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 — بنیادی علم کا جائزہ لیتا ہے کہ ساتھ پروڈکشن ایپلیکیشنز بنائی جاتی ہیں Claude وہ بنیادی ٹیکنالوجیز جن سے

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

---

ہدف امیدوار

مثالی امیدوار ایک **solution architect** جو ہے Claude اور پروڈکشن ایپلیکیشنز ڈیزائن ساتھ کہ ship آپ کرتا ہے 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 لیے کے tools اور resources
- **Prompt engineering** — JSON schemas، few-shot examples، data extraction templates
- **Context windows** — لمبی دستاویزات کے 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 1 میں سے)
اسکورنگ	100–1000 scale، passing score **720**
Guessing penalty	نہیں (ہر سوال کا جواب دیں!)
Scenarios	8 4 میں سے (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 کب استعمال کرنا ہے planning mode 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 مکمل کے ساتھ reports تیار کرنے چاہئیں

Scenario 4: Developer Productivity Tools

ایک 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 کو حاصل Prompts کے ڈیزائن ہونے چاہئیں 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□ کو درست طور پر سنبھالنا

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□ کا برقرار رکھنا، memory strategies، محفوظ execution □ لیے □ کے tool design، اور مبہم یا متضاد
user inputs □□□ کو سنبھالنا شامل

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

اس scenario والے □ کی اطلاع امتحان دینے □ ابھی تک اس candidates لیکن یہ □ دی □□ guide نہ میں شامل نہیں
کریں share میں GitHub Issues کے □ سوالات دیکھ□ □یں، تو انہیں □□□ scenario اگر آپ نہ 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 Claude Messages API کی پیروی کرتا ہے 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 کے رویے کی تعریف کرتا ہے)
- `messages` — conversation history (**بھیجنی ہوتی history تسلسل برقرار رکھنے کے لیے آپ کو مکمل** ہے)
- `tools` — دستیاب tools کی definitions
- `tool_choice` — tool selection strategy

1.2 Message Roles

`messages` array دو conversational roles اور ایک instructional role استعمال کرتا ہے:

- `user` — user messages، بشمول tool results (ایک یا `user`-role message اندر کے `tool_result` content block کی صورت میں بھیجے جاتے ہیں، ایک الگ `tool` role طور پر نہیں کے)
- `assistant` — model responses (history میں بھیجتے وقت شامل کیے جاتے ہیں)، بشمول tool use requests (`tool_use` content blocks)
- `system` — top-level `system` field ہے کے ذریعے سیٹ کیا جا سکتا ہے (پہلے turn سے لاگو ہوتا ہے) یا اس نقطے سے آگے شامل کیا جا سکتا ہے inline کے طور پر `{"role": "system", ...}` `messages` میں (کے تابع — نیچے دیکھیں placement rules لاگو ہوتا ہے، مخصوص

Tool results کو `"tool"` role والے message ایک یہ ایک کے طور پر نہیں بھیجا جاتا `user`-role message کے طور پر بھیجے جاتے ہیں جس کے content میں ایک `tool_result` content block ہے شامل ہوتا:

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

`system` براہ راست `messages` array میں بھی ایک role صرف نہ، ہو سکتا ہے، top-level کے طور پر ظاہر ہو کیونکہ cached prefix کو invalidate بغیر کیے top-level `system` field کے ذریعے اس کا مقصد parameter `system` conversation درمیان کے instructions مخصوص اس کے شامل کرنا ہے۔ placement rules یہ ہیں:

- ایک یہ `user` turn (جس میں بشمول وہ `tool_result` blocks ہوں) یا server tool use والے ہونے پر ختم ہونے کے فوراً بعد آنا چاہیے `assistant` turn کے
- کسی یہ `assistant` turn یا پہلے آنا چاہیے یا array سے آخری element کا ہونا چاہیے
- کسی یہ `tool_use` block اور اس کے `tool_result` پر 400 ایسا کرنے — آ سکتا نہیں درمیان کے error ملتا ہے
- والے بعد میں آنے والوں پر اور بعد (کے درمیان شامل کیے گئے conversation بشمول) messages `system` پہلے turns لیے کے top-level `system` field میں پر فوقیت رکھتے ہیں

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

1.3 Response میں `stop_reason` Field

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

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 مطابق کے handle کریں

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

1.4 System Prompt

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

- `messages` array الگ یہ ہوتا؛ نہیں کا حصہ `system` field میں دیا جاتا ہے

- user messages □□ پر فوقیت رکھتا
- ایک بار load اور پوری conversation □□ میں لاگو ر□تا □وتا □□
- role، constraints، اور output format □□ کی تعریف ک□ لی□ استعمال □وتا □□

امتحان ک□ لی□ ا□م: system prompt کی wording غیر ارادی tool associations طور ک□ مثال □□□ پیدا کر سکتی □میشہ "پر customer ہدایت □جیسی کی تصدیق کریں" کو model `get_customer` کو ضرورت س□ زیاد□ استعمال کرن□ پر مجبور کر سکتی □□، حتیٰ ک□ جب یہ ضروری نہ □و□

1.5 Context Window

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

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

ا□م context-window مسائل:

1. **Lost-in-the-middle effect:** models □لمبی input □ اعتماد س□ زیاد□ معلومات کو آغاز اور اختتام پر موجود ک□ کر سکت□ □یں□ حل: ا□م معلومات شروع یا آخر ک□ قریب کرت□ □یں، مگر درمیان کی تفصیلات process miss رکھیں□
2. **Tool results ہ□ر □ونا: کا جمع** tool call context میں output اگر □□□ شامل کرتا tool 40+ fields واپس کر□ کا بڑا حصہ ضائع □و جاتا □□□ context مگر صرف 5 ا□م □وں، تو
3. **Progressive summarization:** history compress وقت کرت□ numeric values، percentages، اور dates اکثر گم □و کر غیر واضح □و جات□ □یں ("تقریباً"، "کچھ"، "چند")□

---

2 باب: 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 □□ کرن□ Model □□برا result کرتا □□ اور execute اس□ code بناتا □□؛ آپ کا structured tool call request □یں چلاتا — و□ راست code واپس کرتا □□□

2.2 Tool Definition

ہ□ر tool ایک JSON schema □ذریعہ ک□ 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 لکھنے میں انتہائی اہم پہلو:

1. **Description tool selection کا بنیادی mechanism ہے۔** LLM tools کو ان کی descriptions کی بنیاد پر منتخب کرتا ہے۔ Minimal descriptions ("Customer information retrieve کرتا ہے") اُن مواقع پر غلطیوں کا باعث بنتی ہیں جب tools سے دوسرے ایک overlap کرتے ہوں۔
2. **Description میں شامل کریں:**
   - Tool کیا کرتا ہے اور کیا واپس کرتا ہے
   - Input formats اور example values
   - Edge cases اور constraints
   - کب مقابلے میں کب استعمال کرنا ہے tool similar alternatives کے
3. **ایک جیسے یا overlapping descriptions سے بچیں۔** اگر `analyze_document` اور `analyze_content` کی descriptions تو، ہوں تقریباً ایک جیسی model انہیں خلط ملط کر دے گا۔
4. **Built-in tools بمقابلے MCP tools:** agents similar functionality والے built-in tools (Read، Grep) کو MCP tools پر ترجیح دے سکتے ہیں۔ اس سے بچنے کے لیے MCP tool descriptions مضبوط بنائیں — concrete advantages، unique data، یا وہ context جو built-in tools نمایاں کریں فراہم نہیں کر سکتے۔

2.3 `tool_choice` Parameter

`tool_choice` کے ذریعے کنٹرول کرتا ہے model tools کیسے منتخب کرتا ہے:

Value	Behavior	کب استعمال کریں
`{ "type": "auto" }`	Model کے خود طے کرتا ہے tool call کرنا ہے یا text میں جواب دینا ہے	default زیادہ تر حالات میں
`{ "type": "any" }`	Model کو لازماً کوئی tool call کرنا ہوگی	جب آپ کو guaranteed structured output چاہیے
`{ "type": "tool", "name": "extract_metadata" }`	Model کو لازماً ایک مخصوص tool call کرنی ہوگی	جب آپ کو forced first step / execution order چاہیے

□م منظرنامہ:

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

2.4 Structured Output □لیے □کے JSON Schemas

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

- Syntax درست □اعتبار سے □کے JSON □□ کی ضمانت دیتا (braces □وتے □یں غائب نہ□یں، trailing commas نہ□یں □وتے)
- □مطلوبہ structure □□ نافذ کرتا (required fields □یں □وتے □وجود)
- Semantic correctness کی ضمانت نہ□یں دیتا (values □یں □و سکتی □یں غلط بھی پھر)

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 □بمقابلہ optional:** بنائیں جن کی معلومات □میشہ□ دستیاب □و□ fields کو required □و□ صرف اُن گھڑنے □پر مجبور کر سکتی □یں values غائب □ونے □پر data کو Required fields model

2. **Nullable fields:** ایسی معلومات کے لیے `"type": ["string", "null"]` استعمال کریں جو ممکنہ طور پر موجود نہ ہوں Model `null` بجائے اس کے کہ وہ خود ساختہ value واپس کر سکتا ہے بنائے
3. **Enums with `"other"`:** اپنی predefined categories کے لیے سے باہر کی معلومات ضائع ہونے سے بچانے کے لیے `"other"` + ایک detail string شامل کریں
4. **Enum `"unclear"`:** جب اُن صورتوں کے لیے model category ایماندار — ہو نہ ہو کے بارے میں پُراعتماد نہ ہونے سے بہتر ہے غلط category `"unclear"`

2.5 Syntax بمقابلہ Semantic Errors

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

---

3 باب: Claude Agent SDK — Agentic Systems بنانا

> دستاویزات: Agent SDK | Hooks | Subagents | Sessions

3.1 Agentic Loop کیا ہے

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

```
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. مکمل ہونے تک repeat کریں
```

یہ model-driven approach ہے: Claude اپنے context اور پچھلے tool results کی بنیاد پر اگلا tool خود منتخب کرتا ہے۔ یہ hard-coded decision trees جیسا نہیں ہے جو مختلف action sequence سے پہلے fixed ہوتی ہے۔

Anti-patterns (ان سے بچیں):

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

درست طریقہ: completion کا واحد قابلِ اعتماد signal `stop_reason == "end_turn"` ہے۔

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

م□ل 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 سے subagents □ درکار □ (dynamic selection)
- کام subagents کو سونپنا
- نتائج جمع اور validate کرنا
- Errors اور retries handle کرنا
- نتائج user تک پہنچانا

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

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

3.4 Subagents □لیے □کے بنانے □ `Task` Tool

Subagents `Task` tool □ذریعے □کے spawn □یں □جاتے □کیے:

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

Explicit context passing □□ لازمی:

```
# Bad: subagent پاس کے context نہیں ہے
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 □یں □چلتے:

```
# Coordinator ایک response میں شامل یہ ہے:
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 □یں □کی اجازت دیتے □یں

PostToolUse tool result کو model □لے□ سے □پہنچانے □تک intercept □□ کرتا:

```
# مثال: MCP tools کے 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 actions کو block □□ کرتا:

```
# مثال: $500 سے اوپر 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 ہوں — prompts نہیں، hooks استعمال کریں

باب 4: Model Context Protocol (MCP)

> دستاویزات: MCP | Tools | Resources | Servers

4.1 MCP کیا ہے

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

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

4.2 MCP Servers

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

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

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 میں □ہوتے □ استعمال □لیے □کے — tokens خود commit □نہیں کیے جاتے
- □یہ project تمام □کے contributors □□ ہوتا □ دستیاب □لیے □کے

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

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

Servers کا انتخاب:

- Standard integrations (Jira، GitHub، Slack) موجود □سے □لا□ 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 □□□ ہوتا □ استعمال agent □کو بتاتا □□ کہ call fail □□□ ہوئی□

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 کہ فیصلہ سازی کو retry کریں، query یا بدلیں، escalate کریں؟

4.5 MCP Resources

Resources ایسا data جو ہیں agent context لیے کہ حاصل کرنے actions ہیں بغیر طلب کر سکتا کیے:

- Content catalogs (مثلاً تمام project tasks فہرست کی، hierarchical navigation)
- Database schemas (data structure لیے کہ سمجھنے)
- Documentation (API references، internal guides)
- Issue/task summaries

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

---

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

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

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

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

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

5.2 `@path` Syntax (File Imports)

CLAUDE.md بیرونی files کو `@path` □کے ذریعے refer □کے جس سے، □□ configuration modular کر سکتا □و 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 dependencies @package.json میں ہیں
```

`@path` قواعد □کے:

- `@` فوراً بعد □کے file path □آئ (کوئی space □نہیں)
- Relative اور absolute دونوں paths supported □یں
- Relative paths اُس file مطابق □کے resolve □یں جس میں □وتے □وئے import □□
- Maximum import nesting depth 5 □□

□اس سے duplication □ر اور □□ کم □وتی package □متعلقہ صرف standards □میں □وتے □وئے شامل

5.3 `.claude/rules/` Directory

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

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

YAML frontmatter ساتھ □ کا `paths` conditional loading:

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

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

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

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

□□ کام کرتا □یسے □ کیسے □یں:

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

`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 □یں اور □وں □ وابستہ □ سے ضروری نہ □وں

5.4 Custom Slash Commands اور Skills

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

Slash commands reusable prompt templates جو □یں `/name` □ ذریعے □ کے invoke □یں □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/`):

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

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

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

5.5 Skills — `.claude/skills/`

Skills advanced commands ہیں جو SKILL.md frontmatter کے ذریعے configure ہوتے ہیں:

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

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

Frontmatter parameters:

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

Skill بمقابلہ CLAUDE.md کب استعمال کریں:

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

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

- اپنی personal variants □کو بنائیں □سے مختلف ناموں teammates متاثر نہ □ہوں

5.6 Planning Mode □بمقابلہ Direct Execution

Planning mode:

- Model صرف investigate اور plan □□؛ کرتا changes نہیں کرتا
- Codebase explore □کرنے کے لیے 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؟ structure کیا؟)
- Unfamiliar codebase (change □□ سمجھنا ضروری □ہے پہلے □سے کرنے)
- Library migrations 45 جو+ files کو متاثر کریں

Direct execution کب استعمال کریں:

- Single-file fixes واضح stack trace ساتھ □کے
- ایک validation check شامل کرنا
- اچھی طرح سمجھے □گئے، غیر مبہم changes

Combined approach:

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

Explore subagent — codebase explore □کرنے کے لیے 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 □کرنے کے لیے built-in command □□:

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

5.8 `/memory` Command

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

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

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 □ کے 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 □ کے validate □□ کرتا
- Result کو automatically inline PR comments □لیے □ کے پوسٹ کرنے parse □□ کیا جا سکتا

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

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

5.10 `fork_session` اور Session Management

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

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

- saved context ساتھ پچھلی □ کے conversation □□ جاری رکھتا

- Long investigations جو متعدد sessions مفید □ل□ ان ک□ پر پھیلی □وں
- Risk: اگر پچھلی session بعد □ک files تو □وں بدل چکی tool results stale □یں □و سکت□

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

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

- دونوں forks branch point تک context inherit □یں □کرت□
- آزادان□ طور پر □و بعد اس ک□ diverge □یں □کرت□
- Approaches compare □کرن□ یا strategies test مفید □ل□ ک□ کرن□

Resume نئی □بجائ□ ک□ session کب شروع کریں:

- Tool results stale □وں (files □وں بدل چکی)
- کافی وقت گزر چکا □و اور context degrade □و گیا □و
- □پران tool data ساتھ □ک resume: □□ □ک "□م ن□ جو پایا اس کا مختصر خلاص□ □ی □□ بہتر □□ ک□ بجائ□ □ی □کرن□ ک□ کیا جائ□ restart □س "...

---

6 باب: Prompt Engineering — Advanced Techniques

> دستاویزات: Prompt Engineering | Anthropic Cookbook

6.1 Few-shot Prompting

Few-shot prompting جس میں □□ □طریق□ □و expected behavior □ل□ ک□ دکھان□ prompt 4–2 میں input/output examples □یں□ □جات□ شامل کی□

Few-shot text descriptions □□ مؤثر کیوں □زیاد□ □س:

- □م□م instruction □زیاد□ "جیس□ precise □□ سمجھا جا سکتا □س طریقوں کو کئی □وں"
- Example غیر مب□م طور پر expected format اور decision logic □□ دکھاتا
- Model pattern □ئ□ کو cases پر generalize □□ کرتا (صرف examples □یں □راتا □د)

Few-shot examples کی اقسام اور استعمال:

1. **Ambiguous scenarios □ل□ ک□ 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 درکار میں□

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

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 □نکالنے purely rule-based instructions □میں□ ہوتی □متنوع □بات □لیے □کے

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 جنہیں □وتے □ کم JSON syntactically مگر □و درست values inconsistent □ہوں

6.2 Explicit Criteria □بمقابلہ Vague Instructions

Bad (vague):

```
Code comments کی accuracy چیک کریں۔
Conservative صرف — رپورٹ high-confidence findings کریں۔
```

Good (explicit criteria):

```
Comment کو problematic صرف اُس صورت میں flag کریں جب:
1. Comment کا بیان 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 ہی fix ہو چکا ہو

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

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 □□ میں توڑتا:

```
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 سی بہت ایک ساتھ files تو کچھ □□ دیکھتا files □کے bugs miss □د 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 □کرنے □سے پہلے 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

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 □ہوگا کب مؤثر:

- Format errors (date غلط format میں)
- Structural errors (field □غلط جگ□)
- Arithmetic inconsistencies (model □دوبار check □□ کر سکتا)

Retry گا □کر مدد نہ□یں کب:

- Information source document میں موجود □ی نہ□یں
- Required context external □□ (data کسی اور document کیا گیا فرا□م نہ□یں جو □□ میں)

Pydantic بطور validation tool: Pydantic Python library جو □□ schema-based data validation □لیے □کے استعمال □وتی □□□ امتحان کے □لیے ا□م نکات یہ □یں:

- **Structural validation:** types، requiredness، enum constraints JSON بعد □کے code □یں □جاتے □یں چیک کیے میں
- **Semantic validation:** custom validators business logic enforce □یں □کرتے (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 prompt کریں دوبار□
- **JSON Schema generation:** Pydantic models `tool_use` □لیے □کے JSON Schema generate □کر سکتے □یں، جو ایک single source of truth □□ فرا□م کرتا

6.6 Self-correction

Internal contradictions detect کا □کرنے 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 وہ □ مختلف □وں تو Model stated value اور computed value کو discrepancy handle □□□ دیتا □کرنے

باب 7: Message Batches API

> دستاویزات: Message Batches

7.1 Overview

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

Attribute	Value
Savings	synchronous calls **50** میں مقابلہ کی%
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 کو جوڑنے کے لیے `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 ہے؛ 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**	درکار فوری response ہے
10,000 documents process کرنا	**Batch**	Bulk processing؛ savings میں اہم

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` آپ کو یہ کرنے دیتا ہے:

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

7.4 Batch میں Failures کو Handle کرنا

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

5. 5 صرف اُن failed documents □کو دوبار□ submit کریں

7.5 SLA Planning

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

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

---

8 باب: Task Decomposition Strategies

8.1 Fixed Pipelines (Prompt Chaining)

ہر□ step □س □ل□ پ defined □□ ہوتا□:

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

کب استعمال کریں:

- Task structure predictable ہو□ (reviews کی □ ایک □میشہ template follow کریں)
- تمام steps ہوں □ معلوم □س □ل□ پ
- آپ کو stability اور reproducibility □چا□ی□

8.2 Dynamic Adaptive Decomposition

Subtasks intermediate results پر بنیاد کی generate □یں □ ہوت□:

```
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 لکھنے سے external API کے لیے mock add کریں
```

کب استعمال کریں:

- Open-ended investigative tasks
- جب مکمل scope ہو □ □ شروع میں معلوم نہ
- جب ہر□ step □چھل□ step □ کی results ہو□ پر منحصر

8.3 Multi-pass Code Review

10+ files والی pull requests □لیے □کے:

```
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 □ی پر ایک pass □□ کیوں خراب:

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

---

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

9.1 کب Human کی طرف Escalate کریں

Escalation triggers (واضح rules):

Situation	Action
Customer □□کے واضح طور پر “get me a manager”	فوراً escalate کریں؛ حل کرنے کی کوشش نہ کریں
Policy request کو cover □و نہ کرتی	Escalate کریں (مثلاً competitor price matching جب policy خاموش □ہو)
Agent □کر سکے پیش رفت نہ کر	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؛ 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]
NOT: "I can help with your issue, let me..."
```

Resolve بعد □کی کوشش کی □کرنی escalation:

```
Customer: "My refrigerator broke two days after purchase"
Agent: [order □□ چیک کرتا، warranty replacement offer □□ کرتا]
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 replacement یا refund □پیش کر سکتا □ہوں."
Customer: "No, I want to talk to someone!"
Agent: [customer □دوبارہ insist □کرے -> immediate escalation]
```

□□بنیادی اصول: پ□□ emotion acknowledge پھر کریں، concrete solution دیں، اور صرف اسی وقت escalate کریں جب customer human □□کی خواہش د□رائ□□ dissatisfaction پر اظ□ار □□پ□□ کی escalate کریں □نہ (□یا manager □□ برابر نہ□یں □□مانگنی □□)

Policy gap □□لیے □□کی 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 □□کو دینی چاہیے:

```
{
  "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 □ یہ صرف □ وہ — □وتی □یں نہ□یں تک رسائی summary □□□ دیکھتا اور مکمل □ یہ □ لیے اس self-contained □□یہ□ چاہ□ونی

9.4 Confidence Calibration اور Human Oversight

Data extraction systems □ لیے □کہ:

1. **Field-level confidence scores:** model □ر extracted field □ لیے □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 □ لیے □کہ sample audit کریں
- Aggregate 97% accuracy کسی مخصوص document type 40 □ لیے □کہ% errors □□ چھپا سکتی
- Accuracy کو صرف overall □یں نہ، document type اور field □سے حساب □کہ analyze کریں

---

10 باب: Multi-agent Systems میں Error Handling

10.1 Error Categories

Category	Examples	Retryable	Agent action
Transient	Timeout، 503، network failure	Yes	Exponential backoff ساتھ □کہ 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 □□یں کر سکتا □□ recover □□ کرنا □□کیسے	Error type، query، partial results، alternatives واپس کریں
Silent suppression (empty result = success)	Coordinator □□ سمجھتا match □□یں □□ملا، حالانکہ failure تھا	“no results” اور “search failure” میں فرق کریں
ایک failure پر پورا workflow abort کرنا	تمام partial results □□یں □□ضائع □□جاتے	Partial results □□ 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 □□ retry کریں؟
- Partial results استعمال کریں؟
- کسی اور subagent کو delegate کریں؟
- اس section اور بغیر جاری رکھیں □□ 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□ کہ □ پر انحصار کرنا (جو summarization دوران □کہ degrade □□ جاتی □ہو)، key facts کو structured block میں extract کریں:

```
=== CASE FACTS (جب بھی نیا fact آئے update کیا جائے) ===
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
===
```

کیوں نہ □ہو□ summarized کتنی □ہی history میں شامل کریں، چا□□ prompt □ہر block □یا

11.2 Tool Results کو Trimming کرنا

اگر `lookup_order` 40+ fields مگر □□ واپس کرتا current task □میں صرف 5 درکار □لیے □کہ:

```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□ □تھ□ میں رکھ□ن ذ□ کو 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 □ پ□ل□ □س auth.ts vulnerabilities fix کریں.
```

11.4 Scratchpad Files

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

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

جب context degrade □جائ□ (نئی یا session میں)، agent discovery □بجائ□ □ک□ چلان□ دوبار□ scratchpad consult □□□ کر سکتا

11.5 Context □ لیے □ کے بچانے Subagents کو Delegate کرنا

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

Main agent: context 15 میں files ایک بجائے کہ line □□ رکھتا
```

Separate context layer: Multi-agent systems □ر میں subagent محدود context budget — □□ میں کام کرتا □ کہ separate context layer پر طور ایک Coordinator ملتی □□□ information کے لیے ضروری task اسے صرف اپنے □ کرتا □□ کام subagent outputs aggregate □□ کرتا، global state store اور □□ کرتا context allocate کرتا: □ اس سے □□□ "context leakage" ایک جے□اں □وتا، □یں نہ agent window کو ایسی information جو □□ بھر دیتا سے □ دوسروں کے لیے غیر متعلق □و

Subagents □ لیے □ کے constrained context budgets:

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

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

□ر agent اپنی state ایک معلوم 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 □کیے جاتے □ہیں، تو “claim → source” link □□ گم □ہو سکتا:

```
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 کو دونوں کو ساتھ محفوظ رکھیں اور attribution کو اندھا دھند منتخب نہ کریں□ کو coordinator کو فیصلہ کرنے دیں□

12.3 Correct Interpretation کے لیے 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 کو Render کریں

ہر چیز کو ایک ہی format میں نہ ٹھونسیں:

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

باب 13: 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 کے ذریعے specific snippet replace کرنا
Shell command چلانا	**Bash**	git، npm، tests، build

13.2 Incremental Investigation Strategy

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

```
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 وجہ کی fail ہو جائے:

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 □لیے □کے 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 □لیے □کے results واپس کریں
- Tool results conversation history میں append □کریں تاکہ □موثر model اگلا action □کر سکے □طے
- Model-driven decision making (Claude اگلا tool □□ چنتا) □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 □لیے □کے 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
- Subagents isolated context خود بخود □ہیں — وہ □نہیں کرتے inherit کی history coordinator میں کام کرتے □ہیں
- 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 □کم □ہو duplication

- Iterative refinement loops implement کریں (coordinator synthesis evaluate اور کر tasks re-route کر)
- Observability تمام لیں کہ communication coordinator ذریعہ کہ route کریں

1.3 Subagent Calls، Context Passing، اور Spawning Configure کرنا

Key knowledge:

- `Task` tool subagents spawn کرتا ہے؛ coordinator کہ `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` ذریعہ کہ session management

Key skills:

- Previous agents کہ 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 ذریعہ کہ parallel subagents spawn کریں
- Coordinator prompts goals اور quality criteria طور پر لکھیں کہ، step-by-step instructions کہ طور پر نہیں

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

Key knowledge:

- Workflow ordering لیے کہ **programmatic enforcement** (hooks، preconditions) اور **prompt guidance** کہ درمیان فرق
- جب deterministic guarantees درکار ہوں (مثلاً مالی operations پہلے سے identity verification)، prompts کافی نہیں ہوتے
- Escalation دوران کہ 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` کہ 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 کہ consume پہلے سے کرنے intercept میں کرتے

- Hooks outgoing calls intercept کر کے compliance rules enforce ہیں کرتے (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 کرنے کے لیے interception hooks اور escalation کی طرف استعمال کریں redirect کریں
- جب business rules کو guaranteed compliance ہو تو ہمیں prompts کے بجائے hooks منتخب کریں

1.6 Complex Workflows کے لیے Task Decomposition Strategies

Key knowledge:

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

Key skills:

- Predictable multi-aspect reviews کے لیے 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 کریں: پہلے structure map کریں، پھر prioritized plan بنائیں

1.7 Session State، Resuming، اور Forking

Key knowledge:

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

Key skills:

- Named investigation sessions continue کرنے کے لیے `--resume` استعمال کریں
- Approaches parallel compare کرنے کے لیے `fork_session` استعمال کریں
- Resuming (context ابھی current ہے) 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 □کا conversation □□ جاری رکھتا
- Long investigations جو متعدد sessions مفید □لیے ان کے □ہوں پر پھیلی
- Risk: اگر پچھلی session بعد □کے files تو □ہوں بدل چکی tool results stale □ہو سکتے □ہیں

`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
- اس کے بعد وہ آزادانہ طور پر diverge □کرتے □ہیں
- Approaches compare یا □کرنے strategies test مفید □لیے کے □کرنے

Resume نئی □بجائے □کے session کب شروع کریں:

- Tool results stale □ہوں (files □ہوں بدل چکی)
- اور □ہو کافی وقت گزر چکا context degrade □ہو گیا
- □پرانے tool data ساتھ □کے resume: □□ یہ □ہے جو پایا اس کا مختصر خلاصہ □ہے کہ "□ہم نے کرنے کے بجائے یہ بہتر □□ کیا جائے restart □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؛ minimal descriptions unreliable selection □ہیں دیتی
- Input formats، example queries، edge cases، اور applicability boundaries □اہمیت کی □کرنے شامل
- Ambiguous یا overlapping descriptions misrouting □ہیں کا سبب بنتی
- System prompt wording tools غیر ارادی ساتھ □کے associations □□ بنا سکتی

Key skills:

- ایسی descriptions □ہر لکھیں جو tool کو similar alternatives واضح طور پر □سے distinguish کریں

- Functional overlap □لیے □کے ختم کرنے tools rename کریں (مثلاً `analyze_content` -> `extract_web_results`)
- General-purpose tools کو specialized tools □کے جن میں تقسیم کریں input/output contracts واضح □وں

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** درمیان فرق □کے
- 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 ساتھ □کے `retryable: false` استعمال کریں
- Transient failures □لیے □کے subagents میں local recovery □وے صرف؛ کریں errors propagate خود □وے جو □وے 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 □کے بجائے 18 مثلاً) tool selection reliability □□ کم کرتی
- Specialization باہر □سے tools □والے رکھنے agents □میں استعمال کرتے □میں غلط انداز
- 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 □کے بجائے tool call □وے یقینی
- Specific tool force □کریں تاکہ execution order assured □وے

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

Key knowledge:

- MCP server scope: ٹیموں کے لیے project (`.mcp.json`) بمقابلہ experiments کے لیے user (`~/.claude.json`)
- Secrets management کے لیے `.mcp.json` میں environment variable substitution (مثلاً `${GITHUB_TOKEN}`)
- تمام connected MCP servers کے tools connection پر discover ہو جاتے ہیں اور ایک ساتھ available ہوتے 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 کے لیے کرنے 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 ہوتے ہیں 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 کے بجائے topic-focused rule files کے لیے `.claude/rules/` directory

Key skills:

- Hierarchy issues diagnose کریں (نئے team member کو instructions وہ نہیں ملتیں کیونکہ اس لیے 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 بنانا اور 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 چلاتا ہے main session pollute نہ ہو
- 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 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` کے تاکہ glob patterns شامل ہو سکتا ہے مطابق 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 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 بمقابلہ Direct Execution کب استعمال کریں

Key knowledge:

- **Planning mode**: complex tasks، بڑے changes، multiple viable approaches، اور architectural decisions کے لیے
- **Direct execution**: سادہ، اچھی طرح سمجھے گئے changes (مثلاً single validation شامل کرنا)
- Planning mode changes پہلے سے codebase کی safe exploration کے ممکن بناتا ہے
- Explore subagent verbose discovery output isolate کرتا ہے

Key skills:

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

3.5 Progressive Improvement کے لیے Iterative Refinement

Key knowledge:

- Concrete input/output examples expectations communicate ہیں کرنے کا سب سے مؤثر طریقے
- **Test-driven iteration**: پہلے tests لکھیں، پھر failures کی بنیاد پر iterate کریں
- “Interview” pattern: Claude clarifying questions پوچھتا ہے تاکہ 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 concrete input/output examples واضح کرنے کے فراہم کریں
- Implementation پہلے سے expected behavior، edge cases، اور performance requirements کے ساتھ test sets بنائیں
- Design aspects (cache invalidation، failure modes) کے لیے سامنے لانے کے interview pattern استعمال کریں
- Edge cases کے لیے sample inputs اور expected outputs کے ساتھ concrete test cases دیں

3.6 Claude Code کو CI/CD Pipelines میں Integrate کرنا

Key knowledge:

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

Key skills:

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

---

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

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

Key knowledge:

- Explicit criteria vague instructions میں سے زیادہ مؤثر ہوتے ہیں (مثلاً “flag comments only when they contradict code” بمقابلہ “check comment accuracy”)
- “be more conservative” جیسی generic guidance concrete categorical criteria سے کم مؤثر ہے

- False positives developer trust کچھ: درست high false-positive rates میں categories پر اثر انداز ہوتے ہیں کم کر دیتی ہیں trust پر بھی categories

Key skills:

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

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

Key knowledge:

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

Key skills:

- Ambiguous scenarios کے لیے rationale 4–2 کے ساتھ 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 کے ساتھ Structured Output Enforce کرنا

Key knowledge:

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

Key skills:

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

- Extensible categorization □لیے □کے `"unclear"` اور `"other"` □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 □نہی موجود میں retries □میں □ہوتے □فائدے □بے
- Feedback loop design: finding کو trigger والا □کرنے pattern (`detected_pattern`) track کریں
- Semantic errors (totals reconcile □نہیں □ہوتے) □بمقابلے syntax errors (جو `tool_use` □سے address □ہوتے □ہیں)

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) □کے لیے □نہیں
- 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 □نہی کو دوبارہ submit □کر کے failures handle کریں
- Large-scale processing □پہلے □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 کو ساتھ رکھتا □□ اور اپنے فیصلوں کو challenge کرنے کا امکان کم ہوتا □□
- Independent review instances (generation context بغیر کے) subtle issues میں بہتر ہوتی ہیں تلاش کرنے میں
- Multi-pass review: per-file local analysis + cross-file integration pass attention dilution □□ سے بچاتا

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 میں تقسیم کریں تاکہ cross-file dataflow analysis ہو سکے
- Self-rated confidence کے ساتھ verification passes کے استعمال کر کے reviews کو calibrated سے طریقے سے route کریں

---

5 ڈومین: Context Management اور Reliability (15%)

5.1 Critical Information محفوظ رکھنے کے لیے 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 سے process کرتے ہیں، لیکن درمیان کے findings miss کر سکتے ہیں
- Tool outputs relevance میں مقابلے کے context میں disproportionally جمع ہو سکتے ہیں (40+ fields 5 جب درکار ہوں)
- Subsequent API requests میں full conversation history کی اہمیت بھیجنے

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) بمقابلہ attempt-to-resolve (agent scope اندر کہ)
- Sentiment analysis اور model confidence self-ratings case complexity کہ unreliable proxies ہیں
- Multiple customer matches لیے کہ heuristic guessing بجائے کہ additional identifiers مانگنے چاہئیں

Key skills:

- System prompt میں explicit escalation criteria few-shot examples دیں کہ ساتھ
- Human کی explicit request کو بغیر اضافی investigation فوراً کہ execute کریں
- جب policy کسی specific request لیے کہ 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 ہے
- Access failures (timeouts retry decision چاہتے ہیں) اور valid empty results (no matches) میں فرق کریں
- Generic error statuses (“search unavailable”) coordinator سے valuable context چھپا دیتے ہیں
- Silent suppression یا ایک failure پر پورا workflow abort کرنا دونوں anti-patterns ہیں

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 کو subagents میں local recovery صرف کریں؛ non-recoverable errors partial results ساتھ کہ propagate کریں
- Synthesis میں coverage annotate کیا اچھی طرح کریں: supported اور کہاں ہے gaps ہیں باقی

5.4 Large Codebases Investigate وقت کرتے Context Efficiently Manage کرنا

Key knowledge:

- Long sessions میں context degradation: model unstable answers اور مخصوص ہونے لگتا ہے classes کہ بجائے “typical patterns” دیتا ہے کا حوالہ
- Scratchpad files key findings کو context boundaries میں کہ پار محفوظ رکھتے ہیں
- Subagents کو delegate سے کرنے verbose discovery output isolate ہے 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 □پہلے □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 □پہلے □سے document type اور field segment □کے حساب سے accuracy validate کریں

Key skills:

- New error patterns detect □لیے □کے کرنے□ stratified random sampling implement کریں
- Stable performance validate □لیے □کے کرنے□ document type اور field □کے حساب سے accuracy analyze کریں
- Field-level confidence scores output اور کریں 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 “claim → source” mappings محفوظ نہ رہیں
- Aggregation دوران □کے structured mappings برقرار رہنی چاہئیں
- Conflicting statistics کو arbitrarily ایک value □بجائے □کے منتخب کرنے□ attribution ساتھ □کے conflict annotate کر □کے handle کریں
- Temporal differences کو contradiction □لیے □کے بچنے□ سے سمجھنے□ 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 □لیے □کے coordinator کو دیں
- Correct temporal interpretation □لیے □کے publication dates شامل کریں

- Content type مطابق □ کو 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 □□ کہ□ ایک ماہ□ر Claude-based وقت درست حل بناتہ□ trade-off امتحان □□□ یہ□ فیصلہ□ کر سکتا Claude Code، Claude Agent SDK، Claude API، اور Model Context Protocol (MCP) جن — یعنی وہ□ بنیادی ٹیکنالوجیز □□ کہ□ بنیادی علم کو جانچتا □□ سہ□ Claude □یں□ کہ□ ساتھ پروڈکشن ایپلیکیشنز تیار کی جاتی □یں□

□یہ□ کسٹمر سپورٹ کہ□ لیہ□ agentic systems بنانا، امتحان کہ□ سوالات حقیقی دنیا کہ□ منظرناموں پر مبنی □وتہ□ □یں: multi-agent research pipelines ڈیزائن کرنا، Claude Code کو CI/CD میں شامل کرنا، developer productivity tools □سہ دستاویزات بنانا، اور غیر ساختہ□ structured data نکالنا□

□دف امیدوار

مثالی امیدوار ایک **solution architect** جو □□ Claude اور ڈیزائن ایپلیکیشنز پروڈکشن ساتھ کہ□ ship کرتا □□□ آپ کہ□ پاس کم از کم 6 ماہ□ کا عملی تجربہ□ □ونا چاہ□یہ□:

- **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 □یہ□ کہ□ 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 1 □میں سہ)

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 پر مشتمل سنبھالتا ہے agent MCP tools (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`) 80 فیصد سے زیادہ first-contact resolution مناسب escalation کے ساتھ استعمال کرتا ہے%+

Scenario 2: Claude Code ساتھ کے Code Generation

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

Scenario 3: Multi-Agent Research System

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

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 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 □ہو□ حاصل Prompts □کو چا□ئیں کہ ایسے بننے false positives □ہوں□ کم سے کم

Scenario 6: Structured Data Extraction

□□ معلومات نکالتا دستاویزات سے □سسٹم غیر ساخت□، output کو JSON schemas □ذریعہ □کے validate ،□□ کرتا اور high accuracy □اسے □□□ برقرار رکھتا edge cases □چا□ئیں درست طور پر سنبھالنے

Scenario 7: Conversational AI Architecture Patterns

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

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

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

□حصہ I: نظریاتی بنیادیں

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

---

1 باب: Claude API — Model Interaction کی بنیادیں

> دستاویزات: Messages API | Prompt Engineering

1.1 API Request Structure

Claude API ایک request–response model □□□ پر کام کرتا Claude Messages API request میں عموماً یہ □شامل □ہوتا □□:

```
{
    "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 دو conversational roles اور ایک instructional role استعمال کرتا ہے:

- `user` — user messages، بشمول tool results (ایک یا `user`-role message اندر کے `tool_result` content block، ایک الگ کی صورت میں بھیجے جاتے ہیں، `tool` role میں طور پر نہیں کے)
- `assistant` — model responses (history میں بھیجتے وقت شامل ہوتے ہیں)، بشمول tool use requests (`tool_use` content blocks)
- `system` — top-level `system` field ہے جسے ذریعے سیٹ کیا جا سکتا ہے (پہلے turn سے لاگو ہوتا ہے) یا اس نقطے سے آگے inline شامل کیا جا سکتا ہے (`{"role": "system", ...}` کے طور پر `messages` میں مخصوص، لاگو ہوتا ہے placement rules دیکھیں نیچے — تابع کے)

Tool results کو `"tool"` role والے message ایک یا `user`-role message کے طور پر نہیں بھیجا جاتا ہے کے طور پر شامل ہوتا ہے `tool_result` content block میں ایک content پر بھیجے جاتے ہیں جس کے:

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

`system` براہ راست `messages` array میں بھی ایک role صرف نہ ہو سکتا ہے، top-level کے طور پر ظاہر ہو سکتا ہے بغیر کیے invalidate کو cached prefix کے top-level `system` field کے ذریعے اس کا مقصد `system` parameter کے مخصوص اس کے شامل کرنا ہے placement rules میں:

- ایک `user` turn (جس میں بشمول وہ `tool_result` blocks یا server tool use والے پر ختم ہونے یہ کے فوراً بعد آنا چاہیے `assistant` turn
- کسی `assistant` turn یا پہلے آنا چاہیے array سے آخری کا element ہونا چاہیے
- کسی `tool_use` block اور اس کے `tool_result` 400 پر ایسا کرنے — درمیان نہیں آ سکتا error ملتا ہے
- پہلے والوں پر اور بعد (کے درمیان شامل کیے گئے conversation بشمول) messages `system` بعد میں آنے والے top-level `system` field پر فوقیت رکھتے ہیں کے لیے turns کے

اہم بات: ہر محفوظ state کے درمیان Model requests بھیجیے conversation history مکمل میں API request آزاد ہوتی ہے call نہیں رکھتا؛ ہر

1.3 `stop_reason`

Claude API response میں `stop_reason` کے جو بتاتا ہے کہ model کیوں رُکا ہے:

Value	Description	Action
`"end_turn"`	Model نے جواب مکمل کر لیا	user کو نتیجہ دکھائیں
`"tool_use"`	Model tool call چاہتا ہے	tool چلائیں اور result واپس دیں
`"max_tokens"`	Token limit ختم ہو گئی	response truncated؛ limit بڑھا سکتے ہیں
`"stop_sequence"`	Stop sequence مل گئی	application logic کے مطابق handle کریں

Agentic systems میں `"tool_use"` اور `"end_turn"` اہم سب سے signals یہی — ہیں loop control کرتے ہیں

1.4 System Prompt

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

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

اہم امتحان کے لیے: system prompt کی wording tool selection مثال ہے کو غیر ارادی طور پر متاثر کر سکتی ہے ضرورت سے زیادہ `get_customer` کو model کی تصدیق کریں" جیسی ہدایت کے طور پر "ہمیشہ استعمال کرنے پر لے جا سکتی ہے

1.5 Context Window

Context window کل متن □و (tokens) □جس □□ 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:** □لمبا input کی درمیان، □یں □وتی □ؤثر □یادہ معلومات کی آغاز اور اختتام □□ کی details miss □یں رکھیں □ہ معلومات شروع یا آخر میں □م حل: □و سکتی □یں□
2. **Tool results □ر □ونا □کا جمع:** tool call context میں output اگر □□□ شامل کرتی tool 40+ fields واپس □وں تو □م 5 صرف □کر context □□□ ضائع □وتا
3. **Progressive summarization:** history compress پر □کرنا numeric values، percentages، اور dates اکثر vague □یں □و جاتی

باب 2: Tools اور `tool_use`

> دستاویزات: Tool Use

2.1 `tool_use` □□ کیا

`tool_use` □و mechanism □ذریعہ □جس □□ Claude external functions call □□□ کر سکتا Model □□برا راست code □و — چلاتا □یں structured tool call کا آپ؛ □□ بناتا code □اسہ execute □کر □کر result واپس کرتا □□□

2.2 Tool Definition

□ر tool JSON schema □ذریعہ □کی define □□ □وتا:

```
{
    "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 واپس کرتا ہے، کیا کرتا ہے، input format کیا ہے، edge cases کیا ہیں، اور یہ کب استعمال ہونا چاہیے۔
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 خود طے کرتا ہے	default عام
`{ "type": "any" }`	Model کو لازماً tool call کرنی ہوگی	جب structured output چاہیے
`{ "type": "tool", "name": "extract_metadata" }`	خاص tool لازماً call ہوگی	forced first step / order کے لیے

2.4 JSON Schemas for Structured Output

JSON schemas کے ساتھ `tool_use` استعمال کرنا Claude سے structured output کا سب سے reliable طریقہ ہے۔ یہ syntax errors اور کم کرتا ہے required structure، مگر semantic correctness کی ضمانت نہیں دیتا۔

Design اصول:

1. صرف وہ fields required رکھیں جو ہمیشہ ملنے چاہئیں
2. ممکنہ missing data کے لیے nullable fields استعمال کریں
3. `"other"` اور `"unclear"` جیسے enum values شامل کریں تاکہ information ضائع نہ ہو

2.5 Syntax اور Semantic Errors

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

---

باب 3: Claude Agent SDK — Agentic Systems بنانا

> دستاویزات: Agent SDK | Hooks | Subagents | Sessions

3.1 Agentic Loop

Agentic loop میں Claude صرف جواب نہیں دیتا، بلکہ actions کی sequence چلاتا ہے:

1. Claude کو tools ساتھ کا request بھیجیں
2. Response لیں
3. `stop_reason` چیک کریں:
   - `"tool_use"` → tool چلائیں، result history میں ڈالیں، پھر دوبارہ request
   - `"end_turn"` → task مکمل، نتیجہ user کو دیں
4. مکمل ہونے تک دہرائیں

درست signal: completion لیے کا `stop_reason == "end_turn"` دیکھیں۔ Text parsing یا arbitrary iteration limit قابلِ اعتماد نہیں ہیں

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"],
)
```

چار چیزیں: `name`، `description`، `system_prompt`، اور `allowed_tools`۔ Least privilege اصول اپنائیں

3.3 Coordinator اور Subagents

Multi-agent systems اکثر hub-and-spoke topology میں استعمال کرتی ہیں:

```
                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

اصول □م□l: subagents کا context □و □□ ہوتا □الگ parent history خود بخود inherit □□ نہ□یں کرتے

3.4 `Task` Tool

Subagents `Task` tool □ذریعہ □کا spawn □یں□ □وتے□ Coordinator □کا `allowed_tools` میں `"Task"` □ہونا چا□ئیے□□

صحیح context passing:

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

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

3.5 Hooks

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

PostToolUse tool result کو model □ل□ پ□ سے پ□نچانے□ تک 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 □یں□ □وتے□؛ prompt instructions probabilistic □یں□ □وتی□ Critical business rules □لیے□ □کا hooks □یں□ بہ□تر

---

4 باب: Model Context Protocol (MCP)

> دستاویزات: MCP | Tools | Resources | Servers

4.1 MCP □□ کیا

MCP ایک open protocol جو □□ external systems کو Claude سے جوڑتا □□□ اس کے تین بنیادی 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 مطابق استعمال کے

4.3 Configuration

Project config (`.mcp.json`) ٹیم کے لیے:

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

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

4.4 `isError`

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

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

4.5 Resources

Resources □و data جو □یں agent بغیر action □□ پڑھ سکتا □کیا: schemas، docs، content catalogs، summaries□یا □ exploratory tool calls □یں□ □کم کرتا

---

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 □□ سکتا □و پر سطحوں تین:

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

5.2 `@path` Syntax

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

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

5.3 `.claude/rules/`

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

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

API files ہیں کہ 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 □ترین □لیے □کے □□□ Direct execution اور واضح □چھوڑ 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 □□ ہوتا □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 □یہ □□□ دکھایا جاتا vague instructions □سے □کیونکہ □□ مؤثر □زیادہ example ambiguity □□□ کم کرتا

6.2 Explicit Criteria

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

6.3 Prompt Chaining

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

6.4 Interview Pattern

Claude □س clarifying questions پوچھوانا design ambiguity جب خاص طور پر، □□ لاتا □سامنا domain unfamiliar □ہو

6.5 Validation and Retry

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

6.6 Self-correction

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

---

7 باب: Message Batches API

> دستاویزات: Message Batches

7.1 Overview

Message Batches API asynchronous processing 50 :□□ □لیا □کا% savings، up to 24 hours window، اور multi-turn tool calling supported □نہیں؛ `custom_id` □س requests اور responses link □ہیں □ہوتا

7.2 کب استعمال کریں

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

7.3 `custom_id`

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

باب 8: Task Decomposition Strategies

8.1 Fixed Pipelines

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

8.2 Dynamic Decomposition

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

8.3 Multi-pass Code Review

Large PRs □لیے□ per-file analysis کے بعد cross-file integration pass □تاکہ کریں attention dilution □کو□ 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 کرنا□ 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 □تاکہ شامل کریں human operator کو مکمل context □□ملے□

9.4 Confidence Calibration

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

---

10 باب: Multi-agent Systems میں Error Handling

10.1 Error Categories

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

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 gaps باقی 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 کہ□ critical data ضائع نہ□ ہو□

11.2 Tool Results Trimming

PostToolUse hook صرف سہ□ relevant fields رکھیں□

11.3 Position-aware Input

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

11.4 Scratchpad Files

Verbose findings scratchpad □تاکہ میں لکھیں long investigations میں context □□□ر محفوظ

11.5 Subagents کو Delegate کرنا

Exploration subagents اور دیں میں main context میں صرف summary □رکھیں

11.6 Structured State Persistence

□ر agent اپنی state manifest یا JSON file میں export □تاکہ □کر crash recovery □و□ ممکن

---

12 باب: Provenance کو محفوظ رکھنا

12.1 Attribution Loss

Claim ساتھ □کہ source URL، publication date، اور confidence □رکھیں

12.2 Conflicting Data

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

12.3 Dates

Temporal differences کو contradiction □ل□ پ□ سے □سمجھنے publication dates □دیکھیں

12.4 Content Type مطابق □کہ Render

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

باب 13: Claude Code □ کا 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 □لیے □کے
نیا file بنانا	**Write**	scratch □سے
precise edit	**Edit**	unique text match
shell command	**Bash**	git، npm، tests

13.2 Incremental Investigation

ایک ساتھ سب files □لا□؛ پ□ڑھیں □نہ entry points، پھر usages، پھر consumer files □دیکھیں

13.3 Read + Write Fallback

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

□حصہ II: امتحانی ڈومین نوٹس

ڈومین 1: Agent Architecture اور Orchestration (27%)

Agentic loops، coordinator–subagent architecture، `Task` □ذریعے □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 □یں □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 توج

ڈومین 5: Context Management اور Reliability (15%)

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

مشق سوالات □ک: Scenario 7 Conversational AI Architecture Patterns

سوال 61

□آپ ک: **صورتحال** `remove_team_member` tool میں execution کا اثرات □ا□ پ□ سے preview □ل□ کے □کرنے `dry_run: boolean` parameter □□□ Production monitoring □ک □□ معلوم □وتا سے agent راست برا□ `dry_run=false` ساتھ □ک call □ک کر preview step کو bypass □ر □ک □□ یقینی بنانا □□ آپ کو □ یقینی بنانا □□□ کرتا deletion ایک □ا□ پ□ سے preview □و جسے □و user □ explicitly confirm □و □ کیا

□کونسی approach سب سے قابل اعتماد؟

- A) Server-side validation شامل کریں جو `dry_run=false` صرف اسی وقت allow 60 □ا□ جب پچھلا □کر seconds میں یکساں parameters ساتھ □ک `dry_run=true` call □و □وئی
- B) Tool کو confirmation پر طور □ک ضرورت کی annotate کریں اور orchestration layer کو configure کریں □ک annotated tools کو calls forward □ا□ پ□ سے کرنے user کی approval □□مانگ
- C) Tool description میں تفصیلی instructions اور few-shot examples شامل کریں جن میں agent □مطالبہ سے □ا□ پ□ □میشہ □و □ک □و □و کو `dry_run=true` سے call □□کر
- D) دو tools □سے replace کریں: `preview_remove_member` impact details اور ایک single-use confirmation token واپس کرتا □□؛ `execute_remove_member` □و کو token □□ درکار، execution کو preview □سے bind کرتا □□□ **[درست]**

D کیوں: Token-binding approach □ یہ □سے architecturally □ا□ پ□ □ک □□ نا□ممکن □و جاتا preview بغیر □ک execution □و — execute tool کو literally □و token جو صرف □چا□یہ preview tool □ی generate □□□ کر سکتا □یہ واحد approach جو □□ LLM instructions کی پابندی (C)، timing heuristics (A)، یا orchestration infrastructure (B) □بجائے □ک انحصار پر code level پر constraint enforce □□□ کرتی

سوال 62

صورتحال: Production monitoring کا آپ کو معلوم ہوتا ہے کہ `search_catalog` tool 12% وقت fail ہوتا ہے جو کبھی syntax errors 8% پر کامیاب ہوتے ہیں، اور network timeouts 4 retry ہیں جو فی الحال دونوں کامیاب نہیں ہوتے۔ یکساں طریقہ سے error types return فائدہ ہے، جس سے retries ہوتی ہیں۔

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 کے 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

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

سوال 63

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

سے priority کی اصل recommendation user سب سے بہتر طریقہ سے یقینی بناتا ہے کہ approach کونسا ہم آہنگ ہو؟

- A) تضاد کو سامنے لائیں اور user سے پوچھیں کہ کونسی چیز زیادہ اہم ہے **[درست]**
- 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 A: جب 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 ہونے والے recommendation user طریقہ ہے کہ بنیادی طور پر ناقابل مطابقت اہداف ہیں جن کے لیے انسانی فیصلے درکار tolerance

---

سوال 64

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

سب سے زیادہ ممکنہ وجہ کیا ہے؟

- A) Claude کو conversation memory برقرار رکھنے کے لیے vector database connection درکار

- B) Model کی context window exceed □□□ ہو گئی
- C) Claude API کو `session_id` parameter □□□ درکار
- D) آپ کی application `messages` array میں پچھلے messages **[درست]** شامل نہیں کر رہی

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

---

سوال 65

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

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

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

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

سوال 66

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

مؤثر سب سے solution □□؟ کیا

- A) Hybrid approach: پرانے messages □حالیہ ہوئے کرتے خلاصہ کا 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 اور کریں خلاصہ کا running summary □جوڑیں آگے

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

سوال 67

صورتحال: Users جب □ کہ □یں رپورٹ کرتے □یں تو conversations 50 turns latency بڑھ جاتی سے زیادہ □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 والا کی ضرورت internal user profile □□□ بناتا

A کیوں: Claude کی API مکمل طور پر stateless ہے — □□ request کو `messages` array میں پوری conversation history □یسے □یسے □□□ شامل کرنی □وتی conversations ہر □یں، بڑھتی request زیادہ □ہوتی جاتی □□، جو tokens Model calls دونوں بڑھاتا □□□ cost اور directly processing latency کے درمیان کوئی internal state رکھتا □یں 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 نے user reference □□□ کرنے کے بجائے عام جوابات دیتا □□□ کرنے

مؤثر سب سے approach □□؟ کیا

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

C کیوں: Conversation history پر semantic search واحد approach کی جو تین مہینوں □□ discussion کو scale □جبکہ □□ کر سکتی 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 abstractions میں compress □□ کرتی، specific conclusions میں جن کے بارے کھو دیتی □□ users پوچھتے □یں□

سوال 69

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

بہترین solution □□؟ کیا

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

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

سوال 70

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

کیا □□؟ fix سب سے مؤثر

- A) 4–5 □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 □ لیے □کے vulnerable □کیونکہ □□ abstract rules □ہر turn میں model □سے reasoning □□ہیں□ کا تقاضا کرتے □ Verbose rules کو concrete few-shot examples □سے replace کرنا جو correct proficiency-level adaptation demonstrate □کریں، model کو match □ لیے □کے کرنے clear behavioral patterns □ کئی □ہے — □□ دیتا turns میں abstract instructions □زیادہ □سے reliably follow □□□ہوتا

سوال 71

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

□ہے behavioral guidelines □کہاں define □کرنی چاہئیں؟

- A) □ہر 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 □ہونے persistent behavioral constraints اور guidelines □ہر □□□ ڈیزائن کیا گیا □ لیے □کے user message شامل کرنا □□پ□ □سے (A) redundant overhead □□پ □□□ assistant message (C) unreliable □کیونکہ □□ model □□پ□ اپنے statements □سے deviate □□□ کر سکتا Environment variables (D) model □کے behavior □□ڈالتے □ہیں پر اثر نہ□

سوال 72

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

کیا □□؟ approach سب سے مؤثر

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

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

سوال 73

صورتحال: ایک webhook آپ کے system کو notify کرتا □□ کہ user کا package ship ہو گیا □□ جبکہ user actively chat میں ہے آپ چاہتے ہیں کہ assistant اسے اگلے response میں naturally شامل کرے □□ کر رہا □□□

کیا □□؟ approach بہترین

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

D کیوں: Status update کو اگلے user message کے prefix طور پر شامل کرنا کے 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 ہر □ □□□ turn کر سکتا میں tool call force کرنا (C) اس وقت wasteful جب □□ events rare ہوں□

سوال 74

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

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

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

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

سوال 75

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

سب سے زیادہ ممکنہ وجہ کیا □□؟

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

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

سوال 76

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

بہ□ترین solution کیا □□؟

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

A کیوں: Reasonable stated assumptions بڑھنا آگے ساتھ کے user کو informed اور in control □وئے □رکھتے 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 مسئلے حل کیے 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 □کا 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

□یہ موضوعات امتحان میں شامل نہ□یں □ہیں:

- 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 کا 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