Domain 4 — 20%

Prompt Engineering & Structured Output

Domain 4 — 20% of the CCA exam

IN THIS MODULE:

- Explicit criteria vs vague instructions
- Few-shot examples for edge cases
- tool_use JSON schemas for guaranteed structure
- Validation loops and retry-with-feedback
- Batch API for cost-optimized bulk processing

90 PRACTICE QUESTIONS — 30 BEGINNER · 30 INTERMEDIATE · 30 ADVANCED

QUESTION BANK OVERVIEW

Level	Questions	Focus
Beginner	Q1–Q30	Core concepts, terminology, and foundational patterns
Intermediate	Q31–Q60	Application scenarios, design decisions, trade-offs
Advanced	Q61–Q90	Production architecture, edge cases, system design

HOW TO USE THIS MODULE

Work through each level in order. For Beginner questions, aim for 90%+ before moving to Intermediate. For Intermediate, 80%+ before Advanced. Each question includes the correct answer and a full explanation — read the explanation even for questions you answered correctly to understand the underlying principle. The exam tests judgment, not memorization.

BEGINNER QUESTIONS

Questions 1–30

Q1. What distinguishes explicit criteria from vague instructions in prompt engineering?

A) Explicit criteria are longer; vague instructions are shorter

B) Explicit criteria define exactly what to report, what to skip, and concrete conditions for each — eliminating ambiguity. Vague instructions use subjective terms like 'be thorough' or 'be conservative'

C) Explicit criteria use bullet points; vague instructions use prose

D) Explicit criteria are for structured output; vague instructions are for conversational use

■ Correct Answer: B

Explicit: 'Flag bugs only when claimed behavior contradicts code behavior'. Vague: 'Check that comments are accurate'. The explicit version leaves no room for interpretation. Vague versions produce inconsistent results because 'accurate' means different things in different contexts.

Q2. Why does adding 'only report high-confidence findings' to a review prompt fail to improve precision?

A) Claude ignores confidence-based instructions

B) 'High confidence' is a subjective threshold that Claude calibrates differently each call — it doesn't map to a consistent precision level across runs

C) Confidence-based filtering requires extended thinking mode

D) Claude reports all findings regardless of confidence — instructions don't affect this

■ Correct Answer: B

Confidence-based filtering is inherently vague. Claude's internal confidence for the same finding may vary between calls. Explicit categorical criteria — 'report X only when Y condition is met' — produce consistent behavior because the condition is objective and testable.

Q3. When are few-shot examples most valuable in a prompt?

A) When you want Claude to follow instructions more carefully

B) When detailed instructions alone produce inconsistent output — examples show exactly what you want, including for ambiguous edge cases

C) Few-shot examples are only for beginners — expert prompts don't need them

D) When the task requires real-time web data

■ Correct Answer: B

Examples communicate transformation intent more effectively than prose for ambiguous tasks. When 'extract the date in ISO format' produces mixed results, 3 examples (input → output) consistently fix it. Examples are the most effective technique for output format consistency.

Q4. What makes a few-shot example most useful for a classification task?

A) Including as many examples as possible (50+)

B) Showing 2-4 examples that cover the most ambiguous cases — scenarios where reasonable people might disagree — with the correct classification and why

C) Including only positive examples of the desired behavior

D) Using simple, unambiguous examples so Claude learns the pattern easily

■ Correct Answer: B

Edge cases are where inconsistency lives. Claude handles clear-cut cases reliably. It's the borderline cases — 'is this a security issue or a code quality issue?' — where examples with reasoning matter most. Simple unambiguous examples teach what you already handle consistently.

Q5. What is the fundamental limitation of asking Claude to 'respond in JSON'?

A) Claude cannot generate JSON directly

B) JSON output requested via prompt is probabilistic — Claude may add prose, use markdown fences (```json), or produce invalid syntax, especially on edge cases

C) JSON responses are significantly slower than text responses

D) JSON output requires a special API endpoint

■ Correct Answer: B

Prompt-based JSON is unreliable in production. Claude may add explanatory text before the JSON, wrap it in markdown code blocks, or produce invalid JSON on complex nested structures. Tool use with a defined schema is the only guaranteed approach.

Q6. In a tool use JSON schema, when should a field be marked as nullable (type: ['string', 'null'])?

A) Never — all fields should be required strings

B) When the source document may not contain that information — nullable prevents Claude from hallucinating values to satisfy a required field

C) Only for optional metadata fields

D) When the field contains numbers that might be zero

■ Correct Answer: B

Non-nullable required fields without a null option force Claude to fabricate values when source data is missing. A required invoice_number on a document that doesn't have one → Claude invents a plausible-looking invoice number. Nullable fields signal 'it's OK to not have this'.

Q7. What is retry-with-error-feedback in extraction quality loops?

A) Automatically retrying failed API calls after a network error

B) When validation fails, appending the specific validation errors to the conversation and requesting correction — giving Claude concrete guidance on what to fix

C) Retrying with a more capable model after the first model fails

D) Resubmitting the same request without changes until it succeeds

■ Correct Answer: B

Generic retry (same request again) often produces the same error. Retry-with-feedback: append 'extraction failed with: line_items sum (95) != subtotal (100)' and ask for correction. Claude now knows exactly what to fix. This dramatically improves retry success rate for structured extraction tasks.

Q8. What types of errors can tool_use JSON schemas prevent?

A) All errors — tool use schemas guarantee perfect extraction

B) Syntax errors (malformed JSON, wrong types) — but NOT semantic errors like values that don't sum correctly or data in wrong fields

C) Only missing field errors

D) Only type mismatch errors

■ Correct Answer: B

Schema validation is syntactic: the output will be valid JSON with the right types and required fields. But semantics — line items that don't sum to the total, an amount in a date field — are not prevented by schema. Semantic validation requires additional logic in your code.

Q9. What is the Message Batches API's primary advantage?

A) Faster responses than the synchronous API

B) 50% cost reduction for latency-tolerant workloads processed within 24 hours

C) Support for longer context windows than synchronous calls

D) Automatic retry logic for failed requests

■ Correct Answer: B

Batch API pricing is half of synchronous pricing. The tradeoff is latency: results arrive within 24 hours with no guaranteed SLA. Use it for any workload that doesn't require real-time results: nightly reports, batch classification, weekly analysis.

Q10. What is a semantic validation error (as opposed to a schema/syntax error)?

A) An error in the meaning of the prompt

B) An error where the JSON structure is valid but values are logically inconsistent — like line items that don't sum to the stated total

C) An error that occurs when the wrong model is used

D) A validation error that requires human review

■ Correct Answer: B

Semantic errors are logically inconsistent values that pass schema validation. Schema says 'total is a number' — {total: 100, line_items: [50, 30]} is schema-valid but semantically wrong (50+30 ≠ 100). Detecting semantic errors requires validation logic beyond the schema.

Q11. What does the custom_id field in the Batch API enable?

A) Grouping related batch requests for parallel processing

B) Correlating batch request/response pairs — you set it on each request and it appears on the corresponding result for matching

C) Setting processing priority within a batch

D) Identifying the model version used for each request

■ Correct Answer: B

custom_id is your identifier for each item in the batch. When results come back, you use custom_id to match each result to its original request. Without it, you couldn't tell which result corresponds to which document in a 500-item batch.

Q12. Why is the Batch API inappropriate for a pre-merge code review that blocks PR deployment?

A) The Batch API doesn't support code review tasks

B) The Batch API has no guaranteed latency SLA — results may take up to 24 hours, which would block PRs for an unacceptably long time

C) Batch processing produces lower quality reviews than synchronous calls

D) Pre-merge reviews require streaming output which batch doesn't support

■ Correct Answer: B

A pre-merge check must complete in seconds or minutes to provide fast developer feedback. Batch API could take 24 hours. Blocking workflow actions require the synchronous API with guaranteed low latency.

Q13. What is a self-correction validation flow in structured extraction?

A) Claude automatically re-runs extraction until it produces valid output

B) Extracting both a calculated value and a stated value (e.g., calculated_total and stated_total), flagging discrepancies with a conflict_detected boolean for downstream handling

C) Asking a second model to validate the first model's extraction

D) Running the same extraction prompt 3 times and using majority voting

■ Correct Answer: B

Self-correction flows embed validation logic into the schema: extract both the source's stated value AND the value you can independently calculate. If they differ, flag it. This catches data entry errors in source documents as well as extraction errors.

Q14. When is retry-with-feedback NOT effective?

A) When the extraction has semantic errors

B) When the required information simply doesn't exist in the source document — retrying with better instructions won't create data that isn't there

C) When the first extraction was mostly correct

D) When using claude-haiku instead of claude-sonnet

■ Correct Answer: B

Retry fixes format/structure errors where Claude misunderstood how to extract something that IS present. If an invoice doesn't have a vendor name, no amount of retrying will produce one — the information isn't there. Recognize when to accept null vs. when to retry.

Q15. What does the detected_pattern field in a code review findings schema enable?

A) Pattern matching against known security vulnerabilities

B) Tracking which code patterns trigger findings so you can analyze false positive patterns when developers dismiss findings

C) Detecting the programming language and framework automatically

D) Identifying duplicate findings across multiple review runs

■ Correct Answer: B

By tracking which detected_pattern corresponds to each dismissed finding, you can identify systematic false positives: 'whenever Claude sees pattern X in context Y, it incorrectly flags it as issue Z.' This data drives prompt refinement.

Q16. What is the 'format normalization rules + strict schema' combination for?

A) Compressing JSON output for efficiency

B) Handling inconsistent source formatting: schema enforces output structure while normalization rules in the prompt handle varied inputs (e.g., date formats, currency symbols)

C) Converting between different JSON schema versions

D) Ensuring consistent field ordering in JSON output

■ Correct Answer: B

Source documents are inconsistent: dates as '1/15/24', 'Jan 15, 2024', '2024-01-15'. The schema says date must be YYYY-MM-DD. The normalization rule in the prompt says 'convert all dates to YYYY-MM-DD'. Both are needed: schema for structure, rules for source-side variation.

Q17. Why does a high false positive rate in one review category undermine trust in accurate categories?

A) False positives increase review latency, causing developers to skip all reviews

B) Developers learn to dismiss the review tool globally when they frequently encounter incorrect findings, even in categories where findings are accurate

C) False positives consume context window space needed for accurate findings

D) The review model penalizes accuracy when false positive rates are high

■ Correct Answer: B

Psychology of tool trust: when 'unused variable' produces 30% false positives, developers stop trusting the tool. They start dismissing 'security vulnerability' findings without reading them. One noisy category damages the credibility of all categories.

Q18. What is multi-instance review and why does it outperform self-review?

A) Running multiple review passes with the same model instance

B) Using a completely separate Claude instance (fresh context, no generation history) for review — it hasn't formed opinions about the code and catches issues the generating instance would rationalize away

C) Reviewing with multiple models (Claude + GPT + Gemini) and taking the consensus

D) Running the same review prompt multiple times and aggregating results

■ Correct Answer: B

Independent review instances provide structural objectivity. The generating instance 'knows' why it made each decision and rationalizes them. An independent instance sees only the code and judges it on its merits. This is analogous to why peer code review catches more bugs than self-review.

Q19. What does the enum approach with 'other' + detail field accomplish in JSON schemas?

A) It limits valid enum values to exactly what's listed

B) It provides extensible categorization: known values use the enum for reliable classification; novel values use 'other' + a detail string that preserves the specifics without schema violation

C) It prevents Claude from using values not in the enum list

D) It's a workaround for the 50-enum limit in JSON schemas

■ Correct Answer: B

Enums with 'other': 'category: urgent|normal|low|other' + 'category_detail: string'. When a document has a priority level not in your enum, it becomes 'other' with the actual value in category_detail. No data is lost, schema is never violated.

Q20. When should you use the Batch API vs. prompt caching for cost optimization?

A) They are interchangeable — use whichever is easier to implement

B) Use Batch API for latency-tolerant bulk processing (50% cost reduction). Use prompt caching for repeated calls with stable large prompts (up to 90% savings on the cached portion). They can be combined.

C) Batch API is always better — it saves more than caching

D) Prompt caching is deprecated in favor of the Batch API

■ Correct Answer: B

Different tools for different problems: Batch API reduces per-request cost for all content when latency doesn't matter. Prompt caching reduces cost of repeated large static content in real-time workloads. A nightly report pipeline could use both: batch for cost, caching for the repeated system prompt.

Q21. What is 'chain-of-thought prompting' and when should you use it?

A) A technique that chains multiple API calls together

B) Asking Claude to reason step-by-step before producing an answer — improves accuracy on multi-step reasoning tasks like math, logic, and complex analysis

C) Linking prompt templates together for complex workflows

D) A way to chain tool calls in a specific order

■ Correct Answer: B

Chain-of-thought: 'Think through this step by step' or 'Let's reason through this carefully.' For tasks requiring multi-step reasoning, this reduces errors by forcing Claude to work through intermediate steps explicitly rather than pattern-matching to an answer.

Q22. What is the role of negative examples in few-shot prompting?

A) Negative examples confuse Claude — use only positive examples

B) Negative examples show what NOT to report/do, teaching Claude to avoid specific failure modes and improving precision

C) Negative examples are only useful for classification tasks

D) Negative examples require a special 'negative' field in the prompt

■ Correct Answer: B

Negative examples teach discrimination: 'This code is NOT a security issue because [reason]' or 'This date format IS valid — do not flag it' show Claude the decision boundary. Without negative examples, Claude may over-flag (high false positive rate).

Q23. What makes a code review instruction 'actionable' vs. vague?

A) Actionable instructions are longer and more detailed

B) Actionable: specifies exact categories to check, what constitutes a finding in each category, and required output format. Vague: 'review for quality' or 'check for issues'

C) Actionable instructions use bullet points; vague use paragraphs

D) Actionable instructions are written in imperative form

■ Correct Answer: B

Actionable specificity: 'Report SQL injection vulnerabilities — code where user input is concatenated into SQL strings without parameterization. For each finding provide: file, line, vulnerable_pattern, fix_example.' vs 'Check for security issues.' The first leaves no ambiguity.

Q24. What is the 'temperature' parameter's effect on structured output tasks?

A) Temperature has no effect on structured output — use tool_use for that

B) Lower temperature (0.0-0.3) produces more consistent, reproducible structured outputs; higher temperature introduces more variety but can cause schema deviations on complex structures

C) Higher temperature produces better structured output by allowing more creativity

D) Temperature only affects the speed of generation, not the content

■ Correct Answer: B

For structured extraction: temperature=0 to 0.3 maximizes consistency. For creative generation: 0.7-1.0 adds variety. Even at temperature=0, structured output via tool_use is more reliable than relying on temperature alone for schema compliance.

Q25. What does it mean to 'specify the output audience' in a prompt?

A) Specifying which team member will receive the output

B) Telling Claude who will read and use the output — this shapes vocabulary level, assumed background knowledge, and appropriate detail depth

C) A legal disclaimer requirement for AI-generated content

D) Specifying the file format of the output

■ Correct Answer: B

Audience specification changes output: 'Explain for a non-technical executive' vs 'Explain for a senior infrastructure engineer' produces fundamentally different responses to the same question. This is one of the most high-impact, low-effort prompt improvements.

Q26. What is 'output length calibration' in prompt engineering?

A) Setting max_tokens to limit response length

B) Specifying the appropriate response length in the prompt — preventing both verbosity (unnecessary filler) and truncation (missing important details)

C) Post-processing to trim responses to a target length

D) A model parameter that controls output verbosity

■ Correct Answer: B

Length calibration: 'Provide a 3-bullet summary' vs 'Provide a detailed analysis.' Without specification, Claude uses its own judgment — sometimes producing essays when bullets were needed, or bullets when analysis was needed. Explicit length guidance produces appropriately-sized outputs.

Q27. What is the 'system prompt as persona' technique and what are its benefits?

A) Using the system prompt to make Claude pretend to be a different AI

B) Giving Claude a consistent role/persona (e.g., 'You are a senior security engineer') that shapes its perspective, vocabulary, and response style for all turns in the conversation

C) A technique for creating chatbot characters for entertainment

D) Assigning a system prompt ID for tracking purposes

■ Correct Answer: B

Persona-based system prompts produce consistent role-appropriate responses: 'You are a senior security engineer reviewing code for production deployment' produces security-focused, technical, direct responses. The persona carries through the entire conversation without re-stating context on each turn.

Q28. Why should you define 'what not to include' in extraction prompts?

A) Negative constraints reduce output length, saving tokens

B) Without boundaries, Claude may include neighboring context, summaries, interpretations, or formatting that interferes with your parsing logic

C) Negative constraints are processed faster than positive instructions

D) This is a defensive prompt technique against adversarial inputs

■ Correct Answer: B

Explicit exclusions prevent extraction noise: 'Do not include surrounding sentence context, units of measurement, currency symbols, or parenthetical notes — extract only the numeric value.' Without this, Claude's helpful instinct to provide context can break downstream parsers expecting clean values.

Q29. What is 'conditional instruction' in prompt engineering?

A) Instructions that only apply to certain Claude models

B) If-then instructions that cover branching scenarios: 'If the document has a date, extract it in ISO format. If no date is present, return null. If multiple dates are present, return the most recent.'

C) Instructions that are activated by specific trigger words in user input

D) Nested prompts that call sub-prompts based on the first response

■ Correct Answer: B

Conditional instructions handle branching cases that invariably arise in real data. Without conditional guidance, Claude has to decide how to handle edge cases itself — leading to inconsistent behavior across different inputs.

Q30. What does the 'role-task-output format' prompt structure provide?

A) A way to assign multiple roles to Claude in one call

B) A systematic prompt organization: (1) role sets context/expertise, (2) task describes what to do, (3) output format specifies exactly how results should be structured — producing consistent, appropriately-scoped outputs

C) A template format required by the Anthropic API

D) A three-turn conversation pattern for complex tasks

■ Correct Answer: B

Role-task-output: 'You are a data quality analyst (role). Review the following CSV for data quality issues (task). Return a JSON array of findings, each with: row_number, column, issue_type, description (output format).' This structure produces predictable, well-scoped outputs every time.

INTERMEDIATE QUESTIONS

Questions 31–60

Q31. You are building a multi-document extraction pipeline. Documents have varied formats: some have inline citations, others have bibliographies, some embed methodology in footnotes. How do you handle this in few-shot examples?

A) Create a separate extraction prompt for each document format

B) Include few-shot examples demonstrating correct extraction from each format variation, showing Claude how to locate data in each structure

C) Preprocess all documents to a standard format before extraction

D) Use a higher temperature for more flexible interpretation

■ Correct Answer: B

Format-variant few-shot examples: show extraction from 'inline citations' format, 'bibliography' format, and 'footnote methodology' format. Claude learns to recognize and correctly handle each variant. Without these examples, format variations produce missed or wrong extractions.

Q32. A structured extraction validates that line items sum to the stated total. The validation fails. How do you determine if the issue is extraction error or source document error?

A) Always assume extraction error — fix the prompt

B) Include both calculated_total (sum of line_items) and stated_total (from the document) in the schema; if they differ, flag conflict_detected: true and include both values for human review

C) Retry until both values match

D) Use the stated_total — it's always more reliable than the calculated value

■ Correct Answer: B

Conflict_detected: true with both values enables downstream triage: if the document itself has inconsistent numbers (a data entry error in the source), retrying won't help — this needs human review. If it's an extraction error, retry-with-feedback may fix it. The two-value approach distinguishes these cases.

Q33. What is the optimal few-shot example count for a code review task?

A) 1 — more examples consume too many tokens

B) 2-4 carefully chosen examples covering the most common ambiguous cases (e.g., an issue that IS a bug, one that looks like a bug but isn't, one security issue, one acceptable pattern)

C) 20+ examples for maximum coverage

D) No examples — explicit criteria are sufficient

■ Correct Answer: B

2-4 examples is the sweet spot: enough to demonstrate judgment on ambiguous cases without excessive token cost. More than 4 often adds diminishing returns. The examples' quality matters more than quantity — focus on the cases where Claude currently makes mistakes.

Q34. You are designing a batch invoice processing pipeline. Invoices range from 1-50 pages. Some exceed the context limit. How do you handle the Batch API's lack of multi-turn tool calling?

A) Exclude long invoices — batch only supports short documents

B) Pre-process to chunk large invoices; submit each chunk as a separate batch item with custom_id reflecting the source document and chunk number; merge results post-processing

C) Use the synchronous API for long invoices and batch for short ones

D) Compress invoices to fit in the context limit before batching

■ Correct Answer: B

Batch API limitation: no multi-turn within a single request. For long documents: chunk pre-processing, chunk-level batch items with IDs like 'invoice-47-chunk-3', post-processing merge. This adapts to the constraint without abandoning batch savings.

Q35. Your code review system currently reports 45% false positives for 'style issues'. Developers are ignoring all reviews. What is the prioritized fix sequence?

A) Switch to a more accurate model immediately

B) (1) Temporarily disable the style category entirely to restore developer trust. (2) Analyze dismissed findings to identify specific false positive patterns. (3) Rewrite style criteria with explicit 'do not report' examples. (4) Re-enable with refined criteria.

C) Add 'only report critical style issues' to the prompt

D) Increase temperature to produce more varied findings

■ Correct Answer: B

Trust restoration first: disabled category stops immediate harm. Then analyze patterns using detected_pattern data. Then rewrite with explicit criteria and 'do not report' examples for each identified false positive pattern. Re-enable only after validation on test PRs.

Q36. You need to extract financial data from documents written in English, Spanish, and German, each with different number formatting (1,000.00 vs 1.000,00). How does the prompt handle this?

A) Create separate prompts for each language

B) Include format normalization rules in the prompt: 'All monetary amounts must be converted to decimal notation (1234.56) regardless of source formatting' plus examples showing each format's conversion

C) Use a translation step before extraction

D) Use locale: auto in the JSON schema

■ Correct Answer: B

Normalization rules + examples: rule states the target format, examples show how to get there from each source format. '1.000,00 EUR → 1000.00', '1,000.00 USD → 1000.00'. Without examples, Claude may normalize inconsistently across calls.

Q37. When building a validation loop, how many retries are typically appropriate before escalating to human review?

A) Always retry until success

B) 1-2 retries; beyond that, the error is likely unfixable with the same approach (absent information, fundamental format issue) — escalate to human review

C) 5-10 retries to maximize automatic resolution

D) No retries — escalate immediately on first failure

■ Correct Answer: B

Diminishing returns: the first retry often fixes format/structure issues. A second retry with refined instructions handles remaining edge cases. Beyond 2 retries, you're likely facing absent information or a fundamental issue that more retries won't resolve — human review is more efficient.

Q38. You are building a batch pipeline and want to refine the prompt before processing 10,000 documents. What is the optimal approach?

A) Process all 10,000 and fix errors in post-processing

B) Run prompt refinement on a representative sample set (50-100 documents) covering format and content variations; validate precision and recall; only then batch-process the full set

C) Run on 100 documents, fix errors, run 900 more, fix errors — iterate until all 10,000 are done

D) Use the synchronous API for the first 1,000 before switching to batch

■ Correct Answer: B

Front-loaded validation: sample-based refinement on 50-100 representative documents catches prompt issues before they affect 10,000 items. The cost of fixing issues post-batch (re-submission, merging corrections) far exceeds the cost of pre-batch validation.

Q39. A PR review tool reports 60 issues. Developers say most are false positives. You want to identify which categories are causing the problem. What schema field helps most?

A) A severity field (critical/high/medium/low)

B) A detected_pattern field that captures which specific code construct triggered the finding, enabling analysis of which patterns are being over-flagged

C) A confidence field (0-1) that developers can filter on

D) A category field (bug/security/style)

■ Correct Answer: B

detected_pattern provides diagnostic power: if 60% of dismissed findings have detected_pattern='trailing comma in object literal', that's a specific false positive pattern you can fix with a targeted prompt update. Category alone doesn't tell you WHICH specific pattern is wrong.

Q40. How should you structure few-shot examples for a task that requires distinguishing between acceptable code patterns (project convention) and genuine bugs?

A) Only include examples of bugs — show Claude what's wrong

B) Include paired examples: one that LOOKS like a bug but IS an acceptable project convention (with reasoning), and one that IS a genuine bug (with reasoning). This teaches discrimination between the two.

C) Include only acceptable patterns so Claude knows what not to flag

D) Use generic examples not specific to your codebase

■ Correct Answer: B

Contrastive examples teach judgment: 'This global variable looks like an anti-pattern BUT is intentional for performance in this module — do not flag' vs 'This global variable IS a bug because it causes race conditions'. The paired contrast is more informative than either example alone.

Q41. What is the correct approach for calculating the optimal batch submission frequency given: document SLA = 48 hours, batch processing max = 24 hours?

A) Submit every 48 hours

B) Submit every 24 hours — worst case: submitted at start of window, processed in 24 hours = delivered within 48-hour SLA with buffer

C) Submit every 12 hours for a 12-hour safety buffer

D) Use real-time API instead — batch doesn't meet the SLA

■ Correct Answer: B

SLA math: maximum batch processing (24h) + submission-to-processing delay must be <= SLA (48h). Submitting every 24h means worst case: item arrives just after submission cutoff → waits 24h for next batch → 24h processing = 48h total. Exactly meets SLA.

Q42. You have a code review prompt where false positives occur specifically when Claude encounters 'async/await in a loop'. How do you fix this?

A) Remove async/await detection from the prompt entirely

B) Add an explicit 'do not flag' rule with a concrete example: 'Sequential async/await in a loop is acceptable when operations must be ordered — example: [code example]. Only flag when the loop can be parallelized and the delay matters.'

C) Add 'ignore performance issues' to the prompt

D) Increase Claude's context window so it understands the full loop context

■ Correct Answer: B

Pattern-specific 'do not flag' rules with examples are the precise fix. The rule tells Claude when this pattern is acceptable; the example shows it concretely. Generic 'ignore performance issues' would suppress valid performance findings too.

Q43. A batch job runs nightly with 1,000 documents. Some nights 50 documents fail with 'context length exceeded'. What is the efficient handling strategy?

A) Increase max_tokens and resubmit all 1,000 documents

B) Identify failed items by custom_id; chunk those specific documents and resubmit only the chunks as new batch items; merge the chunked results in post-processing

C) Skip oversized documents — they are too complex for batch processing

D) Switch to synchronous API for all documents when any fail

■ Correct Answer: B

Targeted resubmission: use custom_id to identify the 50 failures, chunk only those documents (not all 1,000), resubmit as a smaller batch. This is efficient: 950 documents succeeded — don't reprocess them. Only fix what failed.

Q44. What is the 'Best of N' pattern in prompt engineering for structured output?

A) Generating N responses and taking the first valid one

B) Generating N independent responses to the same prompt and selecting the best using a scoring function — trading cost (N API calls) for quality (best result wins)

C) Running the prompt N times and taking the majority vote

D) Splitting a prompt into N sections and running each separately

■ Correct Answer: B

Best of N: generate 3-5 responses, score each (automated validation or a second model evaluating quality), return the best scoring one. For high-stakes tasks (legal document analysis, medical data extraction), the quality improvement justifies the N× cost.

Q45. How should you handle a JSON schema field that could validly be any of three types: a date string, a boolean, or a number?

A) Use anyOf: [{type: 'string'}, {type: 'boolean'}, {type: 'number'}]

B) Use type: 'string' and document that Claude should serialize the value as a string regardless of semantic type — then parse the string in your application code

C) Create separate schemas for each document type

D) Use type: 'object' with a value field

■ Correct Answer: A

anyOf in JSON schema supports type unions. The schema validator will accept any of the listed types. Claude will choose the most semantically appropriate type based on the source data. This is cleaner than forcing all values to strings.

Q46. You need to validate that an extracted address is a real location without making API calls. What prompt engineering technique provides the best validation?

A) Ask Claude to validate the address after extraction in a separate call

B) Include a plausibility_score field in the schema (0-1) and an extraction_confidence field; include a few-shot example showing what a suspicious vs. normal address extraction looks like

C) Use regex to validate address format after extraction

D) Require extraction confidence > 0.9 before accepting

■ Correct Answer: B

Embedding plausibility assessment in the schema leverages Claude's world knowledge: it can recognize that '999 Mars Boulevard, Jupiter, USA' is implausible and flag low plausibility. Combined with examples showing suspicious vs. normal cases, this provides lightweight validation without external API calls.

Q47. Your batch pipeline processes insurance claims. Some claims have no payout amount because they were denied. Your current schema requires payout_amount. How should you fix the schema?

A) Set payout_amount minimum to 0 to allow zero values

B) Make payout_amount nullable (type: ['number', 'null']) and add a claim_status field (enum: ['approved', 'denied', 'pending']) — null payout is valid for denied claims

C) Remove payout_amount from required fields but keep it non-nullable

D) Use a separate schema for denied claims

■ Correct Answer: B

Semantic validity through schema design: denied claims legitimately have no payout. Nullable + claim_status captures this correctly. Claude won't fabricate a payout amount for denied claims when null is explicitly valid. Adding claim_status provides context for why payout is null.

Q48. What is prompt chaining and when does it outperform a single complex prompt?

A) Linking multiple Claude sessions together

B) Breaking complex tasks into sequential steps where each step's output feeds the next — outperforms single prompts when the task has distinct phases that benefit from focused attention at each step

C) Using multiple models in a pipeline

D) Caching a sequence of prompts for reuse

■ Correct Answer: B

Prompt chaining works because focused attention improves quality. 'Analyze security vulnerabilities' → 'Given these vulnerabilities, assess severity' → 'Given severity assessment, write remediation guidance' outperforms a single 'analyze and provide remediation' prompt because each step gets Claude's full focus.

Q49. You are refining a code review prompt and notice false positives specifically when Claude sees 'for..of loops with await inside'. How do you add the fix?

A) Add 'ignore all for loops' to the prompt

B) Add an explicit negative example to few-shot: 'This code: [for..of with await example] is ACCEPTABLE when operations must be sequential. Only flag if execution could be parallelized and performance matters.'

C) Remove performance review from the prompt scope

D) Add 'await in loops is acceptable' as a general rule

■ Correct Answer: B

Negative few-shot examples with the specific pattern and reasoning are the most precise fix. They teach discrimination: 'this specific pattern in this specific context is acceptable'. General rules ('await in loops is fine') would suppress valid performance issues too.

Q50. How should you design a system that uses the Batch API for primary processing but needs to handle urgent requests that arrive mid-batch?

A) Cancel the current batch and restart with the urgent request included

B) Maintain a dual-track system: synchronous API for urgent requests (real-time), batch API for standard requests (cost-optimized). Route based on urgency flag at submission time.

C) Process all requests in batch — urgency can wait for the next batch window

D) Use streaming for urgent requests within the batch API

■ Correct Answer: B

Dual-track architecture: urgent = synchronous (higher cost, immediate), standard = batch (lower cost, latency-tolerant). This is the standard pattern for mixed-SLA workloads. The routing logic (urgency flag) decides the path at submission time.

Q51. You are building a prompt for a customer sentiment classification system. The current prompt produces 25% 'neutral' classifications for what developers believe are clearly positive or negative. What is the most likely issue?

A) The model needs more training data on neutral sentiment

B) The 'neutral' category boundaries are undefined — add explicit criteria: 'Classify as neutral ONLY when the customer expresses no evaluative language. If any positive or negative modifier exists, classify accordingly, even if mixed.'

C) Reduce the temperature to force more decisive classifications

D) Switch to a 5-category scale to capture more nuance

■ Correct Answer: B

Category boundary definition: 'neutral' is over-used when its definition is vague. Explicit boundaries with negative examples ('do NOT classify as neutral if any evaluative language exists') dramatically reduce ambiguous classifications. The criterion 'no evaluative language' is objective and testable.

Q52. A document extraction system must handle tables with varying structures. Some tables have headers, some don't. Some have merged cells, some have hierarchical headers. How do you handle this in few-shot examples?

A) Include only well-formatted table examples — complex tables are edge cases

B) Include examples of each table variant: header/no-header, merged cells, hierarchical headers — each example shows the input table format and the correct extraction output

C) Pre-process all tables to a standard format before extraction

D) Add 'handle all table formats' to the system prompt

■ Correct Answer: B

Format coverage in examples: each structural variant gets its own example. Without an example for 'no-header tables', Claude may invent headers or fail extraction. Each example teaches one structural pattern. Three examples covering headers, merged cells, and hierarchical headers handles 95% of real-world table variance.

Q53. You want to add a 'self-review' step to your extraction pipeline where Claude checks its own output for consistency before returning it. What is the most effective implementation?

A) Ask Claude to 'review your output' in the same API call

B) Include consistency check fields in the extraction schema: {extracted_data: {...}, consistency_checks: {total_matches_line_items: bool, dates_in_valid_range: bool, required_fields_present: bool}, confidence: number}

C) Make a second API call asking Claude to validate the first call's output

D) Use extended thinking mode which automatically self-reviews

■ Correct Answer: B

Schema-embedded consistency checks force Claude to verify its own work during extraction. consistency_checks.total_matches_line_items: false immediately flags a mathematical inconsistency. This is more efficient than a second API call and catches errors at generation time rather than after.

Q54. Your code review system needs to evaluate 'is this a security vulnerability?' for each finding. What few-shot example design produces the most reliable security classification?

A) Include 20+ diverse security examples to maximize coverage

B) Include paired examples: one code pattern that IS a security vulnerability (with explanation of the attack vector), and one similar-looking pattern that is NOT (with explanation of why it's safe). 3-4 pairs covering the most commonly confused categories.

C) Include only confirmed vulnerabilities — false examples confuse the model

D) Use a binary severity (vuln/not-vuln) without examples

■ Correct Answer: B

Contrastive pairs teach discrimination: 'SELECT * FROM users WHERE id=' + input IS SQL injection (string concatenation, no parameterization). 'SELECT * FROM users WHERE id = ?' with input as parameter is NOT SQL injection. The contrast makes the distinction concrete and actionable.

Q55. You are designing a batch pipeline that extracts data from PDFs in multiple languages. Extraction quality is inconsistent for Japanese documents. What improvement strategy is most effective?

A) Add a translation step to convert Japanese to English before extraction

B) Add Japanese-language few-shot examples to the extraction prompt showing correct field extraction from Japanese invoice formats, including how Japanese date and number formats map to the output schema

C) Use a lower temperature for Japanese documents

D) Flag Japanese documents for manual review only

■ Correct Answer: B

Language-specific examples: Japanese invoices use different date formats (YYYY■MM■DD■), number formats, and layout conventions. Few-shot examples showing Japanese input → standardized JSON output teach Claude the mapping. Translation adds complexity and may lose numeric precision.

Q56. How should you structure a code review prompt to ensure coverage of both security and performance issues without trading one off for the other?

A) Use a single unified prompt that covers all review types

B) Define separate explicit criteria sections for each review dimension: SECURITY: [specific security criteria], PERFORMANCE: [specific performance criteria]. Each finding must be tagged with its dimension. Results show both dimensions independently.

C) Run security review first, then performance in a separate call

D) Ask Claude to 'balance security and performance considerations'

■ Correct Answer: B

Dimension-separated criteria prevent trade-offs: with unified criteria, Claude may unconsciously prioritize one dimension. Explicit separate sections ensure both dimensions get equal attention. Tagged findings enable downstream filtering (show only security findings to the security team).

Q57. You are designing a prompt for extracting medical dosage information from clinical notes. Dosages appear in many formats: '10mg', '10 milligrams', 'ten mg', '0.01g'. How do you handle normalization most reliably?

A) Let Claude normalize whatever format it encounters — it understands units

B) Provide explicit normalization rules: 'Convert all dosages to milligrams (numeric only). Conversion: 1g = 1000mg. Write out numbers: ten = 10. Return: {drug: string, dose_mg: number}' plus examples for each format variant.

C) Extract raw text and normalize with a post-processing library

D) Use separate extraction prompts for each format variant

■ Correct Answer: B

Explicit normalization rules with examples: the rules define the transformation, examples show it in action for each variant. This is more reliable than relying on Claude's unit conversion knowledge alone. Medical context makes errors costly — explicit rules reduce ambiguity.

Q58. How do you prompt Claude to produce a structured output that includes both extracted data AND a confidence assessment for the entire extraction?

A) Make a second API call asking Claude to rate its previous extraction

B) Include confidence fields in the initial extraction schema: {extracted_data: {...}, extraction_confidence: 0-1, low_confidence_fields: ['field1', 'field2'], extraction_notes: 'string for unusual situations'}

C) Use extended thinking mode — it provides confidence automatically

D) Confidence assessment is not possible in single-call extractions

■ Correct Answer: B

Integrated confidence: forcing Claude to assess confidence as part of extraction produces better-calibrated scores than a separate evaluation call. low_confidence_fields enables targeted human review of specific fields. extraction_notes captures unusual situations that affect confidence globally.

Q59. You are building an invoice extraction system and notice that Claude extracts correct values on 95% of invoices but fails on those with non-standard layouts (tabular data in headers, multi-column line items). How do you address this?

A) Accept 95% accuracy — non-standard invoices are rare

B) Add few-shot examples specifically for non-standard layouts; include a layout_detection field in the schema (standard/tabular-header/multi-column) that Claude fills in before extraction; use the detected layout to route to the appropriate extraction approach

C) Pre-process all invoices to a standard layout before extraction

D) Use a different model for non-standard layouts

■ Correct Answer: B

Layout-aware extraction: detecting layout first enables appropriate extraction strategy. Few-shot examples for non-standard layouts teach Claude the correct patterns. The layout field also enables monitoring: if 20% of invoices are 'multi-column', you know which layout type is generating failures.

Q60. How do you design a prompt for summarization that produces consistent length across inputs of varying length (100-word memos to 50-page reports)?

A) Set a fixed max_tokens — length will naturally vary with input

B) Specify proportional length: 'Produce a summary of approximately 10% of the original length, with a minimum of 50 words and maximum of 500 words. For very short inputs (< 100 words), a 2-3 sentence summary is sufficient.'

C) Summarize all inputs to exactly 100 words

D) Let Claude determine the appropriate length based on content density

■ Correct Answer: B

Proportional + bounded length: 10% with min/max handles the full range. A 100-word memo → 50 words (minimum). A 5,000-word report → 500 words (maximum). Special handling for very short inputs prevents over-summarization. Fixed length (option C) is too rigid for variable-length inputs.

ADVANCED QUESTIONS

Questions 61–90

Q61. You are designing a high-accuracy medical document extraction system. What combination of techniques maximizes reliability?

A) Use claude-opus with high temperature for creative interpretation

B) Tool use with nullable fields + retry-with-feedback loop (max 2) + confidence scoring schema + human review queue for low-confidence extractions + few-shot examples for medical terminology variants

C) Multiple models in ensemble with majority voting

D) Strict required fields (no nullables) to force complete extraction

■ Correct Answer: B

Medical extraction defense in depth: (1) tool use ensures schema compliance, (2) nullable fields prevent hallucination of absent data, (3) retry-with-feedback handles correctable errors, (4) confidence scoring routes uncertain extractions to human review, (5) domain-specific few-shot handles medical abbreviations and format variations.

Q62. You need to extract data from 50,000 legal contracts with three extraction schemas (depending on contract type: NDA, SLA, Purchase Agreement). What is the optimal prompt engineering architecture?

A) One universal schema for all contract types

B) Contract type classifier → route to type-specific extraction schema. Each schema uses tool_choice forced selection + type-specific few-shot examples + type-specific validation logic. Batch process by contract type for prompt cache efficiency.

C) Three separate batch jobs running simultaneously

D) One schema with all possible fields, most nullable

■ Correct Answer: B

Type-specific pipelines: classifier determines contract type (can itself be batched), then type-specific extraction with optimized schema and examples. Batching by type maximizes prompt cache reuse (same system prompt per type). Universal schemas are too generic and produce lower accuracy.

Q63. A production extraction system has 2% unexplained failures where validation fails for no apparent reason. Debugging reveals the issue occurs only with documents that contain tables with merged cells. How do you address this?

A) Add 'handle tables with merged cells' to the prompt

B) Add a targeted few-shot example showing correct extraction from a merged-cell table, with a note in the system prompt: 'For tables with merged cells, treat the merged cell value as applying to all spanned rows/columns'

C) Pre-process documents to convert merged cells to repeated values before extraction

D) Flag all documents with tables for human review

■ Correct Answer: B

Root cause + targeted fix: the specific failure pattern (merged cells) gets a specific solution (example + rule). The example shows the visual structure and the correct extraction. Pre-processing (option C) also works but adds pipeline complexity; prompt-based handling is simpler and maintains the extraction in one step.

Q64. You are building a prompt optimization feedback loop. Developer dismissals of review findings are logged with the finding context. After 3 months, you have 50,000 dismissal records. How do you use this data most effectively?

A) Delete all findings that were dismissed more than once

B) Cluster dismissals by detected_pattern; identify top-10 false positive patterns by frequency; create targeted 'do not report' examples for each; A/B test prompt variants before deploying

C) Use the data to fine-tune a custom Claude model

D) Lower confidence threshold for all findings

■ Correct Answer: B

Data-driven prompt improvement: clustering by detected_pattern reveals systematic issues. Top-10 patterns represent 80% of false positives. Targeted examples fix specific patterns without over-generalizing. A/B testing validates improvement before full deployment. This is the systematic prompt engineering improvement cycle.

Q65. You are designing a structured extraction system for financial filings that must achieve 99.9% accuracy on the primary fields. What quality assurance architecture achieves this?

A) A single extraction call with high-quality few-shot examples

B) Multi-stage: (1) extraction with confidence scoring, (2) independent cross-validation extraction, (3) comparison for conflicts, (4) human review queue for conflicts or low-confidence, (5) ongoing false-positive/negative tracking

C) Run the same extraction 10 times and take the majority vote

D) Use claude-opus exclusively — it has 99.9% accuracy

■ Correct Answer: B

99.9% accuracy requires systematic quality assurance, not just a better prompt. Independent cross-validation catches different errors than single extraction. Conflict detection routes genuinely uncertain cases to humans. Ongoing tracking enables continuous improvement. No single model achieves 99.9% without QA architecture.

Q66. How do you design a system where the Batch API handles primary processing but some batch failures should be retried synchronously with richer context?

A) All failures should use the same retry strategy

B) After batch processing: categorize failures by error type. For transient errors (context limits, timeouts): synchronous retry with document chunking and richer prompt context. For semantic validation failures: synchronous retry with specific error feedback. For missing-data failures: route to human review (retrying won't help).

C) Re-submit all failures to a new batch

D) Log failures and process them in the next day's batch

■ Correct Answer: B

Error-type-specific retry strategy: each failure type needs a different response. Transient errors need a different modality (synchronous, chunked). Semantic errors need feedback-enhanced retry. Missing data errors cannot be fixed by any retry — human intervention is needed. One-size-fits-all retry wastes resources and misses the root cause.

Q67. You need to build a self-improving extraction system. When human reviewers correct extraction errors, how should these corrections be incorporated?

A) Manually rewrite the prompt after each correction

B) Store corrections with original input + error + correction as structured examples; periodically analyze patterns; add high-frequency patterns as few-shot negative examples; validate improvement before deploying prompt updates

C) Fine-tune a custom Claude model on correction data

D) Use the corrections as batch API training data

■ Correct Answer: B

Continuous improvement loop: corrections → structured examples → pattern analysis → targeted prompt updates (few-shot negative examples for common errors) → validation. This is prompt-level learning without fine-tuning. Fine-tuning is expensive and slower to iterate than prompt updates.

Q68. A complex extraction task requires understanding document structure, entity resolution, cross-reference validation, and output generation. Single-prompt performance is inconsistent. What architecture solves this?

A) Use extended thinking mode for better reasoning

B) Decompose into a prompt chain: (1) document structure analysis, (2) entity extraction with structure context, (3) cross-reference validation with entities, (4) output generation with validated entities. Each stage receives focused input from the prior stage.

C) Increase temperature for more creative interpretation

D) Use a single larger prompt with all context simultaneously

■ Correct Answer: B

Complex multi-aspect tasks benefit from decomposition: each stage applies focused attention to one concern. Structure analysis informs entity extraction; entities inform validation; validated entities inform output. Single prompts trying to do all four simultaneously show attention dilution.

Q69. You are building a code review system where the same PR should receive different review criteria based on which team owns the changed files (payment team → PCI compliance, auth team → security hardening, data team → privacy). How do you implement this?

A) One universal review prompt for all file changes

B) Detect file ownership from CODEOWNERS; select team-specific review criteria; inject appropriate few-shot examples for that team's standards; force the extraction schema fields relevant to that team's compliance requirements

C) Create separate CI jobs per team

D) Let the reviewer decide which criteria to apply based on the PR description

■ Correct Answer: B

Dynamic context injection based on ownership: CODEOWNERS-based routing selects the right criteria and examples per team. Payment files get PCI-specific security checks and PCI schema fields. Auth files get auth-specific examples. This is targeted review without the overhead of separate CI jobs for each team.

Q70. Your extraction pipeline processes documents in real-time for a user-facing feature and in batch overnight for analytics. The same extraction logic must be used in both modes. How do you architect this?

A) Duplicate the extraction logic — real-time and batch have different requirements

B) Abstract the extraction function to be mode-agnostic (same prompt, same schema, same validation); build a thin routing layer that wraps it in synchronous API calls for real-time and Batch API submission for overnight runs

C) Use different models for real-time (haiku for speed) vs. batch (opus for quality)

D) The Batch API format is incompatible with synchronous extraction — they must be separate

■ Correct Answer: B

DRY extraction architecture: the core extraction logic (prompt + schema + validation) is shared. A routing layer decides the API modality (synchronous or batch). This ensures consistency between real-time and analytics results — same logic, different delivery mechanism.

Q71. When designing a prompt for extracting information from handwritten forms (OCR'd to text), what additional few-shot techniques improve accuracy?

A) The same techniques as for printed documents — OCR doesn't change the approach

B) Add few-shot examples specifically showing OCR artifacts (misspellings, character substitutions, run-together words) and how to correctly interpret them; add a transcription_notes field for ambiguous characters

C) Pre-process OCR text with a spell-checker before extraction

D) Use a lower confidence threshold for handwritten forms

■ Correct Answer: B

OCR-specific examples: handwriting OCR produces characteristic errors ('rn' misread as 'm', '0' vs 'O', etc.). Examples showing these artifacts and correct interpretations teach Claude to handle them. transcription_notes: 'Amount field had illegible digit, estimated from context' preserves uncertainty explicitly.

Q72. You need to batch-process 10,000 invoices. After testing, 500 invoices reliably exceed the context limit. What is the cost-optimal resubmission strategy?

A) Use the synchronous API for all 10,000 to handle the 500 edge cases

B) Submit 9,500 normal invoices to batch (50% savings). For the 500 large invoices: chunk each into sections, create chunk-level batch items (still 50% savings), post-process to merge chunk results.

C) Skip the 500 oversized invoices

D) Use a larger model with bigger context for all 10,000

■ Correct Answer: B

Maximize batch usage: 9,500 invoices go directly to batch. The 500 oversized invoices get chunked — each chunk is a separate batch item. The entire pipeline uses batch pricing (50% savings) rather than falling back to synchronous for edge cases. Chunking enables batch for all items.

Q73. A prompt produces correct output 95% of the time but fails unpredictably on the remaining 5%. You've collected 50 failure examples. What is the systematic improvement process?

A) Add 'be more careful' to the system prompt

B) Cluster the 50 failures by root cause; identify the top 3 failure patterns; create targeted few-shot negative examples for each; A/B test the improved prompt against the original before deploying

C) Switch to a more capable model to reduce the failure rate

D) Increase max_tokens to give Claude more room to think

■ Correct Answer: B

Systematic failure analysis: clustering reveals patterns (e.g., 60% of failures are on 'merged cell tables', 30% on 'multi-currency amounts'). Targeted fixes for each cluster using few-shot examples addresses the actual failure modes. A/B testing before deployment prevents regressions.

Q74. What is the most reliable way to ensure that required fields in a JSON extraction schema are only marked as required when you are certain the source documents always contain them?

A) Mark all fields as required — missing data indicates extraction failure

B) Analyze a representative sample of source documents before finalizing the schema; only mark a field as required if it appears in 99%+ of sampled documents; make all other fields nullable

C) Start with all fields required and remove from required when extraction fails

D) Never use required — make all fields optional

■ Correct Answer: B

Evidence-based schema design: sampling validates assumptions about field presence. A field that appears in 99% of documents can be required (1% missing is likely extraction failure). A field present in 60% should be nullable (40% legitimately missing). This prevents hallucination of absent data.

Q75. You are building a knowledge extraction pipeline for a legal firm. Contracts contain complex clause structures: main clauses, sub-clauses, and exceptions to sub-clauses. Standard extraction produces incorrect nesting 30% of the time. How do you solve this systematically?

A) Extract all clauses as a flat list and post-process the hierarchy

B) Multi-stage extraction: Stage 1 maps document structure (identifies clause IDs and hierarchy), Stage 2 extracts clause content using IDs as anchors, Stage 3 validates hierarchy consistency. Include hierarchical schema {clause_id, parent_clause_id, clause_type, content, exceptions: []}.

C) Use regex to detect clause markers and build the hierarchy

D) Provide 20 few-shot examples covering all hierarchy patterns

■ Correct Answer: B

Structure-first extraction: mapping the hierarchy before content extraction separates two difficult tasks. Stage 1 produces a reliable map (clause IDs, parents). Stage 2 extracts content with known anchors. Stage 3 validates consistency (no orphan clauses, correct exception references). This is more robust than trying to extract structure and content simultaneously.

Q76. How do you design a prompt engineering strategy for a task where ground truth is difficult to establish (e.g., 'tone analysis' of marketing copy)?

A) Skip tone analysis — subjective tasks are not suitable for Claude

B) Operationalize the subjective concept: define specific tone dimensions (formality, energy, warmth) as measurable scales (1-5) with explicit anchors for each level. Include 3-4 calibration examples per dimension where domain experts agree on scores. Test inter-rater reliability between Claude and human experts.

C) Use confidence scores to flag uncertain tone assessments

D) Accept that tone analysis will be inconsistent — this is inherent to subjective tasks

■ Correct Answer: B

Operationalization converts subjective to measurable: formality 1='very casual/slang' to 5='highly formal/corporate'. Expert-calibrated examples anchor the scale. Inter-rater reliability testing reveals where Claude diverges from human judgment. This gives subjective analysis the consistency of objective extraction.

Q77. You are building a medical coding system that maps clinical text to ICD-10 codes. Accuracy is critical. What prompt engineering and validation architecture maximizes accuracy?

A) Few-shot examples with 10 ICD-10 code examples

B) Multi-component accuracy architecture: (1) primary extraction with structured schema {code, description, confidence, supporting_text}, (2) independent validation extraction on same input, (3) comparison for disagreements, (4) disagreement routing to specialist review, (5) ongoing calibration against confirmed diagnoses from EHR system

C) Use the largest Claude model — capability determines accuracy

D) Post-validate codes against the ICD-10 database for validity

■ Correct Answer: B

Medical coding accuracy requires systemic architecture: independent validation catches errors the primary extraction misses, disagreement routing handles genuinely ambiguous cases, ongoing calibration against confirmed diagnoses continuously improves accuracy. ICD-10 database validation (option D) is necessary but only catches invalid codes, not wrong-but-valid codes.

Q78. You are designing a prompt that must work reliably for inputs in 30 languages including right-to-left scripts (Arabic, Hebrew) and ideographic scripts (Japanese, Chinese). What prompt engineering considerations apply?

A) Translate all inputs to English before processing

B) Test extraction quality per language family; define how to handle mixed-language inputs (transliteration, code-switching); specify script-aware normalization rules (Chinese traditional vs simplified); include examples from each script family; specify that output JSON field names are always in English regardless of content language

C) Use separate Claude instances for different language families

D) Claude handles all languages equally — no special considerations needed

■ Correct Answer: B

Multilingual prompt engineering: language families need testing (Claude's accuracy varies by language). Mixed-language inputs need policy. Normalization handles Chinese character variants. JSON field names in English ensure consistent parsing. RTL scripts don't affect JSON structure but may affect date/number parsing—test explicitly.

Q79. How do you build a self-improving prompt pipeline where the prompt automatically evolves based on extraction errors?

A) Prompt improvement requires manual human effort — automation is not possible

B) Automated refinement loop: extraction → validation → error classification → error pattern analysis (cluster similar errors) → generate targeted prompt patches for top error patterns → A/B test patches → deploy winning patch → repeat. Track error rate over time as the improvement metric.

C) Use Claude to generate new prompts from scratch each week

D) Fine-tune a custom model on extraction errors

■ Correct Answer: B

Automated prompt refinement: structured error classification enables pattern detection. Clustering similar errors identifies systemic issues (not one-off noise). Targeted patches address specific error patterns. A/B testing validates improvement before deployment. Error rate tracking measures progress. This implements the continuous improvement cycle.

Q80. A legal contract extraction system must extract obligations (what each party must do) with temporal references (when obligations trigger or expire). Temporal references are complex ('within 30 days of material breach, unless previously waived'). How do you design the extraction schema and prompts?

A) Extract obligations and temporal references as plain text strings

B) Structured temporal schema: {obligation_text, party, trigger_event, trigger_timing: {type: 'relative/absolute/conditional', value: string, reference_point: string}, expiry: {type, value}, conditions: [string], obligation_type: 'must/may/must_not'}. Include few-shot examples for each trigger_timing type.

C) Extract only absolute dates — relative temporal references are too complex

D) Use a separate NLP library for temporal expression parsing

■ Correct Answer: B

Rich temporal schema: structured extraction preserves the complexity of legal temporal references. Trigger_timing type taxonomy (relative/absolute/conditional) covers all cases. Conditions array captures the 'unless' clauses. Few-shot examples for each timing type teach Claude the structural distinctions. Flat text extraction (option A) loses the semantic structure needed for downstream use.

Q81. You are designing a content moderation system that must handle cultural nuance — content acceptable in one cultural context may be inappropriate in another. How do you implement culturally-aware classification?

A) Apply universal content standards — cultural variation cannot be systematically handled

B) Culture-aware classification: include target_market field in input; maintain culture-specific few-shot example sets; classification schema includes {decision, primary_culture_context, cross_cultural_considerations, confidence}. Build a culture-expert review pipeline for uncertain cross-cultural cases.

C) Default to the most restrictive standard across all cultures

D) Use a separate model per cultural region

■ Correct Answer: B

Cultural context in classification: target_market enables culture-specific application of standards. Culture-specific examples teach cultural nuance. Cross_cultural_considerations field surfaces when content is acceptable in one culture but not another. Expert review for borderline cross-cultural cases builds the feedback loop for ongoing improvement.

Q82. How do you implement 'extraction conflict resolution' when multiple sources in a document provide contradictory values for the same field?

A) Take the first occurrence — consistency over recency

B) Extract all occurrences with source location: {value, location, context}. Apply resolution rules documented in the prompt (e.g., 'for price: use the summary table value if present, otherwise the line item total'). Include conflict_detected: true when resolution is ambiguous and both values for human review.

C) Take the most recent occurrence — documents build on earlier content

D) Flag the document as invalid and reject it

■ Correct Answer: B

Conflict resolution with audit trail: extracting all occurrences with locations enables transparent resolution. Explicit resolution rules (summary table beats inline mention) make decisions consistent and auditable. conflict_detected: true preserves both values for human review when rules don't clearly resolve the conflict.

Q83. You need to extract information from handwritten forms that have been OCR-processed. OCR errors are common (l/1, O/0, rn/m, missing spaces). How do you design prompts that are resilient to OCR noise?

A) Fix OCR quality before extraction — garbage in, garbage out

B) OCR-resilient extraction: document OCR error patterns in the prompt; instruct Claude to interpret likely-OCR-corrupted text using context clues; include OCR noise examples in few-shot data; schema includes extraction_confidence and ocr_correction_applied fields for audit

C) Use strict pattern matching — reject extractions with likely OCR errors

D) OCR correction is the preprocessing tool's problem — extraction assumes clean text

■ Correct Answer: B

Context-aware OCR resilience: Claude's language understanding recovers many OCR errors that pattern matching misses ('Iohn' → 'John' based on context). Documenting known error patterns improves recovery rates. ocr_correction_applied flag enables downstream quality tracking. OCR preprocessing (option A) reduces errors but doesn't eliminate them.

Q84. How do you design a Batch API workflow that handles documents arriving continuously throughout the day with varying urgency levels?

A) Accumulate all documents and submit one batch per day

B) Multi-tier submission strategy: urgent documents → synchronous API (immediate). Standard documents → collected into hourly micro-batches for 50% cost reduction with 1-hour maximum latency. End-of-day cleanup batch for any remaining items. Route based on urgency tag at document ingestion.

C) Use the Batch API for everything with urgency handled by polling frequency

D) Submit individual batch requests for each document as it arrives

■ Correct Answer: B

Tiered processing with micro-batching: hourly batches still achieve 50% Batch API savings while limiting maximum latency to 1 hour. Urgent documents use synchronous API without waiting. Individual batch submissions lose the bulk processing efficiency. Daily batch creates SLA risk from late-day document arrivals.

Q85. You are building a system that extracts financial metrics from earnings call transcripts. Some metrics are stated directly ('revenue grew 12%') and others require calculation from stated numbers ('gross margin = (revenue - COGS) / revenue'). How do you design extraction + calculation in a single pass?

A) Extract stated metrics only — calculations introduce error risk

B) Schema with derivation tracking: {metric_name, stated_value (if directly stated), calculated_value (if derived), calculation_formula (if derived), source_values: [values used in calculation], derivation_confidence}. Instruct Claude to show work for calculated metrics.

C) Two separate passes: one for extraction, one for calculation

D) Pre-calculate all derived metrics before extraction to simplify the task

■ Correct Answer: B

Derivation tracking in schema: capturing source_values and calculation_formula makes calculations auditable. If calculated_value is wrong, the source_values and formula show exactly where the error is. Derivation_confidence is naturally lower for calculated metrics (calculation error risk + source value extraction error). Two-pass approach also works but adds latency.

Q86. How do you ensure that your Batch API pipeline for document processing maintains FIFO ordering for documents that must be processed in sequence?

A) Batch API guarantees FIFO processing — no special handling needed

B) Sequence enforcement outside the Batch API: assign sequence_number to documents, track dependencies in a coordinator, only submit document N+1 to batch after document N's result is confirmed. For independent documents, batch freely. Batch API makes no ordering guarantees.

C) Use batch priority flags to enforce ordering

D) Process sequential documents synchronously and batch the rest

■ Correct Answer: B

Batch API ordering reality: no FIFO guarantee within a batch or across batches. Sequence enforcement requires external coordination: track dependencies, gate submission on predecessor completion. For truly independent documents (no ordering requirement), batch freely. Don't introduce artificial sequencing constraints on independent documents.

Q87. A document extraction system must handle both dense technical documents (50,000 tokens) and short forms (500 tokens) using the same extraction prompt and schema. How do you optimize for both?

A) Use different models for short and long documents

B) Adaptive chunking strategy: for documents under 4,000 tokens, single-pass extraction. For longer documents, hierarchical extraction: first pass creates a structured outline with section summaries, second pass extracts detailed fields from relevant sections identified in the outline.

C) Use the full 200K context window for all documents — no special handling needed

D) Truncate all documents to 4,000 tokens for consistency

■ Correct Answer: B

Adaptive strategy: single-pass is optimal for short documents (no overhead). Hierarchical extraction for long documents avoids attention dilution: the outline pass identifies where information lives, targeted section extraction applies focused attention. This is more efficient than loading 50,000 tokens and hoping important details are attended to.

Q88. You need to design a prompt for extracting relationships between entities in text (e.g., 'Company A acquired Company B for $X'). Relationships can span multiple sentences. How do you handle cross-sentence relationship extraction?

A) Extract only within-sentence relationships — cross-sentence is too complex

B) Multi-sentence relationship schema: {relationship_type, subject_entity, object_entity, relationship_attributes (amount, date, etc.), evidence_sentences: [sentence indices where evidence appears], confidence, requires_inference: boolean}. Few-shot examples showing relationships requiring evidence across 2-3 sentences.

C) Pre-process text into single-sentence units before extraction

D) Use a dependency parser to identify relationships, then Claude for details

■ Correct Answer: B

Cross-sentence extraction schema: evidence_sentences array captures the multi-sentence nature of the relationship. requires_inference: true flags relationships that require connecting information across sentences (more error-prone). This schema enables quality auditing: multi-evidence relationships can be spot-checked for accuracy.

Q89. How do you evaluate prompt quality for a structured extraction task at scale when manual evaluation of 10,000 results is infeasible?

A) Sample 100 results randomly and extrapolate quality

B) Automated evaluation pipeline: (1) syntactic validation (schema compliance, 100% automatable), (2) semantic validation (value consistency rules, ~70% automatable), (3) reference dataset comparison for key metrics (precision/recall on labeled subset), (4) drift detection (monitor metric distributions over time for unexpected shifts)

C) Use a second Claude call to evaluate each extraction

D) Only evaluate on cases where downstream processing fails

■ Correct Answer: B

Layered automated evaluation at scale: each layer catches different error types. Syntactic: catches schema violations. Semantic: catches logical inconsistencies. Reference comparison: tracks precision/recall trends. Distribution monitoring: catches prompt drift without labeled data. Combined, these cover >80% of extraction quality without manual review of all 10,000.

Q90. How do you design a prompt that handles 'nested optional structures' in extraction — where a document may have 0, 1, or many instances of a section, each with its own sub-structure?

A) Extract each section type in a separate API call

B) Array-based schema with nullable sections: {mandatory_fields: {...}, optional_section_a: [{field1, field2, ...}] | null, optional_section_b: [{...}] | null}. Instruct Claude: 'If section X is absent, set it to null. If present, extract all instances as array elements.' Include examples for 0, 1, and 2+ instances.

C) Force all documents to have all sections by filling absent ones with defaults

D) Use a two-pass approach: first detect which sections exist, then extract each

■ Correct Answer: B

Nullable arrays for variable sections: empty array vs null vs populated array distinguishes 'section exists but is empty', 'section not present', and 'section has items'. Examples for 0, 1, and 2+ instances cover the full range. Two-pass approach (option D) also works but doubles the API calls.