LLM Prompt Caching: Cost Savings, Invalidation & Workload Design

IntuitionLabs.ai · Adrien Laurent · 2026-09-05 · prompt caching · llm cost optimization · anthropic claude · openai api pricing · cache invalidation · llm inference cost · context caching · api cost savings · prompt engineering

Executive Summary

Large language model (LLM) prompt caching lets an API provider skip reprocessing a repeated block of prompt text, such as a system prompt, a tool definition list, or a retrieved document, and charge a fraction of the normal token price for it on subsequent requests. As of September 2026, some form of prompt caching is documented by every major hosted LLM provider: Anthropic Claude, OpenAI, Google Gemini, AWS Bedrock, and DeepSeek (docs.claude.com) (platform.openai.com) (cloud.google.com) (docs.aws.amazon.com) (api-docs.deepseek.com). Anthropic charges **1.25 times** the base input price to write a 5-minute cache (**2 times** for an extended 1-hour cache) and **0.1 times** the base price to read from it (docs.claude.com) (docs.claude.com). OpenAI's older models write to cache for free and read at a per-model discount of up to 90% (platform.openai.com) (platform.openai.com), while its newest GPT-5.6 generation adopted Anthropic's exact 1.25x-write, 0.1x-read structure (developers.openai.com) (developers.openai.com). Google Gemini discounts cached tokens by 90% on Gemini 2.5 and newer models, 75% on Gemini 2.0 (cloud.google.com) (cloud.google.com), and DeepSeek's disk-based caching carries no separate write charge at all (api-docs.deepseek.com).

This report answers the target question, how much prompt caching actually saves, with a reproducible worksheet rather than a single headline number: given a cache-write multiplier **w**, a cache-read multiplier **r**, and an expected reuse count **N**, the break-even reuse count is **N* = (w minus r) / (1 minus r)**. For Anthropic's 5-minute default (w equals 1.25, r equals 0.1), that threshold is approximately 1.28, meaning the second reuse of a cached prefix within 5 minutes already saves money. Where a provider charges no write premium at all, OpenAI's older models and DeepSeek, the formula shows there is no break-even threshold: caching has no downside from the very first request (platform.openai.com) (api-docs.deepseek.com).

Dated, sourced evidence backs the savings claims rather than assuming them. Anthropic reported reductions of up to 90% in cost and up to 85% in latency for long, repeated prompts at launch (www.anthropic.com), and its own benchmark table shows a 10-turn conversation with a long system prompt cutting time-to-first-token by roughly 75% and cost by roughly 53% (www.anthropic.com). AWS documented equivalent up-to-85%-latency, up-to-90%-cost reductions for Bedrock-hosted models in an April 2025 engineering post (aws.amazon.com), and DeepSeek reported average historical savings above 50% across its user base (api-docs.deepseek.com). Two independent, peer-reviewed systems papers, Prompt Cache and SGLang's RadixAttention, document the same underlying key-value reuse mechanism achieving up to 60 times faster time-to-first-token and up to 6.4 times higher throughput respectively, though in self-hosted research settings rather than commercial API billing (arxiv.org) (arxiv.org).

Cache invalidation is exact and unforgiving: a single changed character anywhere in the cached prefix, a live timestamp, a non-deterministically serialized tool list, or a rewritten earlier conversation turn, misses the cache entirely rather than degrading gracefully (docs.claude.com) (docs.claude.com) (developers.openai.com). The corresponding design discipline, documented near-identically by Anthropic and OpenAI, is to place all static content first, mark a clear breakpoint before the first varying token, and keep tool schemas and conversation history append-only (docs.claude.com) (developers.openai.com) (developers.openai.com).

Introduction and Background

Large language model (LLM) application programming interfaces (APIs) bill by the token, and for many production workloads the same block of text, a system prompt, a tool definition list, a retrieved document, or the early turns of a conversation, is sent to the model over and over again. **Prompt caching** is the mechanism that lets a provider skip reprocessing that repeated prefix and charge a fraction of the normal price for it instead. As of September 2026, prompt caching (sometimes called context caching) is offered in some form by every major hosted LLM API: **Anthropic** Claude, **OpenAI**, **Google** Gemini, **Amazon** Bedrock, and **DeepSeek** (cloud.google.com) (docs.aws.amazon.com) (api-docs.deepseek.com).

The mechanism is simple to state and easy to misuse. A cache is keyed on an exact, byte-identical prefix of the request, covering, in Anthropic's documented render order, the tool definitions, then the system prompt, then the message history, up to a marked cutoff point (docs.claude.com). If a single character changes anywhere before that cutoff, the cache does not partially apply. It misses entirely, and the request is billed and processed as if caching were not in use. That all-or-nothing behavior is why a workload can be technically eligible for caching and still show a near-zero hit rate in production: a timestamp embedded in a system prompt, a non-deterministically serialized tool schema, or a summarization step that rewrites earlier conversation turns will each silently defeat it (developers.openai.com).

This report is a practical, citation-dense reference for engineers and technical decision-makers evaluating or tuning prompt caching. It walks through the mechanics common to every provider, then documents the pricing and behavior of each major implementation as of the observation dates stated throughout (prices and free-tier terms change and should be re-verified against the cited pages before use). It publishes a reproducible break-even worksheet: the formula and

parameter table needed to calculate, for a specific provider and workload, how many times a prefix must be reused before caching pays for itself. It closes with dated, sourced evidence on measured cost and latency outcomes, and with implementation guidance for designing prompts and conversation structures that sustain a high cache-hit rate. For a broader survey of headline API pricing across these same four vendors, see the existing IntuitionLabs comparison of Grok, Gemini, OpenAI, and Claude pricing (intuitionlabs.ai), which this report deliberately does not duplicate; the focus here is the caching mechanism itself, not headline token prices.

What Is Prompt Caching: Mechanics and Taxonomy

Every provider's implementation reduces to the same three parameters, even though the exact multipliers differ: a **write cost** (what it costs to place a prefix into the cache the first time), a **read cost** (the discounted price charged on a subsequent request that matches the cached prefix), and a **time-to-live (TTL)** (how long the cached entry survives before it must be rewritten). A fourth variable, the **minimum cacheable length**, sets a token-count floor below which a prefix is processed normally and never cached at all (docs.claude.com).

Two broad caching modes exist across the market:

- **Manual (explicit) caching**, pioneered by Anthropic, requires the developer to mark a "cache breakpoint" in the request, a specific content block after which everything preceding it becomes eligible to be cached and reused. Anthropic allows up to **4 breakpoints** per request (docs.claude.com); if all four are already used explicitly, the API returns an error rather than silently adding a fifth (docs.claude.com).
- **Automatic (implicit) caching**, the default on OpenAI's pre-GPT-5.6 models, DeepSeek, and Google Gemini, requires no code change: the provider detects a repeated prefix and applies the discount without the developer marking anything (platform.openai.com) (api-docs.deepseek.com) (ai.google.dev).

The market is converging toward offering both. OpenAI's newest model generation, GPT-5.6 and later, added explicit, developer-placed breakpoints on top of its existing automatic caching, giving finer control at the cost of added complexity (platform.openai.com). Google Gemini likewise exposes both an automatic "implicit" mode and a manually declared "explicit" mode with its own separate storage billing (cloud.google.com) (cloud.google.com).

Minimum cacheable length varies by provider and by model within a provider, generally because larger, more capable models need a longer prefix before the fixed overhead of a cache write is worth incurring. Anthropic sets the floor at **1,024 tokens** for its Sonnet and Opus-class models and **4,096 tokens** for the smaller Haiku 4.5 (docs.claude.com) (docs.claude.com). OpenAI's floor is **1,024 tokens** for GPT-5.6 and newer, **2,048 tokens** for older models (platform.openai.com). AWS Bedrock documents a per-model floor as low as 512 tokens for some Claude models and as high as 4,096 tokens for Claude Haiku 4.5 (docs.aws.amazon.com). Google's Gemini 3 family sets a 4,096-token floor (cloud.google.com), while DeepSeek's floor is far smaller, a 64-token storage unit (api-docs.deepseek.com).

Time-to-live is the other axis providers compete on. Anthropic's default cache lives for **5 minutes** and is refreshed at no extra cost every time it is hit, meaning a steady stream of requests spaced under 5 minutes apart keeps the same cache entry alive indefinitely (docs.claude.com); an extended, more expensive **1-hour** TTL is also available (docs.claude.com). OpenAI documents a similar 5-to-10-minute default inactivity window for its in-memory cache, extendable to about an hour, plus a separate 24-hour extended-retention tier for supported models (platform.openai.com) (platform.openai.com). AWS Bedrock's default is likewise **5 minutes** (docs.aws.amazon.com).

Provider Implementations and Pricing

Table 1 below summarizes the caching mechanics of five major API surfaces as of their respective access dates in September 2026.

Provider	Cache mode	Write cost	Read cost (discount)	Minimum tokens	Default TTL
Anthropic Claude	Manual breakpoints (up to 4)	1.25x base input (5-min); 2x base input (1-hour)	0.1x base input (0.025x on select newest models) (docs.claude.com)	1,024 (Sonnet/Opus-class); 4,096 (Haiku 4.5)	5 min, refreshed free on hit; 1-hour paid option
OpenAI (pre-GPT-5.6)	Automatic (implicit) only	No separate write charge	Up to 90% off, varies by model	2,048	5 to 10 min inactivity, up to 1 hour; optional 24-hour tier
OpenAI (GPT-5.6+)	Automatic plus optional manual breakpoints	1.25x standard input rate	0.1x standard input rate	1,024	Same tiers as above
Google Gemini (Vertex AI / AI Studio)	Automatic ("implicit") by default; manual ("explicit") optional (cloud.google.com)	No separate write charge on implicit; explicit mode adds hourly storage billing (cloud.google.com)	90% off (Gemini 2.5 and newer); 75% off (Gemini 2.0) (cloud.google.com) (cloud.google.com)	4,096 (Gemini 3 family) (cloud.google.com)	Provider-managed; explicit caches billed by declared duration (cloud.google.com)
AWS Bedrock	Implicit and explicit, model-dependent (docs.aws.amazon.com)	Model-specific write premium (explicit mode)	Model-specific read discount	512 to 4,096 depending on model (docs.aws.amazon.com)	5 min default; some models support longer (docs.aws.amazon.com)
DeepSeek	Automatic disk-based caching (api-docs.deepseek.com)	No separate write charge (api-docs.deepseek.com)	Up to roughly 90 to 97% off depending on tier and time of day (api-docs.deepseek.com)	64-token storage unit (api-docs.deepseek.com)	Provider-managed

Prices quoted as multipliers of "base input" or "standard input" refer to each provider's own published per-million-token rate for the model in question; consult each provider's current pricing page for the underlying base rate, since those change independently of the caching multipliers (claude.com) (developers.openai.com) (ai.google.dev) (aws.amazon.com).

Anthropic Claude

Anthropic's cache write premium and read discount apply uniformly across its breakpoint model: a 5-minute cache write costs **1.25 times** the base input token price, while a 1-hour extended write costs **2 times** that price. Every subsequent cache read is billed at **0.1 times** the base input price for most current models, though Anthropic documents an even steeper 0.025x multiplier for a subset of its newest models. Anthropic's own worked framing states the read price is "costing only 10% of the base input token price" (www.anthropic.com), and the company's launch materials reported reductions of "up to 90%" in cost and "up to 85%" in latency for long, repeated prompts (www.anthropic.com). Usage

accounting is exposed through three response fields, `cache_creation_input_tokens`, `cache_read_input_tokens`, and `input_tokens` for tokens outside any cache, which is the primary diagnostic for confirming a workload is actually hitting cache in production (docs.claude.com) (docs.claude.com).

OpenAI

OpenAI's caching has historically been fully automatic and free to write: for models before GPT-5.6, there is **no additional cache-write charge**, and cached tokens are simply billed at a per-model discounted rate. That changed with GPT-5.6 and later, which introduced both a manual breakpoint option and, notably, a cache-write charge of **1.25 times** the standard input rate, with reads discounted to **0.1 times** that rate, mirroring Anthropic's multipliers almost exactly. OpenAI's documentation walks through the resulting economics directly: writing a prefix once and reusing it once in full costs 1.35 times the ordinary input cost, against 2 times for processing the same content twice without caching (developers.openai.com).

Google Gemini

Google's Vertex AI and Gemini API both default to **implicit caching**, enabled automatically with no code change and no separate storage fee (cloud.google.com) (ai.google.dev) (cloud.google.com). The discount is **90%** on cached tokens for Gemini 2.5 and newer models, dropping to **75%** for the older Gemini 2.0 family (cloud.google.com). Developers who need guaranteed cache placement rather than best-effort automatic caching can instead declare an **explicit cache**, which persists for a developer-chosen duration but is billed separately for storage, unlike the free-by-default implicit mode.

AWS Bedrock

Bedrock supports both implicit and explicit prompt caching, with availability and minimum token counts that vary by the specific model hosted on the platform, including Anthropic Claude, Amazon Nova, and, through the Responses API, OpenAI's own models (docs.aws.amazon.com) (docs.aws.amazon.com) (docs.aws.amazon.com). Absent an explicit setting, Bedrock applies **the default 5-minute caching behavior**, matching Anthropic's own default when Claude models are served directly (docs.aws.amazon.com).

DeepSeek

DeepSeek's "Context Caching on Disk" is automatic, requires no code or interface changes, and, distinctively among the providers surveyed, carries **no storage fee** for the cached content itself (api-docs.deepseek.com) (api-docs.deepseek.com). DeepSeek's original announcement stated the technology could cut API costs by up to 90% on cache hits (api-docs.deepseek.com), and the company later reported that, across its user base, historical (pre-optimization) data showed average savings of over 50% (api-docs.deepseek.com). The minimum cacheable unit is small relative to other providers, 64 tokens (api-docs.deepseek.com), which makes caching viable for shorter, more granular repeated fragments than Anthropic's or OpenAI's thousand-token-plus floors allow.

Cache Invalidation: What Breaks a Cache Hit

Because a cache hit requires an exact match, invalidation is best understood not as a decay process but as a binary trigger: something in the prefix changed, so the hash of that prefix changed, so the lookup misses. Anthropic's documentation gives a concrete illustration of this: inserting a live timestamp into a system prompt means "the

timestamp differs, so the prefix hash at block 6 differs" on every single request, permanently defeating caching for anything after that point. The same logic explains several other common, easy-to-miss invalidators:

- **Non-deterministic serialization.** Some programming languages randomize the key order when converting a data structure to JSON; if tool definitions are serialized this way, the resulting text differs byte-for-byte between otherwise-identical requests and the cache misses every time.
- **Any change to tool definitions.** Anthropic documents that modifying a tool's name, description, or parameters invalidates the entire cache, not just the portion after the tool block, because tools render first in the prompt (docs.claude.com).
- **Inconsistent auxiliary settings.** Anthropic's own troubleshooting guidance instructs developers to verify that `tool_choice`, image usage, thinking configuration, and output effort settings stay identical between calls, since drift in any of them can invalidate a cache that looks unchanged in its visible text (docs.claude.com).
- **Rewriting conversation history.** OpenAI's guidance is explicit that a cache requires the entire rendered prefix to match, and that developers should "append new messages rather than rewriting earlier turns," since any summarization or compaction step that edits prior turns breaks the match for everything that follows (developers.openai.com).
- **A shared prefix without a breakpoint.** OpenAI separately warns that "a shared prefix is not always a cached prefix": if the static content is not followed by an explicit or automatically detected breakpoint before the varying content begins, no caching occurs even though the text is technically repeated.

Anthropic's matching engine also has a bounded lookback: it checks at most 20 prior block positions per breakpoint when looking for a matching prior cache write (docs.claude.com), meaning an unusually long single turn inserted between otherwise-repeated blocks can push the earlier cache entry out of the window the API is willing to search. Diagnosing a suspected invalidation problem in production, on either Anthropic or OpenAI, comes down to reading the usage fields on the response: Anthropic exposes `cache_read_input_tokens` directly, and OpenAI exposes the equivalent `cached_tokens` and `cache_write_tokens` fields under `usage.input_tokens_details`. A hit rate that unexpectedly drops to zero across otherwise-identical requests is close to always one of the causes above, not a platform outage.

Designing Workloads for Prompt Caching

The practical design goal is to place everything stable, tool definitions, system instructions, retrieved documents, few-shot examples, as early in the prompt as possible, and to push everything that varies per request, the user's specific question, a session ID, a live timestamp, as late as possible, ideally after the last cache breakpoint. Anthropic states this directly: "Place static content (tool definitions, system instructions, context, examples) at the beginning of your prompt," with the breakpoint marking the boundary "at the end of the static prefix, not on the varying block" (docs.claude.com). OpenAI's guidance is functionally identical: "Put stable developer instructions and shared reference material first," with implicit caching automatically placing its breakpoint "at the end of the latest eligible message" (platform.openai.com). For workloads that use a fixed evaluation rubric or reference examples ahead of a per-request input, such as an LLM-based grading or classification pipeline, OpenAI's own worked example places "the fixed rubric and examples" first, with only the item being evaluated coming last, unchanged (developers.openai.com). Because caching removes the per-request cost penalty of a long, example-heavy prompt, Anthropic separately recommends that developers no longer economize on few-shot examples and instead include "20+ diverse examples of high quality answers" once that content is cached (docs.claude.com). Keeping tool schemas byte-stable between calls, rather than swapping which tools are offered per request, is the corresponding OpenAI recommendation: "Preserve tool definitions, ordering, and schemas," using

`tool_choice` to restrict what the model may call instead of removing tool definitions outright. Where OpenAI's caching key is influenced by an application-supplied `prompt_cache_key`, the same stability principle applies: "Reuse the key while its prefix remains useful," rather than generating a new key on every request (developers.openai.com).

A reproducible break-even worksheet

Whether caching a given prefix is worth doing at all is a function of exactly four numbers: the write multiplier **w** (the cache-write price as a multiple of the base input price), the read multiplier **r** (the cache-read price as a multiple of the base input price), and how many times **N** the prefix will actually be reused within the cache's TTL window before it expires or the workload ends. Without caching, N uses of a T-token prefix cost N x T x (base price). With caching, the same N uses cost T x (base price) x [w + (N-1) x r], one write followed by (N-1) discounted reads. Setting the two equal and solving for the break-even reuse count gives:

N* = (w minus r) divided by (1 minus r)

Any workload expected to reuse a prefix more than N* times within the TTL window saves money by caching it; fewer reuses than that, and caching costs more than not caching. *Table 2* applies this formula to the multipliers documented above.

Provider / tier	Write multiplier (w)	Read multiplier (r)	Break-even reuse count (N*)	Interpretation
Anthropic, 5-minute default	1.25	0.10	approximately 1.28	Pays off on the second use of the prefix within 5 minutes
Anthropic, 1-hour extended	2.00	0.10	approximately 2.11	Breaks even against uncached input on the third total request (one cache write and two cache reads). Against the 5-minute tier, compare request timing and the number of 5-minute cache rewrites.
Anthropic, newest 0.025x-read models	1.25	0.025	approximately 1.26	Marginally better break-even than the standard 0.1x read tier
OpenAI, pre-GPT-5.6	1.00 (no write premium)	Model-dependent (up to 0.90 off, i.e. as low as 0.10)	1.00	No threshold: because there is no write premium, even a single reuse is pure savings, with zero downside to caching by default
OpenAI, GPT-5.6+	1.25	0.10	approximately 1.28	Matches Anthropic's 5-minute tier almost exactly
DeepSeek	1.00 (no write premium)	approximately 0.03 to 0.10 depending on tier (api-docs.deepseek.com)	1.00	Same no-downside property as fee-free-write providers

Table 2 shows that the presence or absence of a write premium, not the size of the read discount, is what determines whether a workload needs to think carefully about reuse volume at all. Where a provider charges nothing extra to write the cache (OpenAI's older models, DeepSeek), there is mathematically no scenario in which turning caching on costs more than leaving it off, since the worst case is simply paying the normal, uncached price on the first use. Where a provider charges a write premium (Anthropic on every tier, OpenAI's GPT-5.6 generation), a one-off request that is never repeated within the TTL window is a net loss, typically a small one (25 to 100 percent more than the uncached price),

and the practical implication for workload design is to reserve manual cache breakpoints for content that is genuinely known to recur, such as a fixed system prompt or a document that will be queried multiple times in a session, rather than caching indiscriminately. Because OpenAI's own documentation independently derives the same "1.35x for one reuse versus 2x uncached-twice" result for its GPT-5.6 pricing, this formula can be checked against a live vendor's own numbers rather than taken only on the analysis in this report. Readers can reproduce the worksheet by substituting their own provider's w and r values (from *Table 1* or a current pricing page) and their own expected reuse count into the same N* formula.

Data Analysis and Evidence

Methodology. The figures below are drawn from each provider's own published benchmark or case-study material, from one third-party cloud provider's engineering blog (AWS, publishing measurements for models it hosts on Bedrock), and from two independent peer-reviewed systems papers studying cache-based inference more generally. None of the percentage or multiplier figures below were generated by this report; each is attributed to the page or paper that published it, and vendor-reported figures are explicitly flagged as such rather than presented as independent measurements.

Table 3 collects the dated, sourced quantitative results located during research for this report.

Source	Scenario	Latency change	Cost change	Date
Anthropic	Long, repeated prompts (general)	Up to 85% lower	Up to 90% lower	2026 access
Anthropic benchmark table (anthropic.com)	Many-shot prompting, approximately 10,000-token prompt	Time to first token down approximately 31%	Approximately 86% lower	2026 access
Anthropic benchmark table (anthropic.com)	10-turn conversation with a long system prompt	Time to first token down approximately 75%	Approximately 53% lower	2026 access
AWS (Bedrock) (aws.amazon.com)	General guidance across supported Bedrock models	Up to 85% lower	Up to 90% lower	2025-04-07
DeepSeek (api-docs.deepseek.com)	Aggregate historical usage across DeepSeek's user base, pre-optimization	Not stated	Over 50% lower on average	2024-08-02
Prompt Cache (academic, Gim et al.) (arxiv.org)	Reused attention state across repeated prompt schemas	Time to first token 8x faster (GPU), up to 60x faster (CPU)	Not directly costed	arXiv preprint, 2023
SGLang / RadixAttention (academic) (arxiv.org)	Automatic KV-cache reuse across agent, reasoning, few-shot, RAG, and multi-turn workloads	Not directly stated	Up to 6.4x higher throughput, which lowers effective per-request cost on self-hosted infrastructure	arXiv preprint, 2023

The two academic results are conceptually adjacent to, but methodologically distinct from, the hosted-API prompt caching described elsewhere in this report: both papers study reusable key-value (KV) attention state in self-hosted or research inference stacks rather than a commercial billing feature, and their throughput and latency multipliers should not be read as applicable one-for-one to Anthropic's, OpenAI's, or Google's hosted pricing. They are included because

they document the same underlying mechanism, avoiding redundant computation over a previously seen prefix, from an independent, peer-reviewed angle rather than a vendor's own marketing material. AWS Bedrock's own worked example is more directly comparable to hosted-API use: a single illustrative request in its documentation split into 1,038 cache-read tokens against 37,888 cache-write tokens (aws.amazon.com), a ratio that itself demonstrates why a large, rarely changing document (the 37,888-token write) paired with small, repeated instructions (the 1,038-token read) is close to the ideal caching shape: the expensive write happens once, and the far smaller repeated fragment is what actually gets billed at the discount on every subsequent call.

One important caveat about hit-rate figures specifically: OpenAI's documentation presents illustrative cache-hit-rate numbers, "around 70%" for a single-turn design and "above 90%" for a multi-turn agent design, but explicitly labels the 70% figure as "a hypothetical figure, not a measured deployment result" (developers.openai.com). This report treats that distinction as material and does not present either number as an observed production statistic; readers should measure their own hit rate from their own usage fields rather than benchmark against an illustrative OpenAI example.

Case Studies and Real-World Examples

Notion. In Anthropic's own launch material for prompt caching, Notion is named as an early adopter, with a co-founder quoted directly: the company was "excited to use prompt caching to make Notion AI faster and cheaper" (www.anthropic.com). This is a vendor-published customer endorsement rather than an independently measured case study, and no specific percentage figure is attributed to Notion in the source; it is included here as a documented, named instance of production adoption rather than as a quantified result.

A retrieval-heavy support workload on AWS Bedrock (Hypothetical Example). To illustrate how the write/read split in *Table 3* plays out end to end, consider a customer-support assistant that loads a 38,000-token product manual as context once per session and then answers a sequence of short follow-up questions against it. Using AWS's own published token counts for a comparable request shape, 37,888 write tokens against 1,038 read tokens per follow-up (aws.amazon.com), a session with ten follow-up questions would write the manual into cache once and read the 1,038-token instruction block nine more times at the discounted rate, illustrating the same break-even logic developed in *Table 2*: the single write is recovered after roughly the second reused request, and every question after that is billed at the provider's read discount rather than the full input price for the manual.

Implications and Future Directions

Three trends are visible across the providers surveyed in this report. First, **write pricing is converging rather than diverging**: OpenAI's newest model generation adopted the same 1.25x-write, 0.1x-read structure Anthropic has used since its own caching launch, a specific and unusual coincidence in the exact multipliers chosen by two competing vendors. Second, **manual and automatic caching are becoming complementary rather than competing designs**: Google, AWS Bedrock, and now OpenAI's newest models all expose both an automatic, best-effort mode and a manual, developer-controlled mode within the same API, rather than committing to one design philosophy. Third, **the write-fee-free implementations (DeepSeek, and OpenAI's pre-5.6 model line) demonstrate that a write premium is a pricing choice, not a technical necessity**: nothing about the underlying key-value cache reuse mechanism requires charging more for the first write, and providers that skip the premium remove the break-even calculation in *Table 2* entirely, since caching then has no scenario in which it costs more than not caching.

For organizations designing new LLM-backed workflows, particularly in regulated or evidence-heavy domains such as life sciences, where a governed information layer, standard operating procedures, and long reference documents recur constantly across many similar queries, the practical implication is that prompt structure is now a cost-engineering decision, not just a prompt-quality one. IntuitionLabs, an AI and life-sciences consultancy founded in 2023 (intuitionlabs.ai), works with pharmaceutical and biotech organizations on exactly this kind of governed, high-reuse AI workflow design, where a stable, cacheable foundation of retrieved company documents and standard instructions sits ahead of a varying, per-user query; from that vantage point, the single most common mistake observed across teams new to prompt caching is treating it as an automatic optimization rather than a prompt-architecture discipline that has to be designed in from the start, since retrofitting a volatile element (a live timestamp, a per-user greeting, a randomly ordered tool list) out of an already-shipped prompt is far more disruptive than avoiding it during the initial design. Looking forward, the direction indicated by the GPT-5.6 changes documented in this report, adding manual control on top of an automatic default rather than replacing it, suggests that fine-grained, developer-specified caching will keep expanding across the market even as automatic caching remains the default for developers who do not need that level of control.

Frequently Asked Questions (FAQs)

How does LLM prompt caching work? A provider hashes the prefix of a request (in Anthropic's case, the tool definitions, system prompt, and messages up to a marked point) and stores the model's internal representation of that prefix. If a later request has an exactly matching prefix, the provider skips reprocessing it and serves the cached representation at a steep discount; if even one character differs, the request is processed and billed as a normal, uncached call (docs.claude.com).

What are the actual cost savings from prompt caching? They depend entirely on how many times a prefix is reused within the cache's TTL window. Vendor-reported figures range from roughly 50% average savings across DeepSeek's user base historically to up to 90% for long, heavily repeated prompts on Anthropic and AWS Bedrock (aws.amazon.com). Table 2 in this report gives the exact formula for calculating expected savings for a specific workload rather than relying on a single headline percentage.

How does OpenAI's prompt caching compare with Anthropic's? Anthropic has always charged a write premium (1.25x for a 5-minute cache) and a fixed 0.1x read discount, using manually placed breakpoints. OpenAI's older models charge no write premium at all and cache automatically, while OpenAI's newest GPT-5.6 generation added both manual breakpoints and Anthropic's exact 1.25x/0.1x write/read structure, making the two vendors' newest tiers functionally identical in their pricing mechanics even though their older tiers differ substantially.

What are the most common cache invalidation mistakes? Embedding a live timestamp or other per-request value inside the cached portion of a system prompt, serializing tool definitions non-deterministically, changing any tool's name, description, or parameters, and rewriting or summarizing earlier conversation turns instead of appending to them are the invalidators documented directly in Anthropic's and OpenAI's own troubleshooting guidance.

How can prompt cache hit rate be optimized? Place all static content, tool definitions, system instructions, and reference documents, before any per-request variable content, and mark the breakpoint at the boundary between the two; keep tool schemas and their ordering byte-identical between calls; append new conversation turns rather than editing earlier ones; and monitor the provider's own usage fields, such as Anthropic's `cache_read_input_tokens`, rather than assuming caching is active just because the code compiles.

Besides prompt caching, what other techniques reduce LLM inference costs? Batch processing APIs from both Anthropic and OpenAI offer a flat 50% discount in exchange for asynchronous, non-real-time processing, typically completed within 24 hours (platform.claude.com) (platform.claude.com) (platform.openai.com). Choosing a smaller model for a given task, and, once accuracy is validated, distilling a smaller model from a larger one's outputs, are OpenAI's own documented model-selection cost levers, entirely independent of caching (developers.openai.com) (developers.openai.com). For self-hosted deployments, quantization, reducing the numerical precision of model weights, is a widely studied technique in the academic literature for lowering inference compute and memory cost (arxiv.org).

Conclusion

Prompt caching has moved, over roughly two years, from an Anthropic-specific feature into a near-universal capability across hosted LLM APIs, with Anthropic, OpenAI, Google, AWS Bedrock, and DeepSeek each documenting their own version of the same core mechanism as of September 2026. The economics are governed by three numbers that any team can look up for their own provider and model: the write multiplier, the read multiplier, and the number of times a prefix will realistically be reused before it expires. Where a provider charges no write premium, caching has no downside and should generally be left on. Where a write premium applies, the break-even formula in this report, N* equals (w minus r) divided by (1 minus r), gives a specific, reproducible threshold rather than a rule of thumb. The larger design lesson is structural: caching rewards prompts built with a clear separation between what is stable and what varies, and it punishes, silently and completely, any prompt that mixes the two without a clean boundary. Teams that design for that separation from the outset, rather than retrofitting it later, are the ones positioned to capture the 50 to 90 percent cost reductions documented throughout this report.

For informational purposes. Verify critical medical, legal, financial, regulatory, and technical information with qualified professionals and primary sources.