WORKSHOP 4 OF 5

Prompt Engineering & Structured Output

Reliable, schema-enforced outputs for production systems

WHAT YOU'LL LEARN IN THIS WORKSHOP

- Write explicit criteria that eliminate false positives
- Apply few-shot examples for consistency and edge-case handling
- Enforce JSON schema compliance with tool_use (not markdown)
- Build retry-with-feedback loops for extraction quality
- Use the Batch API for cost-efficient non-blocking workloads

EXAM WEIGHT: 20%	60 Questions | 120 Minutes | Pass Score: 720/1000

BEGINNER Foundations	INTERMEDIATE Application	ADVANCED Production

BEGINNER

1. Prompt Structure & Explicit Criteria

```
# BAD — vague
system = 'Review code for issues. Be thorough but conservative.'
# GOOD — explicit criteria
system = '''
REPORT: bugs (logic vs documented intent), security (injection/auth bypass),
data loss (unhandled exceptions).
DO NOT REPORT: style, refactoring opportunities, comments quality.
For each issue: file, line, category, description, fix.
'''
```

```
# Few-shot: cover the most ambiguous edge cases
system = '''Extract invoice data. Return JSON only.
EXAMPLES:
Input: 'Invoice #INV-001 Jan 15 2024. Total: $1,250.00'
Output: {"invoice_number":"INV-001","date":"2024-01-15","total":1250.00}
Input: 'See attached for pricing. Amount due upon receipt.'
Output: {"invoice_number":null,"date":null,"total":null,
"note":"Insufficient data"}
'''
```

INTERMEDIATE

2. Guaranteed JSON with Tool Use

```
extract_tool = {
"name": "extract_invoice",
"description": "Extract structured invoice data",
"input_schema": {
"type": "object",
"properties": {
"invoice_number": {"type": ["string","null"]},
"date": {"type": ["string","null"]},
"total": {"type": ["number","null"]},
"currency": {"type": "string", "enum": ["USD","EUR","GBP","other"]}
},
"required": ["invoice_number","date","total","currency"]
}
}
response = client.messages.create(
model='claude-sonnet-4-5', max_tokens=1024,
tools=[extract_tool],
tool_choice={'type': 'tool', 'name': 'extract_invoice'},
messages=[{'role': 'user', 'content': f'Extract: {doc_text}'}]
)
tool_block = next(b for b in response.content if b.type == 'tool_use')
data = tool_block.input # already a Python dict
```

ADVANCED

3. Retry Loops & Batch API

```
def extract_with_retry(document_text: str, max_retries=2) -> dict:
messages = [{'role': 'user', 'content': f'Extract: {document_text}'}]
for attempt in range(max_retries + 1):
response = client.messages.create(
model='claude-sonnet-4-5', max_tokens=1024,
tools=[extract_tool],
tool_choice={'type': 'tool', 'name': 'extract_invoice'},
messages=messages)
data = next(b for b in response.content if b.type=='tool_use').input
errors = validate_invoice(data)
if not errors: return data
if attempt < max_retries:
messages.append({'role': 'assistant', 'content': response.content})
messages.append({'role': 'user', 'content':
f'Validation errors:\n' + '\n'.join(errors) + '\nCorrect the extraction.'})
return data
# Batch API — 50% cost reduction for overnight workloads
batch = client.messages.batches.create(requests=[
{'custom_id': f'inv-{id}',
'params': {'model': 'claude-sonnet-4-5', 'max_tokens': 1024,
'tools': [extract_tool],
'tool_choice': {'type': 'tool', 'name': 'extract_invoice'},
'messages': [{'role': 'user', 'content': f'Extract: {doc}'}]}}
for id, doc in docs.items()])
```

Domain 4 — All 60 Practice Questions

TOTAL: 60 QUESTIONS	Beginner: 20	Intermediate: 26	Advanced: 14

BEGINNER

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.

INTERMEDIATE

Q21. 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.

Q22. 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.

Q23. 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.

Q24. 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.

Q25. 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.

Q26. 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.

Q27. 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.

Q28. 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.

Q29. 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.

Q30. 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.

Q31. 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.

Q32. 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.

Q33. 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.

Q34. 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.

Q35. 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.

Q36. 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.

Q37. 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.

Q38. 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.

Q39. 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.

Q40. 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.

Q41. 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.

Q42. 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.

Q43. 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.

Q44. 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.

Q45. 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.

Q46. 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).

ADVANCED

Q47. 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.

Q48. 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.

Q49. 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.

Q50. 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.

Q51. 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.

Q52. 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.

Q53. 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.

Q54. 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.

Q55. 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.

Q56. 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.

Q57. 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.

Q58. 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.

Q59. 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.

Q60. 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.