

Addressing LLM Context Fatigue via Summarization

Aditi Razdan

Singapore American School, 10 Cuscaden Walk, Singapore, 249693, Singapore; aditirazdan1706@gmail.com

ABSTRACT: This paper addresses context fatigue in large language models, where, after a long conversation, the model begins producing low-quality results that do not follow logically from the prior conversation. To address this issue, we created an in-session summarization tool that condenses prior dialogue into a compact context representation, helping the model recover from context fatigue. Our technique is evaluated through controlled, quantitative experiments that measure retention of injected facts and constraints under long conversational drift. Empirically, our experiment demonstrates that the summarization technique substantially mitigates context fatigue, increasing mean recall by nearly two facts on a five-point scale and restoring retention at 250 turns from near-zero performance (0.0–0.4) to approximately four out of five facts (4.0–4.2). Additionally, this Multi-LLM framework allows users to transfer context and conversations from one model instance to another or across different model families, summarizing conversations between Gemini variants or transferring them to locally hosted models.

KEYWORDS: Computer Science, Artificial Intelligence, Large Language Models, Context Management.

■ Introduction

Imagine working with a large language model (LLM) over the course of an afternoon: refining a research question, debugging code, and iterating on explanations until they are just right. At first, the model behaves like a capable conversational partner, tracking your criteria and prior decisions. Over time, however, earlier details begin to fade, and responses grow less consistent. This degradation often manifests as failures to respect earlier constraints or recall facts that were previously introduced and relied upon.

Today's large language models (LLMs), such as Gemini and ChatGPT, are extremely powerful and have increasingly become conversational partners that assist humans across a wide range of tasks. These tasks include deep research and synthesis (e.g., GPT-based deep research tools), exploring and constructing mathematical arguments (work by Georgiev *et al.* on AI-assisted mathematical discovery), software development and debugging (e.g., Claude and other coding-focused LLMs), providing tax or financial guidance through commercial platforms (e.g., GPT-powered assistance integrated into TurboTax), and even offering forms of emotional or social companionship.^{1–5} In many of these settings, users rely on the model to retain key information, such as goals and constraints, over long conversations.

As tasks become increasingly complex, users naturally interact with LLMs for longer, more involved conversations, relying on the model to remember more key details. A key pain point emerges in these extended interactions, though. LLMs are limited in the amount of information they can handle at once, and so when this limit is exceeded, models may behave unpredictably. As conversations progress, LLMs may forget earlier parts of the session, repeat information, or hallucinate facts, producing responses that are incoherent or misaligned with the user's intent. Prior research has referred to this phenomenon as context fatigue or context drift.^{6,7} Importantly, this failure does not

occur abruptly; it worsens as the interaction continues. Notably, such degradation can occur even when the total conversation length remains within a model's nominal context window.

Unlike systems with explicit recovery mechanisms, an LLM experiencing context drift degrades progressively and cannot reliably recover on its own.^{6,7} Once important context is lost, outputs become increasingly inconsistent with the user's earlier intent.

At present, users typically address context drift by starting a new session with a fresh LLM instance. While starting a new session resets the model's context, it requires the user to repeat themselves and summarize earlier parts of the conversation (similar to movies like *Memento* or *50 First Dates*, in which the character must start from scratch every day). This workaround shifts the burden of recovery entirely onto the user and disrupts conversational flow.

Motivated by this gap, *this paper asks whether structured summarization can be used to recover from this kind of degradation, and presents a system and evaluation framework that tests this directly.* We developed a method to refresh the context. When the model begins giving lower-quality responses, the user can trigger a refresh. In addition, this refresh mechanism can be invoked programmatically after a prolonged interaction to evaluate whether coherence and constraint retention can be restored without user intervention. A different background summarization module condenses the conversation, keeping only the most critical points. This allows the conversation to continue without requiring a full restart.

Our approach is model-agnostic: it is essentially a memory/context transfer mechanism that allows one to port the memory of one LLM to another instance of the same LLM or to an entirely different LLM (e.g., transfer context from Ollama to Gemini). In our implementation, this transfer occurs through a structured summary that serves as a self-contained repre-

sentation of prior dialogue. This allows users to leverage the strengths of multiple LLMs seamlessly.

We hypothesize that our approach works well because, at its core, context drift arises from the model's limited context window and its self-attention mechanism.⁸ Prior work suggests that as the length of a conversation grows, transformer-based models increasingly de-prioritize information from earlier or middle portions of the context, even when that information remains relevant to the task at hand.⁶ Our evaluation framework operationalizes this phenomenon by injecting facts and constraints early in a conversation and measuring their retention after many subsequent turns. Our summarization approach addresses this by condensing the conversation while retaining key facts, the overall logical flow, the user's goals, and any unresolved questions.

We implemented this method in a framework that allows users to select multiple LLMs with interface controls to refresh the current context or transfer it to another model. Beyond the interactive system, we also introduce a controlled evaluation setup that quantitatively measures context retention before and after summarization. Our experimental evaluation shows that summarization is fast and that refreshed contexts produce responses that recover factual and constraint-level information more reliably than uninterrupted long-context interactions. These results suggest that refreshing the context is an effective mechanism for mitigating context fatigue, improving both usability and continuity in extended interactions.

■ Background

Large Language Models (LLMs) are a type of artificial intelligence trained on large datasets to better understand and generate human-like language. They learn patterns from data to perform tasks like writing, translation, and answering questions. When generating a response, an LLM draws on its internal knowledge from training, developer instructions, information it can access online or in a database, its conversation history, and the user's immediate question.

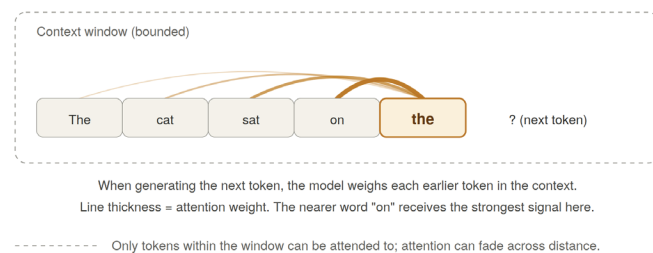


Figure 1: Self-attention lets the model weigh each earlier token in its bounded context window when generating the next one. Nearer tokens receive stronger attention weights, and tokens that fall outside the window cannot be attended to at all, which is the underlying mechanism that produces context fatigue as a conversation grows.

Most modern LLMs are built on the Transformer architecture.⁹ A Transformer processes text one token (a word or part of a word) at a time, and on each step, it uses a mechanism called self-attention to decide which earlier tokens in the conversation are most relevant for predicting the next one. Figure 1 illustrates this relationship: when generating a given word, the model looks back across its input and assigns weights that

reflect how strongly each prior token should influence the output.⁸ Crucially, this lookback is bounded. The model can only attend to tokens that fit within a fixed-size context window, and, as we discuss next, even within that window, its ability to use information effectively can degrade over long inputs.

To focus on context, we define two types of context in LLMs: nominal context and effective context. The nominal context refers to the maximum number of tokens a model can technically process. The effective context, on the other hand, is the portion of that input the model actually uses when generating its response. The cause of this discrepancy is an active area of research. Researchers use terms such as "context fatigue," "context drift," and "Contextual Distraction Vulnerability" (CDV) to describe what happens when a model's conversation history grows beyond what it can effectively handle, often less than the nominal context.¹⁰ This can lead to deterioration in the LLM's performance. This is clearest when the LLM begins to output contradictions, drift from the original topic, and forget earlier details. The model may also lose track of people or entities mentioned earlier in the conversation.^{6,7} These issues have been documented across multiple studies and analyses and are becoming increasingly apparent as users interact with language models over longer and more complex sessions.^{6,7,10,11} Importantly, recent evidence suggests that effective context can degrade even when the total input remains within a model's nominal context limit.

To this day, the precise mechanisms underlying context fatigue remain an open research question. The work of An *et al.* suggests that the primary issue is not memory capacity but rather a lack of training on long-range dependencies.⁷ The authors believe this because, in their experiments, they hide numbers at different positions within long inputs/contexts and find that the model often forgets the ones that appeared earlier in the passage. Testing two smaller models, one trained on longer texts than the other, showed that seeing longer examples helped the model remember better, which led them to conclude that the issue was training rather than memory size.

Conversely, the work of Liu *et al.* identifies a "U-shaped" performance curve. This means that the LLMs are best at recalling information when it's placed at the beginning or end of the context; they perform worst when key details are located in the middle.⁶ The authors argue that when they move the answer to different parts of a long text, the model performs best when the answer is at the start or end and worst when it is in the middle. This consistent U-shaped pattern across tasks and prompts led them to conclude that the model generally struggles with information in the middle of long inputs.

Whatever the root cause may be, the effects of context fatigue are severe and hard to mask: the LLM hallucinates more often and forgets prior conversations. As we show later in this paper, these effects can be observed in failures to retain explicit facts and constraints introduced early and relied upon throughout an extended interaction.

Prior Works:

Developers have attempted several workarounds to prevent context fatigue, and one of the most widely used approaches

is retrieval-augmented generation (RAG).¹² RAG works by storing key information externally, either in a database or on the web, and retrieving it whenever necessary. This enables the model to remain accurate without having to “remember” everything.¹³

Nevertheless, RAG has some limitations. For example, its performance heavily depends on the quality and completeness of its external database; outdated or missing information can lead to inaccuracies.¹²

For instance, a RAG-based assistant retrieving both a valid government report and a misleading blog post can be thrown off by the irrelevant document; a recent study shows that even one “hard distracting” passage reduces answer accuracy significantly compared to giving only the correct passage.¹⁴ In addition, RAG systems are typically designed around explicit document retrieval and are less suited to preserving the evolving goals, constraints, and conversational state that arise organically over a long interactive session.

Another major strategy aims to improve the model's understanding of word positions in long texts. For example, one idea is to modify the model's internal positioning system to leverage its strong understanding of nearby words to handle words that are far apart better.⁷ The goal is to improve the model's use of its full context window without additional data, but this method still requires costly retraining of the entire model and may not fully address the problem of the model's attention weakening over long distances. This is an exciting but still nascent field; hence, no prior work has settled on a final word on how best to address it.

At the application level, commercial LLM products offer partial workarounds to context fatigue, though none target it directly. The most common is simply starting a new conversation, sometimes after the user manually summarizes the prior session, which is a usability workaround that shifts recovery onto the user and disrupts conversational flow. Some systems also maintain lightweight long-term memory; Claude, for example, uses a small preference file (Claude.md) to remember user information across sessions.¹⁵ This form of memory supports personalization across conversations but is not designed to preserve the evolving goals and constraints of an ongoing task-focused session, which is the problem our approach addresses.

■ Problem Statement

In this study, we propose another approach to address context fatigue: a model-agnostic refresh mechanism that compresses the session's key information into a short, structured summary. This approach reduces the need for manual restarts and helps reduce context fatigue. As a benefit, our approach can migrate context from one model to another, allowing a user to use ChatGPT for an initial conversation and then continue seamlessly with Gemini or any other LLM. In our implementation, this migration is achieved through summary-based transfer rather than shared internal state.

The following sections describe our system for refreshing and transferring context and evaluate its impact on performance and accuracy. We also introduce a controlled evaluation

framework that enables testing of this mechanism under a variety of synthetic multi-turn conversational workloads.

■ Methods

Our solution builds on a core architectural idea: using LLMs to generate full-history summaries that act as a compact representation of prior dialogue. This approach allows extended discussions to continue naturally while supporting seamless transitions between models.

The system integrates automated summarization within an intuitive interface that acts as a central control panel for interacting with different LLM providers. By doing so, it helps overcome the limits of traditional context windows and maintains continuity in both within-model and cross-model conversations.

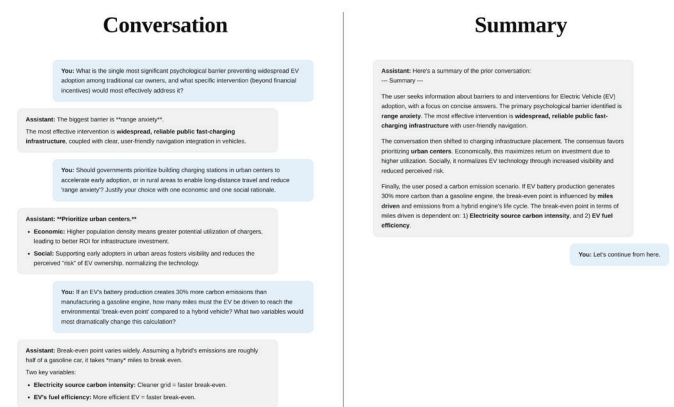


Figure 2: The system condenses a full conversation into a structured summary that preserves key facts and goals. Even in this short example, the summary keeps the core facts, goals, and constraints while dropping conversational filler, and this compression becomes far more pronounced as the underlying conversation lengthens.

Figure 2 illustrates a short exchange and its corresponding summary, showing how the system condenses the main ideas into a concise form that can be reused or transferred across sessions. We used this simple example for illustrative purposes, but we acknowledge that it doesn't fully showcase the strength of our system, since the conversation isn't very long. When the conversation is much longer, the summary's compression is more significant.

At its foundation, the system comprises three core modules: (1) a Context Manager that maintains and tracks both active and summarized conversation states, (2) a Summarization Engine responsible for compressing dialogue into concise and structured digests, and (3) a Multi-LLM Interface that enables model selection and cross-model context transfer by building on the Summarization Engine. Together, these components form an adaptive pipeline that supports continuous dialogue without loss of coherence. While these modules are conceptually distinct, the current implementation emphasizes coordination through message passing rather than persistent shared memory.

The Context Manager serves as the system's persistent memory layer. In practice, it maintains the active conversation state during the running session and synchronizes user preferences, such as the preferred model and feedback style, using a

local SQLite database. One way to think about this database is that it functions similarly to Claude's Claude.md file (previously mentioned in the Prior Works section), except that it works across multiple models. Unlike long-term conversational memory, this database stores only lightweight preference information rather than semantic summaries of dialogue.

When the LLM's responses begin showing signs of context fatigue, the user can trigger the Summarization Engine. Context drift can be identified by the user through qualitative degradation such as repetition, contradictions, or failure to recall recent details. In addition, the system supports summarizing after a prolonged interaction for evaluation, independent of user judgment. The Summarization Engine processes the entire conversation as a single structured prompt. The summarization template explicitly instructs the model to preserve core facts, logical flow, user goals, and unresolved questions. Each summary is generated from the full conversation history rather than from prior summaries, which avoids compounding compression errors across multiple refreshes. This summary is then used as the active context that the model sees on subsequent turns, allowing the conversation to continue without restarting.

Your current active feedback & preferences:

• []

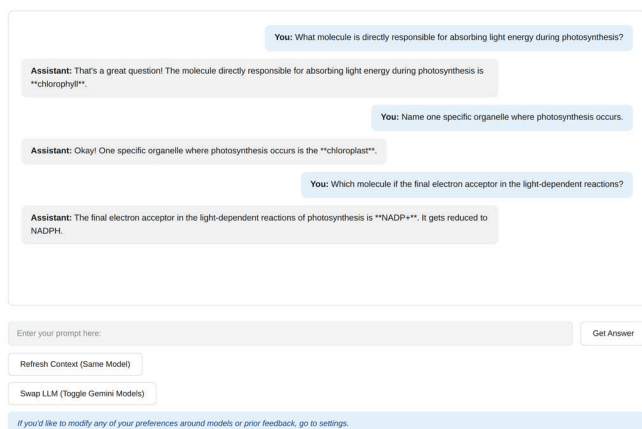


Figure 3: The interface enables controlled context refresh and cross-model transfer, allowing users to recover coherence without restarting the conversation. The image displays a sample conversation between a user and the system. The dedicated "Refresh" and "Transfer" controls let users compress context within a model or move it to another, and visible summary markers show exactly where each context change occurred, keeping users aware of when and how their context is managed.

The interface provides dedicated controls for this process through "Refresh" and "Transfer" functions, allowing users to either compress context within the same model or move summarized conversations to different LLMs with specialized capabilities. This dual-mode structure allows users to preserve conversational continuity while strategically leveraging the strengths of different models. For example, a conversation begun with ChatGPT for creative brainstorming can be transferred to Gemini for concise, factual refinement, without the user having to summarize manually.

Figure 3 shows the cross-LLM chat layout, including the "Refresh" and "Transfer" buttons that manage context compression and model switching.

This layout presents the conversation history with visual distinctions between user prompts, model responses, and transferred contexts. Crucially, it also includes visual markers for summary events, showing where the context was compressed and allowing the user to read the generated summary. The interface prominently features the "Refresh" button for context compression and the "Transfer" button for model switching, with clear visual indicators showing the current active model and a summary of the context state. This design provides users with immediate awareness of context transformations and model transitions.

Before the system summarizes or switches models, it organizes all conversations into a shared message history that stores each exchange as a simple pair of speaker and message, such as `{speaker: "user", message: "Hello"}`. This ensures that every model reads the conversation in a consistent structure. When a user sends a message, the system formats the entire dialogue into a readable transcript that labels each speaker so that different models can follow the same flow of context.

There is no heavy preprocessing or data filtering involved, as the goal is to preserve what was actually said. Once a model responds, the system displays its reply in the chat window with light visual cues that separate user and model turns. When the user chooses to summarize or transfer the chat, that same structured history is passed through the summarization step, and the resulting summary or continuation is added back into the conversation. This design keeps interactions natural while enabling smooth handoffs between models without extra setup or data loss.



Figure 4: The multi-LLM configuration panel supports rapid switching between hosted and local models, demonstrating the portability of summarized conversational state. Because the same structured summary can be loaded into either a hosted Gemini model or a locally run Ollama model, the panel shows that summarized context is portable across model families rather than tied to a single provider.

Figure 4 illustrates how the user can use different LLMs by entering API keys in their terminal and configuring model-specific settings. In the current system, this interface supports both hosted models and locally run models. This centralized configuration enables rapid switching between model providers, in line with the model-agnostic design concept.

The interface is visually divided into the sidebar, the chat window, and the control bar. The sidebar contains preference management options that let users select from multiple models and adjust feedback settings such as tone, format, and verbosity. These preferences are stored locally in SQLite, ensuring user customization persists across sessions without relying on remote servers or external authentication. The chat window displays the conversation as a visually structured dialogue, distinguishing between user prompts, model responses, and summarization events through color-coded formatting. Each summarization event is inserted as a visible summary node that clearly shows when a refresh or transfer has occurred. The control bar includes the "Refresh Context" and "Swap LLM"

buttons, which directly trigger summarization and model-switching events. These controls provide transparency and empower users to decide when and how context management occurs.

Implementation:

The system is implemented using Streamlit, providing a clean, interactive interface for managing multiple LLMs.¹⁶ Users can select which model to interact with, and the interface displays conversation history alongside controls for context management. The current implementation supports both hosted Gemini models and locally run models accessed via an Ollama backend.

Conversation context is stored locally in a structured SQLite database that tracks user settings, such as preferred model and tone, while the active conversation history is maintained in memory during a running session. When the user triggers a context refresh, the Summarization Engine receives the entire conversation as a structured prompt. The summarizer model uses a template to extract core facts, the logical flow, user goals, and unresolved questions.

The system maintains two parallel records of the conversation: a complete history of every message exchanged, and the most recent summary produced by the Summarization Engine. On any given prompt from the user, the model is called with the latest summary followed by all the messages exchanged since that summary was generated. This means that the active context that the model actually sees is bounded.

When the next summarization event is triggered, we use the entire accumulated history (not just the last few messages and summary) to summarize the full conversation.

This design has two characteristics worth noting. First, the summary is limited to 1000 words and focuses only on core facts, logical flow, user goals, and unresolved questions. Summaries, therefore, remain compact even as the underlying conversation grows long, which is what makes them useful as a replacement for the active context. Second, because each summarization passes the full accumulated history to the summarizer in a single call, both the input length and the resulting latency of that inference call grow as the conversation lengthens. This means that on a sufficiently long conversation, the accumulated history could exceed the summarization model's own context window, at which point the Summarization Engine becomes susceptible to the same context fatigue our system is designed to mitigate. Our approach doesn't necessarily eliminate the fundamental constraint of context drift; rather, it extends the duration over which models with bounded context can sustain a coherent conversation. One thing to highlight is that since the summarization model performs a single task (summarizing content), it might be possible that in the future we can have summarization models with very large context windows, since they are trained specifically for this one task rather than being general-purpose models.

Beyond the interactive chat interface, the system includes an evaluation mode that measures context retention under sustained conversational load. Each evaluation run begins by injecting a compact block of five ground-truth facts (a

codename, a numerical budget, a behavioral rule, a forbidden item, and a date) phrased as information the model should remember for later reference. Values are generated from a fixed random seed so that identical configurations produce identical fact sets across runs.

Following this injection, the system replays user turns from one of our synthetic task-oriented datasets while the model under test generates assistant responses live. Because the replayed prompts are unrelated to the injected facts, later recall requires the model to retain the information across many turns rather than rely on repetition. After the conversation reaches a specified length, the model is probed for recall of the injected facts. Depending on the evaluation condition, the system may refresh the context through summarization and, optionally, transfer the conversation to a new model instance before issuing a second probe. This design isolates the effects of interaction length on memory while explicitly testing whether summarization recovers correctness and whether summaries bridge across model instances.

■ Results and Discussion

We evaluate summarization as a mitigation strategy for context fatigue, which we define as the degradation of a model's ability to recall prior constraints and facts over the course of a long conversation. Our central hypothesis is that compressing a prolonged dialogue into a structured summary and re-initializing the model with that summary can recover factual and constraint-level retention while preserving conversational continuity.

We implemented and tested the evaluation pipeline on Gemini 2 (2.5 Flash and 2.0 Flash Lite) and Qwen3-235B, and we report detailed quantitative results for Gemini 2.0 Flash Lite. Gemini was accessed via the hosted API, while Qwen3-235B was executed locally. Evaluating both a cloud-hosted model and a locally deployed model allowed us to measure context drift and refresh across distinct model families under the same protocol. For Gemini, we manually control its context window by supplying only the most recent k tokens (a recency-based truncation), reflecting the bounded sequence processing inherent to transformer-based models.⁸ For Ollama, we can set the context window directly via the `OLLAMA_CONTEXT_LENGTH` environment variable. We set it to approximately 12,000 tokens.

The chosen domains are instruction-heavy and constraint-sensitive, requiring sustained reasoning and careful tracking of prior assumptions. Prompts span definitions, symbolic manipulation, derivations, proofs, applied problem solving, and optimization scenarios, producing extended analytical interaction rather than repetitive exchange and better approximating the structure of real task-focused conversations.

Figure 5 shows the end-to-end evaluation pipeline used to test context retention under sustained conversational load.

Synthetic Dataset Evaluation Runner

Evaluation Mode
☐ Single Run
☒ Batch Grid

Dataset
 Synthetic Calculus (synthetic_calculus_5000.json)

LLM provider
 Ollama

Ollama model name

Batch Evaluation (One-Click Grid)

Datasets to include
 Choose an option

Turn counts (comma-separated)
 50,100,250

Conditions to include
 Choose an option

Max context tokens (approx, 0 = full history)
 0

Output JSON filename
 evaluation_results_171239763.json
 Incremental JSON filename (optional)
 evaluation_results_171239763.json

Figure 5: The evaluation framework isolates context drift and quantifies recovery after summarization, enabling controlled measurement of factual retention across conditions. By injecting a fixed set of facts, replaying many unrelated turns, and then probing recall before and after a refresh, the pipeline isolates the effect of conversation length and directly measures how much retention summarization restores.

We evaluate context retention under two experimental conditions. While the system supports summary-based transfer across model instances, this study's quantitative evaluation focuses exclusively on within-model context refresh. In the baseline condition, the conversation proceeds uninterrupted from fact injection through the full sequence of user turns, after which a single probe is issued. Probes consist of five short questions labeled A through E and require responses in a fixed template to minimize parsing ambiguity.

In the refresh condition, the model is first probed after the long conversation, after which the entire interaction is summarized. A new chat session is initialized using this summary as the sole prior context, and the probe is issued again to measure post-refresh retention.

Our primary metric is match recall on the five injected facts. Each probe response is parsed into five labeled answers (A through E), one per injected fact, using the fixed response template described above. Each labeled answer is then compared to its corresponding ground-truth value using a case-insensitive exact string match after whitespace trimming, yielding a per-item score of 0 or 1. The overall recall score for a probe is the sum of these per-item scores, ranging from 0 to 5. We also retain the LLM's responses to identify which types of constraints (e.g., dates, numerical budgets, or stylistic rules) are most vulnerable to drift.

Each cell in Tables 1 and 2 reports the mean and standard deviation of recall across five independent runs of the same configuration, where each run uses a distinct random seed for fact-block generation. We use exact match because the injected facts are short and unambiguous, and because many real-world constraints (dates, numerical limits, prohibited actions) require precise adherence; relaxing the matching criterion would obscure the effects of context fatigue. Higher scores indicate better fact retention.

Table 1: Results of evaluating the baseline and our refresh summarization system on Gemini 2.0 Flash Lite, varying the number of turns in a synthetic calculus long-term conversation dataset we created. Refresh outperforms the baseline at every turn count, and the gap is widest at 250 turns, where baseline recall falls to 0.0 while refresh still retains 4.2 out of 5 facts.

Calculus	Baseline	Refresh
50 Turns	4.2 ± 0.8	4.8 ± 0.4
100 Turns	3.8 ± 1.1	4.4 ± 0.5
250 Turns	0.0	4.2 ± 0.8

Table 2: Results of evaluating the baseline and our refresh summarization system on Gemini 2.0 Flash Lite, varying the number of turns in a synthetic microeconomics conversation dataset we created. The same pattern holds on this harder dataset: baseline recall drops to 0.4 out of 5 at 250 turns while refresh sustains 4.0, though both conditions show slightly lower recall and higher variance than calculus.

Microeconomics	Baseline	Refresh
50 Turns	4.0 ± 0.7	4.6 ± 0.5
100 Turns	3.2 ± 1.3	4.2 ± 0.8
250 Turns	0.4 ± 0.5	4.0 ± 0.9

Several consistent patterns emerge across both datasets. First, the refresh mechanism consistently outperforms the baseline across all turn counts, with the largest gains observed at 250 turns. At this extreme length, baseline recall approaches zero (calculus: 0.0; microeconomics: 0.4 ± 0.5), while refresh maintains robust performance (calculus: 4.2 ± 0.8; microeconomics: 4.0 ± 0.9). This suggests that summarization is most valuable precisely when context fatigue is most severe, namely after very long conversations, when the model has effectively lost access to early information.

Second, the advantage of refresh increases with conversation length. At 50 turns, the improvement is modest (calculus: +0.6 facts; microeconomics: +0.6 facts). At 100 turns, the gap widens (calculus: +0.6 facts; microeconomics: +1.0 facts). By 250 turns, the difference is substantial (calculus: +4.2 facts; microeconomics: +3.6 facts). This trend supports our hypothesis that summarization becomes increasingly beneficial as conversational drift accumulates.

Dataset Differences:

Notably, the microeconomics dataset proved more challenging than calculus across all conditions. Baseline recall at 100 turns was lower for microeconomics (3.2 ± 1.3) than calculus (3.8 ± 1.1), and refresh recall showed higher variance (microeconomics 4.2 ± 0.8 vs. calculus 4.4 ± 0.5 at 100 turns). This may reflect the greater conceptual density of microeconomics conversations, which require tracking interrelated constraints (supply/demand relationships, elasticity concepts) rather than the more linear derivations in calculus. The higher standard deviations in microeconomics (up to ±1.3) also indicate greater run-to-run variability, suggesting that certain types of content are more susceptible to forgetting.

Reliability and Variance:

The standard deviations themselves provide insight into reliability. Baseline conditions show increasing variance as conversations lengthen (calculus: from ±0.8 at 50 turns to effectively 0 at 250 turns; microeconomics: from ±0.7 to ±0.5),

indicating that while most runs fail by 250 turns, the timing of failure varies. Refresh conditions maintain relatively stable variance across turn counts (± 0.4 - 0.9), demonstrating that summarization not only improves average performance but does so consistently.

The recall is intentionally strict. The injected facts are short, unambiguous, and format-controlled, and many real-world constraints, such as dates, numerical limits, or prohibited actions, require precise adherence.

Using looser or fuzzy matching would obscure the effects of context fatigue and weaken causal interpretation.

All evaluations have fixed settings. Fact blocks are generated from a deterministic random seed, ensuring consistent ground-truth values across runs. Conversational inputs are replayed from static datasets in a fixed order, and all evaluation parameters, including provider, model variant, dataset choice, target number of user turns, and evaluation condition, are explicitly specified through the evaluation interface.

Nevertheless, some sources of variation remain. In particular, model sampling behavior and backend differences can affect recall, and decoding parameters are left at provider defaults.

Discussion:

Since we began this study, related ideas have appeared in commercial tools such as Claude Code, which provides two distinct user-facing commands for context management: `/clear`, which wipes the active conversation entirely, and `/compact`, which asks the model to summarize the conversation and then replaces the history with that summary.¹⁷ These features reflect a growing recognition that long-running conversations require explicit context management rather than uninterrupted accumulation. One important distinction remains, however. Our system treats summarized context as a transferable artifact that can be exported across model instances, whereas Claude Code currently operates within a single-model setting and does not support cross-model context transfer.

The quantitative results support the practical value of this approach. At 250 turns, baseline models retain effectively zero facts; they have, for practical purposes, "forgotten" the conversation's starting constraints. Users experiencing this would need to restart or manually recap. Our refresh mechanism restores approximately 80% of facts (4.0-4.2 out of 5), meaning users can continue seamlessly without repetition. This 80% retention rate after extreme conversational length suggests that summarization captures most critical information while discarding irrelevant drift.

Our approach also has clear limitations. Summaries are interpretive compressions of prior dialogue and therefore cannot preserve all information with perfect fidelity. Any detail that is omitted or misrepresented during summarization is effectively lost in subsequent interactions. In addition, our evaluation focuses on short, explicit facts and constraints, which provide a strict and interpretable signal but do not capture more abstract aspects of conversational continuity, such as evolving reasoning strategies or nuanced preferences. The summarization process itself depends on the underlying model's behavior, which introduces additional sources of variability.

The dataset-dependent differences we observed (microeconomics showing lower recall and higher variance than calculus) highlight an important avenue for future work: understanding which conversation types are most susceptible to context fatigue. If certain topics or reasoning styles are particularly vulnerable, summarization strategies might be tuned accordingly; for example, more frequent refreshes for conceptually dense domains.

There are other ways this problem could be addressed. Retrieval-based systems can reintroduce relevant information from external stores, and architectural changes aimed at improving long-context attention may reduce effective context degradation.^{7,12,13} Each of these approaches targets a different aspect of the problem and comes with its own trade-offs. Future work could explore richer automatic summary representations, adaptive mechanisms that trigger refreshes when degradation is detected, and tighter integration between summarization and retrieval. More broadly, treating context as an explicit, portable object rather than an implicit buffer opens new directions for building LLM systems that better support sustained, flexible interaction over time.

Conclusion

In this study, we explored using summarization to address the limited context supported by LLMs. We built a full chat application from scratch to experiment with this idea, created multi-round synthetic datasets on topics in mathematics and economics, and designed an interface that allows us to toggle between different options (LLMs and evaluation conditions). In sum, this effort allowed us to check whether our generic approach worked for both cloud-based and local models. Our results suggest that strategic summarization can help LLMs remember facts introduced earlier in the conversation, even at extreme conversational lengths where baseline retention collapses.

More broadly, this work points to a shift in how we should think about extended interaction with LLMs. As these models take on longer and more consequential tasks (e.g., tutoring students through a semester, supporting writers through successive drafts, assisting researchers across multiple sessions), the limiting factor is clearly continuity. As conversational systems become a default interface to computation and knowledge, the ability to sustain coherent interaction over time will matter at least as much as the ability to generate a good response for any single question or prompt. Treating context as an explicit, portable object rather than an implicit buffer reframes this limitation as something that can be engineered around.

Acknowledgments

The author thanks her mentor, Sebastian Angel, for providing helpful feedback during the writing of this manuscript.

References

1. OpenAI. *Introducing Deep Research*. 2025. <https://openai.com/index/introducing-deep-research> (accessed April 24, 2026).

2. Georgiev, B.; Gómez-Serrano, J.; Tao, T.; Wagner, A. Z. Mathematical Exploration and Discovery at Scale (2025). arXiv:2511.02864. <https://arxiv.org/abs/2511.02864> (accessed April 24, 2026).
3. Anthropic. *Introducing Claude*. 2023. <https://www.anthropic.com/index/introducing-claude> (accessed Feb 14, 2026).
4. Intuit. *Introducing Intuit Assist: The Generative AI-Powered Financial Assistant for Small Businesses and Consumers* (2023). <https://investors.intuit.com/news-events/press-releases/detail/43/introducing-intuit-assist-the-generative-ai-powered-financial-assistant-for-small-businesses-and-consumers> (accessed April 24, 2026).
5. Liu, B.; Sundar, S. S. Designing for Emotional Support with Social Chatbots. *Proc. ACM CHI Conf. Hum. Factors Comput. Syst.* 2018, 1: 1–12. DOI: 10.1145/3173574.3173851.
6. Liu, N. F.; Lin, K.; Hewitt, J.; Paranjape, A.; Bevilacqua, M.; Petroni, F.; Liang, P. Lost in the Middle: How Language Models Use Long Contexts. *arXiv Preprint*, arXiv:2307.03172 [cs.CL], 2023.
7. An, C.; Zhang, J.; Zhong, M.; Li, L.; Gong, S.; Luo, Y.; Xu, J.; Kong, L. Why Does the Effective Context Length of LLMs Fall Short? *arXiv Preprint*, arXiv:2410.18745 [cs.CL], 2024.
8. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *Adv. Neural Inf. Process. Syst.* 2017, 30: 5998–6008.
9. Ferraris, A. F.; Audrito, D.; Di Caro, L.; Poncibò, C. The Architecture of Language: Understanding the Mechanics Behind Large Language Models. *Cambridge Forum AI Law Gov.* 2025, 1(1): e11. DOI: 10.1017/cfl.2024.16.
10. Hong, K.; Troynikov, A.; Huber, J. Context Rot: How Increasing Input Tokens Impacts LLM Performance. Chroma Technical Report, 2025. <https://research.trychroma.com/context-rot> (accessed April 24, 2026).
11. Zhuang, Y.; Yu, X.; Wu, J.; Sun, X.; Wang, Z.; Liu, J.; Su, Y.; Shang, J.; Liu, Z.; Barsoum, E. Self-Taught Agentic Long Context Understanding. *arXiv Preprint*, arXiv:2502.15920 [cs.CL], 2025.
12. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.-T.; Rocktäschel, T.; Riedel, S.; Kiela, D. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv Preprint*, arXiv:2005.11401 [cs.CL], 2021.
13. Amazon Web Services. *What Is Retrieval-Augmented Generation (RAG)?* 2025. <https://aws.amazon.com/what-is/retrieval-augmented-generation/> (accessed Feb 14, 2026).
14. Amiraz, C.; Cuconasu, F.; Filice, S.; Karnin, Z. The Distracting Effect: Understanding Irrelevant Passages in RAG. *Proc. Annu. Meet. Assoc. Comput. Linguist.* 2025, 63: 18228–18258.
15. Anthropic. *Manage Claude's Memory*. 2025. <https://code.claude.com/docs/en/memory> (accessed Feb 14, 2026).
16. Streamlit. *Streamlit: The Fastest Way to Build and Share Data Apps*. 2024. <https://streamlit.io/> (accessed Feb 14, 2026).
17. Anthropic. *How Claude Code Works* (2026). <https://code.claude.com/docs/en/how-claude-code-works> (accessed April 24, 2026).

■ Authors

Aditi Razdan is a junior at Singapore American School with an interest in artificial intelligence and machine learning systems. She plans to pursue computer science in college, focusing on AI and systems design, particularly at the intersection of technology and social justice.