

DisCEdge: Distributed Context Management for Large Language Models at the Edge

Mohammadreza Malekabbasi
 TU Berlin
 Berlin, Germany
 mm@3s.tu-berlin.de

Minghe Wang
 TU Berlin
 Berlin, Germany
 mw@3s.tu-berlin.de

David Bermbach
 TU Berlin
 Berlin, Germany
 db@3s.tu-berlin.de

Abstract

Deploying Large Language Model (LLM) services at the edge benefits latency-sensitive and privacy-aware applications. However, the stateless nature of LLMs makes managing user context (e.g., sessions, preferences) across geo-distributed edge nodes challenging. Existing solutions, such as client-side context storage, often introduce network latency and bandwidth overhead, undermining the advantages of edge deployment.

We propose *DisCEdge*, a distributed context management system that stores and replicates user context in tokenized form across edge nodes. By maintaining context as token sequences rather than raw text, our system avoids redundant computation and enables efficient data replication. We implement and evaluate an open-source prototype in a realistic edge environment with commodity hardware. We show *DisCEdge* improves median response times by up to 14.46% and lowers median inter-node synchronization overhead by up to 15% compared to a raw-text-based system. It also reduces client request sizes by a median of 90% compared to client-side context management, while guaranteeing data consistency.

Keywords

Edge Computing, Edge Intelligence, Geo-distributed Storage, Large Language Models (LLMs)

1 Introduction

Large language models (LLMs) are becoming key enablers for intelligent, context-aware mobile applications. Their ability to understand and leverage context is what differentiates them from simpler AI models. However, their effectiveness critically depends on managing this context, which becomes a primary challenge in distributed edge environments. While recent advances in edge AI have shown promising results in using LLMs—particularly in robotics [2, 5]—significant challenges remain in extending these capabilities to mission-critical platforms such as life-saving drones [28], autonomous vehicles [29], and even smartphones [39]. These challenges stem from a fundamental trade-off: commodity hardware, reliable latency, and a singleton deployment cannot all be achieved simultaneously [6]. Thus, relying on a single LLM instance—whether on-device, at the edge, or in the cloud—often fails to meet the strict demands of mobile and resource-constrained environments [29].

The trend of offering AI inference as a service follows serverless computing paradigms, where different models can be dynamically loaded and shared among multiple clients, akin to functions in a FaaS platform. The push to migrate LLM inference to the edge is driven by the need for low latency and privacy. However, this migration introduces significant hurdles. Edge devices have limited resources, hindering the performance of large models, especially

when multiple models must run concurrently to serve complex applications like humanoid robots or autonomous vehicles [18, 29]. While many studies have explored cloud-and-device [18, 23, 25, 32, 33, 37] and edge [40] collaboration, the practical deployment of fully distributed LLM inference services at the edge remains largely conceptual [29].

A fundamental barrier to such distributed deployments is context management. LLMs are stateless by design; context—such as user sessions, preferences, and regional data—must be managed externally. In a geo-distributed system, this context must be replicated across nodes to ensure a consistent user experience, which is notoriously difficult as it requires balancing high availability with strong data consistency across unreliable networks. Managing context across a distributed fleet of edge nodes presents a far greater challenge than for single-node LLM services with local context switching [39].

To our knowledge, existing approaches do not adequately ensure that user interactions maintain a consistent context across geo-distributed edge nodes. A common alternative, client-side context storage, introduces additional network latency and increases bandwidth consumption, as the context must be sent with every request. This is particularly problematic for mobile clients, where network constraints are a major limitation [18, 31], and can negate the latency benefits of deploying LLMs at the edge. Inconsistent or stale context can lead to fragmented sessions and irrelevant LLM responses, critically undermining application reliability. Addressing this gap requires a novel approach for managing context in geo-distributed LLM inference services.

We propose *DisCEdge*, a distributed context management system for LLM inference services at the edge. Our system replicates context as tokenized sequences, which efficiently avoids redundant text-to-token processing during inference and reduces user data exposure. Our experiments show that this approach effectively manages user context across geo-distributed LLM nodes, reducing median response times by 14.46%, synchronization network overhead by 15%, and client-to-server network usage by 90% compared to baseline approaches.¹

Our contributions are as follows:

- We design a distributed context management system for LLM inference services at the edge, enabling efficient replication of user context across geo-distributed nodes (§3).
- We implement and open-source a proof-of-concept prototype and the corresponding evaluation setup (§4.1)².

¹We use the term LLM throughout this paper for simplicity, though our approach applies to other context-aware foundation models that use tokenization.

²<https://github.com/ChaosRez/llm-context-management>

- We evaluate the system in a realistic edge environment on commodity hardware, demonstrating reduced response times and network usage compared to server-side raw text and client-side context storage approaches (§4.2).

2 Background and Related Work

To address the challenge of context management for LLMs at the edge, we review two key areas: context management strategies for LLMs (§2.1) and geo-distributed storage systems (§2.2), highlighting the gaps that motivate our system design.

2.1 Context Management in LLMs

LLMs are inherently stateless by design, meaning they do not retain any information about previous interactions or user sessions. Each inference request is treated independently, requiring all relevant context to be explicitly provided with each request. This presents significant challenges for managing user context—session history, preferences, and regional information—especially in distributed deployments.

2.1.1 Interaction Types and Context Requirements. Users typically interact with LLMs through two primary paradigms: chat and instruct models. Chat models are designed to facilitate multi-turn interactive conversations. Their input contains a sequence with roles (system, user, and assistant) to maintain conversational flow. These models are particularly suited for applications requiring session continuity, such as function calling [19], virtual assistants, humanoid robots, and autonomous driving. In contrast, instruct-based models are fine-tuned to follow specific instructions provided in a single prompt and typically do not require session history. Our work primarily targets chat-based interactions where session context is critical.

2.1.2 Context Window Constraints. A primary constraint in managing context is the model’s limited context window size, which defines the maximum number of tokens the model can process in a single request. Concatenating different context data (e.g., regional information, session history, system prompts) can result in an input that exceeds the model’s capacity. Context window sizes vary significantly between models, ranging from 2K tokens for TinyLlama [41] to 1M tokens for Gemini 2.5 Pro [8]. Inputs exceeding this limit must be truncated or summarized [24, 34]. This area of research is actively evolving, with ongoing efforts to develop models with larger context windows [22], extend existing ones [9], and develop techniques to reduce context size while preserving semantic meaning [10, 26].

2.1.3 Tokenization and Its Role. Tokenization converts raw input into numeric token sequences that models can process. This step is critical as it forms the bridge between human-readable text and the model’s input format. Since tokenization is model-dependent, it is typically performed server-side where the model is hosted. The tokenization process can be computationally expensive, especially for large texts or multi-modal inputs (e.g., images, audio, video) [35].

Managing context in a tokenized format offers several advantages for distributed systems. As sequences of integers, tokens are more compact than raw text, reducing network transmission overhead between nodes. They can also be easily concatenated without

repeated tokenization, avoiding redundant processing of session history that would otherwise occur with every request. Despite these benefits, tokenization should not be confused with embedding generation. Unlike embeddings—dense vectors capturing semantic meaning used in RAG systems [12]—tokens are the direct input format for LLM inference.

2.1.4 Context Management in Edge Deployments. While numerous studies have focused on optimizing LLM execution on edge devices [4, 20, 37, 38, 42], these primarily target single-node inference efficiency through techniques such as quantization, model pruning, and memory optimization. Single-node LLM services with local context switching have been explored [39], where multiple user sessions of different applications are managed on a single device. However, managing context across distributed edge nodes where users roam remains largely unaddressed. The critical aspect of maintaining consistent user context across distributed edge deployments, particularly for mobile users, has received limited attention in the literature. This gap motivates our work: how to efficiently replicate and synchronize user context across geo-distributed edge nodes while maintaining consistency and minimizing latency.

2.2 Geo-Distributed Storage Systems and Replica Management

Edge storage systems are designed for efficient data storage and retrieval across geographically distributed nodes. These systems inherently face the CAP theorem trade-off [3], balancing consistency and availability, particularly challenging under network constraints and device mobility.

2.2.1 Edge Storage Systems. Researchers have developed specialized storage systems to address these challenges in edge environments. FogStore [16] introduces a concept of “differential consistency”: clients receive stronger consistency when accessing data relevant to their geographical context, and weaker consistency otherwise.

FReD [27] is a more recent edge storage system that provides a distributed key-value store with a focus on client-centric consistency. Clients can declaratively specify replication schemes, which FReD then executes. FReD nodes exchange data directly via peer-to-peer communication, using a naming service only for metadata and configuration. Clients can dynamically choose which storage node to connect to and move between storage nodes as they roam. FReD groups keys into *keygroups* for which replication and consistency settings can be independently configured. It also provides a client-side middleware that intercepts requests to ensure consistency guarantees such as monotonic reads and read-your-writes for mobile clients, originally proposed by Bermbach et al. [1].

2.2.2 Limitations for LLM Context Management. While these systems provide robust solutions for geo-distributed storage, they are not directly suited for managing LLM session context for mobile users. First, their client-centric consistency models are designed for scenarios where the client itself is mobile and directly interacts with the storage layer. In our architecture, however, the data owner (the mobile user) and the entity managing storage (the static context manager on the edge node) are distinct. The user roams between edge nodes, but the context manager remains stationary

and serves as an intermediary between the client and storage. This architectural difference means existing client-centric consistency models do not directly apply.

Second, these systems often delegate data management complexity to the client—requiring clients to implement consistency protocols, handle replication logic, or manage version vectors. This contradicts our design goal of providing a simple, transparent interface equivalent to a centralized LLM service, where clients are unaware of the underlying distributed nature of the system.

Third, existing edge storage systems do not exploit the unique characteristics of LLM context data. LLM context is inherently sequential (session history), grows monotonically within a session, and benefits from being stored in pre-tokenized form. None of the existing systems leverage these properties to optimize storage, replication, or consistency protocols.

These limitations highlight the need for a specialized approach to context management in distributed edge LLM deployments—one that provides transparent consistency for mobile users, delegates complexity to the edge infrastructure, and exploits the unique characteristics of tokenized LLM context. Our work fills this gap by proposing a context management system tailored specifically for geo-distributed edge LLM services.

3 DisCEdge Architecture

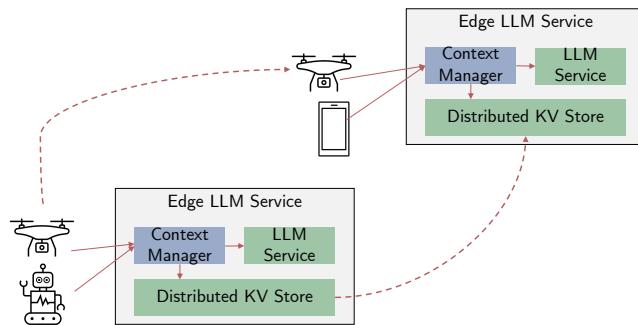


Figure 1: DisCEdge Architecture Overview

In this section, we present the architecture of DisCEdge, our distributed context management system for edge LLMs. The system is designed to efficiently manage user context across geo-distributed edge nodes, enabling low-latency interactions with LLMs as they move. As shown in Figure 1, our modular architecture consists of multiple edge nodes. Each edge node contains a *Context Manager* (§3.1), an *LLM Service* (§3.2), and a *Distributed KV Store* (§3.3). These nodes are accessed by *LLM clients* (§3.4).

Our system, while decentralized, provides the same interface as a centralized LLM service, without introducing data replication complexity on the client side. It employs lightweight tokenized context storage, which benefits data replication and distribution. The remainder of this section details each component.

Our design choice to cache and replicate context in tokenized form offers several advantages. As sequences of integers, tokens are more compact than raw text, reducing network transmission overhead between edge nodes. They can be easily concatenated without

repeated tokenization, avoiding redundant text-to-token processing for session history that must otherwise occur with every request. This optimization occurs earlier in the inference pipeline than embedding generation, positioning our approach as complementary to embedding-based techniques such as RAG.

We focus on managing user session context, which we define as the sequence of interactions within a single conversational session. We prioritize session context because it is both dynamic and critical for personalization, making its consistency a significant challenge in a geo-distributed system. Other context types, such as static user preferences or region-specific data, present simpler consistency requirements and are easily extendable. For instance, user preference is usually minimal and static, and can be stored on the client, while regional data is often localized to a single edge node.

While we focus on LLMs in this work, our tokenization-based approach generalizes to other foundation models that process various modalities (e.g., images, audio, video). Models for images, audio, and video similarly tokenize their inputs before processing. Our system’s core mechanism—managing and replicating pre-tokenized context—applies equally to these modalities, as the architecture is agnostic to the semantic meaning of tokens.

3.1 Context Manager

The Context Manager is the core component of each edge node, acting as an intelligent middleware between the client and the LLM Service. Its primary responsibility is to manage the lifecycle of user context, which includes session history, user preferences, and regional data.

Upon receiving a request, the Context Manager first assigns a user and a session identifier if they are not already present. To provide a strong consistency on top of the eventual consistency of the KV store, our system leverages a lightweight, client-driven consistency protocol. The Context Manager uses a turn counter, provided by the client, to verify that its local version of the session context is up-to-date. This mechanism ensures context consistency before the request is forwarded to the LLM Service. This client-driven approach is well-suited for mobile scenarios, as the client is the ultimate source of truth for the interaction sequence, making it a simple yet robust mechanism to ensure session integrity as users roam between edge nodes.

To improve efficiency, we maintain session context in pre-tokenized form. By doing so, the Context Manager can rapidly construct context-aware prompts for the LLM Service, eliminating the overhead of repeated tokenization for each interaction within a session. Each Context Manager operates in a decentralized manner, making decisions autonomously based on its local state and the information contained within the incoming request.

3.2 LLM Service

The LLM Service is the component responsible for executing language models to generate responses. It acts as an inference framework that receives a pre-tokenized context along the user prompt from the Context Manager. This component is runtime and hardware agnostic, and can be any inference framework that supports tokenized input. Its only requirements are the ability to process

token sequences and to serve the same models—and thus the same tokenizer for each model—as other LLM Services in the network.

A key design feature for efficiency is the direct handling of tokenized data. The Context Manager provides the session context in a pre-tokenized format. The LLM Service is designed to accept this tokenized context and merge it with the user prompt, as illustrated in Figure 2. This approach is efficient because the session context, which grows with every interaction and is typically much larger than the new user prompt, does not need to be re-tokenized. By only tokenizing the smaller, new input and properly concatenating it with the larger, pre-tokenized context, the system avoids redundant processing of a step that must happen with every new request, leading to faster response times.

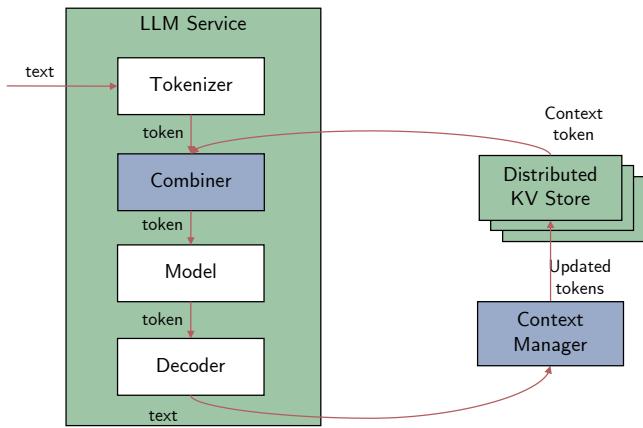


Figure 2: The “LLM Service” as an inference framework, and its abstract integration with the Context Manager.

3.3 Distributed KV Store

The persistence layer of our system is built upon a geo-distributed key-value (KV) store, leveraging existing storage systems designed for edge environments such as FReD [27] or FogStore [16]. Note that this distributed KV store is a data storage system for managing user context across edge nodes, distinct from the LLM’s internal key-value cache (often called KV-cache) used during inference for attention mechanisms. These stores provide the foundational mechanism for replicating and distributing context data across edge nodes.

The edge nodes interact implicitly via the underlying KV store. When a Context Manager updates a session context, it writes to its local KV store replica. The KV store is then responsible for replicating this data. Nodes share a session context only with nodes that are serving the same model, ensuring that context is only replicated where it is relevant. Lastly, each session context has a time-to-live (TTL) to automatically clean up stale data or by client’s explicit request. When a client sends a request to a new node, the Context Manager on that node reads the context from its local replica. If the context is stale (as indicated by the client’s turn counter), the Context Manager retries the read, effectively waiting for the replication from the previous node to complete.

A key aspect of our design is how we manage data consistency. While many distributed data stores rely on client-side middleware to ensure consistency for mobile clients [1], our architecture delegates this responsibility to the Context Manager. This is a more suitable approach for our use case, where the data consumer (the Context Manager on a static edge node) is stationary, but the data owner (the LLM mobile client) roams between nodes.

The Context Manager on each edge node interacts directly with its local replica of the KV store. When a client connects to a new edge node, our system’s consistency protocol, managed by the Context Manager and informed by the client’s turn counter, ensures the local context is up-to-date before processing the request. A key benefit of our design is that this trade-off between consistency and availability is a configurable policy by the client. Similarly, while our prototype does not implement an explicit eviction policy for the short-lived sessions in our experiments, policies such as session-based timeouts can be configured in the Context Manager to manage storage for long-running applications. In cases where network latency or partitions prevent synchronization, the system’s behavior is determined by the application’s requirements. For applications demanding strong consistency (default), the node can notify the client of the failure. Conversely, for those prioritizing availability, it can proceed with the available (potentially stale) context. This flexibility allows our system to support a diverse range of applications with varying needs.

3.4 LLM Clients

The clients use the same request format as the standard LLM Service, but they also include a user identifier, and a session identifier, which both can be assigned by the Context Manager after the first request. To enable our lightweight consistency protocol, the client is responsible for maintaining and sending a simple turn counter with each request within a session.

Further, we assume the clients can determine the closest edge node for their next request, using either a centralized service registry or a geo-aware routing approach introduced in GeoFaaS [21].

4 Evaluation

To showcase the performance of DisCEdge, we implement a proof-of-concept prototype and conduct a series of experiments to evaluate the performance of our distributed context management system in an edge environment.

4.1 Prototype Implementation

We implement DisCEdge, our distributed context management system in Go and on top of the open-source LLM inference framework LLaMA.Cpp [14] which runs quantized models in GGUf format [13]. We slightly modify the LLaMA.Cpp codebase to support our system³. Specifically, we extend the /completion API to accept a pre-tokenized “context” parameter. This allows the inference engine to skip re-tokenizing the existing context, instead only tokenizing the new prompt and prepending the provided context tokens. The Context Manager asynchronously updates the context in the background, after it receives the response from the LLM. It processes

³<https://github.com/ChaosRez/llama.cpp-fastencode>

requests in three context modes: (i) **raw** text mode, where the context is stored as raw text, (ii) **tokenized** mode, where the context is stored as tokenized data, and (iii) **client-side** mode, where the context is stored on the client device and thus the requests are just forwarded to the LLM Service by the Context Manager. Note that, to the LLM Service, the raw mode and client-side mode are the same, as the Context Manager does not modify the request in these modes. Further, since the Context Manager does not store the context in client-side mode, there is no synchronization overhead between edge nodes.

We opt for a self-hosted solution because existing cloud-based LLM services [7, 15, 17, 30], to our knowledge, do not offer the low-level access required to manage tokenized context. Specifically, they lack APIs for submitting pre-tokenized context, which is a core requirement for DisCEdge. This necessitates using an open-source framework that we could modify for our specific needs. We set a keygroup per language model in FReD; this way the user’s context data is isolated per requested model. The context is stored in the FReD distributed key-value store. FReD, similar to systems like Redis, is an in-memory store that provides low-latency access. It also supports asynchronous periodic persistence to disk for durability, but for the purpose of our performance evaluation, all reads/writes are from/to memory. We also implement client-side consistency to replace FReD’s client middleware which is supposed to move with the client, where in our system the Context Manager is FReD’s client and not the LLM clients. The client device is implemented in Python, and runs on a Raspberry Pi 4. The Context Manager checks the retrieved context version against the latest version (turn) stored in the edge node, and if it is newer, it retries to retrieve the context again.

4.2 Experiments

Here we evaluate the efficiency of DisCEdge through two main experiments. The first experiment validates our core design choice of using a tokenized context representation by quantifying its impact on latency, throughput, and network overhead against a raw text approach. The second experiment showcases the superiority of our edge-side architecture for mobile clients by comparing its end-to-end performance and network efficiency against the common alternative of client-side context management. We conduct a series of experiments to evaluate the performance of our distributed context management system in edge computing environments. The context maintained in our experiments only contains the user sessions, which is a sequence of chat *turns*, as managing other types of context, such as user preferences or regional information, is easily extensible. We repeat the experiments three times and report all results, mostly containing a 95% confidence interval. We evaluate several test prompting scenarios with dependence on previous turns of conversation to verify relevance of outputs. In the following experiments, we use a sample 9-turn prompt scenario, since we focus not on the model’s output, but rather on the performance of the context management system. For consistency guarantee settings, we set the retry count to 3, each with a 10ms back off, though through all our experiments the Context Manager never needs to retry more than two times. To measure the edge node synchronization network overhead, we use `tcpdump` to capture the network

traffic on the specific port used by the FReD (storage) nodes for peer communication and `tshark` to analyze the network packets. Although it captures some additional packets, such as TCP handshakes, we decide not to perform intrusive network monitoring by modifying the FReD codebase to log the network usage.

The experimental setup consists of two edge nodes and one client device, all connected within the same local network. We use commodity machines to run the edge nodes and the clients. This setup can be extended to sophisticated edge servers serving multiple AI models. The edge nodes are deployed on an Nvidia Jetson TX2 and one Mac M2, while the client device is a Raspberry Pi 4. We minimize any background processes on the Mac M2 node; still, we observe significantly lower response times from LLaMa.Cpp for the same input/output compared to the TX2 node since TX2 is much older hardware, not supported by the manufacturer, and LLaMa.Cpp is optimized to use hardware acceleration of Apple Silicon. This setup simulates a typical edge computing environment for latency-sensitive applications. The edge nodes and client communicate over a local network to minimize external interference. We use 4-bit quantization Q4_K_M as a balance between performance and compactness of the model. We set the seed to 123, temperature to 0, a max token generation of 128, and verify the number of generated tokens for all experiments to ensure consistent results.

4.2.1 Edge-side Context Management: Tokenized vs. Raw Text. This experiment evaluates the performance of two different edge-side context management strategies: storing context as raw text versus storing it as pre-tokenized data. The goal is to justify our design choice by quantifying the performance gains of avoiding repeated tokenization. We compare two modes: (i) **raw text**, where the server stores the conversation history as plain text and tokenizes the entire context with each new prompt, and (ii) **tokenized**, where the server stores the context in its tokenized form, avoiding repeated tokenization of the history. We conduct the experiment by running a client session against a single edge node (once on an M2 node and once on a TX2 node) for each of the two context modes. To evaluate the trade-offs, we measure and compare the following key metrics:

- **End-to-End Response Time:** To measure the direct impact on user-perceived latency, which is a primary goal for edge-based systems.
- **Tokens Per Second (TPS):** To isolate the effect on the LLM’s inference throughput by minimizing the variable overhead of tokenization.
- **Synchronization Network Overhead:** To demonstrate the benefit of a more compact context representation on inter-node traffic, which is crucial for scalability in a distributed environment.

Result. As shown in Figure 3, tokenized context storage outperforms raw text storage in median response times (14.46% speedup on TX2, and 8.75% on M2). Note that, since the prompt in each turn is different, and thus can have a different length and complexity, the response times are not perfectly linear with the number of tokens in the context. However, the overall trend shows that tokenized context storage consistently leads to lower response times compared to raw text storage. This is because tokenization reduces the

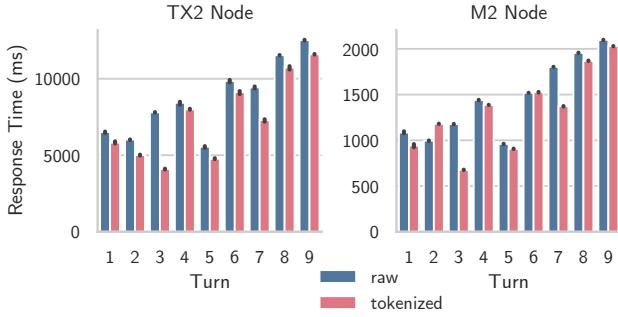


Figure 3: Client-observable response time per turn for tokenized versus raw text context storage on M2 and TX2 nodes. Error bars represent the 95% confidence interval.

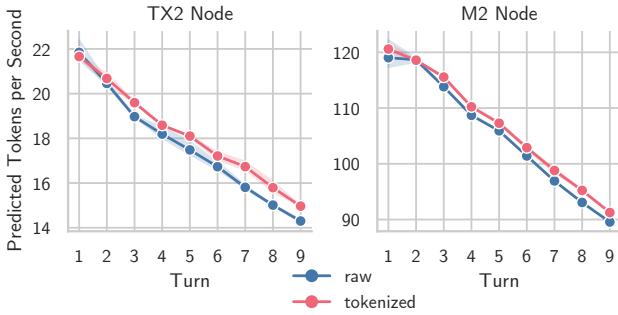


Figure 4: Tokens generated per second (TPS) for tokenized versus raw text context storage. The tokenized approach shows a modest performance improvement, which is more pronounced on the resource-constrained TX2 node.

overhead of processing large text inputs. The results in Figure 4 further confirm this finding, showing that the number of tokens processed per second (TPS) is slightly higher for tokenized context storage compared to raw text storage (2.85% speedup on TX2, 1.41% on M2), but also decreases with the growth of context. While the TX2 is still much slower than the M2 node, tokenized requests are significantly faster than client-side ones, especially when the context is much larger than the prompt. While the asynchronous tokenization step by the Context Manager takes 4ms to 50ms on the TX2 node and is consistently <1ms on the M2 node, this step is performed asynchronously with sending the response to the client. This decreases the client-observable response times.

According to Figure 5, the tokenized context storage reduces the network usage by 13.3% and 15% on M2 and TX2 nodes, respectively. This is because tokenized context storage reduces the size of the context data that needs to be synchronized between edge nodes, leading to lower network overhead.

4.2.2 Edge-Side vs. Client-Side Context Management for Mobile Clients. This experiment evaluates DisCEdge in a mobile client scenario, a key use case for edge computing. The primary objective is to demonstrate that our system maintains context consistency and

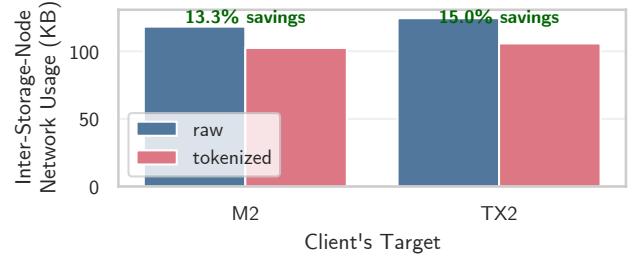


Figure 5: Network overhead for synchronizing context data between edge nodes, comparing tokenized versus raw text storage. Storing context as tokens reduces network usage compared to raw text. The network packets were collected on the M2 node.

performance when a client moves between edge nodes. We compare our proposed edge-side approach (using tokenized context) against a baseline using client-side context management, where the client sends the complete context with each request. This baseline is common but can be inefficient for mobile devices with limited bandwidth, as mentioned in §1. We simulate client mobility by having the client alternate between two different edge nodes after two turns during a conversation. In addition to verifying consistency, we measure and compare the performance of both approaches using two key metrics relevant to mobile users:

- **End-to-End Response Time:** To show that DisCEdge, including the overhead of context synchronization upon handover, is faster for the user than transmitting the full context from the client.
- **Client-to-Server Network Usage:** To quantify the reduction in data sent from the client, a critical factor for mobile devices on metered or unreliable networks, as mentioned in §1.

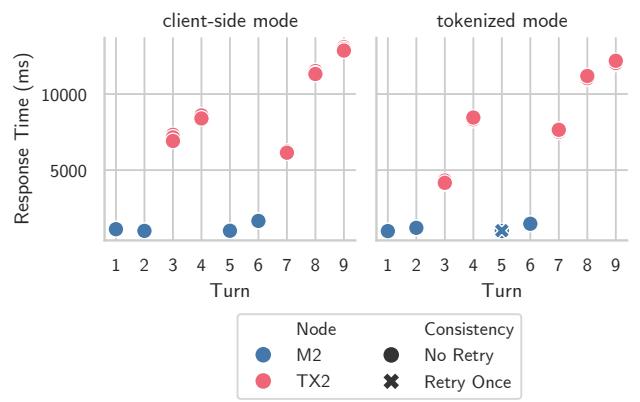


Figure 6: Client-observable response time per turn in a mobile client scenario. The client switches between edge nodes on turns 3, 5, and 7. DisCEdge outperforms the client-side approach, even with the synchronization overhead required after a handover.

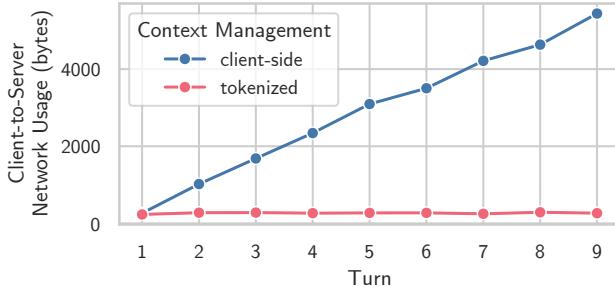


Figure 7: Client-to-server network usage per request turn. With client-side context management, the request size grows linearly as the full conversation history is sent. DisCEdge maintains a constant and minimal request size, only sending the new prompt.

Result. As depicted in Figure 6, DisCEdge reduces response time compared to the client-side baseline. Our approach achieves a median speedup of 5.93%, with a 2.51% improvement on the M2 node and a 6.29% improvement on the TX2 node. The benefits in terms of network usage are even more substantial. Figure 7 shows that our system reduces the client request size by a median of 90%. This is because our edge-side approach maintains a constant request size (dependent only on the new prompt), whereas the client-side approach leads to a linear growth in request size with every turn as the entire conversation history is transmitted. This drastic reduction in client-to-server network traffic is a critical advantage for mobile clients, where bandwidth is often limited and costly.

We observe greater benefits in network usage and response time as the context grows larger. This demonstrates that DisCEdge ensures context consistency for moving clients while significantly improving performance and minimizing network overhead.

5 Discussion

Our evaluation demonstrates the benefits of server-side, tokenized context management for LLM services at the edge. However, our work has several limitations that present opportunities for future research. Our experiments use a single client, which raises questions about scalability. With multiple concurrent users, we expect performance to be bound by LLM inference throughput and potential KV store contention, as each user’s context is managed as a separate key-value pair. A comprehensive multi-tenant scalability analysis is an important next step.

Our evaluation compares against straightforward baselines—raw text and client-side storage—as they represent the most common approaches. We view more advanced techniques, such as context summarization, as complementary rather than mutually exclusive. For instance, summarization could prune context before storage in our tokenized format, helping manage very long-term histories. Our work focuses on the foundational layer of efficient context replication, upon which such methods can be built.

A key threat to external validity is the use of a synthetic prompt scenario. LLM performance is highly dependent on input, and as observed, prompt complexity influences response times. While our

scenario was designed to test growing context, real-world interactions are more varied, and our system’s performance may differ with other prompting scenarios.

Furthermore, our approach is subject to inherent LLM limitations. Recent work shows LLMs may struggle to use information buried in the middle of very long contexts [36]. While our system efficiently manages context delivery by optimizing tokenization, it does not address how the model consumes it. Future work could explore selective retrieval of key information from vector databases to mitigate this issue—techniques that are complementary to our focus on pre-inference token processing.

From an implementation perspective, a more significant optimization could be achieved by directly manipulating the internal so-called KV cache of the LLM [11], avoiding the need to re-process tokens for existing context. However, managing a shared KV cache across multiple clients is complex. Exploring efficient and safe methods to update the KV cache in a multi-tenant, distributed environment is a promising direction for future research.

Moreover, our work focuses on context management and does not address broader edge AI challenges, such as predictive client handover to preemptively synchronize context, or dynamic resource allocation and model management on edge nodes. These areas remain important avenues for future exploration.

Finally, our prototype does not implement an explicit eviction policy for context data, as our experiments focused on a single, finite session. For long-running applications, managing KV-store growth is important. Future work could explore cache eviction strategies, such as Least Recently Used (LRU) or session-based timeouts, within the Context Manager to control storage overhead.

6 Conclusion

Deploying LLM services at the edge is beneficial for latency-sensitive and privacy-aware applications. However, we identified that the stateless nature of LLMs makes managing user context across geodistributed edge nodes challenging. Existing solutions, such as client-side context storage, often introduce network latency and bandwidth overhead, which can undermine the advantages of edge deployment.

In this paper, we proposed DisCEdge, a distributed context management system that stores and replicates user context in tokenized form across edge nodes. By maintaining context as token sequences rather than raw text, our system avoided redundant computation and enabled efficient data replication. We implemented and evaluated an open-source prototype in a realistic edge environment with commodity hardware. We showed DisCEdge improved median response times by up to 14.46% and lowered median inter-node synchronization overhead by up to 15% compared to a raw-text-based storage. It also reduced client request sizes by a median of 90% compared to client-side context management, while guaranteeing data consistency.

Acknowledgments

Funded by the Bundesministerium für Forschung, Technologie und Raumfahrt (BMFTR, German Federal Ministry of Research, Technology and Space) – 16KISK183. We thank Tobias Pfandzelter for helping us with the FReD source code explanation, Mohammad

Mohammadi for reviewing our changes to LLaMa.Cpp source code, and the volunteer reviewers for their valuable feedback.

References

- [1] David Bermbach, Jörn Kuhlenkamp, Bugra Derre, Markus Klems, and Stefan Tai. 2013. A Middleware Guaranteeing Client-Centric Consistency on Top of Eventually Consistent Datastores. In *Proceedings of the 1st IEEE International Conference on Cloud Engineering* (San Francisco, CA, USA) (IC2E 2013). IEEE, New York, NY, USA, 114–123. doi:[10.1109/IC2E.2013.32](https://doi.org/10.1109/IC2E.2013.32)
- [2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusi, Lachy Groom, Karol Hausman, Brian Ichter, et al. 2024. π_0 : A Vision-Language-Action Flow Model for General Robot Control. arXiv preprint arXiv:2410.24164 [cs.LG]. <https://arxiv.org/abs/2410.24164>
- [3] Eric A Brewer. 2000. Towards robust distributed systems. In *PODC*, Vol. 7. Portland, OR, 343–477.
- [4] Yuji Chai, Mujin Kwon, David Brooks, and Gu-Yeon Wei. 2025. FlexQuant: Elastic Quantization Framework for Locally Hosted LLM on Edge Devices. *arXiv preprint arXiv:2501.07139* (2025).
- [5] Guojun Chen, Xiaojing Yu, Neiwen Ling, and Lin Zhong. 2025. ChatFly: Low-Latency Drone Planning with Large Language Models. *IEEE Transactions on Mobile Computing* (2025).
- [6] Kaiyuan Chen, Nan Tian, Christian Juette, Tianshuang Qiu, Liu Ren, John Kubiatowicz, and Ken Goldberg. 2024. FogROS2-PLR: Probabilistic Latency-Reliability For Cloud Robotics. *arXiv preprint arXiv:2410.05562* (2024).
- [7] Cloudflare Workers AI. [n.d.]. Cloudflare Workers AI. <https://developers.cloudflare.com/workers/ai>. Accessed: 2025-04-23.
- [8] Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blstein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261* (2025).
- [9] Yiran Ding, Li Lyra Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. 2024. Longrope: Extending llm context window beyond 2 million tokens. *arXiv preprint arXiv:2402.13753* (2024).
- [10] Weizhi Fei, Xueyan Niu, Pingyi Zhou, Lu Hou, Bo Bai, Lei Deng, and Wei Han. 2023. Extending context window of large language models via semantic compression. *arXiv preprint arXiv:2312.09571* (2023).
- [11] Wei Gao, Xinyu Zhou, Peng Sun, Tianwei Zhang, and Yonggang Wen. 2025. Rethinking Key-Value Cache Compression Techniques for Large Language Model Serving. *arXiv preprint arXiv:2503.24000* (2025).
- [12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2, 1 (2023).
- [13] Georgi Gerganov. 2023. GGUF: GGML Universal File Format. <https://github.com/ggml-org/ggml/blob/master/docs/gguf.md>.
- [14] Georgi Gerganov et al. 2023. llama.cpp: LLM inference in C/C++. GitHub repository. <https://github.com/ggml-org/llama.cpp> Commit a33e6a (Feb 26, 2024); accessed 2025-04-01.
- [15] Google Cloud. [n.d.]. Vertex AI. <https://cloud.google.com/vertex-ai>. Accessed: 2025-04-23.
- [16] Harshit Gupta and Umakishore Ramachandran. 2018. Fogstore: A geo-distributed key-value store guaranteeing low latency for strongly consistent access. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*. 148–159.
- [17] Hugging Face. [n.d.]. Hugging Face Inference Endpoints. <https://endpoints.huggingface.co>. Accessed: 2025-04-23.
- [18] Jeffrey Ichniowski, Kaiyuan Chen, Karthik Dharmarajan, Simeon Adebola, Michael Danielczuk, Víctor Mayoral-Vilches, Nikhil Jha, Hugo Zhan, Edith LLon-top, Derek Xu, et al. 2023. Fogros2: An adaptive platform for cloud and fog robotics using ros 2. In *2023 IEEE international conference on robotics and automation (ICRA)*. IEEE, 5493–5500.
- [19] Ishan Kavathekar, Raghav Donakanti, Ponnurangan Kumaraguru, and Karthik Vaidhyanathan. 2025. Small models, big tasks: An exploratory empirical study on small language models for function calling. *arXiv preprint arXiv:2504.19277* (2025).
- [20] Guangyuan Liu, Yinqui Liu, Jiacheng Wang, Hongyang Du, Dusit Niyato, Jiawen Kang, and Zehui Xiong. 2025. Adaptive Contextual Caching for Mobile Edge Large Language Model Service. *arXiv preprint arXiv:2501.09383* (2025).
- [21] Mohammadreza Malekabbasi, Tobias Pfandzelter, Trever Schirmer, and David Bermbach. 2024. GeoFaaS: An Edge-to-Cloud FaaS Platform. In *Proceedings of the 12th IEEE International Conference on Cloud Engineering* (Paphos, Cyprus) (IC2E '24). IEEE, New York, NY, USA, 66–71. doi:[10.1109/IC2E61754.2024.00014](https://doi.org/10.1109/IC2E61754.2024.00014)
- [22] Kim Martineau. 2024. *What's an LLM context window and why is it getting larger?* <https://research.ibm.com/blog/larger-context-window>
- [23] Matteo Mendula, Paolo Bellavista, Marco Levorato, and Sharon Ladron de Guevara Contreras. 2024. Furcifer: a Context Adaptive Middleware for Real-world Object Detection Exploiting Local, Edge, and Split Computing in the Cloud Continuum. In *2024 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 47–56.
- [24] Mirza Alim Mutasodirin and Radityo Eko Prasojo. 2021. Investigating text shortening strategy in bert: Truncation vs summarization. In *2021 international conference on advanced computer science and information systems (icacsis)*. IEEE, 1–5.
- [25] Avanika Narayan, Dan Biderman, Sabri Eyuboglu, Avner May, Scott Linderman, James Zou, and Christopher Re. 2025. Minions: Cost-efficient collaboration between on-device and cloud language models. *arXiv preprint arXiv:2502.15964* (2025).
- [26] Artur Niederfahrenhorst and Kourosh Hakhamaneshi. 2024. *Fine-tuning LLMs for longer context and better RAG systems*. <https://www.anyscale.com/blog/fine-tuning-lm-for-longer-context-and-better-rag-systems>
- [27] Tobias Pfandzelter, Nils Japke, Trever Schirmer, Jonathan Hasenburg, and David Bermbach. 2023. Managing Data Replication and Distribution in the Fog with FReD. *Software: Practice and Experience* 53, 10 (Oct. 2023), 1958–1981. doi:[10.1002/spe.3237](https://doi.org/10.1002/spe.3237)
- [28] SearchWing Project. [n.d.]. SearchWing: Drones for Sea Rescue. <https://tha.de/searchwing/>. Accessed: 2025-04-18.
- [29] Guanqian Qu, Qiyuan Chen, Wei Wei, Zheng Lin, Xianhao Chen, and Kaibin Huang. 2025. Mobile edge intelligence for large language models: A contemporary survey. *IEEE Communications Surveys & Tutorials* (2025).
- [30] Replicate. [n.d.]. Replicate. <https://replicate.com/home>. Accessed: 2025-04-23.
- [31] Peter Schafhalter, Sukrit Kalra, Le Xu, Joseph E Gonzalez, and Ion Stoica. 2023. Leveraging cloud computing to make autonomous vehicles safer. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 5559–5566.
- [32] Peter Schafhalter, Alexander Krentsel, Joseph E Gonzalez, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. 2025. Bandwidth Allocation for Cloud-Augmented Autonomous Driving. *arXiv preprint arXiv:2503.20127* (2025).
- [33] Guanqun Wang, Jiaming Liu, Chenxuan Li, Yuan Zhang, Junpeng Ma, Xinyu Wei, Kevin Zhang, Maurice Chong, Renrui Zhang, Yijiang Liu, et al. 2024. Cloud-device collaborative learning for multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12646–12655.
- [34] Qingyue Wang, Yanhu Fu, Yanan Cao, Shuai Wang, Zhiliang Tian, and Liang Ding. 2025. Recursively summarizing enables long-term dialogue memory in large language models. *Neurocomputing* 639 (2025), 130193.
- [35] Zichen Wen, Yifeng Gao, Weijia Li, Conghui He, and Linfeng Zhang. 2025. Token Pruning in Multimodal Large Language Models: Are We Solving the Right Problem? *arXiv preprint arXiv:2502.11501* (2025).
- [36] Zijun Wu, Bingyuan Liu, Ran Yan, Lei Chen, and Thomas Delteil. 2024. Reducing Distraction in Long-Context Language Models by Focused Learning. *arXiv preprint arXiv:2411.05928* (2024).
- [37] Daliang Xu, Wangsong Yin, Hao Zhang, Xin Jin, Ying Zhang, Shiyun Wei, Mengwei Xu, and Xuanzhe Liu. 2024. EdgeLLM: Fast On-device LLM Inference with Speculative Decoding. *IEEE Transactions on Mobile Computing* (2024).
- [38] Shengyuan Ye, Bei Ouyang, Liekang Zeng, Tianyi Qian, Xiaowen Chu, Jian Tang, and Xu Chen. 2025. Jupiter: Fast and resource-efficient collaborative inference of generative llms on edge devices. *arXiv preprint arXiv:2504.08242* (2025).
- [39] Wangsong Yin, Mengwei Xu, Yuanchun Li, and Xuanzhe Liu. 2024. Llm as a system service on mobile devices. *arXiv preprint arXiv:2403.11805* (2024).
- [40] Mingjin Zhang, Xiaoming Shen, Jiannong Cao, Zeyang Cui, and Shan Jiang. 2024. Edgeshard: Efficient llm inference via collaborative edge computing. *IEEE Internet of Things Journal* (2024).
- [41] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385* (2024).
- [42] Yue Zheng, Yuhao Chen, Bin Qian, Xiufang Shi, Yuanchao Shu, and Jiming Chen. 2025. A review on edge large language models: Design, execution, and applications. *Comput. Surveys* 57, 8 (2025), 1–35.

A Experiment Details

This appendix provides the complete configuration details for the experiments described in §4.

A.1 Prompt Scenario

The following YAML configuration shows the 9-turn prompt scenario used in our experiments. The scenario simulates a technical conversation about robotics and autonomous systems, with questions that build upon previous responses to test context dependency.

Listing 1: 9-turn prompt scenario for robotics and autonomous systems

```
name: "Robotics_and_Autonomous_Systems_Test"
model_name: "Qwen/Qwen1.5 - 0.5B-Chat"
user_id: "robotics_dev"
messages:
  1. "What are the fundamental components of an autonomous mobile robot?"
  2. "You mentioned sensors. What are the most common types for obstacle avoidance?"
  3. "Can you explain the concept of a PID controller in the context of motor control?"
  4. "Write a simple Python function for a proportional (P) controller."
  5. "In your previous code, what do the `kp` and `error` variables represent?"
  6. "How would you modify that function to include the integral (I) component?"
  7. "Now, let's talk about localization. What is SLAM?"
  8. "What are some of the main challenges when implementing that on a small, low-power robot?"
  9. "Can you compare the EKF SLAM and Particle Filter SLAM approaches?"
```

A.2 Hardware Specifications

Table 1 summarizes the hardware specifications of the devices used in our experiments.

Table 1: Hardware specifications of experimental devices

Device	Specification	Role
Nvidia Jetson TX2	ARM Cortex-A57 (4-core) 8GB unified memory 256-core Pascal GPU	Edge Node
Apple Mac M2	8-core CPU (4P+4E) 16GB unified memory 8-core GPU	Edge Node
Raspberry Pi 4	ARM Cortex-A72 (4-core) 4GB RAM	Client Device