# Context as Material: A Theoretical and Computational Framework for Large Language Model Systems

## Abstract

The paradigm shift in software engineering precipitated by Large Language Models (LLMs)—often characterized as "Software 3.0"—necessitates a fundamental ontological re-evaluation of the medium in which computation occurs. No longer defined by deterministic logic gates or explicit instruction sets, the primary unit of computation has shifted to the "token," and the primary workspace to the "context window." This report proposes the "Context as Material" framework, which posits that the context window must be treated not as an abstract, infinite canvas for text, but as a finite physical resource governed by distinct material properties, thermodynamic-like laws of entropy, and rigorous economic constraints. By synthesizing foundational anthropological theories—specifically Clifford Geertz’s "thick description," Lucy Suchman’s "situated actions," and Bruno Latour’s "Actor-Network Theory"—with contemporary empirical research on "context rot," "lost-in-the-middle" phenomena, and "agentic context engineering," we establish a hierarchical taxonomy of context management. This hierarchy, conceptualized as the **Material Pyramid**, spans five distinct strata: **Foundations**, **Material**, **Physics**, **Architecture**, and **Economics**. Through this lens, we argue that the efficacy of autonomous agents is bounded by the physical dynamics of their context, requiring a transition from the linguistic art of "prompt engineering" to the material science of "context engineering."

## I. Introduction

The history of computer science is largely a history of abstraction. From vacuum tubes to transistors, from assembly language to object-oriented programming, the trajectory has been one of distancing the developer from the physical constraints of the hardware. However, the advent of Large Language Models (LLMs) and the subsequent rise of Generative AI have reversed this trend, reintroducing a form of stubborn materiality to the forefront of software development. In this new era, often termed "Software 3.0" by Andrej Karpathy, the neural network acts as the central processing unit, but the "code" is no longer a static binary; it is a fluid, probabilistic stream of natural language tokens fed into a context window.1

This report argues that the "context window" is not merely a data buffer. It is a material substance that underpins the cognitive capabilities of the artificial agent. Like any material, it occupies space (VRAM), possesses mass (latency), generates friction (inference cost), and is subject to degradation (context rot).3 The prevailing view of context as an ephemeral, abstract "prompt" is insufficient for building robust, autonomous systems. As agents are tasked with increasingly complex, long-horizon workflows—debugging codebases, managing financial portfolios, or navigating legal precedents—the management of this material becomes the defining engineering challenge.

We introduce the "Context as Material" framework to formalize this challenge. Structured as a pyramid, this framework grounds the ephemeral nature of language in the concrete realities of computation. At the base, we find the **Foundations**, borrowing from anthropology and sociology to define what constitutes "meaningful" context in a socio-technical system. Above this lies the **Material**, the raw technical substrate of tokens and Key-Value (KV) caches that physically embody memory.5 This material is governed by the **Physics** of the transformer architecture, which dictates how attention flows, accumulates, or dissipates across the sequence length.6 The **Architecture** layer describes the systems—such as Agentic Context Engineering (ACE) and Dynamic Cheatsheets—designed to structure and refine this material.7 Finally, the **Economics** layer analyzes the cost-value optimization of context as a scarce, tradable asset.8

This report provides an exhaustive analysis of each layer, weaving together theoretical insights with empirical data to offer a comprehensive guide for the design of next-generation AI systems.

## II. Foundations: The Anthropological Substrate

To engineer context effectively, one must first understand what context *is*. In computer science, context is often reduced to "state"—the values of variables at a given time. However, for an LLM effectively simulating human reasoning, "state" is insufficient. We must turn to the "Foundations" layer of the Material Pyramid, which draws upon the rich traditions of anthropology, ethnography, and Science and Technology Studies (STS) to define the semiotic density required for intelligence.

### A. Thick Description and Semiotic Density

The challenge of prompting an LLM to perform a complex task is structurally identical to the challenge of an ethnographer attempting to understand a foreign culture. In his 1973 seminal work, *The Interpretation of Cultures*, Clifford Geertz introduced the distinction between "thin description" and "thick description".9 Geertz used the example of a wink: physically, it is merely a contraction of the eyelid (thin description). However, as a social act, it could be a conspiratorial signal, a mockery, a twitch, or a practice run. To understand the wink, one must describe the "web of significance"—the complex, layered hierarchies of meaning—that surround the physical action.11

In the domain of LLMs, a "thin description" corresponds to a naive, zero-shot prompt: "Translate this text" or "Fix this bug." Such prompts provide the model with the raw signal but lack the semiotic scaffolding necessary to disambiguate intent. The model, trained on the vast, contradictory "culture" of the internet, requires "thick description" to narrow its probabilistic output space to a desired behavior.12 Thick description in context engineering involves layering the prompt not just with instructions, but with provenance, constraints, emotional tone, relational history, and "emic" (insider) perspectives.9

The "Context as Material" framework posits that this "thickness" is a quantifiable property of the context material. Just as a physical material must have sufficient density to support a load, the context must have sufficient "semiotic density" to support complex reasoning.13 Geertz’s assertion that culture is "semiotic" implies that the context window is a space where signs (tokens) interact to produce meaning.9 If the context is too "thin," the semiotic reactions fail to ignite, leading to generic or hallucinated outputs—the AI equivalent of mistaking a wink for a twitch.

### B. Situated Actions and the Failure of Planning

The necessity of dynamic context is further illuminated by Lucy Suchman’s theory of "situated actions," presented in *Plans and Situated Actions* (1987).14 Suchman critiqued the classical AI view that intelligent behavior results from the execution of a pre-formulated plan. Instead, she argued that plans are merely resources for action, but the action itself is "situated"—it emerges from the immediate, improvised interaction between the actor and their environment.14

This distinction is critical for the design of autonomous agents. A rigid "System Prompt" acts as a plan—a static prescription of behavior. However, as the agent interacts with a user or a tool, the "situation" evolves. If the agent relies solely on the static plan (the system prompt), it fails to adapt to the contingencies of the moment.16 Suchman’s work suggests that context is not a static background but an active, fluid constituent of intelligence. The "material" of context must be plastic; it must be capable of being reshaped in real-time by the "situatedness" of the interaction.17

This theoretical stance underpins modern architectures like "Dynamic Cheatsheets" and "Agentic Context Engineering" (ACE), which reject static documentation in favor of context that updates based on the program state.18 These systems acknowledge that intelligence is not "inside the skull" (or the weights) but emerges from the "intra-action" of the agent and its context.17

### C. Actor-Network Theory and Material Semiotics

The agency of the context window itself is best understood through the lens of Bruno Latour’s Actor-Network Theory (ANT).20 ANT posits that agency is not a unique property of humans but is distributed across a heterogeneous network of human and non-human "actants." In an LLM system, the non-human actants—the specific tokens in the prompt, the retrieval algorithm, the KV cache, the GPU memory—are not passive tools. They actively mediate the interaction, enabling or constraining specific outcomes.22

Latour argues that we must treat these non-human actors with "analytical equality".22 In the context of "Context Rot" (discussed in the Physics layer), a "distractor" token is a hostile actant; it actively works to disrupt the network of meaning.3 Conversely, a well-engineered "few-shot example" is an ally, stabilizing the network. The act of context engineering is thus a political act of "parliamentary" assembly: deciding which actants (information chunks) are allowed representation in the "parliament of things" (the context window).21

Furthermore, the concept of "material-semiotics," developed by Donna Haraway and Karen Barad, bridges the gap between the physical and the conceptual.23 Barad’s notion of "intra-action" posits that distinct entities do not precede their interaction; rather, they emerge *through* their entanglement.25 Applied to LLMs, this implies that the "knowledge" of the model is not a static retrieval from a database. It is intra-actively produced at the moment of inference by the entanglement of the model’s weights (past) and the context window (present). The engineer, by defining the "agential cut" (what is included or excluded from context), literally constructs the reality in which the agent exists.27

## III. Material: The Substance of Computation

Having established the theoretical necessity of context, we descend the pyramid to the "Material" layer. Here, we strip away the metaphors of "mind" and "language" to examine the raw, physical substance that enables LLM cognition. In the "Context as Material" framework, context is a tangible resource stored in High-Bandwidth Memory (HBM), occupying physical space and consuming physical energy.

### A. The Materiality of Tokens and the KV Cache

The atomic unit of this material is the "token"—a discrete chunk of information that serves as the fundamental currency of the LLM economy. However, during the inference process, tokens are transmuted into a much heavier, more cumbersome substance: the Key-Value (KV) cache.5

In the Transformer architecture, the attention mechanism calculates the relationship between the current token and every previous token in the sequence. To generate the $N+1$ token, the model needs the Key (K) and Value (V) matrices for tokens $1$ through $N$. Without caching, the model would be forced to recompute these matrices from scratch at every single step, leading to a quadratic increase in computational cost ($O(N^2)$) and making real-time generation impossible.29

The KV cache serves as the "physical memory" of the LLM. It is a dedicated allocation of VRAM (Video RAM) on the GPU where these intermediate representations are stored.5 This cache is the material embodiment of the "context window." It has a definite mass, measured in gigabytes. For example, a 175-billion parameter model with a long context window (e.g., 128k tokens) can generate a KV cache so massive that it exceeds the memory capacity of a single A100 GPU.28

This materiality introduces the "Memory Wall"—a bottleneck where the speed of inference is limited not by the speed of computation (FLOPS) but by the bandwidth available to move this massive KV material between memory and the compute cores.28 We observe here a direct physical manifestation of Haraway’s material-semiotics: the "meaning" (semiotics) the model can generate is strictly bounded by the physical "matter" (VRAM) available to store its history.23 If the material storage runs out, the context is truncated, and the "web of significance" is severed.

### B. The Context Window as RAM: The LLM OS

Andrej Karpathy crystallizes this material view by analogizing the LLM to a Central Processing Unit (CPU) and the context window to Random Access Memory (RAM).1 In this "LLM OS" paradigm, the Large Language Model is the kernel—the core processing engine. The context window is the working memory where the active state of the application is loaded.1

Just as a traditional Operating System (OS) manages RAM to allow multiple applications to run without crashing, an "LLM OS" must manage the context window. It must decide what data to "page in" (retrieve from long-term storage/RAG), what to "page out" (summarize or discard), and how to organize the memory space to prevent "segmentation faults" (hallucinations or context overflow).1

This analogy elevates "Context Engineering" from a linguistic art to a systems engineering discipline. It is akin to memory management in classical computing. The context engineer is responsible for "malloc" (memory allocation) and "garbage collection" (pruning irrelevant tokens). The finite size of the context window (e.g., 8k, 32k, 128k tokens) acts as the physical constraint of the RAM stick. If the engineer fills the RAM with "bloatware" (verbose, irrelevant text), the system thrashes; it spends all its compute cycles processing noise rather than generating signal.31

### C. Software 3.0: Vibe Coding and Natural Language Programming

This material shift marks the transition to "Software 3.0." Software 1.0 was explicit code (C++, Python) written by humans. Software 2.0 was neural networks (weights) learned from data, opaque to human inspection. Software 3.0, as defined by Karpathy, is the programming of these neural networks using natural language prompts.1

In Software 3.0, the prompt *is* the code. However, unlike the rigid syntax of C++, this code is "soft" and probabilistic. Karpathy refers to this as "vibe coding"—programming through intent and persona rather than explicit logic.1 But this softness does not imply a lack of structure. The "Context as Material" framework argues that Software 3.0 requires even *more* rigorous structural engineering because the material (language) is inherently unstable. The engineer must build "containment structures" within the prompt—schemas, delimiters, and few-shot examples—to constrain the probabilistic "fluid" of the model into reliable execution paths.31

## IV. Physics: The Dynamics of Context

If context is a material, it follows that it must be subject to physical laws. It does not behave uniformly; it exhibits varying density, decay, interference patterns, and gravitational pulls. Recent empirical research has uncovered specific "physics" governing how LLMs interact with information distributed across this material substrate.

### A. The "Lost in the Middle" Phenomenon

One of the most significant discoveries in the physics of context is the "U-shaped" performance curve, widely known as the "Lost in the Middle" phenomenon.6 Research by Liu et al. (2023) demonstrates that LLMs are not isotropic in their attention capabilities. When presented with a long sequence of information (e.g., a "haystack" of documents), the model's ability to retrieve a specific fact (the "needle") depends heavily on its position.6

* **Primacy Bias:** Information placed at the very beginning of the context window is retrieved with high accuracy. The model’s attention mechanism seems to "anchor" on the initial tokens, treating them as foundational instructions.
* **Recency Bias:** Information at the very end of the context—closest to the generation point—is also highly salient. These tokens are "fresh" in the recurrence of the autoregressive process.
* **The Trough of Ignorance:** However, information buried in the middle of a long context sequence suffers from significant retrieval degradation. The performance curve dips dramatically in the center, forming a "U" shape.

This phenomenon suggests that the "material" of context has variable density. The edges are dense and solid, providing reliable support for reasoning, while the center is porous and unstable.32 This finding challenges the naive assumption that "more context is better." Simply extending the context window (e.g., to 1 million tokens) does not guarantee that the model can effectively utilize that space.33 The physics of the attention mechanism—specifically the accumulation of attention scores and the limitations of Rotary Positional Embeddings (RoPE)—creates a "fog of war" in the middle of the sequence, where the signal-to-noise ratio drops precipitously.34

### B. Context Rot and Entropy

Complementing the positional bias is the concept of "Context Rot," a term coined to describe the degradation of model performance as the volume of irrelevant information increases.3 Just as physical materials degrade due to entropy, the semantic integrity of the context window degrades as "noise" accumulates.

Experiments by Chroma Research (2025) reveal that this rot is not linear. They evaluated 18 state-of-the-art LLMs using a "Needle in a Haystack" (NIAH) setup with varying degrees of complexity.3 The findings were stark:

* **Distractor Interference:** Adding "distractors"—related but irrelevant information—disproportionately impairs the model's ability to retrieve the target information. The presence of distractors doesn't just dilute the signal; it actively confuses the attention heads, leading to "hallucinations" (confident errors) or "refusals" (conservative abstentions).3
* **Semantic Rot:** The rot is most severe when the distractors are semantically similar to the needle. This suggests an interference pattern where the vector representations of similar concepts overlap, making it difficult for the model to resolve the correct entity.4
* **Shuffled vs. Ordered:** Surprisingly, in some cases, *shuffled* (incoherent) contexts performed better than logically ordered ones.35 This counter-intuitive finding suggests that models may over-attend to the "narrative flow" of ordered text, getting lost in the story, whereas shuffled text forces a more mechanical, keyword-based retrieval.

The physics of context rot implies a **Second Law of Context Dynamics**: in a closed context window (like a long conversation), the ratio of useful information (signal) to useless information (noise) tends to decrease over time. Without active external work (energy input in the form of curation and pruning), the system will inevitably devolve into a state of maximum entropy, characterized by hallucinations and incoherence.3

### C. The Non-Uniformity of Attention

The underlying mechanism driving these phenomena is the non-uniformity of the attention mechanism. While theoretically, the Transformer architecture allows every token to attend to every other token ($O(N^2)$ complexity), in practice, the learned attention weights are not distributed evenly.29 The model learns heuristics during pre-training that prioritize certain positions and token types.

This creates "gravity wells" in the context window. The system prompt (start) and the user query (end) act as massive bodies that warp the attention field, pulling the model's focus. The "middle" tokens, lacking this gravitational pull, drift in a low-attention void. Understanding these physics is crucial for the "Architecture" layer. Engineers cannot simply dump data into the context window; they must structure it to exploit the gravitational pull of the start and end tokens, placing critical instructions and schemas where the physics of the model will naturally highlight them.32

## V. Architecture: Structuring the Material

Recognizing the material properties and physical constraints of context, we move to the **Architecture** layer. This layer deals with the design of systems that can engineer context to maximize utility and minimize rot. The transition here is from the static art of "prompt engineering" to the dynamic systems engineering of "Agentic Context Engineering."

### A. From Prompting to Agentic Context Engineering (ACE)

Traditional prompt engineering treats the prompt as a static artifact—a "spell" cast once to invoke a result. However, for autonomous agents operating over long time horizons, static prompts fail due to the "situated" nature of the task and the inevitable accumulation of entropy.36 The **Agentic Context Engineering (ACE)** framework, introduced by Zhang et al. (2025), represents the architectural response to these challenges.7

ACE treats context not as a static text but as an evolving "playbook." It identifies two primary failure modes in traditional context management:

1. **Brevity Bias:** When context is summarized to save space, the model tends to discard nuanced, domain-specific details in favor of generic abstractions. This "thinning" of the description leads to a loss of competence.19
2. **Context Collapse:** Repeatedly rewriting or summarizing the context history leads to a degradation of information fidelity, similar to the generation loss in repeatedly copying a JPEG image. Over time, the context "collapses" into a noisy, low-fidelity state.19

### B. The Generator-Reflector-Curator Triad

To solve these problems, ACE introduces a modular architecture that separates the cognitive functions of the agent into three distinct roles, creating a "self-improving loop".7

1. **The Generator:** This module is responsible for producing candidate strategies and reasoning paths based on the current context. It generates the "forward pass" of action, proposing how to handle a new query or task.38
2. **The Reflector:** This module acts as the "critic." It analyzes the outcomes of the Generator's actions, distinguishing between success and failure. The Reflector applies a critical gaze to the agent's own history, extracting "lessons learned." This is a computational implementation of Donald Schön’s concept of "reflection-in-action"—the ability of a practitioner to think about what they are doing while doing it.37
3. **The Curator:** This is the memory manager. It integrates the lessons extracted by the Reflector into the persistent context store (the playbook). Crucially, the Curator uses "delta updates"—incremental additions of specific insights—rather than wholesale rewrites of the context. This "append-only" or "patch-based" strategy prevents Context Collapse and ensures that the context grows denser and more refined over time.37

### C. Dynamic Cheatsheets and Focused Retrieval

The ACE framework aligns with the concept of **Dynamic Cheatsheets**.18 Instead of providing the agent with a massive, static manual (which would suffer from Lost-in-the-Middle effects), the system maintains a "cheatsheet" of relevant strategies, code snippets, and heuristics that updates in real-time.

To combat "Context Rot," architectures are also moving away from brute-force RAG (Retrieval-Augmented Generation) toward **Graph-Based Focused Retrieval**.4 In standard RAG, retrieving a chunk of text often brings along "distractors"—irrelevant surrounding sentences that rot the context. In Graph-Based retrieval, the system models the knowledge base as a graph of nodes (concepts) and edges (relations). When the agent needs information, it traverses the graph to retrieve only the specific nodes and edges required, assembling a "pure signal" context with minimal noise.4

This approach allows for "Context-Bench" style evaluations, where agents are tested not just on their ability to answer questions, but on their ability to "engineer" their own context—deciding what to retrieve, what to keep, and what to discard.36 The agent becomes an active curator of its own material reality.

## VI. Economics: The Value of Context

The final layer of the pyramid is **Economics**. The manipulation of context material is not free; it consumes computational energy and financial capital. The "Context as Material" framework allows us to quantify the value of context and optimize its usage in a market-driven environment.

### A. Token Economics: Cache Write vs. Cache Read

The primary cost driver in LLM systems is the processing of tokens. However, the pricing models of major providers like Anthropic reveal a crucial distinction that reflects the underlying materiality of the KV cache.8

* **Cache Write Cost:** Processing *new* tokens is expensive because the model must perform the full matrix calculations to generate the KV states. For Claude 3.5 Sonnet, this costs **$3.75 per million tokens**.8
* **Cache Read Cost:** Reusing *existing* context (cached KV states) is significantly cheaper because the heavy computation is already done. Accessing cached tokens costs only **$0.30 per million tokens**—a **90% reduction** in cost.8

This pricing structure fundamentally alters the economics of context. It transforms context from a "disposable good" (computed once and discarded) into a "durable asset" (computed once and reused). The initial processing of a prompt is a **Capital Expenditure (CapEx)**—an investment in building a material state. Subsequent uses of that prompt are **Operational Expenditures (OpEx)**.

### B. The Attention Budget and Latency

Beyond monetary cost, there is the cost of time (latency). The "mass" of the context window creates inertia. Processing a 100k token context for every turn induces unacceptable latency, often measured in tens of seconds.8

We can conceptualize this as an **"Attention Budget."** An agent has a finite amount of "attention seconds" it can spend per interaction. If the context is too "heavy" (too many tokens), the agent spends its entire budget just "loading" the context, leaving no time for "thinking" (generation).

The use of "Prompt Caching" significantly impacts this budget. By pre-loading heavy context materials (e.g., a codebase, a legal library) into the cache, the "Time to First Token" (TTFT) can be reduced by up to **85%**.8 This allows for "thicker" descriptions (more context) without the latency penalty.

### C. Return on Context (RoC)

The ultimate economic metric for the context engineer is the **Return on Context (RoC)**. This metric asks: "Does the addition of these 1,000 tokens of history improve the outcome sufficiently to justify the incremental cost and latency?"

Empirical results from the ACE framework demonstrate a high RoC. By using dynamic, curated context playbooks, agents achieved:

* **+10.6% performance gain** on general agent benchmarks.7
* **+8.6% performance gain** on finance-specific reasoning tasks.7
* **-86.9% reduction in adaptation latency**.41
* **-75% reduction in rollout costs**.38

These figures prove that treating context as a scarce resource that must be engineered—rather than an infinite bucket to be filled—maximizes the efficient conversion of tokens into intelligence.

## VII. Conclusion

The "Context as Material" framework offers a unified, multi-layered ontology for understanding and engineering Large Language Model systems. It moves the discourse beyond the superficial "tips and tricks" of prompt engineering to a rigorous science of material management.

By traversing the **Material Pyramid**, we see that effective AI systems are built on:

1. **Foundations:** A deep understanding of semiotics and situated action (Geertz, Suchman, Latour) that defines *meaning*.
2. **Material:** A mastery of the physical substrate (KV Cache, VRAM) that defines *capacity*.
3. **Physics:** A respect for the thermodynamic laws (Entropy, Gravity) that define *dynamics*.
4. **Architecture:** The implementation of modular systems (ACE, Generator-Reflector-Curator) that define *structure*.
5. **Economics:** An optimization of the cost-value function (Cache Pricing, RoC) that defines *viability*.

As we enter the era of Software 3.0, the primary skill of the AI engineer evolves. It is no longer about writing the perfect line of code; it is about curating the perfect window of context. It is a shift from the digital to the material, from the abstract to the situated, and from the static to the evolving. The context window is the new silicon; how we refine, structure, and utilize this material will determine the intelligence of the systems we build.

### Table 1: The Material Pyramid of Context

| **Level** | **Core Concept** | **Key Theoretical/Technical Pillars** | **Primary Challenge** |
| --- | --- | --- | --- |
| **I. Foundations** | Semiotics & Agency | Thick Description 9, Situated Action 14, ANT 21, Intra-action 23 | Defining "meaning" and "agency" in non-human systems. |
| **II. Material** | The Token & Cache | KV Cache 29, Context Window as RAM 1, Memory Wall 28 | VRAM limitations, HBM bandwidth bottlenecks. |
| **III. Physics** | Dynamics of Attention | Lost in the Middle 6, Context Rot 3, U-Shaped Curve 32 | Non-uniform attention, entropy of information, distractor interference. |
| **IV. Architecture** | Structured Management | ACE 7, Dynamic Cheatsheets 39, Generator-Reflector-Curator 38 | Context Collapse, Brevity Bias, Signal-to-Noise maintenance. |
| **V. Economics** | Cost-Value Optimization | Prompt Caching 8, Return on Context (RoC), Latency Budget | Balancing "Thick Description" (quality) with inference costs (price). |

### Table 2: Comparative Analysis of Context Paradigms

| **Feature** | **Prompt Engineering (Software 2.0)** | **Context Engineering (Software 3.0)** | **Agentic Context Engineering (ACE)** |
| --- | --- | --- | --- |
| **Metaphor** | Casting a Spell (Incantation). | Managing RAM (OS Kernel). | Evolving a Playbook (Scientific Method). |
| **Temporal State** | Static / Stateless. | Persistent / Stateful. | Dynamic / Self-Improving.7 |
| **Structure** | Text block. | Hierarchical / Graph. | Modular (Generator-Reflector-Curator).38 |
| **Handling Noise** | Ignore / Hope. | Pruning / Garbage Collection. | Active Curation / Delta Updates.37 |
| **Failure Mode** | Hallucination. | Context Rot / Lost in Middle. | Context Collapse.19 |
| **Theoretical Basis** | Linguistics / NLP. | Systems Engineering. | Situated Action / Material Semiotics.23 |

### Table 3: Economic Impact of Context Caching (Claude 3.5 Sonnet)

8

| **Operation** | **Cost (per Million Tokens)** | **Latency Impact** | **Economic Classification** |
| --- | --- | --- | --- |
| **Standard Input** | $3.00 | High (Linear growth) | OpEx (Recurring) |
| **Cache Write** | $3.75 (+25%) | High (Initial Load) | CapEx (Investment) |
| **Cache Read** | $0.30 (-90%) | Low (-85% TTFT) | OpEx (Optimized) |

Note: Data derived from Anthropic pricing models.8 TTFT = Time To First Token.

#### Works cited

1. Software 3.0: Software is Changing Again and Again - Superagentic AI, accessed December 10, 2025, <https://super-agentic.ai/resources/super-posts/software-3-0-software-is-changing-again-and-again>
2. Software 3.0: Karpathy's AI Vision Reshaping Development - WillDom, accessed December 10, 2025, <https://willdom.com/blog/software-3-0-ai-reshaping-development-future/>
3. Papers Explained 445: Context Rot | by Ritvik Rastogi - Medium, accessed December 10, 2025, <https://ritvik19.medium.com/papers-explained-443-context-rot-4bbd72d77631>
4. Context Rot is a Big Problem, and Graphs Are the Promising Fix for Coding Agents - Medium, accessed December 10, 2025, <https://medium.com/@animesh1997/context-rot-is-a-big-problem-and-graphs-are-the-promising-fix-for-coding-agents-30be152c49c6>
5. From Theory to Practice: Demystifying the Key-Value Cache in Modern LLMs, accessed December 10, 2025, <https://alain-airom.medium.com/from-theory-to-practice-demystifying-the-key-value-cache-in-modern-llms-9674e9f904a5>
6. Lost in the Middle: How Language Models Use Long Contexts - ResearchGate, accessed December 10, 2025, <https://www.researchgate.net/publication/378284067_Lost_in_the_Middle_How_Language_Models_Use_Long_Contexts>
7. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models - arXiv, accessed December 10, 2025, <https://arxiv.org/html/2510.04618v1>
8. Prompt caching with Claude | Claude, accessed December 10, 2025, <https://www.anthropic.com/news/prompt-caching>
9. A Summary of Thick Description The Interpretation of Cultures - Scribd, accessed December 10, 2025, <https://www.scribd.com/document/168611383/A-Summary-of-Thick-Description>
10. accessed December 10, 2025, <https://www.scirp.org/reference/ReferencesPapers?ReferenceID=2132836#:~:text=Article%20citationsMore%3E%3E-,Geertz%2C%20C.,York%2C%20NY%3A%20Basic%20Books.>
11. Thick description | Literary Theory and Criticism Class Notes - Fiveable, accessed December 10, 2025, <https://fiveable.me/literary-theory-criticism/unit-8/thick-description/study-guide/9FRywJGWCH1SlgzL>
12. Thick Description - Google Docs, accessed December 10, 2025, <https://docs.google.com/document/d/1bMbIEc8MjErUYW8S4socA8O7QgNZ-sBd3fRXHetQnMw/preview?hgd=1>
13. Thick description - Wikipedia, accessed December 10, 2025, <https://en.wikipedia.org/wiki/Thick_description>
14. Human-Machine Reconfigurations: Plans and Situated Actions: 2nd Edition - ResearchGate, accessed December 10, 2025, <https://www.researchgate.net/publication/265092509_Human-Machine_Reconfigurations_Plans_and_Situated_Actions_2nd_Edition>
15. accessed December 10, 2025, <https://www.biblio.com/9780521337397#:~:text=Plans%20and%20Situated%20Actions%3A%20The,Press%202nd%20Edition%20%7C%209780521337397%20%7C%20Biblio>
16. [PDF] Plans and Situated Actions: The Problem of Human-Machine Communication (Learning in Doing: Social, | Semantic Scholar, accessed December 10, 2025, <https://www.semanticscholar.org/paper/Plans-and-Situated-Actions%3A-The-Problem-of-in-Suchman/5416463537f8c6be1199951b4fd6f8d5dae14920>
17. Social Situatedness of Natural and Artificial Intelligence - DiVA portal, accessed December 10, 2025, <http://www.diva-portal.org/smash/get/diva2:3017/FULLTEXT02>
18. Dynamic Cheatsheet: Adaptive Reference - Emergent Mind, accessed December 10, 2025, <https://www.emergentmind.com/topics/dynamic-cheatsheet>
19. Agentic Context Engineering: Learning Comprehensive Contexts for Self-Improving Language Models | OpenReview, accessed December 10, 2025, <https://openreview.net/forum?id=eC4ygDs02R>
20. Video: Actor Network Theory | Diagram, Critiques & Examples - Study.com, accessed December 10, 2025, <https://study.com/academy/lesson/video/latours-actor-network-theory.html>
21. Actor–network theory - Wikipedia, accessed December 10, 2025, <https://en.wikipedia.org/wiki/Actor%E2%80%93network_theory>
22. Latour's Actor Network Theory - Simply Psychology, accessed December 10, 2025, <https://www.simplypsychology.org/actor-network-theory.html>
23. towards-posthumanist-design - Wild Pear CIC, accessed December 10, 2025, <https://wildpearcic.co.uk/towards-posthumanist-design>
24. Situated Knowledges: The Science Question in Feminism and the Privilege of Partial Perspective, accessed December 10, 2025, <https://commons.princeton.edu/hum583-f21/wp-content/uploads/sites/283/2021/08/Haraway-Situated-Knowledges.pdf>
25. Meeting the Universe Halfway: Quantum Physics and the Entanglement of Matter and Meaning. By Karen Barad. Durham, N.C., accessed December 10, 2025, <https://www.cambridge.org/core/journals/hypatia/article/meeting-the-universe-halfway-quantum-physics-and-the-entanglement-of-matter-and-meaning-by-karen-barad-durham-nc-duke-university-press-2007/5717D38D111DF2ED914359CDCBE6E1E0>
26. Methodologies in New Materialism and Discourse Analysis, accessed December 10, 2025, <https://discourseanalyzer.com/methodologies-in-new-materialism-and-discourse-analysis/>
27. Abstract What constitutes 'knowledge' that we acquire, and how do we acquire it? Through Donna Haraway's theoretical recon - IDA, accessed December 10, 2025, <https://ida.mtholyoke.edu/server/api/core/bitstreams/473399db-eaaa-42ee-bf98-5e7b50d9aa73/content>
28. Scalable Inference with RDMA and Tiered KV Caching | by Nadeem Khan(NK) | LearnWithNK | Nov, 2025, accessed December 10, 2025, <https://medium.com/learnwithnk/scalable-inference-with-rdma-and-tiered-kv-caching-9d7e494a863b>
29. The Secret Behind Fast LLM Inference: Unlocking the KV Cache - Towards AI, accessed December 10, 2025, <https://pub.towardsai.net/the-secret-behind-fast-llm-inference-unlocking-the-kv-cache-9c13140b632d>
30. Andrej Karpathy, accessed December 10, 2025, <https://karpathy.ai/>
31. Andrej Karpathy on Software 3.0: Software in the Age of AI | by Ben Pouladian | Medium, accessed December 10, 2025, <https://medium.com/@ben_pouladian/andrej-karpathy-on-software-3-0-software-in-the-age-of-ai-b25533da93b6>
32. Why Language Models Are “Lost in the Middle” - Towards AI, accessed December 10, 2025, <https://pub.towardsai.net/why-language-models-are-lost-in-the-middle-629b20d86152>
33. Found in the Middle: How Language Models Use Long Contexts Better via Plug-and-Play Positional Encoding - arXiv, accessed December 10, 2025, <https://arxiv.org/html/2403.04797v1>
34. Never Lost in the Middle: Mastering Long-Context Question Answering with Position-Agnostic Decompositional Training - arXiv, accessed December 10, 2025, <https://arxiv.org/html/2311.09198v2>
35. P6: Context Rot – Hamel's Blog, accessed December 10, 2025, <https://hamel.dev/notes/llm/rag/p6-context_rot.html>
36. Context-Bench: Benchmarking LLMs on Agentic Context Engineering | Letta, accessed December 10, 2025, <https://www.letta.com/blog/context-bench>
37. Agentic Context Engineering - Sundeep Teki, accessed December 10, 2025, <https://www.sundeepteki.org/blog/agentic-context-engineering>
38. Agentic Context Engineering (ACE) - Emergent Mind, accessed December 10, 2025, <https://www.emergentmind.com/topics/agentic-context-engineering-ace>
39. Paper page - Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models - Hugging Face, accessed December 10, 2025, <https://huggingface.co/papers/2510.04618>
40. The Complexity Trap: Simple Observation Masking Is as Efficient as LLM Summarization for Agent Context Management - arXiv, accessed December 10, 2025, <https://arxiv.org/html/2508.21433v1>
41. [2510.04618] Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models - arXiv, accessed December 10, 2025, <https://arxiv.org/abs/2510.04618>