WORKSHOP 0 OF 5

Getting Started: Claude API Fundamentals

Your foundation before the 5 exam domains

WHAT YOU'LL LEARN IN THIS WORKSHOP

- Set up your Anthropic API key and Python environment
- Make your first Messages API call with proper structure
- Understand tokens, models, and response anatomy
- Handle streaming responses and basic errors
- Know the difference between claude-opus, sonnet, and haiku

EXAM WEIGHT: Foundations	60 Questions | 120 Minutes | Pass Score: 720/1000

BEGINNER Foundations	INTERMEDIATE Application	ADVANCED Production

BEGINNER

1. Your First Claude API Call

Installing the SDK

```
# Install the official Anthropic Python SDK
pip install anthropic
# Set your API key (never hardcode this in source files)
export ANTHROPIC_API_KEY='sk-ant-...'
```

The Minimal API Call

Every call to Claude goes through the Messages API. The three required fields are **model**, **max_tokens**, and **messages**.

```
import anthropic
client = anthropic.Anthropic() # reads ANTHROPIC_API_KEY from env
message = client.messages.create(
model='claude-sonnet-4-5',
max_tokens=1024,
messages=[
{"role": "user", "content": "What is an agentic loop?"}
]
)
print(message.content[0].text)
```

Key Insight

content[0].text — Claude's response is always a list of content blocks. For simple text replies there is exactly one block of type 'text'. Tool-use responses add more blocks. Always index into content[0].

Response Anatomy

Field	Type	What it means
id	str	Unique message ID (useful for logging)
role	str	Always 'assistant' for responses
content	list	List of ContentBlock objects (text or tool_use)
model	str	Exact model version that served the request
stop_reason	str	'end_turn' | 'tool_use' | 'max_tokens' | 'stop_sequence'
usage.input_tokens	int	Tokens consumed from your prompt

usage.output_tokens	int	Tokens the model generated

Model Selection Guide

Model	Best For	Speed	Cost
claude-opus-4	Complex reasoning, long-context planning	Slow	$$$
claude-sonnet-4-5	Balanced — most production workloads	Medium	$$
claude-haiku-4-5	High-throughput, low-latency tasks	Fast	$

INTERMEDIATE

2. System Prompts, Multi-Turn & Streaming

System Prompts

The system prompt sets Claude's persona, constraints, and output rules. It is passed as a top-level parameter — NOT as a message in the messages array.

```
message = client.messages.create(
model='claude-sonnet-4-5',
max_tokens=1024,
system='You are a code reviewer. Respond only in JSON.',
messages=[{'role': 'user', 'content': 'Review: def add(a,b): return a+b'}]
)
```

Multi-Turn Conversations

```
history = []
def chat(user_message: str) -> str:
history.append({'role': 'user', 'content': user_message})
response = client.messages.create(
model='claude-sonnet-4-5', max_tokens=1024, messages=history
)
assistant_text = response.content[0].text
history.append({'role': 'assistant', 'content': assistant_text})
return assistant_text
```

Streaming Responses

```
with client.messages.stream(
model='claude-sonnet-4-5', max_tokens=2048,
messages=[{'role': 'user', 'content': 'Write a detailed doc'}]
) as stream:
for text_chunk in stream.text_stream:
print(text_chunk, end='', flush=True)
final_message = stream.get_final_message()
print(f'Tokens: {final_message.usage.output_tokens}')
```

ADVANCED

3. Token Management, Caching & Error Handling

Prompt Caching (50–90% Cost Reduction)

```
message = client.messages.create(
model='claude-sonnet-4-5',
max_tokens=1024,
system=[{
"type": "text",
"text": LARGE_KNOWLEDGE_BASE,
"cache_control": {"type": "ephemeral"}
}],
messages=[{'role': 'user', 'content': user_question}]
)
usage = message.usage
print(f'Cache read: {usage.cache_read_input_tokens}')
print(f'Cache write: {usage.cache_creation_input_tokens}')
```

Production Error Handling

```
import anthropic, time
def call_claude_with_retry(messages, max_retries=3):
for attempt in range(max_retries):
try:
return client.messages.create(
model='claude-sonnet-4-5', max_tokens=1024,
messages=messages)
except anthropic.RateLimitError:
wait = 2 ** attempt
time.sleep(wait)
except anthropic.APIStatusError as e:
if e.status_code == 529:
time.sleep(5)
else:
raise
raise RuntimeError('Max retries exceeded')
```

Module 0 — All 60 Practice Questions

TOTAL: 60 QUESTIONS	Beginner: 20	Intermediate: 20	Advanced: 20

BEGINNER

Q1. Which three fields are REQUIRED in every call to client.messages.create()?

A) model, max_tokens, system

B) model, max_tokens, messages

C) model, temperature, messages

D) model, messages, stream

Correct Answer: B

The three required parameters are model (which Claude variant), max_tokens (output budget), and messages (the conversation array). system and temperature are optional.

Q2. Where does the actual response text live in the API response object?

A) response.text

B) response.message

C) response.content[0].text

D) response.output.text

Correct Answer: C

Claude responses return a list of content blocks. For simple text replies, content[0].text contains the response. This is because responses can include multiple blocks (e.g., text + tool_use).

Q3. You want to use Claude for a quick classification task that runs 10,000 times per day at low cost. Which model is the best fit?

A) claude-opus-4 — most capable

B) claude-sonnet-4-5 — balanced

C) claude-haiku-4-5 — fastest and cheapest

D) All models cost the same

Correct Answer: C

claude-haiku-4-5 is optimized for high-throughput, low-latency, low-cost workloads. For tasks that don't require deep reasoning, haiku delivers significant cost savings compared to sonnet or opus.

Q4. What does the stop_reason field in the API response indicate?

A) Whether the API call succeeded or failed

B) Why Claude stopped generating: end_turn, tool_use, max_tokens, or stop_sequence

C) The number of tokens remaining in the context window

D) Which safety filter triggered

Correct Answer: B

stop_reason tells you WHY generation stopped. 'end_turn' means Claude finished naturally. 'tool_use' means Claude wants to call a tool. 'max_tokens' means you hit your token budget. This field drives agentic loop control flow.

Q5. How should you store your Anthropic API key in a production application?

A) Hardcode it as a string constant in your main application file

B) Store it in a .env file committed to your git repository

C) Set it as an environment variable (ANTHROPIC_API_KEY) and read it at runtime

D) Pass it as a URL parameter in API requests

Correct Answer: C

API keys must never be hardcoded or committed to version control. Environment variables are the standard secure approach. The Anthropic SDK reads ANTHROPIC_API_KEY automatically if set, so you don't even need to pass it explicitly.

Q6. What is the purpose of the system parameter in client.messages.create()?

A) It specifies which Claude model to use

B) It sets Claude's persona, constraints, and output format rules for the conversation

C) It defines the maximum number of tokens Claude can generate

D) It enables or disables tool use

Correct Answer: B

The system prompt configures Claude's behavior for the entire conversation — its role, output format, rules, and constraints. It is a top-level parameter, NOT a message in the messages array.

Q7. Claude has no built-in memory between API calls. How do you maintain a multi-turn conversation?

A) Use the session_id parameter to link calls

B) Enable persistent_memory in the API options

C) Manually append each user message and assistant response to the messages array before each call

D) Use the /continue endpoint instead of /messages

Correct Answer: C

Claude is stateless. You must maintain conversation history yourself by building up the messages array: user turn, then assistant turn (from the previous response), then next user turn, and so on.

Q8. What happens if you set max_tokens too low and Claude hasn't finished its response?

A) Claude automatically continues in a follow-up message

B) The API returns an error and no content

C) The response is truncated and stop_reason is set to 'max_tokens'

D) Claude summarizes its response to fit within the limit

Correct Answer: C

When Claude hits max_tokens mid-generation, it stops immediately and returns what it has. The stop_reason will be 'max_tokens' — a signal you should check for and handle programmatically (e.g., by requesting continuation).

Q9. Which model is best suited for complex multi-step reasoning tasks that require deep analysis and long-context planning?

A) claude-haiku-4-5

B) claude-sonnet-4-5

C) claude-opus-4

D) All models perform identically on reasoning tasks

Correct Answer: C

claude-opus-4 is Anthropic's most capable model, designed for complex reasoning, nuanced analysis, and tasks requiring careful multi-step thinking. The tradeoff is higher latency and cost.

Q10. What does the usage object in the API response contain?

A) The total cost of the API call in dollars

B) input_tokens and output_tokens counts for the call

C) The number of API calls remaining in your quota

D) The model version and timestamp

Correct Answer: B

usage.input_tokens is the number of tokens in your prompt (system + messages), and usage.output_tokens is the number Claude generated. This is essential for cost tracking and context window monitoring.

Q11. When would you use streaming (client.messages.stream) instead of a synchronous call?

A) When you need the response to be more accurate

B) When you want to display output progressively as it generates, like in a chat UI

C) When the request involves tool use

D) When using the Batch API

Correct Answer: B

Streaming lets you display tokens as they arrive rather than waiting for the full response. This is ideal for chat interfaces, long-form content generation, or any scenario where showing partial results improves user experience.

Q12. What is the role parameter in a messages array entry, and what are its valid values?

A) role specifies the model; valid values are 'claude' or 'human'

B) role identifies the turn author; valid values are 'user' and 'assistant'

C) role sets permissions; valid values are 'admin' and 'user'

D) role defines message priority; valid values are 'high', 'medium', 'low'

Correct Answer: B

Messages alternate between 'user' turns (your input) and 'assistant' turns (Claude's responses). You must alternate them — two consecutive 'user' messages or two consecutive 'assistant' messages will cause an API error.

Q13. You receive an anthropic.RateLimitError. What should your code do?

A) Immediately retry the same request

B) Wait using exponential backoff (e.g., 1s, 2s, 4s) then retry

C) Switch to a different model and retry immediately

D) Log the error and return an empty response to the user

Correct Answer: B

Rate limit errors (429) mean you're sending too many requests too fast. Exponential backoff — waiting increasingly longer between retries — is the standard approach. Immediate retry just triggers more rate limiting.

Q14. What is a content block in the Claude API response?

A) A section of the system prompt

B) An element in the content list — either type 'text' or type 'tool_use'

C) A batch of multiple API responses

D) A cached portion of the prompt

Correct Answer: B

Claude's response content is a list of blocks. Each block has a type: 'text' for regular responses, 'tool_use' when Claude wants to call a tool. Agentic loops iterate over content blocks to find tool_use requests.

Q15. Which of the following is a valid way to pass the system prompt?

A) messages=[{'role': 'system', 'content': 'You are helpful'}]

B) system='You are helpful' as a top-level parameter to messages.create()

C) {'role': 'user', 'content': '[SYSTEM] You are helpful'} as the first message

D) options={'system': 'You are helpful'}

Correct Answer: B

The system prompt is a dedicated top-level parameter in messages.create(), NOT a message with role='system'. Adding role='system' to the messages array will cause a validation error.

Q16. What does temperature control in the Claude API?

A) The maximum response length

B) The randomness/creativity of responses — higher values produce more varied output

C) The speed of response generation

D) The safety filtering level

Correct Answer: B

Temperature (0.0 to 1.0) controls output randomness. Low values (0.0-0.3) produce deterministic, consistent outputs — good for structured extraction. Higher values produce more creative, varied responses — good for brainstorming.

Q17. You want Claude to always return a JSON object. What is the LEAST reliable approach?

A) Using tool_use with a JSON schema

B) Adding 'respond only in JSON' to the system prompt

C) Using tool_choice to force a specific extraction tool

D) Setting response format via the API schema parameter

Correct Answer: B

Prompt instructions for JSON output are unreliable — Claude may add prose, use markdown fences, or produce invalid JSON especially on edge cases. Tool use with a defined schema is the only guaranteed way to get schema-compliant JSON.

Q18. What does the Anthropic SDK do if you don't pass an API key to the Anthropic() constructor?

A) It generates a temporary API key automatically

B) It reads the ANTHROPIC_API_KEY environment variable

C) It throws an error immediately on initialization

D) It uses a free tier with limited capabilities

Correct Answer: B

The Anthropic SDK automatically reads the ANTHROPIC_API_KEY environment variable if no key is passed explicitly. This is the recommended approach — keep secrets in environment variables, not in code.

Q19. Which stop_reason value indicates that Claude successfully completed its response without hitting any limits?

A) 'success'

B) 'complete'

C) 'end_turn'

D) 'finished'

Correct Answer: C

'end_turn' is the stop_reason when Claude finishes its response naturally. This is the expected value for successful, complete responses. The agentic loop exits when it sees 'end_turn'.

Q20. What is the maximum context window size for current Claude models (claude-sonnet-4-5, claude-haiku-4-5)?

A) 32K tokens

B) 100K tokens

C) 200K tokens

D) 1M tokens

Correct Answer: C

Current Claude models support a 200K token context window, which is roughly 150,000 words or about 500 pages of text. This counts system prompt + all messages + tool results.

INTERMEDIATE

Q21. You are building a multi-turn chat application. After each user message, you call the API and get a response. What must you do before the NEXT user message?

A) Start a new client.messages.create() session

B) Append the assistant's response content to the messages array

C) Clear the messages array and start with only the new user message

D) Call client.sessions.update() with the latest response

Correct Answer: B

Claude has no memory. You must manually maintain conversation history by appending each assistant response (as {'role': 'assistant', 'content': response.content[0].text}) to the messages list before the next call.

Q22. Prompt caching uses cache_control: {type: 'ephemeral'} on a content block. What is the primary benefit?

A) Responses are served faster because the model skips thinking

B) Cached tokens cost approximately 10% of normal input price on re-use, reducing costs by up to 90%

C) The cached content is permanently stored and never expires

D) Cached prompts bypass the context window limit

Correct Answer: B

Prompt caching stores stable prompt sections server-side. Cache writes cost 125% of normal input price, but cache reads cost only 10%. For large repeated knowledge bases (10,000+ tokens used hundreds of times daily), this delivers massive cost savings.

Q23. Which usage fields indicate a prompt cache hit vs. a cache miss?

A) usage.cache_hit: true vs usage.cache_hit: false

B) usage.cache_read_input_tokens (hit) vs usage.cache_creation_input_tokens (miss/write)

C) usage.cached_tokens vs usage.uncached_tokens

D) usage.cache_status: 'hit' or 'miss'

Correct Answer: B

cache_read_input_tokens shows tokens served from cache (cheap). cache_creation_input_tokens shows tokens written to cache (slightly more expensive than normal). Monitor these to verify caching is working as expected.

Q24. Your application calls Claude with the same 15,000-token knowledge base prepended to every request. Cache TTL is 5 minutes. You make 200 calls per minute. What is the correct caching strategy?

A) Cache is useless here — 5 minutes covers less than 1% of daily traffic

B) Apply cache_control to the knowledge base block; it will be re-written every 5 min but read 1,000 times between writes

C) Split the knowledge base into 10 smaller chunks and cache each separately

D) Use the Batch API instead of caching for this volume

Correct Answer: B

With 200 calls/min, each 5-minute cache window serves ~1,000 requests. The knowledge base is written to cache once and read ~999 times at 10% cost. Net savings are substantial even with frequent cache refreshes.

Q25. You want to use streaming and need the complete final message object after the stream ends. How do you get it using the Python SDK?

A) stream.message after the context manager exits

B) Call stream.get_final_message() after iterating text_stream

C) Concatenate all text_chunk values from text_stream

D) stream.complete() returns the full response object

Correct Answer: B

The streaming context manager provides stream.get_final_message() which returns the complete Message object with all fields including usage stats. Concatenating text_stream chunks gives you text only, not the full response.

Q26. What is the correct message structure when appending a tool result back to the conversation?

A) {'role': 'user', 'content': 'Tool result: [result]'}

B) {'role': 'tool', 'content': result}

C) {'role': 'user', 'content': [{'type': 'tool_result', 'tool_use_id': block.id, 'content': result}]}

D) {'role': 'assistant', 'content': [{'type': 'tool_result', ...}]}

Correct Answer: C

Tool results are returned as user-role messages containing a list with type='tool_result'. The tool_use_id must match the id from the tool_use block in Claude's response. Multiple tool results can be in one user turn.

Q27. You are building a customer service bot. The system prompt is 5,000 tokens. Over a long conversation, usage.input_tokens reaches 190,000. What should you do?

A) Increase max_tokens to give Claude more room

B) Summarize the conversation history using a cheap model and restart with the summary

C) Switch to claude-opus which has a larger context window

D) Delete the system prompt to free up tokens

Correct Answer: B

When approaching the context limit, summarize old conversation turns (using haiku for efficiency) and start a new session with the summary as initial context. All current Claude models share the same 200K context limit.

Q28. A 529 status error ('overloaded') is returned from the API. What is the correct response?

A) Switch to a different model immediately

B) Wait a few seconds and retry — Anthropic servers are temporarily overloaded

C) This indicates your account is suspended — contact support

D) Reduce max_tokens and retry immediately

Correct Answer: B

Status 529 means Anthropic's servers are temporarily overloaded. It is a transient error — wait (5-10 seconds) and retry. Unlike 4xx errors which indicate client mistakes, 529 and 5xx errors are server-side and typically resolve quickly.

Q29. You need to pass a 50-page PDF to Claude for analysis. What is the recommended approach?

A) Convert the PDF to text, then pass as a user message

B) Pass the PDF as base64 with type 'document' and media_type 'application/pdf'

C) Split the PDF into individual pages and make 50 separate API calls

D) PDFs are not supported — use only plain text

Correct Answer: B

Claude natively supports PDF input via the document content type. Pass it as base64-encoded data with media_type 'application/pdf'. This is more efficient and preserves document structure better than text extraction.

Q30. What is the difference between stop_sequences and max_tokens as stopping mechanisms?

A) There is no difference — both stop generation at the same point

B) stop_sequences stops when Claude generates a specific token pattern; max_tokens stops after a count regardless of content

C) max_tokens is for text responses; stop_sequences is for tool calls only

D) stop_sequences is deprecated — use max_tokens exclusively

Correct Answer: B

stop_sequences lets you define specific strings (like '###END###') that trigger early stopping when generated. max_tokens is a hard count limit. Both set stop_reason ('stop_sequence' or 'max_tokens') when triggered.

Q31. You need to process 500 invoices overnight with no latency requirement. Which API feature reduces cost by 50%?

A) Prompt caching with cache_control

B) The Message Batches API

C) Streaming responses

D) Reducing max_tokens to 256

Correct Answer: B

The Message Batches API offers 50% cost savings for latency-tolerant workloads. Requests are processed within 24 hours. This is perfect for overnight batch jobs like invoice processing, report generation, or bulk classification.

Q32. What happens to messages with role='assistant' that you manually add to the messages array?

A) They are ignored — only user messages are accepted

B) Claude treats them as its own prior responses and uses them as conversation context

C) They trigger an API validation error

D) They are treated as system prompt additions

Correct Answer: B

You can inject assistant-role messages to prime Claude's behavior or provide example conversation history. This is a technique called 'prefilling' — Claude treats them as its own prior responses and continues from that context.

Q33. Which approach correctly prevents API key exposure in a Node.js application deployed to Vercel?

A) Store the key in a public environment variable prefixed with NEXT_PUBLIC_

B) Hardcode the key but obfuscate it with base64 encoding

C) Store the key in Vercel's environment variables (server-side only) and access via process.env

D) Pass the key from the frontend using localStorage

Correct Answer: C

API keys must only exist server-side. Vercel's server-side environment variables are never exposed to the browser. NEXT_PUBLIC_ variables are exposed client-side — never use them for API keys.

Q34. You want to ensure Claude produces only a specific JSON structure and no prose. What is the most reliable approach?

A) Add 'output ONLY valid JSON, no other text' to the system prompt

B) Use tool_use with a JSON schema, forcing tool_choice to the extraction tool

C) Post-process Claude's response to extract JSON from any prose

D) Use temperature=0 to make output deterministic

Correct Answer: B

Tool use with a defined JSON schema is the only approach that guarantees syntactically valid schema-compliant output. System prompt instructions are probabilistic. temperature=0 reduces randomness but doesn't enforce structure.

Q35. A streaming request is interrupted mid-response due to a network error. What is the best recovery strategy?

A) Retry the full request from the beginning

B) Use the last received event ID to resume from the interruption point

C) Send the partial response to the user with an error message

D) Switch to synchronous (non-streaming) for this request

Correct Answer: A

Claude's streaming API does not support resuming mid-stream. You must retry the full request. To minimize user impact, implement retry logic that automatically restarts the stream and consider showing 'reconnecting...' in the UI.

Q36. What is the correct order of turns in a valid messages array for a multi-turn conversation?

A) assistant, user, assistant, user

B) user, assistant, user, assistant

C) system, user, assistant, user, assistant

D) user, user, assistant, assistant

Correct Answer: B

Messages must strictly alternate: user → assistant → user → assistant. The array must start with a user message. system is a separate top-level parameter, not a message. Two consecutive same-role messages cause a validation error.

Q37. You're building an application that calls Claude with different user queries but always the same 20,000-token knowledge base. Cache TTL is 5 minutes. What cache_control placement maximizes savings?

A) Apply cache_control to each user message

B) Apply cache_control to the knowledge base block in the system prompt

C) Apply cache_control to the model parameter

D) Apply cache_control to the last assistant message

Correct Answer: B

Cache the stable, expensive part — the knowledge base in the system prompt. User messages change with every request and cannot be cached effectively. The system prompt knowledge base is identical across calls and benefits most.

Q38. What does usage.cache_creation_input_tokens represent?

A) Tokens that were read from the cache (cheap)

B) Tokens written to the cache for the first time (125% of normal price)

C) Tokens skipped due to caching (cost $0)

D) The total tokens in the cached block

Correct Answer: B

cache_creation_input_tokens are tokens being written to cache for the first time — they cost 125% of normal input price (slightly more expensive). On subsequent calls they become cache_read_input_tokens at 10% of normal price.

Q39. Your application needs to process 1,000 documents with a 30-hour SLA. The Batch API has a 24-hour processing window. What submission frequency ensures the SLA?

A) Submit all 1,000 at once — 24 hours is within the 30-hour SLA

B) Submit every 6 hours — gives a 6-hour buffer after 24-hour processing

C) Submit every 30 minutes — over-provisioning for safety

D) Use the synchronous API instead — batch is too slow

Correct Answer: B

With a 30-hour SLA and 24-hour maximum batch processing: submitting every 6 hours means the worst-case is submission at hour 0, processing completes at hour 24, well within the 30-hour window. More frequent submission isn't necessary.

Q40. You call the Batch API and some requests fail. How do you identify and reprocess only the failed ones?

A) Re-submit the entire batch — there's no way to identify individual failures

B) Use the custom_id field to correlate results; failed results have result.type != 'succeeded'

C) Check the batch error log in the Anthropic dashboard

D) Failed batch requests are automatically retried by the API

Correct Answer: B

Each batch request has a custom_id you set. In results, filter for result.type != 'succeeded' to find failures. Re-submit only those custom_ids with corrected parameters (e.g., smaller chunks if context was too large).

ADVANCED

Q41. You have a high-volume RAG system: 10,000 calls/day with a 25,000-token knowledge base and 500-token average user query. Cache TTL is 5 minutes, so the KB is re-cached ~288 times/day (1440 min / 5 min). Calculate the approximate daily cost ratio of caching vs no caching.

A) Caching saves ~10% — barely worth the complexity

B) Caching saves ~50% on input costs for the knowledge base portion

C) With 10,000 reads and ~288 writes, roughly 97% of KB tokens are read at 10% cost — approximately 90%+ savings on KB input costs

D) Caching doesn't apply to RAG systems with dynamic queries

Correct Answer: C

9,712 cache reads (97%) at 10% cost vs 288 cache writes at 125% cost. For a 25,000-token KB: reads cost 9,712 x 25K x 0.1 = 24.3M token-equivalents. No caching: 10,000 x 25K = 250M token-equivalents. Net: ~90% savings on the KB portion.

Q42. You are designing a production Claude integration that must handle concurrent requests from 1,000 simultaneous users. What is the primary architectural consideration?

A) Use a single shared Anthropic client instance — it is thread-safe and handles concurrency

B) Create one Anthropic client per user session for isolation

C) Use async/await with a connection pool and implement per-user rate limiting to stay within API rate limits

D) Route requests through a single queue — parallel API calls are not supported

Correct Answer: C

The Anthropic client is thread-safe, so one instance is fine. The real challenge is respecting API rate limits under high concurrency. Use asyncio with httpx, implement a token bucket or semaphore for rate limiting, and handle RateLimitError with backoff.

Q43. A production system needs to process documents of unknown length. Some are 500 tokens, some are 180,000 tokens. What strategy handles both efficiently?

A) Always use 200K max_tokens regardless of document size

B) Estimate document token count first; use full context for large docs, chunk small docs into batches

C) Split all documents into 10,000-token chunks regardless of length

D) Use claude-opus for large documents and claude-haiku for small ones

Correct Answer: B

Adaptive sizing: estimate tokens (chars / 4), then choose strategy. Small docs can be batched together in one request (cheaper). Large docs get dedicated requests. Uniform chunking wastes capacity on small docs and may truncate large ones unnecessarily.

Q44. You need to implement a 'prefill' technique to guide Claude's output format. What does this mean?

A) Adding format instructions to the system prompt before the user message

B) Adding a partial assistant-role message at the end of the messages array that Claude must continue

C) Pre-loading common responses into a cache before user requests arrive

D) Setting initial_content in the API parameters

Correct Answer: B

Prefilling means adding an incomplete assistant message at the end of the messages array, e.g., {'role': 'assistant', 'content': '{"result":'}. Claude will continue generating from that point, effectively forcing a specific start to its output.

Q45. Your streaming application must handle the case where a tool_use block arrives across multiple stream events. What is the correct approach?

A) Each tool_use block always arrives in a single stream event

B) Accumulate content_block_delta events for tool_use blocks until content_block_stop, then parse the complete JSON input

C) Make a separate non-streaming call when tool use is detected

D) tool_use is not supported in streaming mode

Correct Answer: B

In streaming, tool_use input JSON is delivered incrementally via content_block_delta events. You must buffer all deltas for a given content block index until content_block_stop, then JSON.parse the complete accumulated string.

Q46. You need to implement extended thinking to improve Claude's reasoning on a complex problem. What does enabling extended thinking change about the response?

A) It increases max_tokens automatically

B) Claude outputs a thinking block before its response, showing internal reasoning that you can inspect

C) The API call is routed to a special reasoning server with different pricing

D) Extended thinking replaces the regular response with a step-by-step chain

Correct Answer: B

Extended thinking adds a 'thinking' content block to the response containing Claude's internal reasoning. You set a thinking budget (min 1024 tokens). The thinking block helps debug complex reasoning but also costs tokens.

Q47. A long-running agentic workflow has accumulated 150,000 tokens of context. You need to continue the task but are approaching the limit. What is the most reliable continuation strategy?

A) Continue using the same context — 200K is the limit so you have room

B) Ask Claude to summarize the full conversation, start a new session with that summary plus remaining task instructions

C) Delete the oldest 50,000 tokens and continue

D) Increase max_tokens to create more headroom

Correct Answer: B

At 150K tokens, you're at 75% capacity and approaching risk. A structured handoff — summarize key findings and decisions, start fresh — is more reliable than risking context overflow. Deleting arbitrary old content may lose critical task context.

Q48. You are implementing a cost monitoring system for Claude API usage. Which combination of fields gives you the complete per-call cost breakdown?

A) usage.total_tokens only

B) usage.input_tokens, usage.output_tokens, usage.cache_read_input_tokens, and usage.cache_creation_input_tokens

C) usage.input_tokens and usage.output_tokens — cache is free

D) usage.total_cost (a dollar amount field in the response)

Correct Answer: B

Full cost = (input_tokens x input_price) + (output_tokens x output_price) + (cache_creation_input_tokens x 1.25 x input_price) + (cache_read_input_tokens x 0.1 x input_price). The API never returns a dollar cost directly.

Q49. You need Claude to act as a specialized financial advisor persona consistently across thousands of independent API calls, each starting fresh. What is the most maintainable approach?

A) Pass the persona description in every user message

B) Create a system prompt template with the persona and cache_control it; instantiate a new client per request

C) Use a single shared session for all users

D) Train a fine-tuned Claude model with the persona baked in

Correct Answer: B

A cached system prompt is the right pattern: define the persona once in system, apply cache_control for cost efficiency (reused across all calls), and start each user conversation fresh. This ensures consistent, isolated, cost-efficient sessions.

Q50. What is the architectural difference between using tool_choice='any' versus tool_choice={'type':'tool','name':'extract_invoice'} in a structured extraction pipeline?

A) There is no practical difference — both guarantee tool use

B) 'any' lets Claude choose among available tools; forced selection ensures the specific extraction schema is always used regardless of document type

C) 'any' is for single-tool scenarios; forced is for multi-tool

D) Forced selection is only available in the Batch API

Correct Answer: B

'any' guarantees a tool is called but Claude chooses which one — problematic if you have multiple extraction schemas and want a specific one. Forced selection guarantees the exact tool and schema regardless of what Claude might otherwise prefer.

Q51. Your application needs sub-second response times for a user-facing feature. Claude's average response time is 3-5 seconds. What architectural pattern addresses this?

A) Switch to claude-haiku — it responds in under 1 second always

B) Use speculative streaming: show a loading indicator, stream response tokens as they arrive starting immediately

C) Pre-generate responses for all possible user inputs and cache them

D) Reduce max_tokens to 50 to force faster responses

Correct Answer: B

Streaming starts showing tokens within ~1 second of request (first token latency). Combined with a good loading UX, this creates perceived sub-second responsiveness even if total generation takes 5 seconds. haiku is faster but quality may suffer.

Q52. You are implementing a multi-model pipeline where Claude extracts data, GPT validates it, and Claude generates the final report. What is the key data contract design concern?

A) Model APIs are incompatible — use only one provider per pipeline

B) Define a strict JSON schema for the handoff between models; validate at each boundary; never pass raw text between models

C) Always use Claude for all steps for consistency

D) Use XML instead of JSON for cross-model data exchange

Correct Answer: B

Cross-model pipelines require strict data contracts at boundaries. Raw text output from one model fed directly to another creates fragile pipelines. Define JSON schemas, validate at each step, and include explicit error fields for graceful degradation.

Q53. A production system must guarantee that Claude never reveals certain proprietary information in any response. What is the most reliable implementation?

A) Add 'never reveal [information]' to the system prompt — Claude follows instructions reliably

B) Use Anthropic's guardrails API to define topic restrictions at the platform level

C) Post-process all responses with a regex/NLP filter before returning to users

D) Use a combination: system prompt restriction + post-processing filter as defense in depth

Correct Answer: D

Defense in depth: system prompt instructions reduce probability of disclosure, but Claude is probabilistic and can fail. Post-processing filtering provides a deterministic safety net. Neither alone is sufficient for guaranteed compliance.

Q54. You need to implement a feedback loop where Claude evaluates its own output quality and flags low-confidence responses. What schema design enables this most effectively?

A) Add a confidence: number field (0-1) and a requires_human_review: boolean to the output schema

B) Make a second API call asking 'was that good?'

C) Use temperature=0 to maximize confidence

D) Check stop_reason for quality indicators

Correct Answer: A

Embedding confidence scoring directly in the output schema forces Claude to self-evaluate as part of the generation. A requires_human_review boolean enables automated routing. A separate 'was that good?' call adds latency and cost without structural guarantees.

Q55. What is token healing and why does it matter for prompt engineering?

A) Automatic correction of typos in prompts

B) Claude's tokenizer may split words across tokens; carefully placed spaces or prompt endings prevent partial-token artifacts at generation boundaries

C) A technique to compress prompts to reduce token count

D) Recovering from max_tokens truncation by continuing generation

Correct Answer: B

Tokenization artifacts occur when a prompt ends mid-token. Careful prompt design — ending on clean token boundaries, using appropriate spacing — produces more predictable output starts. This matters most for structured output prefilling.

Q56. You are implementing a Claude-powered API that serves external developers. Your system prompt contains proprietary business logic worth protecting. Can developers extract your system prompt via clever user messages?

A) No — system prompts are completely invisible to users

B) Yes — there is no API-level protection; instruct Claude to keep the system prompt confidential and add monitoring for extraction attempts

C) Yes — but only if they use the admin API endpoint

D) System prompts are encrypted and cannot be extracted

Correct Answer: B

System prompts are not technically protected from extraction via prompt injection attempts. Defensive measures include: instructing Claude to never reveal the system prompt, monitoring for extraction patterns, and keeping truly sensitive logic server-side (not in the prompt at all).

Q57. How should you handle the case where a tool result is too large to fit in the context window?

A) Return an error from the tool and stop the agentic loop

B) Truncate the tool result to fit, include a note that it was truncated, and let Claude decide if it needs the rest

C) Start a new session without the large tool result

D) Switch to claude-opus which has unlimited context

Correct Answer: B

Truncation with a clear note is better than silent failure. Include metadata like '[TRUNCATED: 150K chars of 2.3M total — request specific section if needed]'. Claude can then adapt its strategy — requesting targeted subsets rather than the full result.

Q58. You need to implement A/B testing for two different system prompts in production. What is the correct architectural approach?

A) Randomly assign users to prompt A or B at request time; log prompt_version, response_id, and outcome metrics together

B) Run both prompts in parallel for every request and pick the better response

C) Use the Batch API to run both variants and compare offline

D) Test prompts sequentially: one week on A, one week on B

Correct Answer: A

A/B testing Claude prompts requires: random user assignment, logging which variant served each request alongside the response ID, and tying to downstream outcome metrics. Running both in parallel doubles cost and adds latency — acceptable for offline eval but not production traffic.

Q59. A client.messages.create() call raises anthropic.APIStatusError with status_code=400. What does this indicate and what should you do?

A) Rate limit hit — retry with exponential backoff

B) Client error (invalid request) — do NOT retry; fix the request parameters

C) Server error — wait and retry

D) API key expired — regenerate the key

Correct Answer: B

400 errors are client-side validation failures: invalid parameters, malformed messages, exceeded limits in the request itself. Retrying the same request will always fail. Inspect the error body for details and fix the request.

Q60. You are building a multi-tenant SaaS where each customer has different Claude configuration (system prompts, tool sets, model versions). What is the most scalable architecture?

A) One hardcoded configuration for all customers

B) Store per-tenant configuration in a database; load and apply dynamically per request; cache configs with cache_control for efficiency

C) Create a separate Anthropic account per tenant

D) Embed all tenant configurations in a single massive system prompt

Correct Answer: B

Dynamic configuration loading scales to any number of tenants. Database-stored configs are easy to update without code deploys. Caching the system prompt (which is stable per-tenant) dramatically reduces per-request costs. One account handles all tenants.