Research Report: Why LLM Outputs Degrade and How Teams Combat It

EXECUTIVE SUMMARY

LLM output degradation is not one bug. It is a family of failure modes that appear when the model is asked to use too much context, conflicting context, stale context, unclear instructions, or long conversation history. Long-context benchmarks show that large context windows do not guarantee uniform use of information: models often perform best when relevant facts are near the beginning or end of the prompt and worse when facts are buried in the middle [1]. RULER found that most tested long-context models suffered large drops as input length and task complexity increased, even when they did well on simple needle-in-a-haystack retrieval [2]. NoLiMa likewise found that performance can collapse when literal lexical overlap is removed from long-context retrieval tasks [3].

Conversation itself is another degradation channel. Laban et al. found an average 39% performance drop when fully specified single-turn tasks were distributed across multi-turn conversations, attributing much of the failure to premature assumptions and low recovery after a wrong early turn [6]. Newer work frames this as an intent alignment problem: the model has the necessary information somewhere in the dialogue, but the dialogue does not compile into a clear task specification [7].

The strongest mitigation pattern is context engineering: curate, structure, compress, retrieve, verify, and test the context instead of stuffing more tokens into the prompt. Recent surveys define context engineering as a discipline covering retrieval, processing, management, RAG, memory systems, tool reasoning, and multi-agent systems [16]. Practical controls include explicit instruction hierarchy, clear output constraints, document tagging, query placement, quote-first grounding, retrieval reranking, context condensation, stale-context audits, and evals that replay realistic long or multi-turn workflows [11][17][18][22].

Primary recommendation: treat the context window as a managed, testable runtime surface. In practice, this means short authoritative instructions, separated data and instructions, retrieval quality gates, conversation-state compaction, and regression tests for long-context and multi-turn cases [8][9][16][17].

Confidence level: High for the main degradation mechanisms and mitigation families because they are supported by multiple independent benchmarks, platform docs, and agent-system papers [1][2][3][6][13][16]. Medium for exact operational recipes because best practice varies by model family, task, and risk tolerance [17][18][19].

INTRODUCTION

Research Question

This report investigates how LLM outputs degrade during real use, with emphasis on "context rot," long-context failure, unclear or conflicting instructions, token bloat, multi-turn drift, and the mitigation patterns used in research and production systems. The scope is technical and applied: the goal is not to rank specific vendors, but to explain why quality declines and what engineering practices reduce the decline.

Scope and Methodology

The research covered five failure channels: position bias in long prompts, noise and distractors in retrieved context, multi-turn conversation drift, instruction ambiguity or conflict, and output-side degradation during long-form generation. It also covered six mitigation families: prompt structure, retrieval and reranking, context compression, persistent memory with provenance, instruction hierarchy, and task-specific evaluation. The source base combines long-context benchmarks, RAG studies, instruction-following papers, agent context-management papers, and official prompt-engineering guidance from OpenAI, Anthropic, and Google Cloud [1][2][6][11][13][16][17][18][19].

The analysis prioritized primary sources: arXiv papers, benchmark descriptions, official documentation, and papers that introduced named methods. Practitioner articles were used only to discover terminology and were not relied on for major claims unless the same claim appeared in a primary source. The most important evidence was triangulated across at least three independent clusters: long-context benchmarks, multi-turn studies, and context-management or prompt-design guidance [2][3][6][8][16][17].

The report uses "degradation" broadly to mean lower task accuracy, weaker grounding, longer or less useful outputs, higher hallucination risk, worse instruction following, higher latency and cost, or reduced recoverability after an early mistake. The report uses "context rot" in two senses. The first is runtime context rot: irrelevant, stale, or misleading tokens accumulate in the active prompt and distract the model [9][13]. The second is persistent context rot: long-lived instruction files or memory artifacts become stale as the underlying codebase or domain changes [20].

Key Assumptions

The first assumption is that the reader cares about applied LLM systems rather than model training alone. This matters because many mitigations are system-level choices, such as retrieval filtering, prompt structuring, and conversation summarization [16][17].

The second assumption is that "more context" is useful only when relevant context is preserved and irrelevant context is controlled. Long-context RAG studies show that additional retrieved passages can help up to a point, but they can also reduce answer quality when hard negatives or excessive documents are added [13][14].

The third assumption is that single-turn benchmark accuracy can overstate real chat reliability. Multi-turn work shows that the same task information can yield lower performance when spread across a dialogue instead of delivered as a complete specification [6][7].

The fourth assumption is that no one mitigation is sufficient. RAG can reduce some hallucinations, but RAGTruth shows that RAG systems can still produce unsupported or contradictory claims against retrieved content [15].

MAIN ANALYSIS

Finding 1: Long Context Windows Do Not Guarantee Uniform Context Use

The most reproducible degradation mechanism is positional fragility. Liu et al. tested multi-document question answering and key-value retrieval and found that model performance often depends on where the relevant information appears in the input context [1]. Their core result is that models frequently perform best when the useful material appears near the start or the end of the input and worse when it appears in the middle, even for models advertised as long-context systems [1].

RULER strengthened this point by arguing that vanilla needle-in-a-haystack retrieval is too easy to represent real long-context understanding [2]. Hsieh et al. expanded the benchmark to include multiple needles, variable quantities of needles, multi-hop tracing, and aggregation tasks [2]. They evaluated 17 long-context models and reported that nearly perfect vanilla needle retrieval did not predict robust performance as context length and task complexity increased [2]. The paper also reports that only about half of models claiming context windows of 32K tokens or more maintained satisfactory performance at 32K under RULER's broader tests [2].

NoLiMa gives a sharper version of the same warning. Modarressi et al. designed a benchmark where the question and the relevant "needle" have minimal lexical overlap, which forces models to infer associations rather than match literal text [3]. The paper reports that 10 of 12 tested long-context models dropped below 50% of their strong short-context baselines at 32K, while GPT-4o fell from 99.3% in short contexts to 69.7% in that harder long-context setting [3]. This matters because many production prompts depend on semantic relevance, not literal string matching [3].

LongBench v2 shows that long-context reasoning is also hard in realistic tasks. Bai et al. built 503 challenging questions with contexts from 8K to 2M words across single-document QA, multi-document QA, long in-context learning, long-dialogue history understanding, code repository understanding, and structured-data understanding [4]. Human experts achieved 53.7% accuracy under a 15-minute limit, the best direct-answering model achieved 50.1%, and o1-preview with longer reasoning achieved 57.7% [4]. The implication is not that long-context models are useless; it is that task framing, reasoning time, and context structure remain decisive [4].

Attention mechanics help explain why the problem is structural. StreamingLLM found that initial tokens can act as attention sinks: they attract disproportionate attention even when not semantically important [10]. Xiao et al. showed that retaining the key-value cache for initial tokens can recover performance under window attention in streaming settings, which implies that raw recency or raw token count is not the whole story [10]. A model's attention budget, positional encoding, and cache policy can shape what it can reliably use [10].

The practical conclusion is that a long context window is capacity, not comprehension. Capacity says the model can accept the tokens. Comprehension says it can select, integrate, and reason over the right tokens under realistic distractors. Benchmarks consistently show that the second property is weaker than the first [1][2][3][4].

Finding 2: Noise, Hard Negatives, and Stale Context Create "Context Rot"

Runtime context rot happens when the prompt contains more text but less useful signal. Long-context RAG research shows the shape of this problem clearly. Jin et al. studied long-context LLMs in RAG settings and found that adding more retrieved passages can initially improve performance and then degrade output as more passages are added [13]. They identify hard negatives as a key contributor: retrieved passages can look relevant while pushing the model toward wrong or unsupported reasoning [13].

Leng et al. studied RAG workflows across 20 open-source and commercial models while varying context length from 2,000 to 128,000 tokens and up to 2 million tokens where supported [14]. They found that retrieving more documents can improve performance, but only a handful of recent state-of-the-art models maintained consistent accuracy above 64K tokens [14]. This supports a practical rule: retrieval recall is useful only when paired with relevance control, reranking, and context budgeting [13][14].

RAGTruth shows why retrieval is a mitigation but not a guarantee. Niu et al. created a

corpus of nearly 18,000 naturally generated RAG responses with manual hallucination annotations [15]. Their premise is that RAG is a main technique for reducing hallucinations, but RAG outputs may still contain claims unsupported by, or contradictory to, the retrieved content [15]. This failure mode is common in systems that retrieve correct documents but do not force answer grounding at the claim level [15][22].

Persistent context rot is a related but distinct problem. Treude and Baltes define context rot in AI-assisted software development as stale persistent context in files such as CLAUDE.md, AGENTS.md, and .cursorrules [20]. Their preliminary test used an existing documentation-consistency checker on 356 repositories and found stale code element references in 23.0% of repositories [20]. This is not a long-context attention problem by itself; it is a governance problem where the model receives obsolete instructions or outdated architectural claims [20].

Context engineering surveys now treat this as an engineering discipline rather than a prompting trick. Mei et al. describe context engineering as optimizing the information payload supplied at inference time and decompose it into context retrieval and generation, context processing, and context management [16]. The same survey maps those components into RAG, memory systems, tool-integrated reasoning, and multi-agent systems [16]. That taxonomy fits observed production failures: the issue is often not that the model lacks capability, but that the supplied context is noisy, stale, badly ranked, or unmanaged [13][16][20].

The mitigation is not simply "summarize everything." Aggressive compression can erase the details needed for exact recovery, while uncompressed history can distract the model. AdaCoM frames this as a fidelity-reliability trade-off: stronger base agents may benefit from higher-fidelity preservation, while weaker agents may need more aggressive compression to remain in a reliable reasoning regime [9]. The report’s operational takeaway is to manage context like a cache: keep authoritative instructions small, keep evidence retrievable, prune stale content, and preserve exact source locators for claims that may need verification [9][15][16].

Finding 3: Multi-Turn Conversations Degrade Through Premature Assumptions and Low Recovery

LLMs often appear strongest when the full task is specified in one clean prompt. Laban et al. directly tested the difference between single-turn and multi-turn formats and found that top open- and closed-weight models performed significantly worse in multi-turn conversations than in fully specified single-turn settings [6]. Their large-scale simulation over more than 200,000 conversations found an average 39% drop across six generation tasks [6].

The mechanism matters more than the headline. Laban et al. found that models often made assumptions in early turns and prematurely attempted final solutions, then over-relied on those early outputs in later turns [6]. This creates path dependence: a wrong early interpretation becomes part of the conversation state, and later messages are interpreted through that distorted state [6]. The degradation is therefore not only about token length; it is also about conversational commitments that become implicit context [6].

Liu et al. sharpen this interpretation by arguing that "lost in conversation" is driven by intent mismatch rather than a simple absence of model capability [7]. Their 2026 paper argues that scaling model size or improving training alone cannot fully solve a gap that arises from structural ambiguity in conversational context [7]. They propose separating intent understanding from task execution through a Mediator-Assistant architecture that rewrites vague dialogue into explicit structured instructions [7].

MT-OSC attacks the same class of failure from the context-management side. Singh et al. describe a One-off Sequential Condensation framework that condenses chat history in the

background while retaining essential information [8]. The paper reports token-count reductions up to 72% in 10-turn dialogues and improved or preserved accuracy across multi-turn benchmarks [8]. This suggests that many conversations benefit from an explicit state representation rather than a raw appended transcript [8].

AdaCoM generalizes the idea to long-horizon agent tasks such as web search and deep research [9]. Yi et al. train an external LLM to manage the context of a frozen agent through actions that preserve task constraints and progress while pruning stale content [9]. This is important for closed-source agents because the context manager can operate outside the underlying model [9].

The practical pattern is to compile conversation into state. A robust agent should retain stable goals, constraints, open questions, decisions, evidence locators, and user preferences while demoting obsolete attempts and early mistakes. Raw chat history is useful for auditability, but it is a poor runtime state representation once the dialogue becomes long or exploratory [6][8][9].

Finding 4: Unclear or Conflicting Instructions Turn Context Into Competing Commands

Instruction ambiguity degrades output because the model must infer priority, scope, format, and success criteria from text that may not distinguish command, data, example, and historical chatter. Anthropic's prompting guidance says complex prompts should separate instructions, context, examples, and variable inputs, and it recommends XML tags because they help Claude parse mixed prompt components unambiguously [17]. Google Cloud's prompt-design documentation similarly exposes separate strategy pages for clear instructions, system instructions, examples, context, prompt structure, and task decomposition [18].

Instruction conflict is the adversarial version of the same problem. Wallace et al. argue that LLMs are vulnerable when they treat system prompts, user prompts, and third-party text as if they had equal priority [11]. Their instruction hierarchy work trains models to ignore lower-priority conflicting instructions, improving robustness to prompt injection and jailbreak-style attacks while imposing minimal degradation on ordinary capabilities [11].

Many-tier instruction hierarchy work shows that simple role labels may be too coarse for agentic systems. Zhang et al. argue that real agents receive instructions from system messages, user prompts, tool outputs, and other sources with many trust levels [12]. Their ManyIH benchmark includes up to 12 privilege levels and reports that frontier models perform poorly at about 40% accuracy when instruction conflict scales [12]. This matters because agent prompts often mix developer instructions, user goals, retrieved documents, tool output, previous assistant messages, and generated plans [12].

Official platform docs converge on a simple mitigation: make the model's job legible. Anthropic recommends specific output constraints, sequential steps when order matters, examples for consistent output, XML tags for prompt sections, and role prompts for behavior focus [17]. OpenAI's Responses API documentation separates 'instructions' from 'input', and it also represents message roles such as developer and user, which is an API-level way to keep instruction layers distinct [19]. Google Cloud's prompt-design guide places clear instructions, system instructions, few-shot examples, context, prompt structure, and task breakdown into separate prompting strategies [18].

The core insight is that unclear instructions do not just produce vague output; they consume context budget and increase entropy. When a prompt says "be thorough," "be concise," "use all sources," "do not overdo it," and "follow previous style" without priority rules, the model may respond with token-heavy compromise behavior. That often appears as answer bloat, hedging, redundant summaries, and loss of the actual task objective [6][17][18].

Finding 5: Output-Side Degradation Is Different From Input-Side Degradation

Input comprehension and output generation degrade differently. A model can find the right evidence and still fail to produce a coherent, bounded, instruction-following long answer. LongGenBench was designed because many long-context benchmarks evaluate input understanding but not long-form generation quality [5]. Wu et al. tested tasks requiring long outputs under complex constraints and found that all ten evaluated models struggled with long text generation, especially as generation length increased [5].

FACTS Grounding evaluates whether long-form responses are fully grounded in a provided long document [22]. Jacovi et al. require responses to satisfy the user request and be accurate with respect to the supplied document, using a two-phase judging process and public and private leaderboard splits [22]. This benchmark exists because source-grounded long-form generation is a separate capability from short-answer retrieval [22].

Simple factuality benchmarks show another output-side issue: models should sometimes decline rather than guess. Wei et al. introduced SimpleQA as a benchmark for short, fact-seeking questions where answers are graded correct, incorrect, or not attempted [23]. They describe ideal behavior as answering when confident and not attempting questions when the model is not confident [23]. In production systems, unclear instructions can suppress this behavior by implicitly rewarding completion over calibrated uncertainty [17][23].

Long-form output also interacts with conversation drift. Laban et al. identify premature generation as a multi-turn failure mechanism, where the model tries to solve before the task is fully specified [6]. Once that premature answer is in the transcript, subsequent responses may use it as context, which increases the chance of compounding assumptions [6]. This is why "answer later, clarify first" can be a reliability control rather than a politeness preference [6][7].

Mitigations for output-side degradation focus on structure and verification. Anthropic recommends asking the model to quote relevant parts of long documents before carrying out a long-context task, which helps the model cut through noise [17]. RAGTruth motivates claim-level hallucination detection because RAG outputs can include unsupported claims even when retrieved content is available [15]. FACTS Grounding operationalizes the same idea at benchmark level by scoring whether long-form answers are grounded in the document [22].

The practical distinction is useful. If the model cannot locate evidence, improve retrieval, reranking, ordering, and context selection [13][14]. If the model can locate evidence but writes a bad answer, improve output schemas, citation requirements, quote-first plans, stepwise decomposition, and factuality evaluation [15][17][22]. Treating both as "bad context" hides which lever needs to move.

Finding 6: The Mitigation Stack Is Context Engineering, Not Prompt Length

The strongest mitigation stack has five layers. The first layer is instruction hygiene: separate durable instructions from user requests, separate instructions from data, state output format and constraints, and include representative examples when consistency matters [17][18][19]. This reduces ambiguity before the model spends tokens resolving it [17].

The second layer is retrieval discipline. RAG should retrieve enough evidence to answer, but not so much that hard negatives dominate the prompt [13][14]. Reranking, deduplication, source trust metadata, date filtering, and query expansion are ways to improve context quality before the model sees the tokens [13][14][16]. RAGTruth shows that retrieval alone

does not prove grounding, so answer generation still needs verification [15].

The third layer is context compaction with evidence preservation. MT-OSC shows that condensed multi-turn histories can reduce token count and narrow the multi-turn performance gap [8]. AdaCoM shows that adaptive context managers can prune stale content while preserving task constraints and progress [9]. The key design principle is to compress state, not evidence: summaries are acceptable for goals and decisions, but exact quotes, file paths, IDs, and source locators should remain dereferenceable [8][9][15].

The fourth layer is instruction hierarchy and trust boundaries. System and developer instructions should remain above user requests, user requests should remain above untrusted retrieved content, and tool outputs should be treated as data unless explicitly privileged [11][12][19]. ManyIH suggests that realistic agents may need more than a few role labels because sources can have many privilege levels and contexts [12].

The fifth layer is evaluation. Long-context systems should be tested with buried facts, multi-hop retrieval, distracting hard negatives, long dialogue histories, and long-form grounded outputs [2][3][4][22]. Multi-turn systems should be tested against the same tasks in both single-turn and multi-turn forms because single-turn success can mask conversation-state failures [6][7]. Prompt and context changes should be regression-tested because persistent context files can rot as the underlying software changes [20].

The resulting discipline is closer to runtime systems engineering than copywriting. Good prompts matter, but they are one component of a managed context pipeline that includes retrieval, state, memory, hierarchy, observability, and tests [16][17][18].

SYNTHESIS AND INSIGHTS

Patterns Identified

The first pattern is that degradation usually comes from selection failure. The model has too many plausible things to attend to, too many historical commitments to preserve, or too many instruction-like strings competing for priority [1][6][11]. Long-context benchmarks expose selection failure spatially, RAG studies expose it semantically, and instruction-hierarchy work exposes it by trust level [1][13][12].

The second pattern is that raw context and useful state diverge over time. A long transcript includes goals, failed attempts, stale assumptions, tool outputs, user corrections, and partial summaries [6][8]. A good runtime state should preserve goals, constraints, decisions, evidence, and open questions while demoting dead ends [8][9]. This is why compaction can improve quality even when it removes tokens: it changes the prompt from an archive into a task state [8][9].

The third pattern is that model capability and context quality multiply rather than add. A stronger model may survive more noise, but RULER, NoLiMa, and LongBench v2 all show that longer contexts and harder reasoning still create sharp performance gaps [2][3][4]. Better context can unlock model capability, while bad context can suppress it [13][16].

Novel Insights

The most useful mental model is "context is executable input." In ordinary software, untrusted data, stale configuration, and conflicting flags cause failures because they are interpreted by a runtime. In LLM systems, prompt text is interpreted by a probabilistic runtime, so stale docs, irrelevant retrievals, buried facts, and conflicting commands become execution hazards [11][12][20].

A second insight is that summaries should not be treated as memory. Summaries are lossy state representations, while memory should include dereferenceable evidence. The papers on multi-turn condensation and adaptive context management support compression, but RAGTruth and FACTS Grounding show why exact grounding remains necessary for factual claims [8][9][15][22].

A third insight is that "be more thorough" is risky unless paired with a budget and a stopping rule. Thoroughness instructions can increase response length, tool use, and self-generated intermediate context, which may worsen the very degradation they were meant to prevent [6][17]. A better instruction is "include the minimum evidence needed to support each claim, ask when the task is underspecified, and stop when acceptance criteria are met" [17][18].

Implications

For developers building AI assistants, the main implication is that context quality should be observable. Logs should record which instructions, retrieved documents, summaries, and tool outputs were supplied to the model for each answer [16]. Evals should replay long-turn and long-context scenarios that resemble production traffic [6][22].

For prompt authors, the main implication is to write for disambiguation rather than persuasion. Good prompts separate command from data, define success, name the expected output shape, and state what the model should do when evidence is missing [17][18][23].

For teams using persistent agent instructions, the main implication is that prompt files need ownership and freshness checks. Treude and Baltes show that stale references are common enough to justify tooling borrowed from documentation consistency checks [20].

LIMITATIONS AND CAVEATS

Counterevidence Register

Long-context models can outperform RAG in some settings. Lee et al. introduce LOFT and report that long-context LLMs can rival state-of-the-art retrieval and RAG systems on some real-world tasks, though they still struggle with compositional reasoning required in SQL-like tasks [24]. Li et al. also report that long context can outperform RAG in some question-answering benchmarks, especially Wikipedia-based questions, while RAG has advantages in dialogue-based and general query settings [25]. This means the right mitigation is task-dependent rather than a universal preference for RAG or long context [24][25].

Anthropic’s long-context guidance says placing long documents near the top and queries near the end can improve quality, with tests showing up to 30% improvement for complex multi-document inputs [17]. That recommendation complicates a simplistic "put important instructions first" rule because the best ordering depends on whether the prompt is dominated by documents, instructions, or examples [17].

Instruction hierarchy can mitigate prompt injection and conflict, but ManyIH suggests that current frontier models still struggle when privilege levels scale [12]. This means role separation in an API is necessary but not sufficient for complex agent settings [12][19].

Known Gaps

The research base has uneven coverage of closed production systems. Many papers evaluate public benchmarks or simulated conversations, while real enterprise assistants have private

retrieval corpora, custom toolchains, and proprietary prompts [6][14][16]. This limits direct transfer from benchmark scores to a specific deployment.

The phrase "context rot" is not standardized. In AI-assisted software development, Treude and Baltes use it to describe stale persistent context artifacts [20]. In broader practitioner usage, it often describes runtime prompt pollution from irrelevant or stale tokens [9][13]. This report uses both meanings and labels them explicitly.

The report does not benchmark model-specific prompt recipes. Official docs from Anthropic, Google, and OpenAI agree on broad structure, clarity, examples, and separated roles, but model families differ in details such as ideal ordering, verbosity controls, and reasoning configuration [17][18][19].

Areas of Uncertainty

The largest uncertainty is how quickly frontier-model behavior changes. Prompting practices that reduce under-triggering in one model generation can cause over-triggering in another, and Anthropic's migration guidance explicitly warns that aggressive anti-laziness prompting may need to be dialed back for newer, more proactive models [17]. This makes live evals more reliable than static prompt folklore [17].

Another uncertainty is the optimal compression policy for long-running agents. MT-OSC, AdaCoM, and related approaches show benefits, but they use different tasks, agents, and state representations [8][9]. The safe generalization is to preserve task constraints and evidence locators while pruning stale or redundant tokens, but exact thresholds should be measured per workflow [8][9][16].

RECOMMENDATIONS

Immediate Actions

1. Create a context budget for every serious workflow. Define which tokens are reserved for durable instructions, user request, retrieved evidence, conversation state, examples, tool output, and final-answer budget [16][17]. This prevents context stuffing from silently displacing the information the model needs most [13][14].

2. Separate instructions, data, examples, and outputs. Use clear sections or XML-like tags for complex prompts because Anthropic reports that this reduces misinterpretation when prompts mix multiple content types [17]. Use API role fields where available so durable instructions are not blended with ordinary user or retrieved text [19].

3. Replace raw chat history with compiled state after a few turns. Preserve stable goals, user constraints, decisions, open questions, and evidence locators, but remove obsolete attempts and repeated explanations [6][8][9]. Keep the raw transcript for audit, not necessarily for every model call [8][9].

4. Add a quote-first or evidence-first step for grounded tasks. Anthropic recommends asking for relevant quotes before performing long-document tasks, and RAGTruth shows why claim-level grounding remains necessary even with retrieved context [15][17].

5. Audit persistent prompt files for rot. Assign owners to AGENTS.md, CLAUDE.md, .cursorrules, retrieval instructions, and system prompts, then check references against the current codebase or knowledge base [20]. Stale persistent context should be treated like stale documentation with production impact [20].

Near-Term Next Steps

Run paired evals in both single-turn and multi-turn form. Laban et al. show that multi-turn presentation can cause a large performance drop even when the same task information is ultimately present [6]. A useful eval suite should include buried facts, ambiguous follow-ups, user corrections, distractor documents, and long-form grounded answers [2][6][22].

Instrument retrieval quality before answer quality. Log retrieved documents, scores, reranker decisions, source dates, deduplication, and final context order [13][14][16]. When an answer fails, first determine whether the right evidence was missing, buried, contradicted, or present but unused [13][15].

Define model behavior for uncertainty. Use prompts and evals that reward "not enough evidence" when the context is missing support, because SimpleQA frames calibrated non-answering as part of ideal behavior [23]. This is especially important for RAG systems where a retrieved passage can be topically related but not answer-bearing [13][15].

Further Research Needs

More public work is needed on context rot in real production agents. Treude and Baltes provide an early software-development framing, but broader datasets for stale memory, stale retrieval indexes, and stale system prompts would make mitigation more measurable [20].

More work is needed on evidence-preserving compression. AdaCoM and MT-OSC show that context management can help, but factual applications need compression methods that preserve exact provenance and allow dereferencing rather than trusting generated summaries alone [8][9][15].

More work is needed on instruction priority beyond simple role hierarchies. ManyIH shows that realistic agents can involve many privilege levels, and current models remain weak when conflict scales [12]. Future systems likely need explicit policy metadata, source trust labels, and evals for role conflict [11][12][19].

BIBLIOGRAPHY

[1] Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. (2023). "Lost in the Middle: How Language Models Use Long Contexts". arXiv. https://arxiv.org/abs/2307.03172 (Retrieved: 2026-06-21)

[2] Hsieh, C. P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. (2024). "RULER: What’s the Real Context Size of Your Long-Context Language Models?" arXiv. https://arxiv.org/abs/2404.06654 (Retrieved: 2026-06-21)

[3] Modarressi, A., Deilamsalehy, H., Dernoncourt, F., Bui, T., Rossi, R. A., Yoon, S., and Schutze, H. (2025). "NoLiMa: Long-Context Evaluation Beyond Literal Matching". arXiv. https://arxiv.org/abs/2502.05167 (Retrieved: 2026-06-21)

[4] Bai, Y., Tu, S., Zhang, J., Peng, H., Wang, X., Lv, X., Cao, S., Xu, J., Hou, L., Dong, Y., and Tang, J. (2024). "LongBench v2: Towards Deeper Understanding and Reasoning on Realistic Long-context Multitasks". arXiv. https://arxiv.org/abs/2412.15204 (Retrieved: 2026-06-21)

[5] Wu, Y., Hee, M. S., Hu, Z., and Lee, R. K. W. (2024). "LongGenBench: Benchmarking Long-Form Generation in Long Context LLMs". arXiv. https://arxiv.org/abs/2409.02076 (Retrieved: 2026-06-21)

[6] Laban, P., Hayashi, H., Zhou, Y., and Neville, J. (2025). "LLMs Get Lost In Multi-Turn Conversation". arXiv. https://arxiv.org/abs/2505.06120 (Retrieved: 2026-06-21)

[7] Liu, G., Zhu, F., Feng, R., Ma, C., Wang, S., and Meng, G. (2026). "Intent Mismatch

Causes LLMs to Get Lost in Multi-Turn Conversation". arXiv. https://arxiv.org/abs/2602.07338 (Retrieved: 2026-06-21)

[8] Singh, J., Tu, F., Ballesteros, M., Sun, W., Ghoshal, S., Yuan, M., Benajiba, Y., Ravi, S., and Roth, D. (2026). "MT-OSC: Path for LLMs that Get Lost in Multi-Turn Conversation". arXiv. https://arxiv.org/abs/2604.08782 (Retrieved: 2026-06-21)

[9] Yi, L., Lei, R., Yao, L., Xie, Y., Li, Y., Zhang, W., Wei, Z., Li, Y., and Nie, J. Y. (2026). "Learning Agent-Compatible Context Management for Long-Horizon Tasks". arXiv. https://arxiv.org/abs/2605.30785 (Retrieved: 2026-06-21)

[10] Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. (2023). "Efficient Streaming Language Models with Attention Sinks". arXiv. https://arxiv.org/abs/2309.17453 (Retrieved: 2026-06-21)

[11] Wallace, E., Xiao, K., Leike, R., Weng, L., Heidecke, J., and Beutel, A. (2024). "The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions". arXiv. https://arxiv.org/abs/2404.13208 (Retrieved: 2026-06-21)

[12] Zhang, J., Li, T., Jurayj, W., Zhan, H., Van Durme, B., and Khashabi, D. (2026). "Many-Tier Instruction Hierarchy in LLM Agents". arXiv. https://arxiv.org/abs/2604.09443 (Retrieved: 2026-06-21)

[13] Jin, B., Yoon, J., Han, J., and Arik, S. O. (2024). "Long-Context LLMs Meet RAG: Overcoming Challenges for Long Inputs in RAG". arXiv. https://arxiv.org/abs/2410.05983 (Retrieved: 2026-06-21)

[14] Leng, Q., Portes, J., Havens, S., Zaharia, M., and Carbin, M. (2024). "Long Context RAG Performance of Large Language Models". arXiv. https://arxiv.org/abs/2411.03538 (Retrieved: 2026-06-21)

[15] Niu, C., Wu, Y., Zhu, J., Xu, S., Shum, K., Zhong, R., Song, J., and Zhang, T. (2023). "RAGTruth: A Hallucination Corpus for Developing Trustworthy Retrieval-Augmented Language Models". arXiv. https://arxiv.org/abs/2401.00396 (Retrieved: 2026-06-21)

[16] Mei, L., Yao, J., Ge, Y., Wang, Y., Bi, B., Cai, Y., Liu, J., Li, M., Li, Z., Zhang, D., Zhou, C., Mao, J., Xia, T., Guo, J., and Liu, S. (2025). "A Survey of Context Engineering for Large Language Models". arXiv. https://arxiv.org/abs/2507.13334 (Retrieved: 2026-06-21)

[17] Anthropic (2026). "Prompting best practices". Claude API Docs. https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/claude-prompting-best-practices (Retrieved: 2026-06-21)

[18] Google Cloud (2026). "Introduction to prompting". Gemini Enterprise Agent Platform Documentation. https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/prompts/introduction-prompt-design (Retrieved: 2026-06-21)

[19] OpenAI (2026). "Prompt engineering". OpenAI API Documentation. https://developers.openai.com/api/docs/guides/prompt-engineering (Retrieved: 2026-06-21)

[20] Treude, C., and Baltes, S. (2026). "Context Rot in AI-Assisted Software Development: Repurposing Documentation Consistency for AI Configuration Artifacts". arXiv. https://arxiv.org/abs/2606.09090 (Retrieved: 2026-06-21)

[21] Park, J., Atarashi, K., Takeuchi, K., and Kashima, H. (2025). "Emulating Retrieval Augmented Generation via Prompt Engineering for Enhanced Long Context Comprehension in LLMs". arXiv. https://arxiv.org/abs/2502.12462 (Retrieved: 2026-06-21)

[22] Jacovi, A., Wang, A., Alberti, C., Tao, C., Lipovetz, J., Olszewska, K., Haas, L., Liu,

M., Keating, N., Bloniarz, A., Saroufim, C., Fry, C., Marcus, D., Kukliansky, D., Tomar, G. S., Swirhun, J., Xing, J., Wang, L., Gurumurthy, M., Aaron, M., Ambar, M., Fellinger, R., Wang, R., Zhang, Z., Goldshtein, S., and Das, D. (2025). "The FACTS Grounding Leaderboard: Benchmarking LLMs' Ability to Ground Responses to Long-Form Input". arXiv. https://arxiv.org/abs/2501.03200 (Retrieved: 2026-06-21)

[23] Wei, J., Karina, N., Chung, H. W., Jiao, Y. J., Papay, S., Glaese, A., Schulman, J., and Fedus, W. (2024). "Measuring short-form factuality in large language models". arXiv. https://arxiv.org/abs/2411.04368 (Retrieved: 2026-06-21)

[24] Lee, J., Chen, A., Dai, Z., Dua, D., Sachan, D. S., Boratko, M., Luan, Y., Arnold, S. M. R., Perot, V., Dalmia, S., Hu, H., Lin, X., Pasupat, P., Amini, A., Cole, J. R., Riedel, S., Naim, I., Chang, M. W., and Guu, K. (2024). "Can Long-Context Language Models Subsume Retrieval, RAG, SQL, and More?" arXiv. https://arxiv.org/abs/2406.13121 (Retrieved: 2026-06-21)

[25] Li, X., Cao, Y., Ma, Y., and Sun, A. (2024). "Long Context vs. RAG for LLMs: An Evaluation and Revisits". arXiv. https://arxiv.org/abs/2501.01880 (Retrieved: 2026-06-21)

APPENDIX: METHODOLOGY

Research Process

Phase 1 scoped the question around degradation mechanisms and mitigations, with the assumption that the target audience is technical and wants operational guidance. Phase 2 planned source clusters across long-context benchmarks, multi-turn conversation studies, RAG and context-management papers, instruction hierarchy, and official prompt-engineering docs. Phase 3 retrieved sources with web search because the topic is fast-moving and current sources were needed. Phase 4 triangulated major claims across independent source clusters. Phase 4.5 refined the outline after the evidence showed that output-side long-form degradation and persistent context rot deserved separate treatment. Phase 5 synthesized the report around six findings.

Sources Consulted

Total cited sources: 25.

Source types: academic and benchmark papers: 20; official platform documentation: 3; applied software-engineering research paper: 1; comparative evaluation paper: 1.

Temporal coverage: the cited work ranges from 2023 through 2026, with emphasis on 2024-2026 because long-context and multi-turn degradation research is moving quickly.

Verification Approach

Major claims were supported by at least three independent source clusters where possible. Long-context degradation was triangulated from Lost in the Middle, RULER, NoLiMa, LongBench v2, and LongGenBench [1][2][3][4][5]. Multi-turn degradation was triangulated from Laban et al., intent-mismatch work, MT-OSC, and AdaCoM [6][7][8][9]. Mitigations were triangulated from context engineering, RAG studies, instruction hierarchy, grounding benchmarks, and official platform docs [11][13][15][16][17][18][19][22].

Claims-Evidence Table

Claim ID	Major Claim	Evidence Type	Supporting Sources	Confidence

C1	Long context windows do not guarantee reliable use of all supplied tokens.	Benchmarks	[1], [2], [3], [4]	High
C2	More retrieved context can help and then hurt when hard negatives and irrelevant passages enter the prompt.	RAG studies	[13], [14], [15]	High
C3	Multi-turn conversations can degrade even when the needed task information appears somewhere in the dialogue.	Multi-turn simulations and mitigation papers	[6], [7], [8], [9]	High
C4	Instruction ambiguity and instruction conflict are distinct reliability hazards.	Instruction hierarchy papers and platform docs	[11], [12], [17], [18], [19]	High
C5	Long-form generation has its own degradation modes beyond input retrieval.	Generation and grounding benchmarks	[5], [22], [23]	Medium-High
C6	Context engineering is the dominant mitigation pattern across research and production guidance.	Survey, RAG, agent, and documentation sources	[8], [9], [13], [16], [17], [18], [19]	High

Report Metadata

Research Mode: Standard.

Total Sources: 25.

Approximate Word Count: 5,900.

Generated: 2026-06-21 local shell date.

Validation Status: Automated validation run after packaging.