Claude Certified Architect — Foundations Certification

مطالعہ گائیڈ (سرکاری امتحانی گائیڈ کی بنیاد پر)

تعارف

سرٹیفیکیشن اس بات کی تصدیق کرتی ہے کہ ایک ماہر حقیقی **Claude Certified Architect — Foundations** Claude Code، فیصلہ کر سکتا ہے یہ امتحان trade-off حل نافذ کرتا وقت درست Claude-based دنیا کہ Claude Agent SDK، Claude API، اور Model Context Protocol (MCP) یعنی — کہ بنیادی علم کا جائزہ لیتا ہے کہ ساتھ پروڈکشن ایپلیکیشنز بنائی جاتی ہیں Claude وہ بنیادی ٹیکنالوجیز جن سے

امتحانی سوالات حقیقت پسندانہ صنعتی منظرناموں پر مبنی ہوتے ہیں: کسٹمر سپورٹ کہ لیے agentic systems بنانا، multi-agent research pipelines ڈیزائن کرنا، Claude Code کو CI/CD میں ضم کرنا، developer productivity tools سے دستاویزات بنانا، اور غیر ساختہ structured data نکالنا

ہدف امیدوار

مثالی امیدوار ایک **solution architect** جو ہے Claude اور پروڈکشن ایپلیکیشنز ڈیزائن ساتھ کہ ship آپ ہے کرتا کہ پاس کم از کم 6 ماہ کا عملی تجربہ ہونا چاہیے:

- **Claude Agent SDK** — multi-agent orchestration، subagents کو delegate کرنا، tool integration، lifecycle hooks
- **Claude Code** — CLAUDE.md، MCP servers، Agent Skills، planning mode
- **Model Context Protocol (MCP)** — backend integration کہ لیے tools اور resources
- **Prompt engineering** — JSON schemas، few-shot examples، data extraction templates
- **Context windows** — لمبی دستاویزات کہ ساتھ کام، multi-agent context passing
- **CI/CD pipelines** — automated code review، test generation
- **Escalation and reliability** — error handling، human-in-the-loop

---

امتحانی فارمیٹ

پیرامیٹر	قدر
سوال کی قسم	Multiple choice (4 درست میں سے 1)
اسکورنگ	100–1000 scale، passing score **720**
Guessing penalty	نہیں (ہر سوال کا جواب دیں!)
Scenarios	8 میں سے 4 (randomly selected)

امتحانی مواد: 5 ڈومینز

ڈومین	وزن
1. Agent architecture and orchestration	**27%**
2. Tool design and MCP integration	**18%**
3. Claude Code configuration and workflows	**20%**
4. Prompt engineering and structured output	**20%**
5. Context management and reliability	**15%**

امتحانی منظرنامہ

Scenario 1: Customer Support Agent

آپ ایک agent میں جو بناتے ہوئے Claude Agent SDK استعمال کرتے ہوئے returns، billing disputes، اور account issues کو سنبھالتا ہے agent MCP tools (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`) 80 ہدف کے ساتھ استعمال کرتا ہے+% first-contact resolution مناسب escalation کے ساتھ

Scenario 2: Claude Code کے ساتھ Code Generation

آپ Claude Code کو development میں استعمال کرتے ہیں تیز کرنے کے لیے: code generation، refactoring، debugging، documentation آپ کو اسے CLAUDE.md configuration اور custom slash commands کے ساتھ integrate کرنے کے سمجھنا ہے، اور کرنا planning mode کب استعمال کرنا ہے

Scenario 3: Multi-Agent Research System

ایک coordinator agent specialized subagents کو tasks کے سونپتا ہے: web research، document analysis، synthesis، اور report generation سسٹم کو citations کے ساتھ مکمل reports تیار کرنے چاہئیں

Scenario 4: Developer Productivity Tools

یہ agent engineers کو unfamiliar codebases explore کرنے، boilerplate code اور بنانے، routine tasks automate کرنے میں مدد دیتا ہے Built-in tools (Read، Write، Bash، Grep، Glob) اور MCP servers استعمال کیے جاتے ہیں

Scenario 5: Claude Code for Continuous Integration

Claude Code کو CI/CD pipeline میں ضم کریں تاکہ automated code reviews، test generation، اور pull request feedback ہو حاصل Prompts کے چاہئیں ہونے ڈیزائن ایسے false positives کم سے کم ہوں

Scenario 6: Structured Data Extraction

□□ معلومات نکالتا □□ دستاویزات سے □ سسٹم غیر ساخت□، output کو JSON schemas □ ذریعہ □ کا validate □□، کرتا
اور high accuracy □ اسے □□□ برقرار رکھتا edge cases □□یہ□ چاہ□ سنبھالنا کو درست طور پر

Scenario 7: Conversational AI Architecture Patterns

آپ multi-turn conversational systems میں جن میں □یں □ ڈیزائن کرتے context window management، متعدد turns
میں instructions □نا ر□ برقرار کا، memory strategies، محفوظ execution □ لیے □ کے tool design، متضاد یا مبہم اور
user inputs □□□ شامل سنبھالنا کو

Scenario 8: Agentic AI Tools *(مواد موجود نہیں — ہماری مدد کریں!)*

اس scenario □والے □ کی اطلاع امتحان دینے candidates کی ابھی تک اس □ لیکن یہ □□ دی □ guide میں شامل نہیں
کریں share میں GitHub Issues کے □ سوالات دیکھ□ □یں، تو انہیں □□□ scenario اگر آپ نے □ اصل امتحان میں اس
شامل کر سکیں□ آپ کا تعاون □ر اس شخص کی مدد کر□ گا جو امتحان کی تیاری کر coverage تاکہ □م مکمل
□□□ ر□نا

سرکاری دستاویزات

وسیلہ□	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 (non-interactive mode)	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 (code examples)	https://github.com/anthropics/anthropic-cookbook

---

□حصہ I: نظریاتی بنیادیں

یہ حصہ وہ تمام نظریہ احاطہ کرتا ہے جس کی آپ کو امتحان میں کامیابی کے لیے ضرورت ہے مواد کو کے مطابق — اس سے آپ کو domains کے مطابق ترتیب دیا گیا ہے، نہ کہ امتحانی concepts ٹیکنالوجیز اور موضوع کی زیادہ گہری سمجھ پیدا کر سکتے ہیں

---

1 باب: Claude API — Model Interaction کی بنیادیں

> دستاویزات: Messages API | Prompt Engineering

1.1 API Request Structure

Claude API ایک request–response model ہے جو کی پیروی کرتا Claude Messages API کی ہر request میں یہ شامل ہوتا ہے:

```
{
  "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"}
}
```

اہم فیلڈز:

- `model` — model selection (`claude-opus-4-6`, `claude-sonnet-4-6`, `claude-haiku-4-5`)
- `max_tokens` — response میں زیادہ سے زیادہ tokens کی تعداد
- `system` — system prompt (model کا رویہ کیسا ہونا چاہیے)
- `messages` — conversation history (**بھیجنی ہوتی history تسلسل برقرار رکھنے کے لیے آپ کو مکمل** ہوتی ہے)
- `tools` — دستیاب tools کی definitions
- `tool_choice` — tool selection strategy

1.2 Message Roles

`messages` array تین roles استعمال کرتا ہے:

- `user` — user messages
- `assistant` — model responses (history میں بھیجتے وقت شامل کیے جاتے ہیں)
- `tool` — tool call results (role واضح طور پر set ہے؛ یہ `tool_result` content block کی صورت میں ظاہر ہوتا ہے)

اہم: انتہائی اہم: API request میں آپ کو **مکمل conversation history** بھیجنی چاہیے Model requests کے درمیان state رکھتا نہیں — محفوظ call آزاد ہوتا ہے

1.3 Response میں `stop_reason` Field

Claude API response میں `stop_reason` کیوں روکی generation نے model شامل ہوتا ہے، جو بتاتا ہے کہ:

Value	Description	Action
`"end_turn"`	Model نے response مکمل کر لیا	نتیجہ user کو دکھائیں
`"tool_use"`	Model tool call کرنا چاہتا ہے	Tool execute کریں اور result واپس دیں
`"max_tokens"`	Token limit ختم ہو گئی	Response truncated ہے ممکن ہے; limit بڑھانی پڑے
`"stop_sequence"`	Stop sequence مل گئی	اپنی application logic کے مطابق handle کریں

Agentic systems میں `"tool_use"` اور `"end_turn"` یہی — یہ سب سے اہم ہیں agent loop کو کنٹرول کرتے ہیں۔

1.4 System Prompt

System prompt ایک خاص instruction جو ہے context اور behavioral rules یہ ہے متعین کرتا:

- `messages` array الگ ہے یہ ہوتا نہیں کا حصہ `system` field ہے میں دیا جاتا
- user messages ہے پر فوقیت رکھتا
- ایک بار load اور پوری ہے conversation ہے ہوتا میں لاگو رہتا
- role، constraints، اور output format ہے ہوتا استعمال لیے کے تعریف کی

امتحان کے لیے اہم: system prompt کی wording غیر ارادی tool associations طور کے مثال پیدا کر سکتی ہے کو ضرورت سے زیادہ `get_customer` کو model کی تصدیق کریں" جیسی ہدایت customer پر، "ہمیشہ استعمال کرنے پر مجبور کر سکتی ہے، حتیٰ کہ جب یہ ضروری نہ ہو

1.5 Context Window

Context window text (tokens) جسے ہے کی کل مقدار ہے model ایک وقت میں process اس میں ہے کر سکتا شامل ہے:

- System prompt
- مکمل message history
- Tool definitions
- Tool results

اہم context-window مسائل:

1. **Lost-in-the-middle effect:** models لمبے input سے اعتماد زیادہ کو موجود معلومات کے آغاز اور اختتام پر کے کر سکتے ہیں۔ حل: اہم معلومات شروع یا آخر کے قریب miss کرتے ہیں، مگر درمیان کی تفصیلات process رکھیں۔
2. **Tool results کا جمع ہونا:** tool call context میں output اگر ہے شامل کرتا tool 40+ fields واپس کرے کا بڑا حصہ ضائع ہو جاتا ہے context مگر صرف 5 اہم ہوں، تو
3. **Progressive summarization:** history compress وقت کرتے numeric values، percentages، اور dates اکثر گم ہو کر غیر واضح ہو جاتے ہیں ("تقریباً"، "کچھ"، "چند")

باب 2: Tools اور `tool_use`

> دستاویزات: Tool Use

2.1 `tool_use` □□ کیا

`tool_use` ایک mechanism جو □□ Claude کو external functions call □□□ کی اجازت دیتا □کرنا Model □□برا راست code □و — چلاتا □نہیں structured tool call request کا آپ؛ □□ بناتا code □اسا execute اور □□ کرتا result واپس کرتا □□□

2.2 Tool Definition

ہر tool ایک JSON schema □ذریعہ □کا define □□ کیا جاتا:

```
{
  "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": []
  }
}
```

Tool description □کا انتہائی اہم پہلو:

1. **Description tool selection کا بنیادی mechanism □□□** LLM tools کی ان کو descriptions کی بنیاد پر □□□ منتخب کرتا Minimal descriptions (“Customer information retrieve □□ کرتا”) اُن مواقع پر غلطیوں کا باعث بنتی □ہیں جب tools □سا □دوسرا ایک overlap □ہوں □کرتا
2. **Description □میں شامل کریں:**
   - Tool □□ کیا کرتا □□ اور کیا واپس کرتا
   - Input formats اور example values
   - Edge cases اور constraints
   - □کہ مقابلہ □میں کب استعمال □ہو tool similar alternatives □یہ
3. **یا □جیسا ایک overlapping descriptions □سا بچیں □اگر** `analyze_content` اور `analyze_document` کی descriptions تو، □ہوں تقریباً ایک جیسی model □گا □دے خلط ملط کر انہیں
4. **Built-in tools □بمقابلہ MCP tools:** agents similar functionality □والے built-in tools (Read، Grep) کو MCP tools □لیے □کہ □بچنا □اس سا □ہیں □سکتا □پر ترجیح دے MCP tool descriptions مضبوط بنائیں — concrete advantages، unique data، □یا context جو نمایاں کریں built-in tools □□سکتا فراہم نہیں کر

2.3 `tool_choice` Parameter

`tool_choice` □کا □□ کنٹرول کرتا □یہ model tools □□ منتخب □کیسا:

Value	Behavior	کب استعمال کریں
`{ "type": "auto" }`	Model □ کہ □□ کرتا خود ط□ tool call کرنا یا □□ text □□ میں جواب دینا	default زیاد□ تر حالات میں
`{ "type": "any" }`	Model کو لازماً کوئی tool call کرنا □وگی	جب آپ کو guaranteed structured output □چا□ئیے
`{ "type": "tool", "name": "extract_metadata" }`	Model کو لازماً ایک مخصوص tool call کرنی □وگی	جب آپ کو forced first step / execution order □چا□ئیے

□م منظرنام□:

- `tool_choice: "any"` + multiple extraction tools → model ب□ترین tool منتخب کرتا □□، مگر پھر بھی structured output □□ ملتا
- Forced selection → جب آپ کو کسی خاص پ□لی action □چا□ئیے کی ضمانت (مثلاً enrichment □□ سے پ□لے `extract_metadata`)

2.4 Structured Output □ لیے □ کے JSON Schemas

حاصل کرنے کا **سب سے قابلِ** structured output سے Claude استعمال کرنا `tool_use` ساتھ □ کے JSON schemas □ یہ □□□ طریقے **اعتماد**:

- Syntax درست □ اعتبار سے □ کہ JSON □□ کی ضمانت دیتا (braces □وت□ میں □ غائب، trailing commas نہ□یں □وت□)
- □مطلوب structure □□ نافذ کرتا (required fields □یں □ □وت□ موجود)
- Semantic correctness کی ضمانت نہ□یں دیتا (values □یں □و سکتی □و غلط پھر بھی)

Schema design — بنیادی اصول:

```
{
  "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"]
}
```

Schema design rules:

1. **Required □بمقابل optional:** صرف اُن fields کو required □و□ دستیاب □میشہ□ معلومات بنائیں جن کی values □یں□ غائب □ون□ پر data کو Required fields model گھڑنا□ پر مجبور کر سکتی
2. **Nullable fields:** □لیے □ایسی معلومات کے □ `"type": ["string", "null"]` طور پر □ممکن استعمال کریں جو Model □موجود نہ □وں□ `null` □خود ساخت□ وہ □ کے □ اس کے بجائے □□ واپس کر سکتا value □□بنائے
3. **Enums with `"other"`:** اپنی predefined categories □لیے □کے بچانے □سے □ونے ضائع معلومات کی □باہر □سے `"other"` + ایک detail string □شامل کریں
4. **Enum `"unclear"`:** جب □لیے □کے صورتوں اُن model category ایماندار — □و □نہ میں پُراعتماد □کے بار□ □سے بہتر □□□ category غلط `"unclear"`

2.5 Syntax □بمقابل Semantic Errors

Error type	Example	Mitigation
Syntax	Invalid JSON، غلط field type	JSON schema □کے ساتھ `tool_use` (ختم کر دیتا □□)
Semantic	Totals match □نہیں کرتے، value غلط field میں، hallucination	Validation checks، feedback ساتھ □کے retry، self-correction

باب 3: Claude Agent SDK — Agentic Systems بنانا

> دستاویزات: Agent SDK | Hooks | Subagents | Sessions

3.1 Agentic Loop □□ کیا

Agentic loop خودکار task execution کا بنیادی pattern □□□ Model □و — صرف جواب نہیں دیتا actions کی ایک sequence □□ انجام دیتا:

```
1. Claude کو tools کے ساتھ request بھیجیں
2. Response وصول کریں
3. stop_reason چیک کریں:
   - "tool_use" -> tool execute کریں، result history میں append کریں، step 1 پر واپس جائیں
   - "end_turn" -> task مکمل، result user کو دکھائیں
4. repeat کریں مکمل ہونے تک
```

□ی **model-driven approach □□:** Claude □اپنا context □پچھلا اور tool results کی بنیاد پر اگلا tool خود منتخب □ی □□□ کرتا hard-coded decision trees □جاں □□ مختلف □سے action sequence □سے □لا□ fixed □□□ □وتی

Anti-patterns (ان سے بچیں):

- Completion detect □لیے □کے □کرنے assistant text parse کرنا (“Task completed”)
- Primary stop condition طور پر □کے arbitrary iteration limit استعمال کرنا (مثلاً `max_iterations=5`)
- □چیک کرنا کہ □ی assistant □نے textual content □اسے، یا نہیں □□ دیا completion signal سمجھنا

□**درست طریقہ:** completion اعتماد قابلِ واحد کا signal `stop_reason == "end_turn"` □□□

3.2 `AgentDefinition` Configuration

`AgentDefinition` Claude Agent SDK میں agent configuration object □□:

```
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"],
    # Coordinator کے لیے:
    # allowed_tools=["Task", "get_customer", ...]
)
```

م□l parameters:

- `name` / `description` — agent کی شناخت اور وضاحت
- `system_prompt` — □ہدایات والا system prompt
- `allowed_tools` — □اجازت یافتہ tools کی فہرست (least privilege اصول)

3.3 Hub-and-Spoke: Coordinator اور Subagents

Multi-agent architecture عموماً hub-and-spoke topology □□ میں بنائی جاتی:

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

Coordinator کی ذمہ□ داریاں:

- Task کو subtasks میں تقسیم کرنا
- □ط□ کرنا کہ□ کون سہ□ subagents میں□ درکار (dynamic selection)
- کام subagents کو سونپنا
- نتائج جمع اور validate کرنا
- Errors اور retries handle کرنا
- نتائج user تک پہ□نچانا

اصول□م: subagents کا context □□□ **الگ □وتا**

- Subagents خود بخود coordinator کی conversation history inherit □نہیں کرتہ□
- تمام ضروری context subagent prompt □□ میں **واضح طور پر** دینا □وتا
- Subagents calls درمیان □کہ memory share □نہیں کرتہ□
- تمام communication coordinator □□ گزرتی □کہ ذریعہ□ (observability اور error control □لیہ□)

3.4 Subagents □لیہ□ بنانہ□ کہ□ Task Tool

Subagents `Task` tool □ذریعہ□ کہ spawn □میں□ جاتہ□ □کیہ□:

```
# Coordinator کہ allowedTools میں "Task" ہونا چاہیہ
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

Explicit context passing □□ لازمی:

```
# Bad: subagent پاس کہ context ہہ نہیہ
Task: "Analyze the document"

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

Parallel spawning: coordinator ایک response میں متعدد `Task`s call □□ کر سکتا — subagents parallel میں□ چلتہ□:

```
# Coordinator ایک کہ response میں یہ شامل ہو:
Task 1: "Search for articles about X"
Task 2: "Analyze document Y"
Task 3: "Search for articles about Z"
# تینوں ایک ساتھ چلتے ہیں
```

3.5 Agent SDK میں Hooks

Hooks agent lifecycle مخصوص کے points پر interception اور transformation کی اجازت دیتے ہیں۔

PostToolUse tool result کو model تک پہنچانے سے پہلے intercept کرتا:

```
# مختلف مثال: MCP tools کہ تاریخ سے formats normalize کریں
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Unix timestamp -> ISO 8601
    # "Mar 5, 2025" -> "2025-03-05"
    return normalized_result
```

Outgoing-call interception hook policy والے کی خلاف ورزی کرنے والے actions کو block کرتا:

```
# اوپر سے 500$ مثال: refunds block کریں
@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)
```

Hooks اور prompt instructions میں فرق

Attribute	Hooks	Prompt instructions
Guarantee	**Deterministic** (100%)	**Probabilistic** (>90%, 100% نہیں)
کب استعمال کریں	Critical business rules، financial operations، compliance	General preferences، recommendations، formatting
مثال	Refunds > $500 block کریں	“Try to solve before escalating”

قاعدہ: جب failure ہوں کہ مالی، قانونی، یا حفاظتی نتائج — prompts نہیں، hooks استعمال کریں

باب 4: Model Context Protocol (MCP)

> دستاویزات: MCP | Tools | Resources | Servers

4.1 MCP □□ کیا

Model Context Protocol (MCP) ایک open protocol جو □□ external systems کو Claude □□□ جوڑتا □سے MCP تین بنیادی resource types define □□ کرتا:

1. **Tools** — functions جنہیں agent actions □لیے □کے انجام دینے □□ call کر سکتا (CRUD operations، API calls، command execution)
2. **Resources** — context □لیے □کے data □جسے agent □□ پڑھ سکتا (documentation، database schemas، content catalogs)
3. **Prompts** — عام tasks □لیے □کے predefined prompt templates

4.2 MCP Servers

MCP server ایک process جو □□ MCP protocol implement اور □□ کرتا tools/resources جب □□□ فراہم کرتا آپ MCP server □سے connect □ہیں □کرتے:

- تمام tools خود بخود discover □ہیں □ہو جاتے
- تمام connected servers □کے tools □ہیں □ہوتے □ایک ساتھ دستیاب
- Tool descriptions □کے □ہیں کرتی □طے model گا □کرے استعمال کیسے □انہیں

4.3 MCP Servers کو Configure کرنا

Project configuration (`.mcp.json`) — team usage □لیے □کے:

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

□اہم نکات:

- `.mcp.json` project root اور □□ ہوتا محفوظ میں version control □□ ہوتا میں
- Environment variables (`${GITHUB_TOKEN}`) secrets □ہیں □ہوتے □استعمال □لیے □کے — tokens خود commit □نہیں کیے جاتے

- □ project تمام □ contributors □□ □وتا دستیاب □ لیے □ کے

User configuration (`~/.claude.json`) — personal/experimental servers □ لیے □ کے:

- User □ کے home directory □□ □وتا میں محفوظ
- Version control □ذریعے □ کے share □وتا □نہیں
- Personal experiments اور testing موزوں □ لیے □ کے

Servers کا انتخاب:

- Standard integrations (Jira، GitHub، Slack) موجود □ سے □□ پہلے □ لیے □ کے community MCP servers کو ترجیح دیں
- اپنی custom servers صرف unique، team-specific workflows بنائیں □ لیے □ کے

4.4 MCP میں `isError` Flag

جب MCP tool میں error تو □□ آتا response میں `isError: true` □ یہ □□□ □وتا استعمال agent □ کو بتاتا □□ کہ call fail □□□ □وئی

Structured error (اچھا):

```
{
  "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
  }
}
```

Generic error (anti-pattern):

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

Generic error agent کیا — □نہیں دیتا کوئی معلومات □ لیے □ کے فیصلہ □ سازی کو retry کریں، query یا بدلیں، escalate کریں؟

4.5 MCP Resources

Resources ایسا data جو □میں agent context □ لیے □ کے حاصل کرنے □ کے actions □□ بغیر طلب کر سکتا □کیے:

- Content catalogs (مثلاً تمام project tasks کی فہرست □، hierarchical navigation)

- Database schemas (data structure □لیے □کے □سمجھنے)
- Documentation (API references، internal guides)
- Issue/task summaries

Resource □کا فائدہ: agent □لیے □کے □سمجھنے □یہ کو exploratory tool calls □کے پڑتی نہیں ضرورت کی data موجود □فراہم کرتا □□□ “map” فوری Resource کیا □□□

---

5 باب: Claude Code — Configuration اور Workflows

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

5.1 CLAUDE.md Hierarchy

CLAUDE.md □و instruction file(s) جو □ہیں Claude Code اس کی تین سطحی □□□ استعمال کرتا hierarchy □□:

```
1. User-level: ~/.claude/CLAUDE.md
   - صرف اسی user پر لاگو
   - VCS ذریعہ □کے share □ہوتا نہیں
   - ذاتی preferences اور working style

2. Project-level: .claude/CLAUDE.md یا root CLAUDE.md
   - تمام project contributors پر لاگو
   - VCS ذریعہ □کے manage □□ □ہوتا
   - Coding standards، testing standards، architectural decisions

3. Directory-level: subdirectories میں CLAUDE.md
   - اسی جب directory کی files □□ □ہوتا لاگو □ہو تو کام □کے ساتھ
   - codebase مخصوص □کے حصہ □کے اُس کے conventions
```

□ایک نئی **عام غلطی:** team member کو project instructions □و □کیونکہ ملتیں نہیں `.claude/CLAUDE.md` (project-level) □کے بجائے □کے `~/.claude/CLAUDE.md` (user-level) □میں رکھی گئی تھیں

5.2 `@path` Syntax (File Imports)

CLAUDE.md بیرونی files کو `@path` □ذریعے □کے refer □جس سے، □□ کر سکتا configuration modular □جاتی □□:

```
# Project CLAUDE.md

Coding standards @./standards/coding-style.md میں بیان کیے گئے ہیں
Test requirements @./standards/testing-requirements.md میں ہیں
Project overview @README.md اور میں dependencies @package.json میں ہیں
```

`@path` قواعد کے:

- `@` فوراً بعد کے file path آئے (کوئی space نہیں)
- Relative اور absolute دونوں paths supported ہیں
- Relative paths اُس file مطابق کے resolve جس میں ہیں ہوتے ہیں import کے
- Maximum import nesting depth 5 ہے

اس سے duplication اور کم ہوتی ہے package متعلقہ میں صرف standards شامل ہوتے ہیں

5.3 `.claude/rules/` Directory

`.claude/rules/` monolithic CLAUDE.md جو، کے متبادل ہے، rules کو topic کے حساب سے organize کے کرنے کے لیے استعمال ہوتا ہے:

```
.claude/rules/
  testing.md          -- testing conventions
  api-conventions.md  -- API conventions
  deployment.md       -- deployment rules
  react-patterns.md   -- React patterns
```

YAML frontmatter ساتھ کے `paths` conditional loading:

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

API files کے لیے کے ساتھ async/await کے explicit error handling استعمال کریں
ہر endpoint standard response wrapper واپس کرے
```

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

Tests میں describe/it blocks استعمال ہونے چاہئیں
Data factories استعمال کریں، hardcoding نہیں
Database mock نہ کریں — test database استعمال کریں
```

کیسے کام کرتا ہے:

- Rule صرف اس وقت load جب □□ □وتا □ Claude Code ایسا file edit جو □کر `paths` pattern □سے match کرتا □و
- □اس سے context اور tokens □غیر متعلقہ — □میں □بچتے □ rules load □وتے □نہیں
- Glob patterns آپ کو file type □کے لحاظ سے □ conventions apply □□چاہے، □میں □دیتے □کرنے files codebase میں □وں □کہیں بھی (tests مفید □ہت □کے لیے)

`paths` □والے `.claude/rules/` □بمقابلہ directory-level CLAUDE.md کب استعمال کریں:

- `.claude/rules/` with `paths` — جب conventions کئی directories میں بکھری files (tests، migrations) پر □لاگو □وں
- Directory-level CLAUDE.md — جب conventions کسی خاص directory اور □کہیں اور □وں □سے وابستہ □وں ضروری نہ □وں

5.4 Custom Slash Commands اور Skills

> **نوٹ:** □موجود Claude Code version میں custom commands (`.claude/commands/`) کو skills □بناتے commands `/name` formats □کر دیا گیا □□□ دونوں unify □کے ساتھ (`.claude/skills/`) □□□ supported اب بھی format کا ذکر کرتی □□ — و□ `.claude/commands/` □میں امتحانی گائیڈ

Slash commands reusable prompt templates جو □میں `/name` □کے ذریعے □کے invoke □میں □وتے □وتے:

`.claude/commands/` format (legacy، supported):

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

`.claude/skills/` format (current):

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

Project commands (`.claude/commands/` یا `.claude/skills/`):

- VCS اور □میں □وتے □میں محفوظ repo clone □میں □وتے □میں دستیاب □کرنے والے سب کے لیے
- ٹیم میں consistent workflows □میں □یقینی بناتے

User commands (`~/.claude/commands/` یا `~/.claude/skills/`):

- ذاتی commands جو VCS □کے ذریعے □کے share □وتے □نہیں
- انفرادی workflows □کے لیے

5.5 Skills — `.claude/skills/`

Skills advanced commands جو □میں SKILL.md frontmatter □کے ذریعے □کے configure □میں □وتے □وتے:

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

کا تجزیہ کریں۔ code structure میں directory دی گئی
تیار کریں۔ report پر ایک architectural patterns اور Dependencies
```

Frontmatter parameters:

Parameter	Description
`context: fork`	Skill کو isolated subagent □□□ میں چلاتا Verbose output main session کو pollute کرتا نہیں
`allowed-tools`	□ کون سے □ کو محدود کرتا □□ یہ tools □گے □وں دستیاب (security — مثلاً skill files delete کر نہیں □سکے گی اگر اجازت نہ □ہو)
`argument-hint`	جب skill parameters بغیر □کے invoke □ہو تو argument □کا اشارہ □مانگنے

Skill □بمقابلہ CLAUDE.md کب استعمال کریں:

- **Skill** — مخصوص task □لیے □کے on-demand invocation (review، analysis، generation)
- **CLAUDE.md** — □والے □ہونے □لوڈ □ہمیشہ general standards اور conventions

Personal skills (`~/.claude/skills/`):

- اپنی personal variants □کے بنائیں تاکہ □سے مختلف ناموں teammates □وں متاثر نہ

5.6 Planning Mode □بمقابلہ Direct Execution

Planning mode:

- Model صرف investigate اور plan □□؛ کرتا changes کرتا نہیں
- Codebase explore □لیے □کے □کرنے Read، Grep، Glob □□ استعمال کرتا
- ایک implementation plan □جسے □□ تیار کرتا user approve □□ کرتا
- بغیر side effects □کے safe exploration

Planning mode کب استعمال کریں:

- □بڑے changes (درجنوں files)
- Multiple plausible approaches (microservices: boundaries □کیسے define □ہوں؟)
- Architectural decisions (کون سا framework کیا؟ structure؟)
- Unfamiliar codebase (change □□ سمجھنا ضروری □□پہلے □سے □کرنے)
- Library migrations 45 جو+ files کو متاثر کریں

Direct execution کب استعمال کریں:

- Single-file fixes واضح stack trace ساتھ □کا
- ایک validation check شامل کرنا
- changes اچھی طرح سمجھ□ گئ□، غیر مبہ□م

Combined approach:

1. Investigation اور design □لیے □کا planning mode
2. User plan approve □کر
3. Approved plan implement □لیے □کا □کرنے direct execution

Explore subagent — codebase explore □لیے □کا □کرنے specialized subagent:

- Verbose output کو main context □سے isolate □□ کرتا
- صرف summary □□ واپس کرتا
- Multi-phase tasks میں context-window exhaustion □□ روکتا

5.7 `/compact` Command

`/compact` context compress □لیے □کا □کرنے built-in command □□:

- Pچھلی history کو summarize □کر کا context window □□ خالی کرتا میں جگ□
- Long investigation sessions جب □□ ہوتا □ہوتا میں استعمال verbose tool output □سے context بھر جائ□
- Risk: exact numeric values، dates، اور specific details summarization □میں □ہو سکت□ میں گم

5.8 `/memory` Command

`/memory` sessions درمیان □کا memory manage □لیے □کا □کرنے built-in command □□:

- `CLAUDE.md` file editing □تاکہ، □□ کھولتا □لیے □کا notes، preferences، اور context محفوظ کیے □جا سکیں
- Information sessions اور □□ رہتی □برقرار پار □کا startup پر خود بخود load □□ ہوتی□
- Project conventions، user preferences، frequently used commands، اور current work context محفوظ کرنے □کا □لیے مفید
- ہر □ session میں وہی instructions کا متبادل سمجھانے □دوبار□

5.9 Claude Code CLI for CI/CD

`-p` (یا `--print`) flag:

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

- Non-interactive mode: prompt process □□ کرتا، stdout پر print اور، □□ کرتا exit □□ ہو جاتا□
- User input کا انتظار نہیں کرتا
- CI/CD pipelines میں Claude □چلانے کا واحد درست طریقہ

CI □لیے □کا structured output:

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

- `--output-format json` — output کو JSON □□ میں دیتا
- `--json-schema` — output کو schema مطابق □کو validate □□ کرتا
- Result کو automatically inline PR comments □لیے □کو پوسٹ کرنے □کو parse □□ کیا جا سکتا

Session context isolation: وہی Claude session □نے جس code generate کیا □ہو، اکثر review میں کم مؤثر □ہوتی □□ (model اپنی reasoning context اور اپنے □□ فیصلوں کو ساتھ رکھتا challenge کرنے □کا امکان کم □ہوتا □□)□ Review □لیے □کو independent instance □استعمال کریں

Duplicate comments □سے بچاؤ: نئے commits □کے بعد دوبارہ □کو review کرتے □وقت، پچھلے review results context □کو صرف نئے یا Claude میں شامل کریں اور unresolved issues report □کی □ہدایت دیں□ کرنے

5.10 `fork_session` اور Session Management

`--resume <session-name>` named session کو resume □□ کرتا:

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

- saved context پچھلی ساتھ □کو conversation □□ جاری رکھتا
- Long investigations جو متعدد sessions مفید □لیے □ہوں ان کو پھیلی □ہوں
- Risk: اگر پچھلی session بعد □کے files تو □ہوں بدل چکی tool results stale □ہیں □ہو سکتے

`fork_session` shared context □سے independent branch □□ بناتا:

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

- دونوں forks branch point تک context inherit □کرتے □ہیں
- آزادانہ □طور پر اس کے بعد وہ diverge □کرتے □ہیں
- Approaches compare یا □کرنے strategies test مفید □لیے □کو کرنے

Resume نئی □بجائے □کو session کب شروع کریں:

- Tool results stale □ہوں (بدل چکی □ہوں files)
- کافی وقت گزر چکا □ہو اور context degrade □ہو گیا □ہو
- □پرانے tool data ساتھ □کو resume □□ □ہے: "□ہم نے جو پایا اس کا مختصر خلاصہ □ہے □□ کرنے □کو بجائے □یہ بہتر □□ کہ □سے "... restart □کیا جائے

6 باب: Prompt Engineering — Advanced Techniques

> دستاویزات: Prompt Engineering | Anthropic Cookbook

6.1 Few-shot Prompting

Few-shot prompting جس میں □□ □طریقہ □و expected behavior □لیے □کے □دکھانے prompt 4–2 میں input/output examples □ہیں□ جاتے □ہیں شامل کیے

Few-shot text descriptions □□ مؤثر کیوں □زیادہ □سے:

- □مبہم instruction □زیادہ“ □جیسے precise □□ سمجھا جا سکتا □سے طریقوں کئی کو ”□ں
- Example پر طور مبہم غیر expected format اور decision logic □□ دکھاتا
- Model pattern □نئے کو cases پر generalize □□ کرتا (صرف examples ہیں نہ□راتا د□)

Few-shot examples کی اقسام اور استعمال:

1. **Ambiguous scenarios □لیے □کے examples:**

```
Request: "My order is broken"
Action: Call get_customer -> lookup_order -> check status.
Rationale: “broken” damaged item کو آپ ؛□□ سکتا ہو بھی order details □میں درکار

Request: "Get me a manager"
Action: فوراً escalate_to_human call کریں.
Rationale: Customer پر واضح طور human □کریں کوشش کی حل خود □□□ کرنے رہا مانگ
```

2. **Output formatting □لیے □کے examples:**

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

3. **Acceptable □بمقابلہ problematic code □لیے □کے □الگ کرنے examples:**

```
// Acceptable (flag نہ کریں):
const items = data.filter(x => x.active);

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

4. **Different document formats □سے extraction □کے examples:**

```
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. **Informal measurements □لیے □کے examples:**

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

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

Few-shot خاص طور پر informal اور non-standard measurement units جو □□ میں مؤثر □نکالنا purely rule-based instructions □میں □ہوتی □ہت متنوع □کے لیے

Prompts میں format normalization rules: جب structured output □لیے □کے strict JSON schemas استعمال کریں، prompt میں normalization rules شامل کریں:

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

□اس سے semantic errors □ہاں جیسے □ہوتے □کم JSON syntactically □درست مگر values inconsistent □ہوں

6.2 Explicit Criteria □بمقابلہ Vague Instructions

Bad (vague):

```
Code comments کی accuracy □چیک کریں
Conservative صرف — □رمیں high-confidence findings report □کریں
```

Good (explicit criteria):

```
Comment کو problematic صرف اُس صورت میں flag کریں جب:
1. Comment بیان کا actual code behavior □سے CONTRADICT □کرتا ہو
2. Comment کسی غیر موجود function یا variable □کا حوالہ دے
3. TODO/FIXME comment □کسی ایسے bug کی طرف اشارہ کرے جو code □میں پہلے □سے fix □ہو چکا ہو

Flag کریں □نہ:
- □ایسے comments جو صرف style □کے لحاظ سے پرانے ہوں
- معمولی wording inaccuracies
- Missing comments (الگ □یہ category □□)
```

Severity criteria examples □کے ساتھ define کریں:

```
CRITICAL: User کے لیے runtime failure
  Example: Payment process کرتے وقت NullPointerException

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

MEDIUM: Logic bug بغیر فوری impact کے
  Example: Wrong sorting، off-by-one error

LOW: Code quality
  Example: Duplication، چھوٹے data کے لیے suboptimal algorithm
```

6.3 Prompt Chaining

Prompt chaining ایک complex task کو focused steps کی sequence □□ توڑتا میں:

```
Step 1: auth.ts analyze کریں (صرف local issues)
       -> Output: auth.ts میں issues کی list

Step 2: database.ts analyze کریں (صرف local issues)
       -> Output: database.ts میں issues کی list

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

□□ کیوں اہم □ہے:

- **Attention dilution** جب—□□ بچاتا سے model بہت سی ایک ساتھ files تو کچھ □□ دیکھتا files کے bugs miss □ سکتا shallow commentary □□ کر سکتا اور باقی پر
- ہر file لیے کے consistent analysis quality □□ یقینی بناتا
- Cross-file interactions الگ لیے کے analysis □□ کی اجازت دیتا

Prompt chaining بمقابلہ dynamic decomposition کب استعمال کریں:

- **Prompt chaining** — predictable، repeatable tasks (code review، file migrations)
- **Dynamic decomposition** — open-ended investigations جہاں subtasks execution ہوتے کے دوران واضح ہیں

6.4 “Interview” Pattern

Solution implement □لے پہلے سے کرنے Claude clarifying questions □□ پوچھتا:

```
Claude: "API کے لیے caching implement کرنے سے پہلے چند سوالات:
1. آپ TTL یا event-based میں سے کون سی cache invalidation strategy پسند کرتے ہیں؟
2. اگر cache unavailable ہو تو کیا stale data قابلِ قبول ہے؟
3. Caching per-user ہونی چاہیے یا global؟
4. Cache کرنے کے لیے expected data volume کیا ہے؟"
```

□□ کب مفید □ہے:

- Unfamiliar domain (fintech، healthcare، legal systems)
- غیر واضح implications والے tasks (cache strategies، failure modes)
- متعدد viable approaches جہاں بہترین انتخاب context پر منحصر ہو

6.5 Validation اور Retry-with-Feedback

جب extracted data validation fail کرے:

```
Step 1: Document سے data extract کریں
Step 2: Validate کریں (Pydantic، JSON Schema، business rules)
Step 3: اگر error ہو — context کے ساتھ retry کریں:
  - Original document
  - Previous (incorrect) extraction
  - Specific error: "Field 'total' = 150, but sum(line_items) = 145. Re-check values."
```

Retry کب مؤثر ہوگا:

- Format errors (date غلط format میں)
- Structural errors (field غلط جگہ)
- Arithmetic inconsistencies (model دوبارہ check کر سکتا ہے)

Retry کب مدد نہیں کرے گا:

- Information source document میں موجود ہی نہیں
- Required context external ہے (data کسی اور document میں ہے جو فراہم نہیں کیا گیا)

Pydantic بطور validation tool: Pydantic Python library جو ہے schema-based data validation کے لیے استعمال ہوتی ہے۔ امتحان کے لیے اہم نکات یہ ہیں:

- **Structural validation:** types، requiredness، enum constraints JSON بعد کے code میں چیک کیے جاتے ہیں
- **Semantic validation:** custom validators business logic enforce کرتے ہیں (sum of items equals total؛ start_date < end_date)
- **Validate–retry loops:** Pydantic validation failure پر error message بنا کر Claude کو error context کے ساتھ دوبارہ prompt کریں
- **JSON Schema generation:** Pydantic models `tool_use` کے لیے JSON Schema generate کر سکتے ہیں، جو ایک single source of truth ہے فراہم کرتا ہے

6.6 Self-correction

Internal contradictions detect کرنے کا pattern:

```
{
  "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}
  ]
}
```

آپ `conflict_detected` دونوں نکالتا □□ — اگر و□ مختلف □وں تو Model stated value اور computed value کو discrepancy handle □□□ دیتا □کرنا

---

7 باب: Message Batches API

> دستاویزات: Message Batches

7.1 Overview

Message Batches API کو آپ asynchronous processing □لیے □کے requests □کے batches submit □□ دیتا □کرنا:

Attribute	Value
Savings	synchronous calls **50** میں مقابلہ □کے%
Processing window	**24** □زیادہ سے □زیادہ **hours** (latency SLA guarantee نہیں)
Multi-turn tool calling	**Supported** نہیں (ایک request = ایک response)
Correlation	request اور response □لیے □کے جوڑنے کو `custom_id` field

7.2 Batch API بمقابلہ Synchronous API کب استعمال کریں

Task	API	Why
Pre-merge PR check	**Synchronous**	Developer 24 hours انتظار کر رہا □□؛ قابلِ قبول نہیں
Overnight tech-debt report	**Batch**	Result 50 صبح تک چاہیے؛ savings%
Weekly security audit	**Batch**	50 فوری نہیں؛ savings%
Interactive code review	**Synchronous**	فوری response □□ درکار
10,000 documents process کرنا	**Batch**	Bulk processing؛ savings میں □م□

7.3 `custom_id` کا استعمال

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

`custom_id` □□ دیتا □ کرنا □ یہ آپ کو:

- Result کو original document □سے link کرنا
- Failure کی صورت میں صرف failed documents □دوبارہ submit کرنا
- Successful documents □کو دوبارہ process بچنا □سے کرنا

7.4 Batch میں Failures کو Handle کرنا

1. 100 documents کا batch submit کریں
2. 95 succeed 5 ؛کریں fail □وں (context limit exceeded)
3. Failures کو `custom_id` □سے identify کریں
4. Strategy بدلیں (مثلاً long documents کو chunks کریں میں تقسیم)
5. 5 صرف اُن failed documents □کو دوبارہ submit کریں

7.5 SLA Planning

30 اگر آپ کو hours میں result اور □یہ□چا Batch API 24 کو hours □یں □تک لگ سکتا:

- Submission window: 30 - 24 = **6 hours**
- Batches کو deadline 24 کم از کم □سے hours □لا□پ submit □وگا کرنا
- Frequent submissions 4 □لیے □کے-hour windows میں تقسیم کریں

---

8 باب: Task Decomposition Strategies

8.1 Fixed Pipelines (Prompt Chaining)

ر□ step □سے □لا□پ defined □□ وتا□:

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

کب استعمال کریں:

- Task structure predictable ہو (reviews کی ایک جیسی template follow کریں)
- تمام steps کو پہلے سے معلوم ہوں
- آپ کو stability اور reproducibility چاہیے

8.2 Dynamic Adaptive Decomposition

Subtasks intermediate results پر بنیاد کی generate میں ہوتے ہیں:

```
1. "Legacy codebase کے لیے tests add کریں"
2. -> پہلے structure map کریں (Glob, Grep)
3. -> 3 ملا: modules بغیر tests 2 کے، partial coverage کے ساتھ
4. -> Prioritize: payments module سے شروع کریں (high risk)
5. -> کام کے دوران: external API پر dependency دریافت ہوئی
6. -> Adapt: tests پہلے سے لکھنے external API کے لیے mock add کریں
```

کب استعمال کریں:

- Open-ended investigative tasks
- جب مکمل scope شروع میں معلوم نہ ہو
- ہر جب step پچھلے step کے results پر منحصر ہو

8.3 Multi-pass Code Review

10+ files والی pull requests کے لیے:

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

14 files کی ایک پر pass کیوں خراب ہے:

- Attention dilution: کچھ files کی deep analysis، کچھ کی shallow
- Inconsistent comments: ایک file میں pattern flag، دوسری میں approve
- Missed bugs: cognitive overload کی وجہ سے واضح errors میں چھوٹ جاتے ہیں

---

9 باب: Escalation اور Human-in-the-Loop

9.1 کب Human کی طرف Escalate کریں

Escalation triggers (واضح rules):

Situation	Action

Customer □□ واضح طور پر کہ “get me a manager”	فوراً escalate کریں؛ حل کرنے کی کوشش نہ کریں
Policy request کو cover نہ کرتی ہو	Escalate کریں (مثلاً competitor price matching جب policy خاموش ہو)
Agent پیش رفت نہ کر سکے	مناسب تعداد میں attempts کے بعد escalate کریں
Threshold سے اوپر financial operation	Escalate کریں (ترجیحاً hook کے ذریعے enforce، prompt کے ذریعے نہیں)
Customer وقت تلاش کرتے multiple matches	مزید identifiers مانگیں؛ اندازے نہ لگائیں

جو reliable trigger □□ نہیں:

Unreliable method	Why it fails
Sentiment analysis	Customer کا mood case complexity سے correlate نہیں کرتا
Model self-rated confidence (1–10)	Model confidently غلط ہو سکتا ہے؛ calibration □□ کمزور
Automatic classifier	Overengineering□□ ممکن ؛ training data ہو نہ موجود

9.2 Escalation Patterns

Immediate escalation:

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

Resolve کرنے کی کوشش کے بعد escalation:

```
Customer: "My refrigerator broke two days after purchase"
Agent: [order چیک کرتا ہے، warranty replacement offer کرتا ہے]
اگر customer مطمئن نہ ہو -> escalate
```

Nuanced escalation (acknowledge → resolve → reiteration پر escalate):

```
Customer: "This is outrageous, I'm very unhappy with the quality!"
Agent: [frustration acknowledge] "میں آپ کی frustration سمجھتا ہوں."
       [resolution offer] "میں refund یا replacement پیش کر سکتا ہوں."
Customer: "No, I want to talk to someone!"
Agent: [customer دوبارہ insist کرے -> immediate escalation]
```

بنیادی اصول: پہلے emotion acknowledge کریں، پھر concrete solution دیں، اور صرف اسی وقت escalate کریں جب customer human □□ کی خواہش دہرائے dissatisfaction پر اظہار پہلے کے escalate نہ کریں (یہ manager □□ برابر نہیں کے مانگنے)

Policy gap پر escalation:

```
Customer: "Competitor X has this item 30% cheaper—give me a discount"
Policy: صرف اپنی site پر price adjustments cover □□ کرتی
Agent: [escalate — policy competitor price matching کو cover نہیں کرتی]
```

9.3 Structured Handoff Protocols

Escalation پر agent کو structured summary human □یا□ کو دینی چا□یے:

```
{
  "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"
}
```

Human operator کو full conversation transcript □یا صرف □و — □وتی □یں نہ□یں تک رسائی summary □□□ دیکھتا □ونی چا□یے□□ self-contained اس لیے □یا مکمل اور

9.4 Confidence Calibration اور Human Oversight

Data extraction systems □یا □کہ:

1. **Field-level confidence scores:** model □ر extracted field □یا □کہ confidence score □□ نکالتا
2. **Calibration:** labeled validation sets □کہ استعمال کر thresholds tune کریں
3. **Routing:**
   - High confidence + stable accuracy -> automated processing
   - Low confidence یا ambiguous sources -> human review

Stratified random sampling:

- High-confidence extractions □سے باقاعدگی بھی □یا □کہ sample audit کریں
- Aggregate 97% accuracy کسی مخصوص document type 40 □یا □کہ% errors □□ چھپا سکتی
- Accuracy کو صرف overall □یں نہ، document type اور field □سے حساب □کہ analyze کریں

باب 10: Multi-agent Systems میں Error Handling

10.1 Error Categories

Category	Examples	Retryable	Agent action
Transient	Timeout، 503، network failure	Yes	Exponential backoff ساتھ □کا retry
Validation	Invalid input format، missing required field	No (input fix کریں)	Request modify کریں اور retry کریں
Business	Policy violation، threshold exceeded	No	User کو سمجھائیں؛ alternative پیش کریں
Permission	Access denied	No	Escalate کریں

10.2 Error-handling Anti-patterns

Anti-pattern	Problem	Correct approach
Generic status “search unavailable”	Coordinator decide □نہیں کر سکتا کہ recover □□ کرنا □کیسے	Error type، query، partial results، alternatives واپس کریں
Silent suppression (empty result = success)	Coordinator □□ سمجھتا match □نہیں □ہوا تھا failure ملا، حالانکہ	“no results” اور “search failure” میں فرق کریں
ایک failure پر پورا workflow abort کرنا	تمام partial results □ضائع □ہو جاتے □ہیں	Partial results □کے ساتھ جاری رکھیں؛ gaps annotate کریں
Subagent میں infinite retries	Latency اور resources کا ضیاع	Local recovery (1–2 retries)، پھر coordinator تک propagate

10.3 Structured Subagent Error

```
{
  "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"
}
```

□ coordinator □□ ضروری معلومات دیتا □ لیے □ کہ □ کرنے □ فیصلہ کو:

- Modified query ساتھ □ کے retry کریں؟
- Partial results استعمال کریں؟
- کسی اور subagent کو delegate کریں؟
- اس section بغیر جاری رکھیں اور □ کے gap annotate کریں؟

10.4 Final Synthesis میں Coverage Annotations

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

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

### Music (PARTIAL COVERAGE — search agent timeout)
[partial results]
⚠️ Note: اس section کی coverage search agent timeout □□ محدود □ سے وجہ کی.

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

11 باب: Production Systems میں Context Management

11.1 Facts کو Separate Block میں Extract کریں

Conversation history □بجائ□ □ک پر انحصار کرن□ (جو summarization دوران □ک degrade □□ جاتی □□و)، key facts کو structured block میں extract کریں:

```
=== CASE FACTS (جب بھی نیا fact آئ□ update □کیا جائ□) ===
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
===
```

□ی block □ر prompt □□چا، میں شامل کریں history □ی کتنی summarized □و□ □ن کیوں

11.2 Tool Results کو Trimming کرنا

اگر `lookup_order` 40+ fields مگر □□ واپس کرتا current task □یں درکار 5 صرف □لی□ □ک:

```python
# PostToolUse hook: صرف relevant fields رکھیں
@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"]
    }
```

□س اس context اور □□ بچتا noise □□□ کم □وتی

11.3 Position-aware Input

Lost-in-the-middle effect □وئ□ □ذہن میں رکھتے □ک critical information رکھیں:

```
[KEY FINDINGS — اوپر]
3 critical vulnerabilities ملیں...

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

[ACTION ITEMS — آخر میں]
Priority: merge ہونے سے پہلے auth.ts vulnerabilities fix کریں.
```

11.4 Scratchpad Files

Long investigations میں agent key findings scratchpad file میں لکھ سکتا ہے:

```
# investigation-scratchpad.md
## Key findings
- PaymentProcessor in src/payments/processor.ts inherits from BaseProcessor
- refund() تین places سے call ہوتی ہے: OrderController، AdminPanel، CronJob
- External PaymentGateway API limit: 100 req/min
- Migration #47 نے refund_reason (NOT NULL) add کیا — 01-12-2024
```

جب context degrade ہو جائے (نئی یا session میں)، agent discovery بجائے چلانے دوبارہ scratchpad consult کر سکتا ہے

11.5 Context بچانے کے لیے Subagents کو Delegate کرنا

```
Main agent: "payments module کی dependencies investigate کریں"
  -> Subagent (Explore): 15 files read کرتا ہے، imports trace کرتا ہے
  -> Returns: "Payments depends on AuthService, OrderModel, and the external
PaymentGateway API"

Main agent: context میں 15 files ایک جگہ کے بجائے 2 line رکھتا ہے
```

Separate context layer: Multi-agent systems میں ہر subagent محدود context budget — میں کام کرتا ہے اور صرف اپنے task کے لیے ضروری information ملتی ہے Coordinator ایک separate context layer کے طور پر context allocate کرتا ہے، اور global state store کرتا ہے، subagent outputs aggregate کرتا ہے اور کام کرتا ہے: ہر subagent کو ایسی information جو بھر دیتا ہے اس سے agent window نہیں ہوتا، جہاں ایک "context leakage" ہوتا ہے دوسروں کے لیے غیر متعلق ہو

Subagents کے لیے constrained context budgets:

- Minimal context بھیجیں: specific task + ضروری data
- Subagent کو raw data dumps بجائے structured results واپس کرنے کی ہدایت دیں
- `allowedTools` سے subagent کے toolset کو limit کم — کریں tools کم مطلب distractions اور کم context cost

11.6 Structured State Persistence (crash recovery □لیے □کے)

ہر□ agent اپنی state معلوم ایک location پر export □□ کرتا:

```
// 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"]
}
```

Coordinator resume پر ایک manifest load □□ کرتا:

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

---

12 باب: Provenance کو محفوظ رکھنا

12.1 Attribution Loss Problem

جب متعدد sources □سے results summarize تو، □ہیں جاتے □کیے “claim → source” link □□ کھو سکتا □ہے گم:

```
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 Conflicting Data کو Handle کرنا

جب دو sources مختلف values دیں:

```
{
  "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": "Methodology اور time period فرق کا"
}
```

کسی ایک value دونوں کو □کریں □نہ منتخب کو اندھا دھند attribution اور ساتھ محفوظ رکھیں □کہ coordinator کو فیصلہ □کرنا □دیں

12.3 Correct Interpretation □لیے □کے Dates شامل کریں

Dates بغیر □کے temporal differences کو contradiction □□ سمجھ لیا جا سکتا:

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

12.4 Content Type مطابق □کے Render کریں

ہی □ر چیز کو ایک format ٹھونسیں □نہ میں:

- Financial data -> tables
- News اور analysis -> prose
- Technical findings -> structured lists
- Time series -> chronological ordering

13 باب: Claude Code □ کے Built-in Tools

13.1 Tool Selection Reference

Task	Tool	Example
نام/pattern □ سے files تلاش کرنا	**Glob**	`**/*.test.tsx`, `src/components/**/*.ts`
File contents اندر □ کے search کرنا	**Grep**	Function name، error message، import
File مکمل پڑھنا	**Read**	Analysis □ لیے □ کے file load کرنا
نیا file بنانا	**Write**	Scratch □ سے file create کرنا
□موجود file میں precise edit	**Edit**	Unique text match □ ذریعے □ کے specific snippet replace کرنا
Shell command چلانا	**Bash**	git، npm، tests، build

13.2 Incremental Investigation Strategy

تمام files ایک ساتھ پڑھیں □ نہ سمجھ بوجھ کو رفتہ رفتہ □ بنائیں:

```
1. Grep: entry points تلاش کریں (function definition، export)
2. Read: ملنے والی files پڑھیں
3. Grep: usages تلاش کریں (import، calls)
4. Read: consumer files پڑھیں
5. جب تک مکمل picture نہ مل جائے repeat کریں
```

13.3 Fallback: Edit □ بجائے □ کے Read + Write

جب Edit non-unique text match □ سے □ کی وجہ سے fail □ جائے □ تو:

1. Read — مکمل file content لوڈ کریں
2. Content کو programmatically modify کریں
3. Write — updated version لکھ دیں

□حصہ II: امتحانی ڈومین نوٹس

ڈومین 1: Agent Architecture اور Orchestration (27%)

1.1 Autonomous Task Execution کے لیے Agentic Loops Design کرنا

Key knowledge:

- Agent loop lifecycle: Claude request بھیجیں، `stop_reason` (`"tool_use"` بمقابلہ `"end_turn"`) چیک کریں، tools execute کریں، اگلی iteration کے لیے results واپس کریں
- Tool results conversation history میں append کریں تاکہ ہوتے ہوئے model اگلا action طے کر سکے
- Model-driven decision making (Claude اگلا tool کے چنتا ہے) بمقابلہ hard-coded decision trees

Key skills:

- Flow control: جب `stop_reason = "tool_use"` ہو تو loop جاری رکھیں اور `"end_turn"` پر رک جائیں
- Iterations درمیان کے tool results کو context میں append کرنا
- Anti-patterns بچنا سے: completion کے لیے assistant text parse کرنا، primary stopping mechanism کے طور پر arbitrary iteration limits استعمال کرنا

1.2 Multi-agent Systems (Coordinator–Subagent) Orchestrate کرنا

Key knowledge:

- Hub-and-spoke architecture: coordinator تمام inter-agent communication، error handling، اور routing کا مالک ہوتا ہے
- Subagents isolated context خود بخود وے — وے میں کام کرتے ہیں coordinator کی history inherit نہیں کرتے
- Coordinator responsibilities: task decomposition، delegation، result aggregation، dynamic selection of subagents
- Coordinator کی overly narrow decomposition کا خطرہ

Key skills:

- Research coverage کو subagents کے درمیان تقسیم کریں تاکہ duplication کو کم
- Iterative refinement loops implement کریں (coordinator synthesis evaluate اور کرے tasks re-route کرے)
- Observability تمام communication coordinator کے ذریعے کے route کریں

1.3 Subagent Calls، Context Passing، اور Spawning Configure کرنا

Key knowledge:

- `Task` tool subagents spawn کرتا ہے؛ coordinator کے `allowedTools` میں `"Task"` شامل ہونا چاہیے
- Subagent context prompt میں explicitly شامل ہونا چاہیے؛ subagents parent context inherit نہیں کرتے
- `AgentDefinition` configuration: descriptions، system prompts، tool constraints
- Alternatives explore کرنے کے لیے `fork_session` کے ذریعے session management

Key skills:

- Previous agents کے full outputs subagent prompt میں شامل کریں
- Context pass وقت کرتے data اور metadata کو separate کرنے کے لیے structured formats استعمال کریں
- ایک coordinator turn میں متعدد `Task` calls کے ذریعے parallel subagents spawn کریں
- Coordinator prompts goals اور quality criteria کے طور پر لکھیں، step-by-step instructions کے طور پر نہیں

1.4 Enforcement اور Handoff Patterns کے ساتھ Multi-step Workflows Implement کرنا

Key knowledge:

- Workflow ordering کے لیے **programmatic enforcement** (hooks، preconditions) اور **prompt guidance** کے درمیان فرق
- جب deterministic guarantees درکار ہوں (مثلاً مالی operations سے پہلے identity verification)، prompts کافی نہیں ہوتے
- Escalation دوران کے structured handoff protocols (customer ID، reason، recommended action)

Key skills:

- ایسے programmatic preconditions جو downstream calls کو block کریں جب تک پہلے steps مکمل نہ ہوں (مثلاً `get_customer` کے verified ID تک واپس کرنے `process_refund` block کریں)
- Multi-aspect customer requests کو الگ items میں تقسیم کریں
- Human کو escalate وقت کرتے structured summaries produce کریں

1.5 Tool Calls Intercept اور کرنے Data Normalize کرنے کے لیے Agent SDK Hooks

Key knowledge:

- Hook patterns (مثلاً `PostToolUse`) tool results کو model کے consume کرنے سے پہلے intercept میں کرتے ہیں
- Hooks outgoing calls intercept کر کے compliance rules enforce میں کرتے ہیں (مثلاً threshold اوپر سے refunds block کرنا)
- Hooks **deterministic guarantees** دیتے ہیں، جبکہ prompt instructions **probabilistic compliance** دیتی ہیں

Key skills:

- `PostToolUse` hooks □سے data formats normalize کریں (Unix timestamps، ISO 8601، numeric status codes)
- Policy-violating actions block □لیے کے □کرنے interception hooks اور استعمال کریں escalation کی طرف redirect کریں
- جب business rules کو guaranteed compliance □ہو تو □چاہیے prompts □بجائے کے hooks منتخب کریں

1.6 Complex Workflows □لیے کے Task Decomposition Strategies

Key knowledge:

- **Fixed pipelines** (prompt chaining) □بمقابلے intermediate results کی بنیاد پر **dynamic adaptive decomposition**
- Prompt chaining: sequential steps (□ہر file کو الگ analyze پھر، کریں integration pass)
- Adaptive investigation plans جو discovered information کی بنیاد پر subtasks generate □کرتے □ہیں

Key skills:

- Predictable multi-aspect reviews □لیے کے prompt chaining؛ استعمال کریں open-ended investigations □کے □لیے dynamic decomposition
- Large code reviews کو per-file analysis + separate cross-file integration pass میں تقسیم کریں
- Open-ended tasks کو degrade □ہونے پر: کریں structure map پھر، کریں prioritized plan بنائیں

1.7 Session State، Resuming، اور Forking

Key knowledge:

- Named sessions continue □لیے کے □کرنے `--resume <session-name>`
- Shared context □سے independent investigation branches □لیے کے □بنانے `fork_session`
- Resume وقت □کرتے agent کو file changes □کی اہمیت بتانے
- Structured summary □کے ساتھ نئی session، stale results □کے ساتھ resuming □زیادہ سے reliable □ہو سکتی □□

Key skills:

- Named investigation sessions continue □لیے کے □کرنے `--resume` استعمال کریں
- Approaches parallel compare □لیے کے □کرنے `fork_session` استعمال کریں
- Resuming (context ابھی current □□) اور نئی session □شروع کرنے (results stale □ہیں) کے □درمیان انتخاب کریں

5.10 `fork_session` اور Session Management

`--resume <session-name>` named session کو resume □□کرتا:

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

- saved context پچھلی ساتھ کے conversation کو جاری رکھتا ہے
- Long investigations جو متعدد sessions مفید ہیں ان کے لیے پھیلی ہوں
- Risk: اگر پچھلی session بعد کے files تو بدل چکی ہوں tool results stale ہیں ہو سکتے

`fork_session` shared context سے independent branch بناتا ہے:

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

- دونوں forks branch point تک context inherit کرتے ہیں
- آزادانہ طور پر اس کے بعد وہ diverge کرتے ہیں
- Approaches compare کرنے یا strategies test کرنے کے لیے مفید

Resume نئی بجائے کے session کب شروع کریں:

- Tool results stale ہوں (files بدل چکی ہوں)
- کافی وقت گزر چکا ہو اور context degrade ہو گیا ہو
- پرانے tool data ساتھ کے resume کرنے کے بجائے یہ بہتر ہے کہ "ہم نے جو پایا اس کا مختصر خلاصہ یہ ہے: ... restart کیا جائے" سے

---

2 ڈومین: Tool Design اور MCP Integration (18%)

2.1 Clear Descriptions ساتھ کے Tool Interfaces Design کرنا

Key knowledge:

- Tool descriptions **بنیادی** وہ **mechanism** ہیں جن سے LLM tools select کرتا ہے؛ minimal descriptions unreliable selection دیتی ہیں
- Input formats، example queries، edge cases، اور applicability boundaries کی اہمیت شامل کرنے
- Ambiguous یا overlapping descriptions misrouting کا سبب بنتی ہیں
- System prompt wording tools ساتھ غیر ارادی associations بنا سکتی ہے

Key skills:

- ایسی descriptions لکھیں جو tool کو similar alternatives پر واضح طور پر distinguish کریں
- Functional overlap ختم کرنے کے لیے tools rename کریں (مثلاً `analyze_content` -> `extract_web_results`)
- General-purpose tools کو specialized tools میں تقسیم کریں جن کے input/output contracts واضح ہوں

2.2 MCP Tools □لیے □کے Structured Error Responses Implement کرنا

Key knowledge:

- MCP tool responses میں `isError` flag
- **Transient errors** (timeouts)، **validation errors** (bad input)، **business errors** (policy violations)، اور **access/permission errors** کے درمیان فرق
- Generic errors ("Operation failed") صحیح recovery decisions میں □ روک دیتے □
- Retryable اور non-retryable errors میں فرق

Key skills:

- `errorCategory` (transient/validation/permission)، `isRetryable`، اور human-readable message جیسی structured metadata واپس کریں
- Business-rule violations واضح □لیے □کے user-facing explanations کے ساتھ □کے `retryable: false` استعمال کریں
- Transient failures □لیے □کے subagents میں local recovery □ صرف وہ کریں؛ errors propagate کریں جو وہ □ خود حل نہ □ کر سکیں
- Access failures (retry decision) اور valid empty results (no matches) میں فرق کریں

2.3 Agents میں Tools Allocation اور `tool_choice` Configure کرنا

Key knowledge:

- □زیادہ بہت tools (5–4 □بجائے □کے 18 مثلاً) tool selection reliability □□ کم کرتی
- Specialization □بہتر سے □ tools □والے □رکھنے agents □میں غلط استعمال کرتے □ انہیں
- Scoped tool access: صرف role-relevant tools + محدود cross-role utilities
- `tool_choice`: `"auto"`, `"any"`, اور forced tool selection (`{"type": "tool", "name": "..."}`)

Key skills:

- □ہر subagent کا toolset صرف relevant tools تک محدود رکھیں
- General tools کو constrained alternatives بدلیں □سے (مثلاً `fetch_url` -> `load_document`)
- `tool_choice: "any"` □استعمال کریں تاکہ text answer □بجائے □کے tool call □ یقینی □ہو
- Specific tool force □کریں تاکہ execution order assured □ہو

2.4 MCP Servers کو Claude Code اور Agent Workflows میں Integrate کرنا

Key knowledge:

- MCP server scope: □لیے □کے ٹیموں project (`.mcp.json`) □بمقابلہ experiments □لیے □کے user (`~/.claude.json`)
- Secrets management □لیے □کے `.mcp.json` میں environment variable substitution (مثلاً `${GITHUB_TOKEN}`)

- تمام connected MCP servers کے tools connection پر discover ایک ساتھ اور available ہو جاتے ہیں
- MCP resources بطور “content catalogs” (task summaries، database schemas) exploratory tool calls کم کرتے ہیں

Key skills:

- Shared MCP servers کو project `.mcp.json` میں env-var-based tokens کے ساتھ configure کریں
- Personal/experimental servers کو `~/.claude.json` میں رکھیں
- Standard integrations کے لیے community MCP servers کو custom servers پر ترجیح دیں

2.5 Built-in Tools (Read, Write, Edit, Bash, Grep, Glob) منتخب اور Apply کرنا

Key knowledge:

- **Grep**: file contents اندر کے search (function names، error messages، imports)
- **Glob**: name/extension patterns سے files تلاش کرنا
- **Read/Write**: full-file operations؛ **Edit**: unique text matches کے ذریعے precise changes
- اگر Edit non-unique matches سے وجہ کی fail ہو تو Read + Write پر واپس جائیں

Key skills:

- Content search کے لیے Grep اور pattern-based discovery کے لیے Glob استعمال کریں
- سمجھ بوجھ رفتہ رفتہ بنائیں: entry points کے لیے Grep، پھر flows trace کرنے کے لیے Read
- Wrapper modules کے ذریعے function usage trace کریں

---

3 ڈومین: Claude Code Configuration اور Workflows (20%)

3.1 Hierarchy، Scope، اور Modular Organization کے ساتھ CLAUDE.md Configure کرنا

Key knowledge:

- CLAUDE.md hierarchy: user (`~/.claude/CLAUDE.md`)، project (`.claude/CLAUDE.md` یا root `CLAUDE.md`)، اور directory-level (subdirectories میں CLAUDE.md)
- User-level settings صرف ایک user اور نہیں ہوتے پر لاگو ہوتے ہیں VCS کے ذریعے share نہیں
- External files refer کرنے کے لیے `@path` syntax (مثلاً `@./standards/coding-style.md`) تاکہ CLAUDE.md modular بنے
- Monolithic CLAUDE.md بجائے کے topic-focused rule files کے لیے `.claude/rules/` directory

Key skills:

- Hierarchy issues diagnose کریں (نئی team member کو instructions وہ نہیں ملتیں کیونکہ وہ اس لیے نہیں user-level میں تھیں، project-level میں نہیں)
- ہر package کے CLAUDE.md میں standards selectively include کرنے کے لیے `@path` (مثلاً `@./standards/testing.md`) استعمال کریں
- بڑے CLAUDE.md کو متعدد `.claude/rules/` files (testing.md، api-conventions.md، deployment.md) میں تقسیم کریں

3.2 Custom Slash Commands اور Skills اور بنانا Configure کرنا

Key knowledge:

- `.claude/commands/` میں **Project commands** (VCS کے ذریعے shared) بمقابلہ `~/.claude/commands/` میں **user commands**
- `.claude/skills/` میں SKILL.md frontmatter: `context: fork`، `allowed-tools`، `argument-hint`
- `context: fork` skill کو isolated subagent context تاکہ وہ main session pollute نہ ہو میں چلاتا ہے
- Personal skill variants `~/.claude/skills/` میں مختلف ناموں سے رکھ سکتے ہیں

Key skills:

- Project slash commands `.claude/commands/` میں رکھیں تاکہ پوری team انہیں حاصل کرے
- Verbose output والے skills isolate کرنے کے لیے `context: fork` استعمال کریں
- Skill استعمال کرنے کے tools کے لیے محدود کرنے کے لیے `allowed-tools` استعمال کریں
- Required parameters مانگنے کے لیے `argument-hint` استعمال کریں

3.3 Conditional Convention Loading کے لیے Path-specific Rules استعمال کرنا

Key knowledge:

- `.claude/rules/` files میں YAML frontmatter `paths` تاکہ وہ شامل ہو سکتا ہے glob patterns مطابق کے rules activate ہوں
- Path-scoped rules صرف matching files edit وقت کرتے load ہیں ہوتے، context اور tokens بچتے ہیں
- Glob-based path rules directory-level CLAUDE.md جب ہیں سے بہتر ہو سکتے ہیں conventions کئی directories میں apply ہوں (مثلاً tests)

Key skills:

- `paths: ["terraform/**/*"]` کے ساتھ `.claude/rules/` files صرف بنائیں تاکہ matching files پر load ہوں
- Glob patterns (`**/*.test.tsx`) استعمال کریں تاکہ location قطع نظر سے file type کے لحاظ سے conventions apply ہوں
- جب conventions codebase تو ہوں پھیلی بھر میں directory-level CLAUDE.md بجائے کے path-specific rules کو ترجیح دیں

3.4 Planning Mode □بمقابل Direct Execution کب استعمال کریں

Key knowledge:

- **Planning mode**: complex tasks، □بڑ changes، multiple viable approaches، اور architectural decisions □لیے کا
- **Direct execution**: □گئ سمجھ□ اچھی طرح ،□ساد changes (مثلاً single validation شامل کرنا)
- Planning mode changes □□پہلے □سے codebase کی safe exploration □□ ممکن بناتا
- Explore subagent verbose discovery output isolate □□ کرتا

Key skills:

- Architectural consequences □والے tasks (microservices، 45+ files والی migrations) □لیے □کے planning mode استعمال کریں
- واضح stack trace اور single file □والے fixes □لیے □کے direct execution استعمال کریں
- Multi-phase tasks میں context-window exhaustion □لیے □کے روکنے Explore subagent استعمال کریں
- Approaches کو combine کریں: discovery □لیے □کے plan، implementation □لیے □کے execute

3.5 Progressive Improvement □لیے □کے Iterative Refinement

Key knowledge:

- Concrete input/output examples expectations communicate □میں □مؤثر طریقے □سے سب کا کرنے
- **Test-driven iteration**: □□پہلے tests پھر ،لکھیں failures کی بنیاد پر iterate کریں
- "Interview" pattern: Claude clarifying questions □تاکہ □□ پوچھتا non-obvious design considerations □سامنے آئیں
- تمام issues ایک message میں کب دیں (interdependent) اور sequentially کب (independent)

Key skills:

- Transformation requirements 3–2 □لیے □کے واضح کرنے concrete input/output examples فراہم کریں
- Implementation □□پہلے □سے expected behavior، edge cases، اور performance requirements ساتھ □کے test sets بنائیں
- Design aspects (cache invalidation، failure modes) □لیے □کے لانے □سامنے interview pattern استعمال کریں
- Edge cases □لیے □کے sample inputs اور expected outputs ساتھ □کے concrete test cases دیں

3.6 Claude Code کو CI/CD Pipelines میں Integrate کرنا

Key knowledge:

- Automated pipelines میں non-interactive mode □لیے □کے `-p` (یا `--print`) flag
- CI میں structured output □لیے □کے `--output-format json` اور `--json-schema`
- CLAUDE.md project context (testing standards، review criteria) □□ فراہم کرتا
- **Session context isolation**: code generate کرنے والی same session □سے review □ہوتی کرنے میں کم مؤثر □□

Key skills:

- Interactive input پر hang □ لیے □ کے □ بچنے □ سے □ ہوئے CI میں Claude Code کو `-p` ساتھ چلائیں □
- Structured results (مثلاً inline PR comments) □ لیے □ کے `--output-format json` + `--json-schema` استعمال کریں
- □ نئے commits □ بعد دوبارہ □ کے run □ وقت پچھلے کرتے review results شامل کریں (صرف new/unfixed issues report کریں)
- Test generation quality □ لیے □ کے □ بہتر کرنے CLAUDE.md میں testing standards اور available fixtures document کریں
- □ نئے tests generate وقت □ کرتے existing test files کو context □ تاکہ میں شامل کریں duplication اور □ ہو □ نے style consistent □□ ر

---

4 ڈومین: Prompt Engineering اور Structured Output (20%)

4.1 Accuracy □ لیے □ کے □ بہتر کرنے Explicit Criteria ساتھ □ کے Prompts Design کرنا

Key knowledge:

- Explicit criteria vague instructions میں □ ہوتے □ مؤثر □ زیادہ سے (مثلاً "flag comments only when they contradict code" □ بمقابلہ "check comment accuracy")
- "be more conservative" جیسی generic guidance concrete categorical criteria □□ سے کم مؤثر
- False positives developer trust کچھ: میں □ ہوتے □ پر اثر انداز categories میں high false-positive rates درست categories پر بھی trust میں □ کم کر دیتی

Key skills:

- Review criteria define کیا: کریں report □□ کرنا (bugs، security) اور کیا ignore □□ کرنا (minor style)
- High false-positive rates والی categories پر عارضی طور disable کریں
- □ ہر level □ لیے □ کے code examples ساتھ □ کے explicit severity criteria define کریں

4.2 Output Consistency □ لیے □ کے □ بہتر کرنے Few-shot Prompting استعمال کرنا

Key knowledge:

- Few-shot examples consistently formatted، actionable output مؤثر سے سب کا کرنے پیدا method میں □
- Few-shot مبہم cases (tool selection، test coverage gaps) □□ کو دکھا سکتا
- Few-shot model □ نئے کو patterns پر generalize صرف، □□ میں مدد دیتا کرنے defaults میں نہیں میں □ دراڑ
- Few-shot extraction tasks میں hallucinations □□ کم کر سکتا

Key skills:

- Ambiguous scenarios □لیے □کے rationale 4–2 ساتھ □کے targeted examples دیں
- Output format (location، issue، severity، suggested fix) □والے دکھانے examples شامل کریں
- □ایسے examples جو دیں acceptable code patterns اور real issues میں فرق دکھائیں
- مختلف structures □والے documents سے درست □extraction کے examples دیں

4.3 `tool_use` اور JSON Schemas ساتھ □کے Structured Output Enforce کرنا

Key knowledge:

- `tool_use` with JSON Schemas schema-conformant output اور □دینے کی ضمانت JSON syntax errors ختم □کرنے کا سب سے reliable □□ طریقہ
- `tool_choice: "auto"` میں model text؛ □□ واپس کر سکتا `"any"` لازماً میں اسے tool call □کرنی □ہوتی؛ □□ forced selection specific tool □□ منتخب کرتی
- Strict JSON Schemas syntax errors مگر □میں ختم کرتی semantic errors □میں روکتیں (totals؛ □میں ملتے values غلط fields میں)
- Schema design: required vs optional fields؛ enums ساتھ □کے “other” + detail string extensibility □لیے □کے

Key skills:

- JSON Schemas ساتھ □کے extraction tools define کریں اور `tool_use` results □سے data parse کریں
- متعدد schemas □ہوں تو structured output □لیے □کے یقینی بنانے `tool_choice: "any"` استعمال کریں
- Specific tool call force کریں: `tool_choice: {"type": "tool", "name": "extract_metadata"}`
- Source میں information کی صورت میں □ہونے □ نے values fabricate □لیے □کے بچانے □سے □ہونے fields optional/nullable رکھیں
- Extensible categorization □لیے □کے `"other"` اور `"unclear"` □جیسے enum values + detail fields استعمال کریں

4.4 Extraction Quality □لیے □کے Validation، Retries، اور Feedback Loops Implement کرنا

Key knowledge:

- Retry-with-error-feedback: correction guide □لیے □کے کرنے retry prompt میں concrete validation errors شامل کریں
- اگر information source تو □ہو □ نے موجود میں retries □میں □ہوتے □ فائدے بے
- Feedback loop design: finding کو trigger والا □کرنے pattern (`detected_pattern`) track کریں
- Semantic errors (totals reconcile □ہوتے □میں) بمقابلے syntax errors (جو `tool_use` □سے address □ہوتے □میں)

Key skills:

- Original document، incorrect extraction، اور specific validation errors ساتھ □کے follow-up prompts دیں
- جب retry □ہو □ فائدے بے (required info صرف external document □ہو میں) تو اس کی شناخت کریں

- False positives analyze □لیے کے □کرنے findings میں `detected_pattern` fields شامل کریں
- Discrepancies detect □لیے کے □کرنے `calculated_total` اور `stated_total` دونوں extract □کے کر self-correction design کریں

4.5 Efficient Batch Processing Strategies Design کرنا

Key knowledge:

- Message Batches API: 50% savings، 24-hour تک processing window، latency SLA guarantee □نہیں
- Batch processing non-blocking tasks (overnight reports، audits) اور □□ مناسب □لیے کے blocking tasks (pre-merge checks) □لیے نہیں کے
- Batch API ایک single request میں multi-turn tool calling support □نہیں کرتا
- `custom_id` fields batch اندر □کے request/response correlate □میں کرتی

Key skills:

- Blocking checks □لیے کے synchronous API اور overnight/weekly workloads □لیے کے Batch API استعمال کریں
- SLA needs مطابق □کے batch submission cadence plan کریں (مثلاً 30-hour guarantee 24 □لیے □کے-hour processing 4 ساتھ □کے-hour windows)
- Failed documents (identified by `custom_id`) □صرف انہی کو دوبارہ submit □کر کے failures handle کریں
- Large-scale processing □پر□ سے sample ساتھ □کے prompts iterate کریں

4.6 Multi-instance اور Multi-pass Review Architectures Design کرنا

Key knowledge:

- Self-review limitations: model اپنی reasoning context ساتھ رکھتا □□ اور اپنے فیصلوں کو challenge کرنے کا امکان کم □ہوتا □□
- Independent review instances (generation context بغیر □کے) subtle issues □میں بہتر □ہوتی میں تلاش کرنے
- Multi-pass review: per-file local analysis + cross-file integration pass attention dilution □□ بچاتا □سے

Key skills:

- Changes review □لیے کے □کرنے generation context بغیر □کے second independent Claude instance استعمال کریں
- Multi-file reviews کو per-file passes + integration passes □تاکہ میں تقسیم کریں cross-file dataflow analysis □ہو سکے
- Self-rated confidence ساتھ □کے verification passes □کر کے استعمال reviews کو calibrated □سے □طریقے route کریں

ڈومین 5: Context Management اور Reliability (15%)

5.1 Critical Information کے لیے محفوظ رکھنے کے Conversation Context Manage کرنا

Key knowledge:

- Progressive summarization خطرات کے: numeric values، percentages، اور dates vague summaries میں بدل جاتے ہیں
- Lost-in-the-middle effect: models long inputs کے شروع اور آخر کو زیادہ reliable سے طریقے process کرتے ہیں، لیکن درمیان کے findings miss کر سکتے ہیں
- Tool outputs relevance میں مقابلے کے context میں disproportionally جمع ہو سکتے ہیں (جب 5 fields 40+ درکار ہوں)
- Subsequent API requests میں full conversation history بھیجنے کی اہمیت

Key skills:

- Transactional facts کو summarized history ایک باہر سے persistent “case facts” block میں نکالیں
- Verbose tool outputs کو relevant fields تک trim کریں
- Key findings کو aggregated data شروع میں کے explicit section headings کے ساتھ رکھیں
- Subagents سے structured outputs میں metadata (dates، sources) شامل کروائیں

5.2 Effective Escalation Patterns Design کرنا اور Ambiguity Resolve کرنا

Key knowledge:

- Suitable escalation triggers: human کی explicit request، policy gaps/exceptions، پیش رفت نہ ہو پانا
- Immediate escalation (explicit request) بمقابلے attempt-to-resolve (agent scope اندر کے)
- Sentiment analysis اور model confidence self-ratings case complexity کے unreliable proxies ہیں
- Multiple customer matches کے لیے کے heuristic guessing بجائے کے additional identifiers مانگنے چاہئیں

Key skills:

- System prompt میں explicit escalation criteria few-shot examples کے ساتھ دیں
- Human کی explicit request کو بغیر اضافی investigation فوراً کے execute کریں
- جب policy کسی specific request کے لیے کے ambiguous یا silent ہو تو escalate کریں
- Tool results میں multiple matches ہوں تو additional identifiers مانگیں

5.3 Multi-agent Systems میں Error Propagation Strategies Implement کرنا

Key knowledge:

- Structured error context (failure type، query، partial results، alternatives) coordinator کو smarter recovery □□ دیتا
- Access failures (timeouts retry decision میں □تا□ چا□) اور valid empty results (no matches) میں فرق کریں
- Generic error statuses (“search unavailable”) coordinator □سے valuable context میں □ چھپا دیتے □
- Silent suppression یا ایک failure پر پورا workflow abort کرنا دونوں anti-patterns میں□

Key skills:

- Structured error context واپس کریں: failure type، کیا attempt کیا گیا، partial results، possible alternatives
- Access failures کو valid empty results سے □ الگ کریں
- Transient failures □لیے □کے subagents میں local recovery صرف کریں؛ non-recoverable errors partial results ساتھ □کے propagate کریں
- Synthesis میں coverage annotate کریں: کیا اچھی طرح supported □اور □□ gaps میں □باقی

5.4 Large Codebases Investigate وقت □کرتے Context Efficiently Manage کرنا

Key knowledge:

- Long sessions میں context degradation: model unstable answers اور مخصوص □□ لگتا □دینے classes □کے بجائے “typical patterns” □□ دیتا □حوالے کا
- Scratchpad files key findings کو context boundaries میں □ □کے پار محفوظ رکھتے □
- Subagents کو delegate □سے □کرنے verbose discovery output isolate □□ جاتی □ہو
- Structured state persistence crash recovery □□ ممکن بناتی

Key skills:

- Specific questions □لیے □کے subagents spawn □کریں جبکہ high-level coordination main agent میں رکھیں
- Key findings محفوظ رکھنے □ اور بعد میں refer □لیے □کے کرنے scratchpad files استعمال کریں
- □اگلے phase □کے subagents spawn □پلے □سے کرنے key findings summarize کریں
- Long investigations دوران □کے context usage □لیے □کے کم کرنے `/compact` استعمال کریں

5.5 Human Oversight اور Confidence Calibration □کے ساتھ Workflows Design کرنا

Key knowledge:

- Aggregate metrics (97 مثلاً% overall accuracy) specific document types یا fields □پر کمزوری چھپا سکتے □ہیں
- Stratified random sampling high-confidence extractions میں error rates measure □□ کرتا

- Field-level confidence calibration labeled validation sets کے ذریعے
- Automation پر مبنی document type اور field segment کے حساب سے accuracy validate کریں

Key skills:

- New error patterns detect کرنے کے لیے stratified random sampling implement کریں
- Stable performance validate کرنے کے لیے document type اور field کے حساب سے accuracy analyze کریں
- Field-level confidence scores output کریں اور labeled data کے ذریعے review thresholds calibrate کریں
- Low-confidence یا ambiguous-source extractions human review کی طرف route کریں

5.6 Multi-source Synthesis میں Provenance Preserve کرنا اور Uncertainty Handle کرنا

Key knowledge:

- Summarization دوران کے attribution اگر کھو جاتی ہے “claim → source” mappings محفوظ نہ رہیں
- Aggregation دوران کے structured mappings برقرار رہنی چاہئیں
- Conflicting statistics کو arbitrarily ایک value بجائے کے منتخب کرنے کے attribution ساتھ کے conflict annotate کر کے handle کریں
- Temporal differences کو contradiction کے لیے سمجھنے سے بچنے کے publication/collection dates شامل کریں

Key skills:

- Subagents سے “claim → source” mappings (URL، document name، quotes) output کروائیں
- Reports کو stable findings اور disputed ones الگ میں structure کریں
- Conflicting values کو annotations اور محفوظ رکھیں ساتھ کے reconciliation کے لیے coordinator کو دیں
- Correct temporal interpretation کے لیے publication dates شامل کریں
- Content type مطابق کے render کریں: financial data tables میں، news prose میں، technical findings structured lists میں

Claude Certified Architect — Foundations Certification

مطالعہ گائیڈ (سرکاری امتحانی گائیڈ کی بنیاد پر)

تعارف

Claude Certified Architect — Foundations سرٹیفیکیشن اس بات کی تصدیق کرتی ہے کہ ایک ماہر Claude-based حل بناتے وقت درست trade-off امتحان میں یہ فیصلے کر سکتا ہے Claude Code، Claude Agent SDK، Claude API، اور Model Context Protocol (MCP) جن بنیادی ٹیکنالوجیز — یعنی وہ بنیادی علم کو جانچتا ہے کے ساتھ پروڈکشن ایپلیکیشنز تیار کی جاتی ہیں Claude سے

امتحان کے سوالات حقیقی دنیا کے منظرناموں پر مبنی ہوتے ہیں: کسٹمر سپورٹ کے لیے agentic systems بنانا، developer productivity میں شامل کرنا، CI/CD کو Claude Code ڈیزائن کرنا، multi-agent research pipelines بنانا، اور غیر ساختہ دستاویزات سے structured data نکالنے tools

---

ہدف امیدوار

مثالی امیدوار ایک **solution architect** جو Claude کے ساتھ پروڈکشن ایپلیکیشنز ڈیزائن اور ship کرتا ہے۔ آپ کے پاس کم از کم 6 ماہ کا عملی تجربہ ہونا چاہیے:

- **Claude Agent SDK** — multi-agent orchestration، subagents کو delegate کرنا، tool integration، lifecycle hooks
- **Claude Code** — CLAUDE.md، MCP servers، Agent Skills، planning mode
- **Model Context Protocol (MCP)** — backend integration کے لیے tools اور resources
- **Prompt engineering** — JSON schemas، few-shot examples، data extraction templates
- **Context windows** — لمبی دستاویزات، multi-agent context passing
- **CI/CD pipelines** — automated code review، test generation
- **Escalation and reliability** — error handling، human-in-the-loop

---

امتحانی فارمیٹ

پیرامیٹر	قدر
سوال کی قسم	Multiple choice (4 درست میں سے 1)
اسکورنگ	100–1000 scale، passing score **720**
Guessing penalty	نہیں (ہر سوال کا جواب دیں!)
Scenarios	8 میں سے 4 (randomly selected)

امتحانی مواد: 5 ڈومینز

ڈومین	وزن
1. Agent architecture and orchestration	**27%**
2. Tool design and MCP integration	**18%**
3. Claude Code configuration and workflows	**20%**
4. Prompt engineering and structured output	**20%**
5. Context management and reliability	**15%**

امتحانی منظرنامہ

Scenario 1: Customer Support Agent

آپ ایک agent جو میں بناتے ہوئے Claude Agent SDK استعمال کرتے ہوئے returns، billing disputes، اور account issues کو سنبھالتا ہے agent MCP tools (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`) کے ساتھ 80 فیصد+ first-contact resolution مناسب escalation کے ساتھ استعمال کرتا ہے

Scenario 2: Claude Code کے ساتھ Code Generation

آپ Claude Code کو development میں استعمال کرتے ہوئے تیز کرنے کے لیے: code generation، refactoring، debugging، documentation کے ساتھ CLAUDE.md configuration اور custom slash commands کو آپ کو اس کے استعمال کو سمجھنا ہے planning mode کرنا ہے، اور integrate

Scenario 3: Multi-Agent Research System

ایک coordinator agent specialized subagents کو tasks کے سونپتا ہے: web research، document analysis، synthesis، اور report generation کے ساتھ مکمل رپورٹس تیار کرنی چاہئیں citations کے ساتھ سسٹم کو

Scenario 4: Developer Productivity Tools

یہ agent engineers کو unfamiliar codebases explore کرنے، boilerplate code اور routine tasks بنانے، automate کرنے میں مدد دیتا ہے Built-in tools (Read، Write، Bash، Grep، Glob) اور MP servers استعمال ہوتے ہیں

Scenario 5: Claude Code for Continuous Integration

Claude Code کو CI/CD pipeline میں ضم کریں تاکہ automated code reviews، test generation، اور pull request feedback حاصل ہو Prompts کے ایسے بننے چاہئیں کہ false positives کم سے کم ہوں

Scenario 6: Structured Data Extraction

سسٹم غیر ساختہ دستاویزات سے معلومات نکالتا ہے، output کو JSON schemas کے ذریعے validate کرتا ہے، اور high accuracy اسے برقرار رکھتا ہے edge cases کو درست طور پر سنبھالنے چاہئیں

Scenario 7: Conversational AI Architecture Patterns

آپ multi-turn conversational systems میں جن میں ڈیزائن کرتے ہیں context window management، متعدد turns میں instructions کا برقرار رہنا، memory strategies، محفوظ execution کے لیے tool design، اور مبہم یا متضاد user inputs کو سنبھالنا شامل ہے

Scenario 8: Agentic AI Tools *(مواد موجود نہیں — ہماری مدد کریں!)*

اس scenario والے کی اطلاع امتحان دینے candidates کو ابھی تک اس guide میں شامل نہیں دی ہے لیکن یہ اصل امتحان میں اس اگر آپ نے scenario کے سوالات دیکھے ہیں، تو انہیں GitHub Issues میں share کریں شامل کر سکیں آپ کا تعاون ہر اس شخص کی مدد کرے گا جو امتحان کی تیاری کر coverage تاکہ ہم مکمل رہنا ہے

□حصہ I: نظریاتی بنیادیں

یہ حصہ وہ بنیادی نظریہ بیان کرتا ہے جس کی آپ کو امتحان میں کامیابی کے لیے ضرورت ہے۔ مواد کو topic کے مطابق — اس سے قطع domains کے مطابق ترتیب دیا گیا ہے، نہ کہ امتحانی concepts اور technologies کی گہری سمجھ بنتی ہے۔

---

1 باب: Claude API — Model Interaction کی بنیادیں

> دستاویزات: Messages API | Prompt Engineering

1.1 API Request Structure

Claude API ایک request–response model ہے۔ Claude Messages API request میں عموماً یہ شامل ہوتا ہے:

```
{
    "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"}
}
```

اہم فیلڈز:

- `model` — model selection (`claude-opus-4-6`, `claude-sonnet-4-6`, `claude-haiku-4-5`)
- `max_tokens` — response میں زیادہ سے زیادہ tokens
- `system` — system prompt (model کی تعریف کردہ رویہ)
- `messages` — conversation history (**full history بھیجنا ضروری ہے**)
- `tools` — available tools کی definitions
- `tool_choice` — tool selection strategy

1.2 Message Roles

`messages` array میں تین roles ہیں جو ہوتے ہیں:

- `user` — user messages
- `assistant` — model responses (history میں ہوتے ہیں وقت شامل بھیجتے)

- `tool` — tool call results (عملاً `tool_result` content block)

ہر □م بات: محفوظ state کو □ درمیان Model requests بھیجیں□ conversation history میں مکمل API request آزاد □وتی □□□ call □یں رکھتا؛ □ر

1.3 `stop_reason`

Claude API response میں `stop_reason` □ کو □□ جو بتاتا □□ □وتا □□ model کیوں رُکا □:

Value	Description	Action
`"end_turn"`	Model جواب مکمل کر لیا □	user کو نتیج□ دکھائیں
`"tool_use"`	Model tool call □□ چا□تا	tool چلائیں اور result واپس دیں
`"max_tokens"`	Token limit ختم □و گئی	response truncated ؛□□ limit □یں بڑھا سکت□
`"stop_sequence"`	Stop sequence مل گئی	application logic مطابق □ کو handle کریں

Agentic systems میں `"tool_use"` اور `"end_turn"` □م سب سے □م signals □ی — □یں loop control □رت□ □یں□

1.4 System Prompt

System prompt ایک خاص instruction جو □□ context اور behavioral rules □ □□□ متعین کرتا:

- `messages` array الگ؛ □وتا □یں □ حصہ کا `system` field □□ میں دیا جاتا
- user messages □□ پر فوقیت رکھتا
- ایک بار load □□ □و کر پوری گفتگو میں لاگو □تا
- role، constraints، اور output format □□ □وتا استعمال □ لیے کی وضاحت کو

امتحان کے لیے □م: system prompt کی wording tool selection مثال □□□ کو غیر ارادی طور پر متاثر کر سکتی □میشہ □ طور پر customer کو □ ہدایت □ جیسی model کو `get_customer` □ زیادہ ضرورت سے استعمال کرنے پر لے جا سکتی □□□

1.5 Context Window

Context window کل متن □ (tokens) جسے □□ model ایک وقت میں process اس میں شامل کر سکتا □□□ □□:

- System prompt
- مکمل message history
- Tool definitions
- Tool results

□م مسائل:

1. **Lost-in-the-middle effect:** □ لمبے input کی درمیان، □یں مؤثر □وتی زیادہ معلومات اختتام اور آغاز کے □ حل: □م معلومات شروع یا آخر میں رکھیں□ details miss □و سکتی □یں□

2. **Tool results کا جمع ہونا:** ہر tool 40+ fields واپس کرتی ہے۔ اگر output میں tool call context شامل کرنا ہو تو ہم صرف 5 مگر کارآمد context کرتے ہیں۔ باقی ضائع ہوتا ہے۔
3. **Progressive summarization:** history compress کرنے پر numeric values، percentages، اور dates اکثر vague ہو جاتی ہیں۔

باب 2: Tools اور `tool_use`

> دستاویزات: Tool Use

2.1 `tool_use` کیا ہے

`tool_use` وہ mechanism ہے جس کے ذریعے Claude external functions call کر سکتا ہے۔ Model براہ راست code چلاتا نہیں — وہ structured tool call کا آپ؛ آپ code بناتا ہے، execute کرتا ہے، result واپس کرتا ہے۔

2.2 Tool Definition

ہر tool JSON schema کے ذریعے define ہوتا ہے:

```
{
    "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": []
    }
}
```

Tool description کے اہم نکات:

1. Description ہی tool selection کا بنیادی mechanism ہے۔ Minimal descriptions غلط selection کا سبب بنتی ہیں۔
2. Description میں یہ شامل کریں: tool کیا کرتا ہے، کیا واپس کرتا ہے، input format کیا ہے، edge cases کیا ہیں، اور یہ کب استعمال ہوتا ہے۔
3. Overlapping descriptions سے بچیں۔
4. Built-in tools اور MCP tools میں overlap ہو تو MCP tool کو بہتر اور واضح بنائیں۔

2.3 `tool_choice`

Value	Behavior	کب استعمال کریں
`{ "type": "auto" }`	Model □□ کرتا خود ط□	default عام
`{ "type": "any" }`	Model کو لازماً tool call کرنی □وگی	جب structured output چا□یے
`{ "type": "tool", "name": "extract_metadata" }`	خاص tool لازماً call □وگی	forced first step / order □ک□ لیے

2.4 JSON Schemas for Structured Output

JSON schemas ساتھ □ک `tool_use` استعمال کرنا Claude □سے structured output □لینے کا سب سے reliable □یقی□ □□□ طریقہ syntax errors اور □□ کم کرتا required structure، مگر □□ نافذ کرتا semantic correctness کی ضمانت نہ□یں دیتا□

Design اصول:

1. □صرف و fields required □چا□ئیں □ملنے □میشہ □جو رکھیں
2. □ممکنہ missing data □لیے □ک nullable fields □استعمال کریں
3. `"other"` اور `"unclear"` □جیسے enum values □شامل کریں تاک information □ضائع نہ □ہو

2.5 Syntax اور Semantic Errors

Error type	Example	Mitigation
Syntax	Invalid JSON، غلط field type	JSON schema ساتھ □ک `tool_use`
Semantic	Totals match □نہ□یں کرتے، value غلط field میں	Validation checks، feedback retry

---

3 باب: Claude Agent SDK — Agentic Systems بنانا

> دستاویزات: Agent SDK | Hooks | Subagents | Sessions

3.1 Agentic Loop

Agentic loop میں Claude □بلکہ، صرف جواب نہ□یں دیتا actions کی sequence □□ چلاتا:

1. Claude کو tools ساتھ □ک request بھیجیں
2. Response لیں
3. `stop_reason` چیک کریں:
   - `"tool_use"` → tool چلائیں، result history □میں ڈالیں، پھر دوبار□ request

- `"end_turn"` → task □نتیج، مکمل user کو دیں
4. مکمل □ونا□ تک د□رائیں

درست signal: completion □لیا□ کا `stop_reason == "end_turn"` □دیکھیں Text parsing یا arbitrary iteration limit □یں□ نہیں قابلِ اعتماد

3.2 `AgentDefinition`

```
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"],
)
```

م□l: `name`، `description`، `system_prompt`، اور `allowed_tools` □ Least privilege □اصول اپنائیں

3.3 Coordinator اور Subagents

Multi-agent systems اکثر hub-and-spoke topology □یں استعمال کرتی:

```
                Coordinator
               /     |      \
    Subagent1  Subagent2  Subagent3
```

Coordinator task □□ توڑتا کو، subagents □□ نتائج جوڑتا □□، کام سونپتا □□، منتخب کرتا □□، errors handle کرتا اور، □□ user تک final □□□ جواب پ□نچاتا

م□l اصول: subagents کا context □ و؛ □□ □وتا الگ parent history خود بخود inherit □□ نہیں کرتا

3.4 `Task` Tool

Subagents `Task` tool □ذریعہ □کا spawn □یں□ □وتا□ Coordinator □کا `allowed_tools` میں `"Task"` □ونا چا□ئیے□□

صحیح context passing:

- `Task: "Analyze the document"` کافی نہیں
- `Task: "Analyze this document. Full text: ... Output schema: ..."` □□ درست

Coordinator ایک response میں multiple `Task`s □□ذا□، □□ بھیج سکتا parallel subagents □یں□ ممکن

3.5 Hooks

Hooks lifecycle مخصوص □کا points پر work □یں□ کرتا□

PostToolUse tool result کو model □□□ پہنچانے سے پہلے normalize □□ کر سکتا:

```
@hook("PostToolUse")
def normalize_dates(tool_result):
        return normalized_result
```

PreToolUse policy violation block □□ کر سکتا:

```
@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)
```

یاد رکھیں: hooks deterministic ہوتے ہیں؛ prompt instructions probabilistic ہوتی ہیں۔ Critical business rules کے لیے hooks بہتر ہیں۔

---

4 باب: Model Context Protocol (MCP)

> دستاویزات: MCP | Tools | Resources | Servers

4.1 MCP □□ کیا

MCP ایک open protocol جو □□ external systems کو Claude تین بنیادی □□□ اس کے ساتھ جوڑتا ہے resource types ہیں:

1. **Tools** — actions کے لیے functions
2. **Resources** — context data (documentation، schemas، catalogs)
3. **Prompts** — predefined task templates

4.2 MCP Servers

MCP server ایک process جو □□ tools/resources expose □□□ کرتا Connect پر ہونے tools discover ہو جاتے ہیں اور descriptions ہیں ہوتے کے مطابق استعمال

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

Project config version control میں ہوتی ہے؛ secrets environment variables میں آتی ہے User config (`~/.claude.json`) personal/experimental servers کے لیے ہوتی ہے

4.4 `isError`

MCP tool errors میں `isError: true` استعمال کریں Generic errors کے بجائے structured error واپس کریں:

```
{
    "isError": true,
    "content": {
        "errorCategory": "transient",
        "isRetryable": true,
        "message": "The service is temporarily unavailable."
    }
}
```

4.5 Resources

Resources وہ data ہیں جو agent بغیر action کے پڑھ سکتا ہے: schemas، docs، content catalogs، summaries یا exploratory tool calls میں کمی کرتے ہیں

---

5 باب: Claude Code — Configuration اور Workflows

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

5.1 CLAUDE.md Hierarchy

CLAUDE.md کے تین سطحوں پر ہو سکتا ہے:

- **User-level:** `~/.claude/CLAUDE.md` — صرف اسی user پر لاگو
- **Project-level:** `.claude/CLAUDE.md` یا root `CLAUDE.md` — پوری team □لیے □کے
- **Directory-level:** subdirectories میں CLAUDE.md — مخصوص folder conventions

5.2 `@path` Syntax

CLAUDE.md میں `@path` □سے external files refer □کیے تاکہ □یں جا سکتے □کیے config modular □و:

```
Coding standards @./standards/coding-style.md میں میں
Test rules @./standards/testing-requirements.md میں میں
```

5.3 `.claude/rules/`

`.claude/rules/` rules کو topic □سے لحاظ □کے organize اور □□ کرتا `paths` □ذریعے □کے conditional loading □□ دیتا:

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

API files میں کے لیے async/await اور explicit error handling استعمال کریں
```

5.4 Commands اور Skills

`.claude/commands/` project commands □جبکہ، □□ لیے □کے `~/.claude/commands/` personal commands □□لیے □کے `.claude/skills/` میں frontmatter ساتھ □کے skills □یں□ ملتی

5.5 Skills اور `context: fork`

`context: fork` skill کو isolated context □□□ میں چلاتا `allowed-tools` □سے tool access اور، محدود کریں `argument-hint` □سے required input □مانگیں

5.6 Planning Mode

Planning mode بڑی architectural تبدیلیوں، multiple approaches، اور unfamiliar codebases □بہترین □لیے □کے □□□ Direct execution اور واضح □چھوٹے changes □□□ بہتر □لیے □کے

5.7 `/compact` اور `/memory`

`/compact` context compress اور، □□ کرتا `/memory` notes اور preferences □□□ سنبھالنے میں مدد دیتا

5.8 Claude Code CLI for CI/CD

`-p` / `--print` non-interactive mode □□ □وتا استعمال □لیے □کے:

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

Structured output □لیے □کے `--output-format json` اور `--json-schema` استعمال کریں□

---

6 باب: Prompt Engineering — Advanced Techniques

> دستاویزات: Prompt Engineering | Anthropic Cookbook

6.1 Few-shot Prompting

Few-shot prompting 4–2 میں examples کر □د expected behavior □ی □□□ دکھایا جاتا vague instructions □سے کیونکہ □□ مؤثر □زیاد example ambiguity □□□ کم کرتا

6.2 Explicit Criteria

Vague instructions □بجائ□ کے clear criteria کیا دیں: flag کیا، □□ کرنا ignore اور، □□ کرنا severity □کیسے decide کرنی □□□

6.3 Prompt Chaining

Complex task کو focused steps میں توڑیں: local analysis، پھر integration pass□ی □ attention dilution کم کرتا □□□

6.4 Interview Pattern

Claude □سے clarifying questions پوچھوانا design ambiguity جب خاص طور پر، □□ لاتا □سامنے domain unfamiliar □و□

6.5 Validation and Retry

Extraction fail تو □و original document، incorrect output، اور specific validation error ساتھ □کے retry □کریں

6.6 Self-correction

Model stated value اور calculated value دونوں نکال□؛ conflict تو □و discrepancy handle □کریں

---

7 باب: Message Batches API

> دستاویزات: Message Batches

7.1 Overview

Message Batches API asynchronous processing 50 :□□ □لیے □کے% savings، up to 24 hours window، اور multi-turn tool calling supported □نہیں `custom_id` □سے requests اور responses link □ہیں □ہوتے

7.2 کب استعمال کریں

Pre-merge checks اور interactive tasks □لیے □کے synchronous API، □جبکہ overnight reports اور bulk jobs □کے □لیے Batch API □استعمال کریں

7.3 `custom_id`

`custom_id` result کو original document □جوڑنے □سے، failures identify صرف اور، □کرنے failed items □دوبارہ submit □□□ میں مدد دیتا □کرنے

---

8 باب: Task Decomposition Strategies

8.1 Fixed Pipelines

Predictable tasks □لیے □کے fixed pipeline □جیسے، استعمال کریں Document → Extraction → Validation → Final output□

8.2 Dynamic Decomposition

Open-ended tasks میں subtasks intermediate results □ہیں □ہتے کی بنیاد پر

8.3 Multi-pass Code Review

Large PRs □لیے □کے per-file analysis بعد □کے cross-file integration pass □کریں تاکہ attention dilution □ہو □کم

باب 9: Escalation اور Human-in-the-Loop

9.1 کب Escalate کریں

Explicit user request، policy gaps، progress ہونا □ نہ، threshold اوپر □ سے financial action، یا ambiguous matching کی صورت میں escalate □کریں Sentiment analysis اور self-rated confidence unreliable □ہیں

9.2 Escalation Patterns

□ مسئلہ □ پہلے resolve کرنے کی کوشش کریں، پھر اگر policy gap یا explicit insistence □ہو تو human کو hand off □کریں

9.3 Structured Handoff

Escalation پر customer ID، issue summary، actions taken، اور recommended action □شامل کریں تاکہ human operator کو مکمل context □□ملے

9.4 Confidence Calibration

Field-level confidence scores، calibration on labeled data، اور stratified sampling □استعمال کریں

باب 10: Multi-agent Systems میں Error Handling

10.1 Error Categories

Transient errors retryable □ہیں، validation errors input fix □ہیں مانگتی، business errors policy-driven □ہوتی ہیں، اور permission errors generally escalate □ہیں کی جاتی

10.2 Anti-patterns

Generic “search unavailable” response، silent suppression، یا پورا workflow abort □□□ کرنا نقصان دہ Structured errors □واپس کریں

10.3 Structured Subagent Error

Structured error میں failure type، attempted query، partial results، alternative approaches، اور coverage impact □شامل کریں

10.4 Final Synthesis میں Coverage

Report میں واضح کریں کہ کون سی sections fully covered ہیں، کون سی partially covered ہیں، اور کہاں باقی gaps ہیں

---

11 باب: Production Systems میں Context Management

11.1 Facts Separate Block میں

Transactional facts کو persistent case facts block میں رکھیں تاکہ summarization باوجود critical data ضائع نہ ہو

11.2 Tool Results Trimming

PostToolUse hook صرف relevant fields رکھیں

11.3 Position-aware Input

Important findings اور شروع میں action items آخر میں رکھیں تاکہ lost-in-the-middle effect کم ہو

11.4 Scratchpad Files

Verbose findings scratchpad میں لکھیں تاکہ long investigations میں context محفوظ رہے

11.5 Subagents کو Delegate کرنا

Exploration subagents میں دیں اور main context میں صرف summary رکھیں

11.6 Structured State Persistence

ہر agent اپنی state manifest یا JSON file میں export کرے تاکہ crash recovery ممکن ہو

---

12 باب: Provenance کو محفوظ رکھنا

12.1 Attribution Loss

Claim کے ساتھ source URL، publication date، اور confidence رکھیں

12.2 Conflicting Data

اگر sources مختلف values دیں تو دونوں کو attribution ساتھ محفوظ رکھیں اور conflict flag کریں

12.3 Dates

Temporal differences کو contradiction پڑھ سمجھنے سے publication dates دیکھیں

12.4 Content Type مطابق کے Render

Financial data tables میں، news prose میں، technical findings structured lists اور، time series میں chronological order میں دکھائیں

---

13 باب: Claude Code کے Built-in Tools

13.1 Tool Selection Reference

Task	Tool	Example
pattern سے file تلاش کرنا	**Glob**	`**/*.test.tsx`
file contents میں search	**Grep**	function name, error message
file پڑھنا	**Read**	analysis کے لیے
نیا file بنانا	**Write**	scratch سے
precise edit	**Edit**	unique text match
shell command	**Bash**	git، npm، tests

13.2 Incremental Investigation

سب ایک ساتھ files پڑھیں؛ نہ entry points، پھر usages، پھر consumer files دیکھیں

13.3 Read + Write Fallback

اگر Edit fail ہو تو Read تبدیلی کریں، پھر programmatically Write کریں

حصہ II: امتحانی ڈومین نوٹس

ڈومین 1: Agent Architecture اور Orchestration (27%)

Agentic loops، coordinator–subagent architecture، `Task` □ذریعہ □کے spawning، hooks، escalation، اور multi-step workflows □پر فوکس کریں

ڈومین 2: Tool Design اور MCP Integration (18%)

Tool descriptions، `tool_choice`، JSON schema output، structured errors، MCP servers، resources، اور built-in tools □کے فرق کو سمجھیں

ڈومین 3: Claude Code Configuration اور Workflows (20%)

CLAUDE.md hierarchy، `.claude/rules/`، commands، skills، planning mode، `context: fork`، اور CI/CD integration □میں □م□ا

ڈومین 4: Prompt Engineering اور Structured Output (20%)

Few-shot prompting، explicit criteria، prompt chaining، validation/retry loops، JSON schema enforcement، اور batch-friendly prompting □دیں □پر توجہ

ڈومین 5: Context Management اور Reliability (15%)

Summarization risks، lost-in-the-middle، trimming tool output، confidence calibration، escalation، اور provenance preservation □یاد رکھیں

Scenario 7 مشق سوالات □کے: Conversational AI Architecture Patterns

سوال 61

صورتحال: آپ □کے `remove_team_member` tool میں execution کا اثرات □□□پ□ل□ سے preview □کرنے □کے لیے □کرنے □کے □□□ Production monitoring □کے □□ معلوم □ہوتا □□ سے agent راست □برا `dry_run: boolean` parameter □کرتا □□□ آپ کو یہ یقینی بنانا □□ □کے □ہر bypass کو preview step کر □کے call □کے ساتھ `dry_run=false` □□□پ□ل□ سے preview □جسے □ہو □ user □نے explicitly confirm □ہو □کیا ایک deletion

کونسی □□؟ approach سب سے □قابل اعتماد

- A) Server-side validation شامل کریں جو `dry_run=false` صرف اسی وقت allow 60 □کرے □جب پچھلا □ہوئی □ہو call `dry_run=true` □کے ساتھ parameters میں یکساں seconds
- B) Tool کو confirmation پر طور □کی ضرورت □کریں اور annotate کریں orchestration layer کو configure کریں □کے annotated tools کو calls forward □□□پ□ل□ سے □کرنے user کی approval □□مانگ

- C) Tool description میں تفصیلی instructions اور few-shot examples شامل کریں جن میں agent □سے مطالبہ □ہو کہ وہ ہمیشہ پہلے `dry_run=true` سے call □کرے
- D) دو tools □سے replace کریں: `preview_remove_member` impact details اور ایک single-use confirmation token واپس کرتا □□؛ `execute_remove_member` کو وہ token □□ درکار، execution کو preview □سے bind کرتا □□□ **[درست]**

D کیوں: Token-binding approach □یہ □سے architecturally □پہلے کہ پہلے □□ جاتا □ہو ناممکن preview بغیر □کے execution □ہو — execute tool کو literally □وہ token جو صرف □چاہیے preview tool □ہی generate □□□ کر سکتا □یہ واحد approach جو □□ LLM instructions کی پابندی (C)، timing heuristics (A)، یا orchestration infrastructure پر انحصار کرنے بجائے (B) code level پر constraint enforce □□□ کرتی

سوال 62

صورتحال: Production monitoring کا آپ □کے معلوم □ہوتا □□ کہ `search_catalog` tool 12% وقت fail □ہوتا □□% network timeouts جو retry 4 □میں، اور 8% query syntax errors پر کامیاب □ہوتے □میں جو کبھی □میں، جس سے بہ فائدہ return یکساں طریقے سے error types کامیاب نہیں □ہوتے□□ فی الحال دونوں retries □ہوتی □میں□

آپ tool کی error handling □کیسے modify کریں گے□؟

- A) System prompt میں few-shot examples شامل کریں جو demonstrate □کریں کہ network errors اور syntax errors □میں فرق کیسے کریں□
- B) تمام errors پر uniformly exponential backoff retry logic apply □کریں
- C) Tool اندر □کہ network timeouts □کے لیے automatic retry with backoff implement □کریں؛ syntax errors کو parameter validation details **[درست]** □کے ساتھ فوری واپس کریں□
- D) تمام errors کو `retryable` boolean flag اور error type details □کے ساتھ واپس کریں□

C کیوں: Tool level پر transient errors □کے لیے retries handle کرنا صحیح abstraction boundary □□ — tool کو agent پر کر سکتا □□ بغیر deterministic retry logic implement □کہ بار□ میں یقینی علم □□ اور وہ error type □کہ انحصار کہ □کہ وہ flag (D) interpret یا □کرے prompt instructions (A) follow □□کرے Uniform backoff (B) syntax errors □□گے □ہوں جو کبھی کامیاب نہیں □□ پر وقت ضائع کرتا

سوال 63

صورتحال: Investment strategy کی کئی □کہ گفتگو کی turns میں، user "میری risk tolerance اور □بہت کم □□" نہ کہا □کہ "کرنا چاہیے invest؟" اب وہ پوچھتا □□: "مجھے کس چیز میں returns maximize پھر "میں اپنے

□سے priority کی اصل recommendation user سب سے بہتر طریقے سے یقینی بناتا □□ کہ approach کونسا □م آہنگ □ہو؟

- A) تضاد کو سامنے لائیں اور user **[درست]** □□□ اہم □سے پوچھیں کہ کونسی چیز زیادہ
- B) دونوں scenarios الگ الگ □کے لیے recommendations □فراہم کریں
- C) سب سے حال میں بیان کی گئی preference □کے ساتھ آگے بڑھیں
- D) تضاد کو حل کیے بغیر balanced portfolio □تجویز کریں

کیوں: جب **A** user کی preferences لانا اور clarification براہ راست متضاد ہوں، تو تضاد سامنے مانگنا ہی واحد low risk کرنا اور Returns maximize کے اصل ارادے سے ہم آہنگ ہو recommendation user طریقے کے بنیادی طور پر ناقابل مطابقت اہداف ہیں جن کے لیے انسانی فیصلے درکار tolerance ہے

سوال 64

صورتحال: Users کئی conversation turns میں playlist preferences refine کرتے ہیں۔ User مجھے "jazz کے "پسند ہیں؟ genres پوچھتا ہے" "آپ کو کون سے Claude، بعد messages پسند ہے" کہنے کے دو

سب سے زیادہ ممکنہ وجہ کیا ہے؟

- A) Claude کو conversation memory کے لیے برقرار رکھنے کے لیے vector database connection درکار ہے
- B) Model کی context window exceed ہو گئی ہے
- C) Claude API کو `session_id` parameter درکار ہے
- D) آپ کی application `messages` array میں پچھلے messages **[درست]** شامل نہیں کر رہی

کیوں D: Claude پاس کے server-side memory ہے — یہ نہیں API call stateless ہے اور ہر request کے `messages` array میں مکمل conversation history بغیر کے شامل کیے، Claude کے پاس پچھلے turns کا کوئی علم نہیں ہوتا Vector databases (A) اور `session_id` (C) Claude کے architecture کا حصہ نہیں ہیں؛ دو ناممکن ہے (B) context window overflow کے لیے exchange کے messages

---

سوال 65

صورتحال: 40 منٹ کے cooking session کے بعد، conversation 78,000 tokens تک پہنچ گئی ہے History میں شامل ہیں۔ آپ کو اہم معلومات discussion اور عمومی، clarified cooking terms، recipe scaling، allergies کم کرنے ہیں۔ tokens محفوظ رکھتے ہوئے

کے درمیان بہترین توازن رکھتا ہے؟ token reduction اور approach preservation کونسا

- A) پوری conversation history کا خلاصہ کریں
- B) 20,000 صرف سب سے حالیہ tokens رکھیں
- C) اہم structured data (allergies، quantities، preferences) عمومی، نکالیں discussion اور، کا خلاصہ کریں حالیہ exchanges کو verbatim **[درست]** رکھیں
- D) مکمل conversation externally store اور کریں semantic search متعلقہ حصے کے ذریعے retrieve کریں

کیوں C: Hybrid approach کم سب سے cost زیادہ پر سب سے valuable ہے معلومات محفوظ رکھتی Allergies (summarization ہوتے ہیں extract میں compact structured block ایک critical facts جیسے recipe quantities اور exchanges ہوتی ہے، اور حالیہ discussion summarize عمومی، (سے بچاتے ہوئے precision loss کے دوران conversational coherence کے لیے verbatim رہتے ہیں Options A اور B اہم dietary information کھو سکتے ضرورت سے زیادہ پیچیدہ ہے کے لیے cooking session ایک D ہیں؛

سوال 66

صورتحال: Users طویل □یں کہ assistant کے □ہلے topics اور preferences کا conversations میں رپورٹ کرتے □یں □ رکھتی message pairs □□□ صرف آخری 25 implementation حساب کھو دیتا □□□ آپ کی موجود□

کیا □□؟ solution سب سے مؤثر

- A) Hybrid approach: پرانے messages کا خلاصہ کرتے □وئے حالیہ messages verbatim **[درست]** رکھیں□
- B) مکمل conversation history پر vector similarity search□
- C) Window 50 کو message pairs □تک بڑھائیں
- D) ہر turn میں dropped messages کریں اور running summary کا خلاصہ □آگے جوڑیں

کیوں A: Hybrid approach □□ دونوں پہلوؤں کو حل کرتی □□ مسئلہ: exact recent context کو محفوظ رکھتی □□ برقرار compressed representation کی preferences جبکہ پہلے کی (conversational coherence □م □ا □کے لیے) کو miss □□□ رکھتی Window بڑھانا (C) □□□ صرف اسی مسئلہ کو ملتوی کرتا □□ Vector search (B) □م context کو semantically مشابہ نہ □و□ current query سے کر سکتی □□ جو

سوال 67

صورتحال: Users جب □یں کہ conversations 50 turns رپورٹ کرتے □یں تو latency بڑھ جاتی زیادہ سے □وتے □یں اور □□ costs بھی□

بنیادی وجہ کیا □□؟

- A) ہر API request کے ساتھ پوری conversation history **[درست]** □□□ شامل □وتی
- B) Model gradually لمبے responses generate □□□ کرتا
- C) History بڑھنے کے ساتھ database operations □یں سست □و جاتے □یں
- D) Model زیادہ processing کی ضرورت والا internal user profile □□□ بناتا

کیوں A: Claude کی API مکمل طور پر stateless ہے — □□ request کو `messages` array میں پوری conversation history □ہر request زیادہ بڑھتی □یں، ہر conversations شامل کرنی □وتی □□□ جیسے جیسے tokens، جو □□ جاتی □ Model calls دونوں بڑھاتا □□□ cost اور directly processing latency کے درمیان کوئی internal state رکھتا □نہیں (D □□ غلط)□

سوال 68

صورتحال: تین مہینوں کے weekly sessions بعد، conversation history 85,000 tokens جب □□□ تک بڑھ گئی discussions پچھلی assistant کے بارے میں کیا نتیجہ اخذ کیا تھا؟"، تو theme کے isolation پوچھتا □□ "□م نہ user reference □□□ عام جوابات دیتا کرنے کے بجائے

کیا □□؟ approach سب سے مؤثر

- A) Rolling window truncation□
- B) Key conclusions capture □وئے progressive summarization کرتے□
- C) Relevant exchanges retrieve □لیے کے کرنے semantic embeddings **[درست]** □
- D) Discussion conclusions mark □لیے کے کرنے structured XML tags شامل کریں

C کیوں: Conversation history پر semantic search واحد approach جو تین مہینوں کی □□ discussion کو scale کر سکتی □□ جبکہ demand پر specific relevant exchanges surface □□□ کر سکتی Rolling window truncation (A) history کا بیشتر حصہ ضائع کر دے گی Progressive summarization (B) discussions کو پوچھتے ہیں users کھو دیتی □□ جن کے بارے میں specific conclusions، کرتی □□ compress میں abstractions

---

سوال 69

صورتحال: QA testing دوران کے، Claude 15–10 پہلے turns میں system prompt guidelines follow، کرتا □□ token limits کے اندر □□□ ابھی Conversation آ جاتی □□□ deviation میں responses لیکن بعد کے

کیا □□؟ solution بہترین

- A) Behavioral guidelines کو پہلے user message میں منتقل کریں
- B) 20 turns بعد نئی کے conversation شروع کریں
- C) Conversation breakpoints پر guidelines reinforce کرنے والے user-role messages insert **[درست]** کریں
- D) Non-compliant responses کو regenerate کرنے کے لیے post-response validation استعمال کریں

C کیوں: Behavioral reminders کا periodic injection instruction drift □□ کرتا راست مقابلہ کا براہ، history accumulate ہونے کے ساتھ constraints re-establish □□ کر کے Guidelines کو پہلے user message میں منتقل کرنا (A) ان کا authority □□□ کم کرتا Post-response validation (D) corrective □□ نہ کے □ اور preventive significant latency □□□ شامل کرتی

سوال 70

صورتحال: آپ کے AI tutor 2,800 token system prompt میں جو □□ teaching methodology اور adaptation rules کو نظرانداز کرنا شروع کر دیتا □□□ assistant proficiency levels، کے بعد turns کرتا □□□ define 12

کیا □□؟ fix سب سے مؤثر

- A) 5–4 ہر turns میں reminders inject کریں
- B) Verbose rules کو proficiency-level adaptation demonstrate کرنے والے few-shot examples سے replace کریں **[درست]**
- C) Critical rules کو system prompt کے آخر میں رکھیں
- D) Responses evaluate کریں اور اگر difficulty level match نہ ہو تو regenerate کریں

B کیوں: Declarative rules 2,800 والا token system prompt drift کے لیے vulnerable کیونکہ □□ abstract rules کا تقاضا کرتے ہیں reasoning سے model ہر turn میں concrete few-shot examples سے Verbose rules کو match کرنے کے لیے model کو، correct proficiency-level adaptation demonstrate کریں جو replace کرنا clear behavioral patterns کئی ہے — □□ دیتا turns میں abstract instructions زیادہ سے reliably follow □□□ ہوتا

سوال 71

صورتحال: آپ کے assistant کو enthusiastic tone اپنی، برقرار رکھنا، reasoning explain کرنا، اور clarifying questions پوچھنے ہیں یہ behavioral guidelines کہاں define؟ ہونی چاہئیں

□ونی چا□ئیں؟ define □کہاں behavioral guidelines □یں

- A) □ر user message □□□ پ□□ □سے prepend □کریں
- B) System prompt **[درست]** □میں
- C) □□□ پ assistant message □میں
- D) Environment variables □میں

B کیوں: System prompt خاص طور پر پوری conversation میں apply والی □ونے persistent behavioral constraints اور guidelines □ر □□□ ڈیزائن کیا گیا □□□ لیے □کہ user message شامل کرنا □□□ پ□□ □سے (A) redundant overhead □□□ پ assistant message (C) unreliable □کیونکہ □□ model □□□ اپنے statements □سے deviate □□□ کر سکتا Environment variables (D) model □کہ behavior □□□ ڈالتے □□ پر اثر نہیں

سوال 72

صورتحال: Users repetitive response openings □جیسے "یقیناً!" اور "میں مدد کرنے میں خوش □وں!" رپورٹ کرتے □یں□

مؤثر □سب سے approach □□؟ کیا

- A) Direct response opening □کے ساتھ partial assistant message append **[درست]** □کریں
- B) Temperature setting □کم کریں
- C) Greetings □لیے □کہ □ثانے responses کو post-process □کریں
- D) System prompt میں ان phrases کی بچنے □سے instructions □شامل کریں

A کیوں: Assistant □کہ response کو direct □جواب کی شروعات سے prefill کرنا generation level پر greeting patterns □□ روکتا — model prefill □سے continue □نئے □بجائے □□ کرتا opening phrases generate □□کرنے System prompt instructions (D) مدد کر سکتی □یں لیکن کم reliable □یں□ Post-processing (C) ایک fragile workaround □□□ Temperature (B) randomness control □□ کرتا، specific phrase patterns □نہیں

سوال 73

صورتحال: ایک webhook □کہ آپ system کو notify □کہ □□ کرتا user کا package ship □جبکہ □□ گیا □و user actively chat □کہ □یں □□ آپ چا□تے □□ کر رہا □□ assistant □اگلا اسے response میں naturally □شامل کر□

□ترین □□؟ approach کیا

- A) Shipping status کو system prompt □میں شامل کریں
- B) فوری synthetic user message □بھیجیں
- C) □ر turn میں assistant کو status tool call □کرنے پر مجبور کریں
- D) Status update □اگلا کو user message □کہ prefix □کہ طور پر append **[درست]** □کریں

D کیوں: Status update □اگلا کو user message □کہ prefix کرنا □کہ طور پر شامل کرنا flow کو interrupt بغیر □کیے natural conversation boundary پر real-time context inject □□□ کرتا System prompt modify کرنا (A) session rebuild □□□ کی ضرورت Synthetic user message (B) natural dialogue flow کو disrupt □ر □□□ کر سکتا turn میں tool call force کرنا (C) اس وقت wasteful جب □□ events rare □وں□

سوال 74

صورتحال: Users □ لیے کے "پارٹی اکثر venue book □ جیسے" کریں requests □ میں □ بھیجتے Assistant 4+ clarifying questions 35 □ جس سے، □□ پوچھتا % abandonment □□□ ہوتا □

کونسا approach trade-off □ سے □ بہترین طریقہ کو improve □□؟ کرتا

- A) Hidden defaults □ بڑھیں آگے ساتھ کے
- B) تمام clarifying questions ایک compound message □ میں پوچھیں
- C) Assumptions explicitly بیان کریں اور corrections **[درست]** □ بڑھیں آگے □ ہوئے □ مدعو کرتے
- D) Structured intake form □ استعمال کریں

C کیوں: Assumptions explicitly بیان کرنا اور آگے بڑھنا user مفید، فوری کو response غلط □ جبکہ □□ دیتا کو نہیں بتاتے کہ کیا user (A) Hidden defaults کرنے کی صلاحیت برقرار رکھتا □□□ correct کو assumptions □ کیا گیا assume Compound question list (B) پھر بھی user سے upfront effort □□□ کا تقاضا کرتی Structured form (D) کم friction □□□ شامل کرتا زیادہ □ کی بجائے

سوال 75

صورتحال: آپ کا assistant contractor-persona system prompt ابتدائی □□□ استعمال کرتا turns rules follow □ کرتے □ میں، لیکن turn 7 تک assistant generic advice □□□ دیتا Conversation length 2,500 صرف tokens □□□

سب سے زیادہ ممکنہ وجہ کیا □□؟

- A) System prompts صرف initial behavior establish □ میں □ کرتے
- B) Turns accumulate ساتھ کے □ ہونے model کی attention □□□ کمزور □ ہوتی
- C) Accumulated assistant responses system prompt کو اثر کے dilute **[درست]** □ میں □ کرتے
- D) System prompt □□□ صرف ایک بار بھیجا جاتا

C کیوں: جیسے □ جیسے system prompt □ کے، جمع □ ہوتے □ میں assistant responses میں conversation history assistant-generated content □ کے بڑھتے □ ہوئے proportion کا text کرنے والے reflect کو behavioral constraints □ کے بجائے اپنے پہلے system prompt instructions پر بھی token lengths چھوٹے Model کم □ ہوتا جاتا □□□ relative کرتا □□□ compound کو drift، کرتا □□ follow کو زیادہ patterns □ کے outputs

---

سوال 76

صورتحال: Users مبہم requests "کیا آپ کرتے □ میں جیسے □ report؟ میں مدد کر سکتے □ میں" Assistant کئی سوالات پوچھ کر respond □□ کرتا (کون سی report؟ کیا مدد ؟ deadline ؟□□ کیا)، 40 جس سے % abandonment □□□ ہوتا □

بہترین solution □□؟ کیا

- A) Reasonable assumptions کریں، انہیں explicitly کریں، اور بیان کریں adjustments **[درست]** □ کی پیشکش کریں
- B) Respond □ پہلے □ سے کرنے smaller model ساتھ کے ambiguity classify □ کریں
- C) Assumptions بغیر بیان کیے predefined interpretations □ استعمال کریں
- D) Assistant ہر □ کو turn ایک میں clarifying question □ تک محدود کریں

A کیوں: Reasonable stated assumptions بڑھنا آگ□ ساتھ □کے user کو informed اور in control □ہوئ□ □رکھت□ back-and-forth □□□ کو مکمل طور پر ختم کرتا Silent predefined interpretations (C) users کو confuse کرتی □یں جب response □سے □ارادے □کے ان match □ہو□ □نہ One-question limit (D) پھر بھی back-and-forth turns کی □□□ ضرورت Smaller classification model (B) core UX بغیر □حل کیے □مسائل latency اور infrastructure complexity □□□ شامل کرتا

عملی مشقیں

Exercise 1: Multi-tool Agent with Escalation Logic

Goal: tool integration، structured errors، اور escalation ساتھ □کے agent loop □بنائیں

Exercise 2: Configuring Claude Code for Team Development

Goal: CLAUDE.md، custom commands، path-specific rules، اور MCP servers configure □کریں

Exercise 3: Structured Data Extraction Pipeline

Goal: JSON schema، validation/retry loops، اور Message Batches API ساتھ □کے extraction pipeline □بنائیں

Exercise 4: Designing and Debugging a Multi-agent Research Pipeline

Goal: subagent orchestration، explicit context passing، error propagation، اور source tracking implement □کریں

Appendix: Technologies and Concepts

Technology	Key aspects
Claude Agent SDK	agent loops, `stop_reason`, hooks, `Task`, `allowedTools`
MCP	servers, tools, resources, `isError`, `.mcp.json`
Claude Code	CLAUDE.md, `.claude/rules/`, `.claude/commands/`, `.claude/skills/`, planning mode
Claude Code CLI	`-p`, `--output-format json`, `--json-schema`
Claude API	`tool_use`, `tool_choice`, `stop_reason`, `max_tokens`

Message Batches API	50% savings, 24-hour window, `custom_id`
JSON Schema	required/optional, nullable, enum values
Few-shot prompting	ambiguous scenarios □لیے □کے examples
Prompt chaining	sequential decomposition
Context window	token budget, lost-in-the-middle
Confidence calibration	field-level scoring, stratified sampling

---

Out-of-Scope Topics

□یں موضوعات امتحان میں شامل نہ□یں:

- Claude models کی fine-tuning
- Authentication، billing، یا account management
- مخصوص programming languages کی تفصیلی implementation
- MCP servers کی hosting/infrastructure
- Claude کی internal architecture یا model weights
- Constitutional AI، RLHF، یا safety training
- Embeddings یا vector databases کی implementation details
- Computer use / browser automation
- Vision / image analysis
- Streaming API یا SSE
- Rate limiting اور detailed API cost calculations
- OAuth اور API key rotation details
- Cloud-provider-specific configs
- Performance benchmarks
- Prompt caching implementation details
- Token counting algorithms

Preparation Recommendations

1. Claude Agent SDK ایک ساتھ □کے agent بنائیں
2. Claude Code کو real project پر configure کریں
3. MCP tools design اور test کریں
4. Data extraction pipeline بنائیں
5. Prompt engineering practice کریں
6. Context management patterns سیکھیں

7. Escalation اور human-in-the-loop workflows سمجھیں
8. Practice exam حل کریں