

CONTEXT ENGINEERING

FIELD GUIDE

The practical playbook for building better context in RAG, AI agents, and production LLM systems.

RAW INPUT

100,000

NOISY TOKENS



CURATED CONTEXT

Only what matters

RETRIEVE / RERANK / PRUNE / CHECKPOINT / CACHE

A million-token window is capacity.

Useful context is an engineering decision.

FIRST, THE DISTINCTION

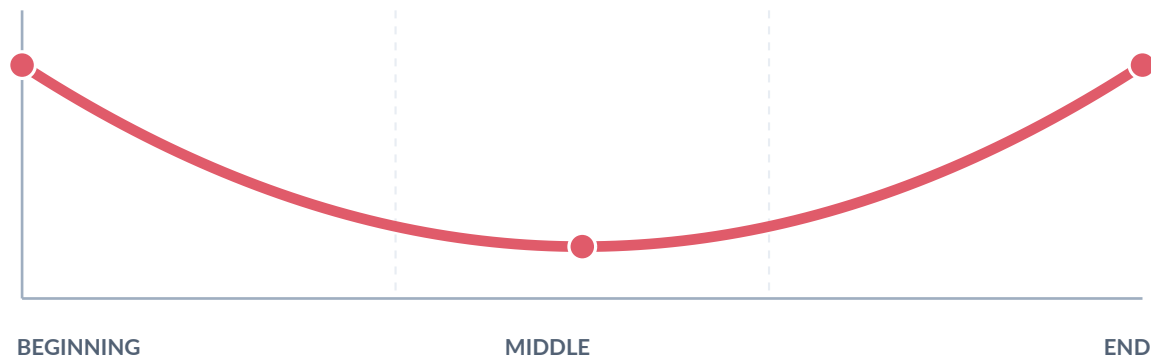
Can the model fit it?

Can it actually use it?

A model's context window tells you how many tokens it can accept under its API limits. It **does not** guarantee equal recall, reasoning quality, or usefulness for every token you send.

RELEVANT EVIDENCE POSITION / CONCEPTUAL

Higher task performance



The U-shape was observed in landmark research. It is **not** a universal performance curve. Some newer models perform very well on simple needle-retrieval tests.

CAPACITY

How much input/output the model can handle.

RELEVANCE

Whether the included material helps this specific decision.

SUFFICIENCY

Whether all necessary facts are available to answer.

Design target: the smallest context that is **sufficient, fresh and trustworthy** for the next decision.

Reading: Liu et al. (2024), *Lost in the Middle*. Google Research (2025), *Retrieval Quality at Context Limit*.

A BETTER WAY TO THINK ABOUT IT

Build a context pipeline

Do this at **every meaningful model call**, not only once when the chat starts. Context is selected, assembled, measured, and refreshed as the agent works.

THE SIX-STEP LOOP

01

UNDERSTAND

Intent, constraints, permissions



02

FETCH

Docs, tools, history, state



03

SELECT

Rank, verify, de-duplicate



06

CHECKPOINT

Save verified state + refs



05

REASON

Answer or request more



04

PACK

Budget, order, cite

Checkpoint state becomes the starting point for the next call.

WHERE GOOD SYSTEMS GET DIFFERENT

- **Retrieval is task-aware**

An exact error code may need lexical search. An architectural explanation may need semantic search. Often use both.

- **Rank for coverage over similarity**

One highly similar paragraph can still miss an exception, a timeline, or contradictory evidence.

- **Use source IDs and versions**

Keep timestamps, document versions, and source pointers attached to every claim you elevate.

- **Separate facts from hypotheses**

A generated summary is not evidence. Preserve its supporting excerpts and mark uncertainty.

The loop: retrieve, check sufficiency, assemble, act, update structured state.

INDEX, FIND, THEN FILTER

Three layers before the model

These techniques solve different problems. They are **not** interchangeable, and none is a shortcut around retrieval quality.

01 / CORPUS PREPARATION

Semantic and structural chunking

Split source material at useful boundaries: sections, functions, decisions, tables, or logical topics. Attach **document ID, version, timestamp, heading, and access scope** to each chunk. Overlap only where it demonstrably protects meaning.

Watch out: chunks can lose cross-section dependencies. Expand neighbors when needed.

02 / QUERY-TIME EVIDENCE

Hybrid retrieval + reranking

Use lexical search for exact strings (IDs, errors, version numbers) and dense retrieval for semantic matches. Rerank for the **current question**, not generic topic similarity. Keep diverse evidence when the answer needs multiple steps or conflicting sources.

Watch out: retrieval recall can improve while answer accuracy stays flat.

03 / TOKEN ALLOCATION

Dynamic pruning and deduplication

Remove repeated logs, superseded tool responses, stale documentation, and irrelevant chat turns. Preserve **negative evidence, exceptions, source pointers, and decision constraints** even when they are short and easy to overlook.

Watch out: blind truncation can remove the evidence that changes the answer.

Research grounding: Anthropic, Contextual Retrieval (2024). Google Research, Sufficient Context (2025).

AGENTS NEED A DIFFERENT STRATEGY

Three layers across turns

As agents call tools, their contexts grow. The goal is to preserve continuity without repeatedly shipping the entire session transcript.

04

Progressive disclosure / just-in-time fetch

Keep file paths, document IDs, and database keys as pointers. Load the exact slice only when the agent needs it. This keeps large tool catalogs and documents out of every inference call.

Watch out: Don't hide material the next decision actually requires.

05

Compaction + structured state

When a session grows, checkpoint verified facts, decisions, pending questions, constraints, and source references. Then continue from that state, while leaving full artifacts retrievable.

Watch out: Summaries can silently erase qualifiers. Verify and pin critical details.

06

Prefix caching / KV-cache reuse

During inference, the KV cache holds attention states for processed tokens. Provider prompt caching may reuse matching prefix computations, improving repeat-call cost or latency. Keep **stable** instructions and tool schemas before changing evidence.

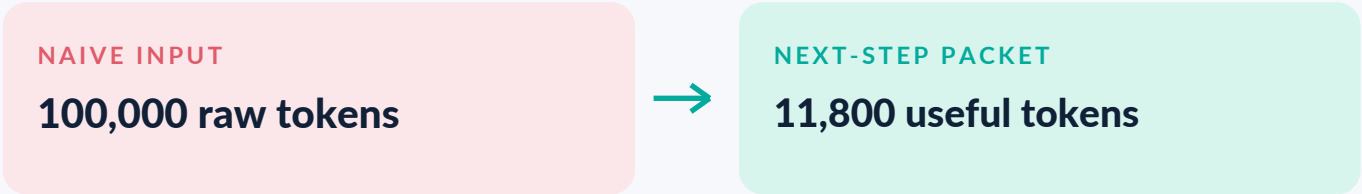
Watch out: Caching speeds up computation. It does not make weak evidence reliable.

Three different things. Context is the current input. Memory is stored facts and artifacts. KV cache is reused model computation.

EXAMPLE / INCIDENT INVESTIGATOR AGENT

What caused the latency spike?

The user asks: “Checkout p99 jumped from 170 ms to 1.8 s after the 14:05 deploy. Why?” Assume the agent has 100,000 tokens of logs, traces, diff output, and conversations. (Illustrative scenario.)



AN EXAMPLE TOKEN BUDGET

Stable instructions + tools	1,400
Goal, constraints, current state	1,100
Before/after metrics	2,000
Deploy diff + config changes	2,800
Grouped errors with counts	2,300
Representative traces	1,200
Missing evidence + source IDs	1,000
TOTAL CURATED INPUT	11,800

KEEP

Evidence with IDs: p99 split by version (M1), DB lock wait trace (T2), transaction-scope diff (D3), stable DB CPU (M4). State says “**lock contention is a hypothesis.**”

LEAVE RETRIEVABLE

Thousands of repeated log lines, unrelated historical docs, and stale chat. Preserve pointers. **Next step:** compare v41 vs v42 traces and confirm wait events before claiming root cause.

All counts and telemetry above are illustrative, not experimental results or product benchmarks.

STATE IS NOT EVIDENCE

A reusable context packet

Give the model a clear goal, independently attributable facts, and the next information gap. Do not ask it to infer all of this from an enormous transcript.

CONTEXT PACKET / SCHEMATIC JSON

```
{
  "task": "Explain checkout p99 regression",
  "constraints": [
    "Read only until rollback approval"
  ],
  "time_window": "14:00-14:40 UTC",
  "verified_facts": [
    {"claim": "p99 170ms -> 1.8s on v42",
     "source": "M1", "version": "v42"}
  ],
  "hypotheses": [
    {"claim": "DB lock contention",
     "status": "UNCONFIRMED"}
  ],
  "evidence": ["M1", "T2", "D3", "M4"],
  "missing": ["v41 trace comparison",
             "DB wait-event metrics"],
  "next_action": "fetch DB wait-event metrics"
}
```

THREE NON-NEGOTIABLES

VERIFIED

Facts link back to raw evidence.

HYPOTHESIS

Unconfirmed claims stay labeled.

RETRIEVABLE

Keep source IDs. Fetch full text later.

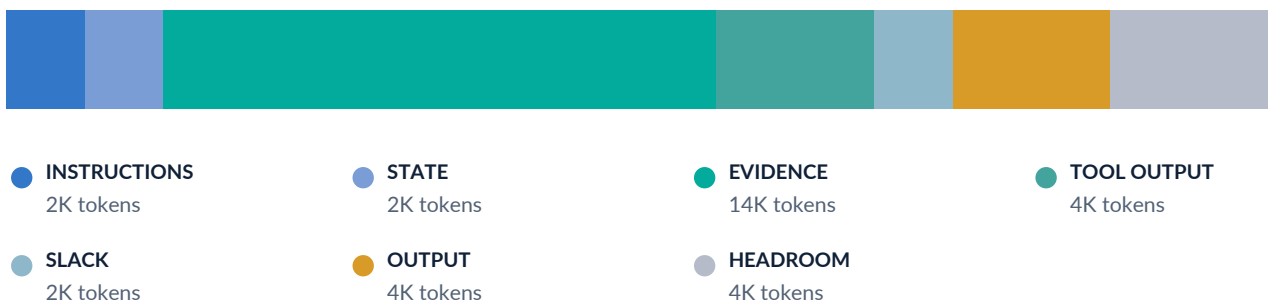
Template tip: preserve exact error strings, timestamps, decisions, and approval requirements verbatim.

MAKE THE TOKEN BUDGET EXPLICIT

Budget the next call

Example only: for a model/request with a **32K total-token budget**, reserve output and headroom before selecting evidence. Actual limits and accounting vary by provider.

ILLUSTRATIVE 32K WINDOW / NOT A RULE OF THUMB



COST IS MORE THAN TOTAL INPUT TOKENS

Approximate cost = uncached input × input rate + cached reads × cache-read rate + cache writes × write rate + output × output rate.

Model and provider pricing vary. Reused prefixes can have different read/write prices and retention rules.

THE OPTIMIZATION ORDER

- 01 Fix evidence quality**
Missing facts cannot be compressed into existence.
- 02 Cut low-value tokens**
Prune duplicate, old, and irrelevant content.
- 03 Exploit safe reuse**
Keep stable prefixes stable. Check cache hit and write cost.
- 04 Measure completed work**
Track cost and latency per **correct** task. Per-call numbers hide retries.

Provider reference: [OpenAI API Prompt Caching docs](#) (rates and retention may change).

DID IT ACTUALLY GET BETTER?

Measure the entire task

A shorter prompt is not a win if it misses evidence, invents a root cause, or triggers an expensive retry.

01 / ANSWER QUALITY

Correctness on held-out real tasks, including conflicting and multi-hop evidence.

02 / EVIDENCE SUFFICIENCY

Is the answer supportable from the selected context? Does it abstain when context is missing?

03 / POSITION SENSITIVITY

Move decisive facts to start, middle, end.
Compare full task success. A needle lookup is too easy to pass.

04 / STATE RETENTION

After compaction, test critical requirements, version IDs, pending steps, and approval constraints.

05 / COST PER SUCCESS

Include retrieval, LLM calls, output, cache read/write charges, retries, and tool costs.

06 / LATENCY & CACHE

Track p95 end-to-end latency, time to first token, and actual cached-token reuse.

RUN A SIMPLE THREE-WAY ABLATION

Variants: A = raw long context | B = retrieved + reranked | C = B + structured state + cache.

Keep fixed: model and test set. Compare correctness, evidence coverage, latency, and full-task cost.

Security rule: retrieved documents and tool outputs are untrusted data, not instructions or permissions.

QUICK REFERENCE

What to build next

Use this as your implementation checklist. You do not need a complex “context framework” to get started.

PRODUCTION CHECKLIST

- ☐ Define what “correct” means for the next task.
- ☐ Retrieve with metadata, version, and access checks.
- ☐ Use lexical + semantic search where it helps.
- ☐ Rerank for coverage and sufficiency.
- ☐ Remove duplicates, not decisive counter-evidence.
- ☐ Pass structured state plus cited excerpts.
- ☐ Preserve IDs to fetch details on demand.
- ☐ Keep stable prefixes stable for caching.
- ☐ Test missing information and position effects.
- ☐ Track cost per successful end-to-end task.

SELECTED SOURCES

Liu et al. (2024)[Lost in the Middle](#)**Anthropic (2025)**[Effective context engineering](#)**Anthropic (2024)**[Contextual Retrieval](#)**Google Research (2025)**[Sufficient Context for RAG](#)**Google Research (2025)**[Retrieval Quality at Context Limit](#)**OpenAI API docs**[Prompt Caching](#)

THE ONE IDEA TO KEEP

Context engineering is a decision. Pick what the model must know **now**, prove it is enough, and keep the rest retrievable.

Research and provider behavior change fast. Benchmark with your own model and workload.

Want to build this properly?

Join the ArchCrux AI Engineering cohort. Live sessions on RAG, AI agents and context engineering, taught design-first by a working architect.

LIVE COHORT · WAITING LIST OPEN

Join the cohort waiting list →

www.archcrux.com/cohort

WRITTEN BY

Amit

Principal Architect. 14 years building distributed systems, now building production AI systems.