BUILD WITH IT

Context Engineering

The window is a budget. Learn to spend it.

First edition, September 2026

Addaly

Series editor: Dr Mustafa Mun

Draft, open for academic review

BUILD WITH IT

Context Engineering

The window is a budget. Learn to spend it.

Series editor: Dr Mustafa Mun

Addaly

First edition, September 2026 · Draft, open for academic review

Context Engineering

First edition, September 2026, build 62a26075.

Written by AI models to a written standard, with Dr Mustafa Mun as series editor. Exam keys checked blind by a second model: 3,665 of 3,666 agreed. Academic review invited.

The manuscript was drafted with the assistance of a large language model and was edited and published under his name. That is stated plainly here rather than left to be discovered, because a reader is entitled to know how a book they are trusting was made.

How to cite: *Mun, M. (2026). Context Engineering. Addaly. addaly.com/learn/context-engineering.*

This book is the printed form of the course of the same name at addaly.com/learn/context-engineering. The website is the living version and is corrected first. Where the two differ, the website is right.

Free to read, free to download, free to print, and free to teach from. There is no paid edition and there never will be.

Every diagram in it is drawn from data by code, not generated as a picture, so the numbers on the page are the numbers in the argument. The answers to every check, test and examination question are at the back.

9 modules · 87 lessons · 30 figures · 9 case studies · 70,463 words · about 13 hours

Brief contents

1

MODULE 1

The window as an engineering surface

Account for every token in a request — system prompt, tool definitions, history, retrieved material and output reserve — count them with the tokeniser the provider actually uses, and predict from that budget what the call will cost, how long its first token will take, and which part gets dropped when it overflows.

Lesson 1 · 8 min

The window is a budget, not a container

THE ONE THING TO KEEP

Every token costs money, latency and attention, so treat the window as spend, not storage.

Fitting is not working

You have a 200,000-token window. Your document is 180,000 tokens. It fits.

That is not the same as it working. Three things get worse as you add tokens, and they get worse at different rates.

Money. Input tokens are billed on every call, forever. At $3 per million input tokens, a 200k-token prompt costs $0.60 to send. Ten thousand calls a day is $6,000 a day, $180,000 a month. The same task trimmed to 6k tokens costs $0.018 a call — about $180 a day. Same model, often the same answers, roughly 33x the bill. For a team in Lagos or Lahore paying in dollars from local revenue, that difference is the whole business case.

Latency. Before the model emits a single token, it reads yours. Prefill scales with input length. A 200k-token prompt typically means several seconds before the first character appears; an 8k prompt means a few hundred milliseconds. Users forgive a slow answer. They do not forgive a slow blank screen.

Attention. This is the one people miss. A model does not read your context the way a database reads a row. It attends across all of it, and every irrelevant passage you add is a passage that can win the competition for relevance. Add the whole product manual and you have not just added the paragraph about refunds — you have added forty paragraphs that *look* like they are about refunds.

What the benchmarks hide

Needle-in-a-haystack tests — hide one odd sentence in 500k tokens, ask the model to find it — are close to saturated on frontier models. Teams read that and conclude long context is solved.

It is not the same task. The needle is semantically alien to its surroundings, and there is exactly one. Real work asks the model to combine four facts scattered across a long document, where three near-duplicates of each fact also appear and two of them are outdated. Accuracy on that shape of task degrades with length on every model measured, and it degrades well before the window fills.

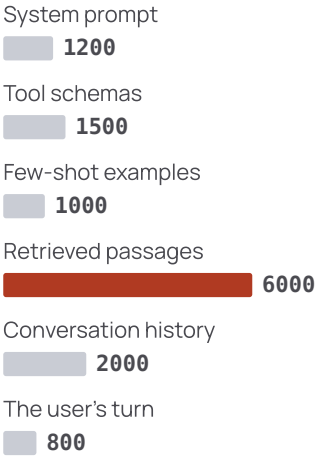
The honest summary: long context is real and getting better, and it still costs you accuracy, not only money.

Spend it, do not fill it

Treat the window like a monthly budget in a currency you actually earn. Give every component a line item, in tokens, and enforce it in code rather than hoping.

Figure 1.1

A 12,500-token ledger for one request



The same allocation the code below enforces. Retrieved passages are nearly half of it, which is why the selection module comes before everything else. The point of writing it down is not the trimming — it is the warning that fires the day somebody uploads a PDF of scanned invoices and retrieval quietly starts returning forty thousand tokens.

```
BUDGET = {
    "system":       1_200,   # role, constraints, output
contract
    "tools":        1_500,   # tool schemas
    "examples":     1_000,   # few-shot, static
    "retrieved":    6_000,   # ~8-12 chunks after reranking
    "history":      2_000,   # trimmed, oldest dropped first
    "user_turn":      800,
}
# total 12,500 in, leaving room for output

def assemble(parts, budget):
    out = {}
    for name, limit in budget.items():
        text = parts.get(name, "")
        n = count_tokens(text)
        if n > limit:
            log.warning("over budget: %s used %d of %d", name,
n, limit)
            text = trim(text, limit)          # per-part policy,
not a blind cut
        out[name] = text
    return out
```

The logging line matters more than the trimming. When retrieval quietly starts returning 40k tokens because someone uploaded a PDF of scanned invoices, you want a warning, not a surprise invoice.

The question to ask of every token

For each block you are about to include: *if I delete this, which specific answer gets worse?*

If you cannot name the answer, delete the block and measure. Most context bloat is defensive — someone added the full policy document because a single question about refunds went wrong once, and nobody ever took it out. Six months later that block is in every call, on every request, in the bill every month.

The cheapest optimisation available to you is usually deletion.

What this course does

Everything that follows is a way of spending that budget better: where a token should sit so the model uses it, which tokens can be cached so they cost a tenth, when to fetch tokens instead of carrying them, how to cut documents without cutting meaning, and how to prove a change helped rather than believing it did.

BEFORE YOU MOVE ON

A team replaces a retrieval step (5 chunks, about 3,000 tokens) with pasting their entire 90,000-token product manual into every call. The manual fits comfortably in the model's window. Accuracy on specific policy questions gets noticeably worse. What is the most likely explanation?

A. The manual needs converting to clean Markdown; the model is being confused by PDF formatting artefacts.

B. The 90,000 tokens contain many passages that resemble the answer, and the correct one now competes with dozens of plausible distractors.

C. The prompt exceeded the usable context and the provider silently truncated the middle of the manual.

D. The model's knowledge cutoff predates the manual, so it falls back on outdated training data instead of reading the text.

The answer and why: p. 489

Lesson 2 · 9 min

Nobody typed this

THE ONE THING TO KEEP

Context engineering starts the moment a program, not a person, assembles what the model reads — and the questions that matter become selection, order, cost, provenance and proof rather than wording.

A request nobody wrote

At 03:14 a support assistant answers a question about a delayed refund. The request that reached the model was 9,800 tokens long. A human typed fourteen of them.

Here is where the rest came from, in the order the code appended it:

- **620 tokens** of system prompt, written by a product manager in March and edited twice since.
- **1,150 tokens** of tool definitions — six functions, their JSON schemas serialised into the prompt by the SDK.
- **2,900 tokens** of conversation history, four earlier turns from the same customer.
- **4,100 tokens** of retrieved material: nine passages pulled from a help centre by a vector search that ran forty milliseconds earlier.
- **14 tokens** of what the customer actually asked.
- **1,000 tokens** held back so the answer has room to exist.

No person read that request before it was sent. No person will read it afterwards unless something goes wrong. It was built by a function, from parts, most of which change on every call.

That function is the thing this course is about.

Prompting ends where assembly begins

Prompt engineering is what you do when you are the one typing. It is real work, and it has a ceiling: you can only tune the words you personally control. The moment your system starts inserting material you have never read — a customer's earlier messages, a passage from a document somebody uploaded this morning, the output of a tool that returned 40,000 characters of JSON — the words stop being the interesting variable.

What matters then is:

- **What got selected**, out of everything that could have been selected.
- **In what order**, because order changes whether a fact is used.
- **At what cost**, because every token is billed, and prefill time scales with how many there are.
- **From where**, because material has a provenance and some of it is hostile.
- **How you would prove** that yesterday's change was an improvement.

Those five questions are not answered by better wording. They are answered by code, budgets and measurement. That is why the discipline has a different name.

What is engineering about it

Call it engineering when these things become true of your context:

1. **It has a budget.** A number, written down, that the assembled request must not exceed, with a reserve for output.
2. **It is deterministic.** Given the same inputs, the assembler produces the same bytes. If it does not, you cannot debug it.
3. **It is testable.** You can call the assembler in a unit test without calling a model, and assert on what came out.
4. **It is versioned.** The whole call — instructions, retrieval settings, model id, sampling parameters — has an identifier, and a log line carries it.

5. **It has failure modes you can name.** Overflow, truncation, a cache miss, a poisoned passage, a tool result that swallowed the instructions.

If none of those are true, you do not have a context problem yet. You have a prompt, and prompt-level thinking will serve you fine.

The honest limit

Context engineering does not make a model correct. It changes what the model has to work with, and nothing more. A model given the right passage can still misread it; a model given a perfect budget can still invent a number.

The mechanism is worth stating plainly, because it sets the ceiling on everything else in this course. A language model produces a distribution over next tokens conditioned on the tokens in front of it. Better context shifts that distribution — sometimes dramatically. It does not add a verification step, a memory of what is true, or a refusal to guess. Those have to be built around the call, not inside the window.

So the honest claim is narrow and still worth a course: most production failures blamed on the model are failures of what the model was given. The passage was not retrieved. The instruction was 8,000 tokens above the question. The tool returned a wall of JSON and the schema at the top fell out of the window. You can fix all three without touching the model.

What you need to follow along

Everything measurable in this course is measurable for free, on a laptop, without a GPU:

- `tiktoken` (pip install tiktoken) counts tokens for OpenAI-family tokenisers offline.
- Hugging Face `transformers` gives you the exact tokeniser and chat template of any open-weights model, again offline.
- `llama.cpp` or Ollama runs a 3B–8B model locally so you can run position and ordering experiments without a bill.

- Anthropic and OpenAI both expose a token-counting endpoint, free of inference charges, when you need their exact number.

The paid path exists — hosted evaluation platforms, managed vector databases, observability suites — and none of it is required to learn any of this. A Python file, a JSON file of test cases and a text editor will take you through every lesson here.

Where this course goes

Nine modules, in the order the problem actually appears: what is in the window and what it costs; choosing what earns a place; the retrieval pipeline that fills it; position and ordering; cache and latency; conversation and memory; tools and agents; trust boundaries; and finally how to prove a change helped and ship it so it can be rolled back.

The first thing to do is stop calling it a prompt.

BEFORE YOU MOVE ON

A team′s assistant sends 9,800-token requests of which the user typed 14 tokens. They spend a week rewording the 620-token system prompt and see no improvement. What does this most likely indicate?

A. The system prompt is too short relative to the request and should be expanded until it is a larger share of the window.

B. The model is too small for the task and the fix is a larger model.

C. The variable 94% of the request — retrieval, history, tool definitions — is where the behaviour is being decided, and it has not been touched.

D. Rewording cannot help any request, because instructions are ignored once the context exceeds a few thousand tokens.

The answer and why: p. 490

CASE STUDY · MODULE 1

The question bank that fitted, and the bill that did not

An illustrative case, built from documented patterns.

A coaching centre in Kota with 3,400 enrolled students, six weeks before the January session, running a doubt-solving assistant over its own eleven-year question bank.

Ranjan Bhatt ran two people and a server. The assistant they had built did one thing and did it well: a student typed a doubt, and it answered using the centre's own question bank and the worked solutions the faculty had written over eleven years. Faculty trusted it because the material was theirs.

The implementation was the simplest one available. The physics bank as plain text is about 180,000 tokens. The model's advertised window was 200,000. So every request sent the whole bank.

It fitted. For four months nobody looked any further, because fitting and working feel identical from the outside.

Two things arrived in the same December week. The first was the December invoice: ₹11.4 lakh, against ₹1.6 lakh in August. Usage had roughly quadrupled as the session approached; the bill had grown with it, exactly as it should have, and nobody had done the arithmetic in advance. At roughly ₹250 per million input tokens, one doubt cost about forty-five paise to answer before the model wrote a word. Eighty thousand doubts in a week is a number a coaching centre reaches without noticing.

The second was a complaint from the senior physics faculty, and it was worse. Students were reporting answers that quoted a formula correctly and then applied it to the wrong case — rotational inertia for a hollow cylinder used on a solid one. The solutions in the bank were right. The assistant was reading 180,000 tokens containing forty near-identical worked examples and picking a neighbouring one.

The decision

Ranjan had two routes and six weeks.

Keep sending everything. Nothing to build. The faculty's material stays intact, nothing can be missed by a retriever that was never written, and there is no new component to break in the middle of the session. The cost is the bill, which will roughly double again by the exam, and the accuracy problem, which they had no reason to believe would improve.

Write a budget and retrieve. Give each block of the request a token allocation, count with the tokeniser the provider actually uses, and fetch eight to twelve relevant solutions instead of all of them. The cost is two weeks of the only two engineers, during the six weeks when a broken assistant is most visible, and a real chance that the retriever misses the one solution a student needed and the assistant answers from nothing.

The second option also carried a risk Ranjan could not size. The faculty's objection was not to the bill. It was that they had spent eleven years writing that bank and did not want a program deciding which parts of it a student was allowed to see.

What made the argument tractable

Ranjan's colleague Meera spent an afternoon on a measurement rather than an opinion. She took fifty doubts whose correct solution everybody agreed on, and built contexts at 8,000, 32,000, 100,000 and 180,000 tokens, each containing the right solution at a random position among real distractors from the same chapter.

At 8,000 tokens the assistant answered forty-six of fifty. At 180,000 it answered thirty-one. The material was identical. The only thing that changed was how much of it was in the window at once.

That number cost about ₹900 of inference and it ended the meeting. It also reframed the faculty's objection: the choice was not between showing students everything and showing them some of it. It was between showing the model everything and having it use the wrong part, or showing it twelve solutions and having it use the right one.

YOUR ANSWERS

1. The bank fitted inside the advertised window. Why did accuracy still fall as the window filled, and why did nobody notice for four months?

2. The faculty objected that a program should not decide which of their solutions a student sees. Was that objection answered, or overruled?

3. Ranjan's team wrote the budget ledger before building the retriever, and it saved them five days. Why would that order help?

STOP HERE

Write your answers before you turn the page. How it played out starts on p. 60.

This page is blank so that how it played out stays out of view until you turn the page.

CASE STUDY · MODULE 1

How it played out

They wrote the ledger first and the retriever second, and shipped in nine days rather than fourteen because the ledger made the retriever's target obvious: 6,000 tokens of solutions, 1,200 of instructions, nothing else. The January bill was ₹1.9 lakh on twice August's traffic. On the same fifty doubts the assistant answered forty-seven. The faculty objection was settled by a line under each answer naming the solutions used, which turned out to be the feature teachers liked most, because they could check it. The measurement Meera ran now runs monthly against a fixed fifty, and the budget assertion throws in continuous integration if any block exceeds its allocation — which it did twice in February, both times because a new chapter had been ingested with the page footers still attached.

The questions, discussed

1. The bank fitted inside the advertised window. Why did accuracy still fall as the window filled, and why did nobody notice for four months?

The advertised length is an API limit, not a quality guarantee. As input grows, attention is spread across more positions, the correct solution sits further from the question, and the distractors in this corpus are unusually dangerous because forty near-identical worked examples are on-topic and wrong. Nobody noticed because nothing failed: the request was accepted, the response was fluent, and the failure looked like the model being slightly careless rather than like a system fault.

2. The faculty objected that a program should not decide which of their solutions a student sees. Was that objection answered, or overruled?

Answered, and by evidence rather than argument. The measurement showed that sending everything did not mean the model used everything — at 180,000 tokens it used the wrong solution nineteen times in fifty. So the real choice was between an implicit, unexplainable selection made by attention and an explicit one made by a retriever that could be logged, cited and

corrected. The citation line under each answer gave the faculty something they had never had with the old system: the ability to see which of their solutions was actually used.

3. Ranjan's team wrote the budget ledger before building the retriever, and it saved them five days. Why would that order help?

The ledger turns an open-ended engineering problem into a target with a number on it. Once solutions had 6,000 tokens rather than as many as would fit, the retriever's job was fully specified: return what fits in 6,000 tokens, ranked. Without the allocation the team would have been tuning k against a moving definition of enough. The ledger also gives an assertion that fails loudly in CI, which is how they later caught page footers being ingested as content.

MODULE 1 TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record. The answers, and the reason behind each, are in the key on p. 488. If two go wrong, re-read the module before the exam.

1 A team interpolates a customer's message into the system prompt with string formatting, reasoning that the system role carries more authority. What have they actually done?

A. Nothing, because the API strips any text that did not come from the developer's own template

B. Placed the customer's text in the position the model was trained to treat as most authoritative

C. Added cost and nothing else, because role labels exist only for the provider's logging

D. Made the customer's text exempt from the moderation applied to ordinary user turns

2 Your local tokeniser counts a request at 7,200 tokens and the usage field on the response says 9,400. What is the likeliest explanation?

A. The provider rounds billed tokens up to the nearest hundred for accounting

B. The local tokeniser is faulty and should be swapped for a different library

C. The model expands long prompts internally before billing, which is normal above 4,000 tokens

D. A whole block is missing from your count, usually tool definitions or the chat template

3 A request sits comfortably under the model's context limit and the API rejects it anyway. What has usually happened?

A. The requested completion length counts against the same limit, and the sum does not fit

B. The context limit applies per conversation rather than per request

C. The prefix cache was cold, and a cold call counts each token twice

D. Billing and enforcement use different tokenisers, so the enforced count is higher than yours

4 An assembler keeps the last N characters of an over-long request with a slice like text[-N:]. What symptom does that produce?

A. The question disappears, so the model summarises the material instead of answering it

B. The oldest conversation turns disappear, which is the behaviour you wanted anyway

C. The instructions and the output contract disappear, so the answer is plausible and wrongly shaped

D. Nothing visible, because the model reconstructs the missing framing from the retrieved passages

5 Why does a near-perfect needle-in-a-haystack score fail to predict performance on real long-context work?

A. The haystack is synthetic text, and models do worse on any genuine document

B. Those tests run at short lengths and the results are extrapolated to long ones

C. The harness finds the needle itself, so the model is never actually asked to search for it

D. The needle shares rare tokens with the question and nothing else competes for attention

6 A snapshot test on the assembled request begins failing every night at midnight. What is the defect?

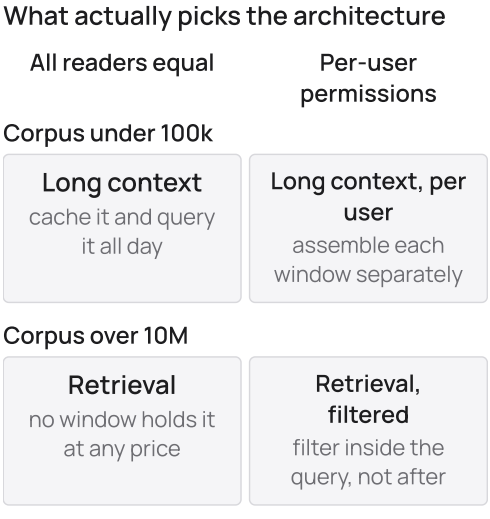
A. The retrieval index is rebuilt overnight and returns different passages

B. The assembler reads the clock itself instead of being handed the date

C. The provider rotates its chat template at the start of each day

D. The stored snapshot expires after twenty-four hours and has to be regenerated

Figure 2.1



Window size only changes the budget. Corpus size, permissions and freshness choose the shape, and a permission boundary settles it fastest: a long-context prompt containing everything the user might be allowed to see is a data leak waiting for the wrong prompt.

The version that is actually dying

Not retrieval. The naive 2023 pipeline: fixed 512-token chunks, one dense embedding model, cosine similarity, top 5, no reranking. That is being replaced, and it should be.

What replaced it:

Hybrid search. Dense embeddings handle paraphrase — "my payment bounced" finding a page titled "declined transactions". They are poor at rare literal strings, because an embedding is a lossy compression and a part number carries almost no semantic signal. Ask a support bot about error `CU-4102` or part `MX-880-B` and dense-only retrieval misses. BM25 matches it exactly. Run both, fuse the rankings (reciprocal rank fusion is four lines of code and works), and you stop losing identifiers.

Reranking. Retrieve 50 candidates cheaply, then score each against the query with a cross-encoder that reads query and passage together. It is slower per item and far more accurate, and 50 candidates is a small enough set to afford it.

Retrieval that feeds long context. These are not opposites. Retrieve 40 chunks instead of 5, rerank to 15, spend 10k tokens instead of 2k. Bigger windows made retrieval budgets more generous rather than making retrieval unnecessary.

How to choose

Situation	Take
Corpus under ~100k tokens, stable	Long context. Cache the prefix.
Corpus over ~1M tokens	Retrieval. No real choice.
Query names an exact identifier	Hybrid, with BM25 carrying it
Answer is a property of the whole	Long context, or a map-reduce pass
Per-user permissions	Retrieval, filtered at query time
Content changes hourly	Retrieval
High volume, one shared document	Long context plus caching
You need clickable sources	Retrieval

The measurement that settles arguments

Build 50 real questions with known answers. Run three configurations: retrieval only, full context, and retrieval into a generous context. Record accuracy, p95 latency, and cost per query for each.

Most teams find the third wins on accuracy and the first wins on cost, and then they have an actual decision to make instead of an opinion to defend.

EXAMINATION

Context Engineering — final examination

This paper covers the whole course:

- accounting for every token in a request and predicting what it costs
- choosing what earns a place in the window and saying what was left out
- the retrieval pipeline that fills it, from parsing awkward documents to verifying a citation
- position and reading order
- cost, prefix caching and latency
- conversation, structured state and compaction
- tools, schemas and the agent's window
- untrusted text and the trust boundary
- shipping a context change with evidence that it helped

63 questions, the whole pool. The site draws 25 for a sitting, pass mark 70%. Work any 25 under your own conditions. Answers from page 548.

1 Module 1 · The window as an engineering surface

Which of these makes a context problem an engineering problem rather than a prompting one?

A. The prompt has been rewritten by somebody with real domain expertise
B. The assembler is deterministic: the same inputs produce the same bytes
C. The system has moved to a model with a substantially larger context window
D. The instructions live in a separate template file so they can be reviewed

2 Module 1 · The window as an engineering surface

A tool result, the model's previous answer and the user's earlier message all sit in the state part of a window. Which of them is a guess?

A. The tool result, because the system behind it may have been wrong
B. The user's earlier message, because people misremember what they said
C. None of them, because anything already in the window has been validated once
D. The model's own previous output

3 Module 1 · The window as an engineering surface

An assembled request exceeds its budget. Which block should be the last thing cut?

A. The instructions and the output contract
B. Retrieved passages, lowest-ranked first
C. Conversation history, oldest first, or compacted rather than dropped
D. Few-shot examples, reduced from four to two and then measured

4 Module 1 · The window as an engineering surface

A 30,000-token contract is asked about on every call. Where does it belong in the request?

A. In the system prompt, because it is stable and ought to cache
B. Retrieved in chunks, because no stable block should be that large
C. As the first user message, so it caches and its provenance stays clear
D. At the end of the request, immediately beside the question about it

5 Module 1 · The window as an engineering surface

You need to cut a document to exactly 2,000 tokens. Why is slicing by characters wrong?

A. Characters are billed separately from tokens by most providers
B. A character slice always removes the end, which is where conclusions live
C. The tokeniser re-encodes the text afterwards, which changes the count again
D. The counts diverge, and a slice can split a multi-byte sequence

6 Module 1 · The window as an engineering surface

Which number is worth alerting on in a context budget?

A. The mean number of tokens per request
B. The ninety-ninth percentile of total tokens, and how often cuts fire
C. The number of requests that failed with a context-length error from the API
D. The ratio of input tokens to output tokens across the day

7 Module 1 · The window as an engineering surface

A parser stopped at fifty pages, a paginated tool result was never read past page one, and a query carries a LIMIT 100 written three years ago. What do these share?

A. They are all rejected by the API with a clear and attributable error
B. They are all fixed by raising the context budget for the request
C. They truncate upstream, so nothing in the model call looks wrong
D. They matter only when a request is already close to the context limit

Module 1 · The window as an engineering surface

The module starts on p. 11.

MODULE 1 TEST

Module 1 test, Q1, p. 62
A team interpolates a customer's message into the system prompt with string formatting, reasoning that the system role carries more authority. What have they actually...

Answer B: Placed the customer's text in the position the model was trained to treat as most authoritative

Why: The system role's authority is a training prior expressed by position and delimiters inside one token sequence. There is no flag in the tensor that says obey this one, and the API checks nothing. Interpolating text you did not write into that position hands whoever wrote it the strongest place in the request.

Module 1 test, Q2, p. 62
Your local tokeniser counts a request at 7,200 tokens and the usage field on the response says 9,400. What is the likeliest explanation?

Answer D: A whole block is missing from your count, usually tool definitions or the chat template

Why: When a local estimate and the billed number differ by more than about fifteen per cent, you are not looking at rounding — you are missing a block. The local count sees the strings you passed. The billed count includes the template's delimiters and role labels, the serialised tool schemas, and any image patches. Log the billed number on every call, because it is the only count you can reason about afterwards.

Module 1 test, Q3, p. 63
A request sits comfortably under the model's context limit and the API rejects it anyway. What has usually happened?

Answer A: The requested completion length counts against the same limit, and the sum does not fit

Why: On most APIs the limit is the sum of what you send and what you asked for back, so max_tokens is subtracted from the space available for input. This is why the output reserve is the first subtraction in a budget ledger rather than the last: a reserve added at the end produces a request that looks fine by your own arithmetic and fails at the boundary.

Module 1 test, Q4, p. 63
An assembler keeps the last N characters of an over-long request with a slice like text[-N:]. What symptom does that produce?

Answer C: The instructions and the output contract disappear, so the answer is plausible and wrongly shaped

Why: Keeping the tail keeps the question and the last passages and throws away the top of the request, where the instructions live. Cutting the back instead removes the question and produces a summary of the material, which is a distinctive symptom worth recognising. The cut you actually want takes the middle — oldest history, lowest-ranked passages — and it only happens if somebody writes it.

Module 1 test, Q5, p. 63

Why does a near-perfect needle-in-a-haystack score fail to predict performance on real long-context work?

Answer D: The needle shares rare tokens with the question and nothing else competes for attention

Why: The needle is lexically distinctive and there is exactly one of it, so attention has an easy target. Paraphrase it so it shares no distinctive words, add real distractors, or require several facts to be combined, and scores fall well below the advertised length on models that scored perfectly on the easy version. Measure the curve on your own material and budget below where it bends.

Module 1 test, Q6, p. 64

A snapshot test on the assembled request begins failing every night at midnight. What is the defect?

Answer B: The assembler reads the clock itself instead of being handed the date

Why: A pure assembler produces the same bytes from the same inputs, so anything it reads outside its arguments — the clock, the network, an unseeded shuffle — makes it untestable. Inject the date. The usual response to a test that fails at midnight is to delete the test, which removes the one check that would have caught a stray timestamp sitting above the cache breakpoint.

THE LESSON CHECKS

Lesson 1, p. 16

A team replaces a retrieval step (5 chunks, about 3,000 tokens) with pasting their entire 90,000-token product manual into every call. The manual fits comfortably…

Answer B: The 90,000 tokens contain many passages that resemble the answer, and the correct one now competes with dozens of plausible distractors.

Why: Every token costs money, latency and attention, so treat the window as spend, not storage.

A The manual needs converting to clean…: Formatting does affect parsing, so it feels like the obvious first fix, but it would not explain a drop relative to chunks drawn from the same document.

C The prompt exceeded the usable context…: Silent truncation does happen with some client libraries, but the stated premise is that it fits, and providers error rather than trim on overflow.

D The model's knowledge cutoff predates the…: Cutoffs genuinely cause stale answers, but text placed in the context is read regardless of when it was written.

Lesson 2, p. 20

A team's assistant sends 9,800-token requests of which the user typed 14 tokens. They spend a week rewording the 620-token system prompt and see no…

Answer C: The variable 94% of the request — retrieval, history, tool definitions — is where the behaviour is being decided, and it has not been touched.

Why: Context engineering starts the moment a program, not a person, assembles what the model reads — and the questions that matter become selection, order, cost, provenance and proof rather than wording.

A The system prompt is too short…: Treats influence as proportional to token share; a longer instruction block competes for the same attention and costs more, without addressing which material was selected.

B The model is too small for…: Jumps to the model when 94% of the request was assembled by code that has not been examined; a larger model reading the same wrong passage answers wrongly more fluently.

D Rewording cannot help any request, because…: Overgeneralises from one failed experiment into a false mechanism; instructions are followed at long context, they are just competing with far more material.

Lesson 3, p. 24

A request is 12,000 tokens against a 16,000-token limit, and the API rejects it. `max_tokens` is set to 6,000. Why?

Answer A: The limit applies to input and output together, so 12,000 plus a 6,000-token reserve exceeds 16,000.

Why: Sort every token in the window into instructions, exemplars, knowledge or state — the four differ in half-life, trust, cache behaviour and failure mode, and almost every layout decision follows from which kind a block is.

B The tool definitions are being counted…: Invents a double-count; tool definitions are serialised once, and their cost is real but singular.

C The request exceeded the model's effective…: Confuses a quality degradation, which is real and unenforced, with a hard API limit, which is arithmetic on the declared numbers.

D History has not been compacted, and…: Assumes tokens are weighted by origin; every token counts once regardless of which of the four kinds it belongs to.

BUILD WITH IT

Context Engineering

The window is a budget. Learn to spend it.

WHAT IT TEACHES	For people who have already written the clever prompt and hit its ceiling. This course treats the context window as an engineering surface: what you put in it, in what order, from where, at what cost, and how you prove a change was an improvement.
WHAT IS INSIDE	Nine modules and 87 lessons, 30 figures drawn from data, nine case studies, nine module tests, a 63-question examination, and every answer with the reason behind it.
LICENCE	CC BY-NC-SA 4.0. Copy, print, share and adapt it for any non-commercial purpose, with credit, under the same licence. There is no paid edition.
THE COURSE	Free to read, with the lessons, the films and the examination, at **addaly.com/learn/context-engineering**

Scan to open the course

Addaly

First edition, September 2026 · build 62a26075