2. Créer des fichiers `.claude/rules/` avec frontmatter YAML pour différentes zones de code (`paths: ["src/api/**/*"]`, `paths: ["**/*.test.*"]`)
3. Créer un skill de projet sous `.claude/skills/` avec `context: fork` et `allowed-tools`
4. Configurer un serveur MCP dans `.mcp.json` avec des variables d'environnement + une surcharge personnelle dans `~/.claude.json`
5. Tester le mode planification vs l'exécution directe sur des tâches de complexité différente

Domaines : 3 (Configuration de Claude Code), 2 (Outils et MCP)

Exercice 3 : Pipeline d'extraction de données structurées

Objectif : Schémas JSON, `tool_use` pour la sortie structurée, boucles de validation/nouvelle tentative, traitement par lots.

Étapes :

1. Définir un outil d'extraction avec un schéma JSON (champs requis/optionnels, enums avec « other », champs nullable)
2. Construire une boucle de validation : en cas d'erreur, réessayer avec le document, l'extraction incorrecte et l'erreur de validation précise
3. Ajouter des exemples few-shot pour des documents de structures différentes
4. Utiliser le traitement par lots via la Message Batches API : 100 documents, gérer les échecs via `custom_id`
5. Router vers les humains : scores de confiance au niveau du champ, analyse par type de document

Domaines : 4 (Ingénierie de prompts), 5 (Contexte et fiabilité)

Exercice 4 : Concevoir et déboguer un pipeline de recherche multi-agents

Objectif : Orchestration de sous-agents, passage de contexte, propagation d'erreurs, synthèse avec suivi des sources.

Étapes :

1. Un coordinateur avec 2+ sous-agents (`allowedTools` inclut `"Task"`, le contexte est passé explicitement dans les prompts)
2. Exécuter les sous-agents en parallèle via plusieurs appels `Task` dans une seule réponse
3. Exiger une sortie de sous-agent structurée : affirmation, citation, URL de la source, date de publication
4. Simuler un timeout de sous-agent : renvoyer un contexte d'erreur structuré au coordinateur et continuer avec des résultats partiels
5. Tester avec des données contradictoires : préserver les deux valeurs avec attribution ; séparer les conclusions confirmées des conclusions contestées

Domaines : 1 (Architecture d'agents), 2 (Outils et MCP), 5 (Contexte et fiabilité)

Annexe : Technologies et concepts

Technologie	Aspects clés
Claude Agent SDK	AgentDefinition, boucles d'agent, `stop_reason`, hooks (PostToolUse), engendrement de sous-agents via Task, `allowedTools`

Model Context Protocol (MCP)	Serveurs MCP, outils, ressources, `isError`, descriptions d'outils, `.mcp.json`, variables d'environnement
Claude Code	Hiérarchie CLAUDE.md, `.claude/rules/` avec motifs glob, `.claude/commands/`, `.claude/skills/` avec SKILL.md, mode planification, `/compact`, `--resume`, `fork_session`
Claude Code CLI	`-p` / `--print` pour le mode non interactif, `--output-format json`, `--json-schema`
Claude API	`tool_use` avec schémas JSON, `tool_choice` ("auto"/"any"/forcé), `stop_reason`, `max_tokens`, prompts système
Message Batches API	50 % d'économies, fenêtre jusqu'à 24 heures, `custom_id`, pas d'appels d'outils multi-tours
JSON Schema	Requis vs optionnel, champs nullable, types enum, "other" + détail, mode strict
Pydantic	Validation de schéma, erreurs sémantiques, boucles validation/nouvelle tentative
Outils intégrés	Read, Write, Edit, Bash, Grep, Glob — objectif et critères de sélection
Prompting few-shot	Exemples ciblés pour situations ambiguës, généralisation à de nouveaux motifs
Chaînage de prompts	Décomposition séquentielle en passes ciblées
Fenêtre de contexte	Budgets de tokens, résumé progressif, « perdu au milieu », fichiers bloc-notes
Gestion des sessions	Reprise, `fork_session`, sessions nommées, isolation du contexte
Calibrage de la confiance	Scoring au niveau du champ, calibrage sur ensembles étiquetés, échantillonnage stratifié

Sujets hors périmètre

Les sujets adjacents suivants ne seront **PAS** à l'examen :

- Fine-tuning des modèles Claude ou entraînement de modèles personnalisés
- Authentification, facturation ou gestion de compte de la Claude API
- Implémentation détaillée dans des langages de programmation ou frameworks spécifiques (au-delà de ce qui est nécessaire pour la configuration d'outils/schémas)
- Déploiement ou hébergement de serveurs MCP (infrastructure, réseau, orchestration de conteneurs)
- Architecture interne de Claude, processus d'entraînement ou poids du modèle
- Constitutional AI, RLHF ou méthodologies d'entraînement à la sécurité
- Modèles d'embedding ou détails d'implémentation de bases de données vectorielles
- Computer use (automatisation du navigateur, interaction avec le bureau)

- Capacités d'analyse d'images (Vision)
- API de streaming ou server-sent events
- Limitation de débit (rate limiting), quotas ou calculs détaillés de coût d'API
- OAuth, rotation de clés d'API ou détails de protocole d'authentification
- Configurations spécifiques aux fournisseurs cloud (AWS, GCP, Azure)
- Benchmarks de performance ou métriques de comparaison de modèles
- Détails d'implémentation de la mise en cache de prompts (au-delà de savoir qu'elle existe)
- Algorithmes de comptage de tokens ou spécificités de la tokenisation

---

Recommandations de préparation

1. **Construire un agent avec le Claude Agent SDK** — implémenter une boucle d'agent complète avec appel d'outils, gestion d'erreurs et gestion des sessions. Pratiquer les sous-agents et le passage explicite de contexte.
2. **Configurer Claude Code pour un vrai projet** — utiliser la hiérarchie CLAUDE.md, les règles spécifiques au chemin dans `.claude/rules/`, les skills avec `context: fork` et `allowed-tools`, et l'intégration de serveurs MCP.
3. **Concevoir et tester des outils MCP** — rédiger des descriptions qui différencient les outils similaires, renvoyer des erreurs structurées avec catégories et drapeaux de nouvelle tentative, et tester face à des demandes utilisateur ambiguës.
4. **Construire un pipeline d'extraction de données** — utiliser `tool_use` avec des schémas JSON, des boucles de validation/nouvelle tentative, des champs optionnels/nullable et le traitement par lots via la Message Batches API.
5. **Pratiquer l'ingénierie de prompts** — ajouter des exemples few-shot pour les scénarios ambigus, des critères de revue explicites et des architectures multi-passes pour les grandes revues de code.
6. **Étudier les modèles de gestion du contexte** — extraire les faits des sorties verbeuses, utiliser des fichiers bloc-notes et déléguer la découverte aux sous-agents pour gérer les limites de contexte.
7. **Comprendre l'escalade et l'intervention humaine** — quand escalader (lacunes de politique, demande explicite de l'utilisateur, incapacité à progresser) et les workflows de routage basés sur la confiance.
8. **Passer un examen blanc** avant le vrai. Il utilise les mêmes scénarios et le même format.