Recovering and Rehosting Mobile Local LLM Conversations and Contexts via Memory Forensics

Haichuan Xu, David Oygenblik, Runze Zhang, Mingxuan Yao,
Muhammad Ibrahim, Brendan Saltaformaggio
Georgia Institute of Technology

***Abstract*—Advances in edge computing devices have enabled local large language model (LLM) apps to execute entirely on-device, powering tasks such as document summarization and virtual assistance while preserving user privacy. However, this also removes centralized logging and enforcement, allowing adversaries to fine-tune or deploy modified models that generate malicious content. When such apps are used in planning crimes, it is hard for forensic investigators to gather evidences because disk logs from app storage are cleared once conversation histories are deleted from the UI by suspects. Existing memory forensics techniques also cannot reconstruct the context embeddings retained within the LLM runtime states. We present ORISA, a memory forensics framework for recovering and rehosting deleted conversation contexts from local LLM apps. ORISA leverages the batch tokens and key–value (KV) caches preserved in process memory to reconstruct token sequences and restore session-level attention states. ORISA recovers previous conversations and outputs a rehosted app-agnostic session that remembers forgotten context from the original session, allowing investigators to reveal suspect's prior interactions and ask additional information using the live session. We evaluated ORISA across 60 session configurations involving 10 Android local LLM apps and 3 model architectures. ORISA recovered 100% of tokens and KV cache tensors within the active context window, and an average of 32.60% and 38.59% more forgotten context. ORISA's rehosted sessions revealed an average of 169.6% knowledge compared to the original session.**

1. Introduction

Local LLM apps are increasingly more popular as edge computing devices grow more powerful [1]–[3]. They power a variety of applications — from search-engines to virtual assistants [4], document summarization [5] to personalized tutoring [6] — all while running directly on the user's own device. By operating locally, such models give users greater control over their data and privacy compared to cloud-hosted alternatives, because no proprietary provider processes or logs every query and response. Because of their strong privacy protections, a suspect who uses a local LLM app to plan or coordinate a crime makes their traces covert. When this happens, it is crucial for an investigator to reveal the previous conversations the suspect had with the LLM app, and even better, to interact with the LLM to understand and reveal the suspect's additional actions, goals, and motives.

Unfortunately, no research exists to provide investigators with such evidence. Investigators have two options for where to look for evidence: in app storage (internal and external storage for an Android app [7]) on disk and in the runtime memory of the LLM process. Existing disk forensics work focuses on extracting OS and app storage artifacts such as sensitive data [8], [9] and log messages [10], [11]. However, when the suspect deletes the conversation from the app UI, corresponding disk artifacts will also be removed, leaving no evidence for recovery. Traditional memory forensic techniques can recover kernel objects and app/OS specific data structures [12], [13]. Data structure recovery from LLM apps would enable evidence recovery, but it falls short of rehosting the LLM so the investigator can interact with it. Few prior memory forensic work recovered and rehosted deep learning (DL) model internals [14], and Android app GUIs [15], [16]. However, no prior work has considered rehosting a complex layered software stack required for LLM. Local LLM conversations are mapped into high-dimensional embedding spaces (KV cache tensors) and unstructured batch tokens. The fundamental challenge that necessitates a novel, LLM-specific memory forensics framework is that logically evicted context is dereferenced by the context manager and lacks structural boundaries. Existing data-structure recovery tools rely on static OS profiles or object pointers, which inherently fail when the LLM dynamically shifts and dereferences tensor buffers or wraps attention memory during continuous generation. Consequently, no existing research has been able to take advantage of the forgotten context beyond the model's context limit, realign, extend, and rehost them into a live session.

LLM apps commonly use KV cache to preserve the attention of the model and batch tokens to represent the vocabulary of the context [17]. To continue generation for a conversation, both are required to represent the existing conversation context. During our preliminary study (§3), we found that local LLM apps preserve both the KV cache and the batch tokens in memory until overwritten by new conversations. This gives us a unique vantage point to reconstruct previous conversation context by recovering both from memory. However, unlike application-level chat records, tokens and KV cache are encoded, unstructured,

and lack boundary markers, making their interpretation challenging. Furthermore, the active regions are dereferenced once a session is deleted in the UI, making the recovery and validation difficult.

Additionally, since LLMs have intrinsic context limits, local LLM apps keep track of the active KV cache region according to the session's context limit. We discovered that since the available allocated memory for the app is usually larger than that required to represent the model's context, logically evicted and forgotten context still resides in memory before being physically cleared and recycled by the OS. In fact, ORISA found an average of over 30% of tokens and KV tensors beyond the context limit are still present (§5.1). This gives us an opportunity to recover extra forgotten context from an earlier conversation and rehost it into a live session for investigators to interact with. However, recovering these logically evicted tensors are challenging because the LLM sessions only book-keeps positions of tensors within the currently active context window. Even when recovered, the session will never incorporate them to continue inference without properly extending and realigning the active region.

Based on these insights, we developed ORISA[1], an automated system for recovering conversation histories and rehosting the context of a local LLM app into a live session for investigators to interact with. ORISA takes an LLM app process's memory image and its app storage as input. ORISA first performs model binary extraction from disk (§4.1). ORISA then does batch token recovery (§4.2), giving investigators evidences and a starting point for interrogation. Next, ORISA conducts KV cache recovery and active KV extension (§4.3, §4.4) to recover and realign both active and forgotten context from the previous session. ORISA then reconstructs and rehosts the recovered context into an app-agnostic session (§4.5), allowing investigators to ask additional questions to the live session with previous conversation context. Finally, we will open-source ORISA upon the paper's acceptance.

We have evaluated ORISA with 60 configurations of LLM app sessions consisting of 10 Android Local LLM apps, with 3 different models under 2 different context windows. For tokens and KV cache recovery, ORISA recovered 100% of batch tokens and KV cache tensors from within the sessions' active context window and 32.60% and 38.59% beyond it. ORISA's rehosted session retained an average of 169.6% of knowledge compared to the original context, enabling investigators to interrogate the rehosted session with the forgotten context of the session. Finally, ORISA's rehosted sessions with the same context window preserved the response behaviors of the original session, allowing investigators to treat the rehosted sessions' interrogation outputs as credible evidence.

2. Overview

Cloud-hosted LLMs are traditionally deployed on centralized GPU/TPU clusters, where all user prompts and model outputs are processed on remote servers. These providers routinely log requests, responses, telemetry, and security metadata for service quality and abuse detection. Major technology companies can also disclose such server-side data to law enforcement under legal processes such as subpoenas, warrants, or national security orders [18]–[20]. As a result, cloud LLM interactions leave extensive traces, including network traffic, server logs, and access metadata that investigators can obtain during criminal inquiries.

In contrast, local LLMs run entirely on the user's device, typically optimized via runtimes such as llama.cpp [21], GGML [22], or other on-device inference backends. For suspects seeking to avoid cloud visibility, local models are especially attractive. Prompts and responses never leave the device, eliminating server-side audit trails or logs that investigators can obtain via legal processes. In these cases, the device itself becomes the only meaningful evidence source, motivating the need for forensic techniques that can recover and rehost the local LLM context.

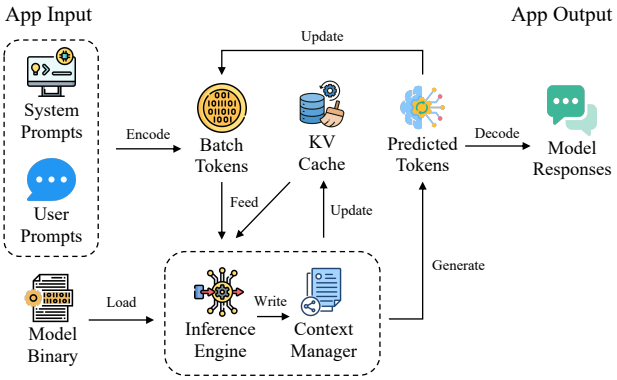

Figure 1: Overview of key components that enable inference in a local LLM app's chat session.

2.1. Key Components of Local LLMs

Figure 1 shows the overview of a mobile LLM app's workflow of generating chat session responses from user input. Local LLM inference necessitates several artifacts that have to reside on disk and in memory, giving investigators a unique vantage point to recover evidence.

Model Binary. To load a local LLM model, the model binary itself has to reside on disk, as shown in Figure 1. This binary contains the serialized model with the learned parameters specific to the model. It also contains key metadata about the model architecture, including vocabulary size, embedding dimension, number of layers, RoPE parameters, etc. For a local LLM app to instantiate a live session, it has to first load the model binary from disk to the inference engine.

Batch Tokens. When a user interacts with a local LLM app, their textual input is first passed to the tokenizer, which encodes each piece of text into a unique integer ID from the model's vocabulary, named as batch tokens, as shown

1. **O**n-device **R**ecovery of **I**nteractions and **S**ession **A**ttention

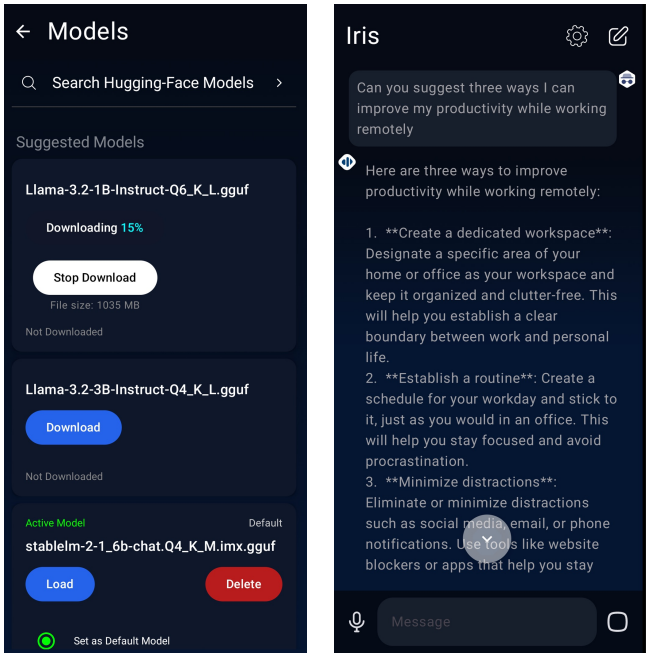

Figure 2: IRIS's offline model download screen and chat screen, allowing users to use custom open-source models for completely local inference.

in Figure 1. These integers are then stored in token buffers in process memory. The token buffer acts as the linearized memory of the textual conversation so far. It also provides positional references used to index the KV cache (discussed below). Without this buffer, the model has no way to align its internal cache to the original text. Recovering and deserializing them allows investigators to reconstruct previous prompts and responses of a session as evidence.

KV Cache. The KV cache is a collection of tensors that store the LLM's internal attention state for every token already processed, managed by the inference engine's context manager, as shown in Figure 1. It represents the current conversation context and is updated with each new token. The K layer stores the projection of previous tokens into the key space used by self-attention layers, while the V layer stores the corresponding value projections. Even when a suspect deletes the original conversation text from the device's UI, the KV cache preserves the semantic embedding of those tokens. Recovering and rehosting the KV cache together with batch tokens allows investigators to resume generation of the session from where it was left off to further gather evidence.

2.2. Threat Model

We assume that the suspect of interest is an average, normal user of a mobile device and utilizes a local LLM app on an Android device to coordinate and plan a crime. The suspect opens the app, engages in conversation with the local LLM, then deliberately deletes the chat history via the app UI, leaving no logs or transcripts on disk in app storage. They

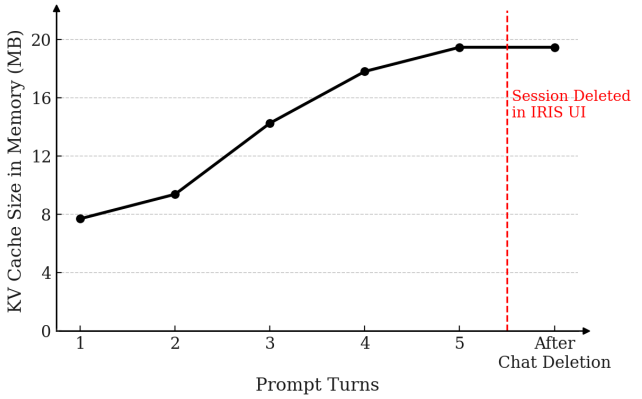

Figure 3: The size of KV cache in memory with respect to conversation turns. Although dereferenced after chat session deletion in IRIS UI, the KV cache still physically remains in memory.

rely purely on local execution; thus, no server-side log is available. At the time of seizure, the device is still running the LLM app's process. More advanced and technical suspects who can rewrite the app or use anti-forensics techniques such as those that obfuscate or encrypt the app's memory are discussed in §7. Although they can still be handled by ORISA with additional perpendicular research, they are outside the scope of this paper.

3. Problem and Opportunity

During our preliminary study, we found a popular Android local LLM app on the Google Play Store [23] named IRIS [24] with over 10K+ downloads. It is advertised as "Privacy-Focused" [25] and works completely offline without an internet connection. Figure 2 shows the model download and chat UI of the app. It allows users to download custom open-source models from Hugging Face [26] and use them for local inference. While interacting with the app, we dumped both the external (under `\sdcard\Android\data`) and internal (under `\data\data`) storage directory of the app and found that no cache or log of chat sessions is stored on disk. Clicking the new chat button also deletes the existing session. No way exists in the app UI to resume a previous chat session once deleted. This makes us wonder if there are still traces in memory that could reconstruct a previous chat session, if it has already been deleted in the app.

We then instrumented the app and dumped the memory image of the app process as we interacted and prompted the model in the app. In the memory image, we referenced `llama_batch` pointer and found token IDs that, when detokenized, matched with our previous prompts. In addition, when we examine the region pointed by `llama_kv_cache` pointer in `llama_context`, we found KV cache tensors that grow with conversation turns. Figure 3 shows the KV cache size in memory with respect to the turns of the current conversation, growing from 7.71 MB to 19.48 MB. We then

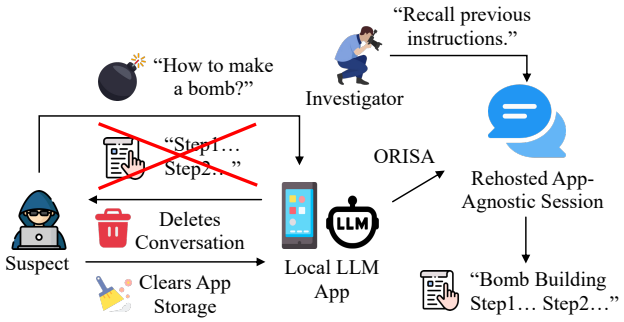

Figure 4: Rehosting a deleted conversation context into a live session allows investigators to interrogate and extract forensics evidence.

deleted the chat session after turn 5 by pressing the new chat button in the IRIS UI and dumped the memory image again. We found that although the `llama_kv_cache` pointer is dereferenced in the context block, when examining the same region before the deletion, the KV cache tensor values remain intact and are not physically cleared. This gives us a unique opportunity to rehost a previous session from batch tokens and KV cache recovered from memory, even after the session was cleared through the app UI.

Opportunity in Forensics Investigation. Figure 4 shows an example of applying this opportunity in a forensics investigation. Suppose a suspect uses a local LLM app with a jailbroken model to plan a crime and asks it, "How to build a bomb?" and then deletes the responses and clears the app storage. When an investigator wants to determine if the LLM assisted in the suspect's crime planning and confiscates the device, they turn to ORISA for help. ORISA recovers deleted conversation context in memory and rehosts it into a live app-agnostic session. The investigator can then interrogate the rehosted session and ask it to recall previous bomb making instructions and ask additional questions regarding the suspect's intentions. The responses then reveal previous illicit instructions and serve as evidence to convict the suspect.

4. Methodology

ORISA's goals are to reconstruct previous prompts and responses from a local LLM app session, recover active and forgotten context of the LLM session, and rehost them into a live session to enable live interrogation by investigators. Figure 5 shows the overview of ORISA's design consisting of 5 phases. The design consists of two stages: Artifact Extraction (Phases 1 and 2), which leverages foundational disk and memory forensic methodologies to establish a baseline, and Context Re-instantiation (Phases 3, 4, and 5), which introduces ORISA's novel contributions to overcome the LLM semantic gap. ORISA takes a mobile LLM app's process memory and app storage on disk as input. ORISA first locates and extracts the model binary and metadata from disk (§4.1). ORISA then recovers the previous session's batch tokens from memory and reconstructs them into the prompt history (§4.2). Next, ORISA recovers the KV cache representing both the active and logically evicted forgotten context from memory (§4.3). ORISA then extends the KV cache's active region to incorporate the forgotten context into a larger context window and realigns the KV tensors (§4.4). Finally, ORISA reconstructs and rehosts the recovered context into an app-agnostic session (§4.5), enabling investigators to conduct a live interrogation of the previous app session.

4.1. Model Binary Extraction From Disk

The first phase relies on traditional artifact extraction primitives. To enable session rehosting, ORISA first needs to extract the exact model binary used in the session as batch tokens, and KV cache only makes sense given a specific model. Pinpointing the exact model file used in the original run is challenging, as multiple model files may reside on disk, which could be used in different sessions. Even when a model with the same type is recovered, a slight variation in the model's hyperparameters, such as embedding dimension or head counts, will make the rehosting invalid.

However, we discovered in our preliminary study that during runtime, LLM apps' context embeds key descriptors of the model file into memory, including the model architecture identifier, and hyperparameters such as the number of layers, trained context length. Additional constants, such as the quantization layout, are also embedded in the KV cache configuration, which can be used to identify different versions of the same model type. These parameters together form a unique fingerprint to recover the exact model file used by the session. To accurately extract the exact same model used in the original run, ORISA uses these model metadata recovered from memory to locate and validate the model file.

Since the apps use local LLMs, the model file has to reside on the disk. ORISA scans the LLM app's both external (under`\sdcard\Android\data`) and internal (under`\data\data`) app storage directories to locate and extract the model file. Similar to established file carving methodologies that utilize magic bytes and contiguous block recovery [27], ORISA first extracts all files within the two directories with a matching file header with the model file format, e.g. `0x47475546` for `GGUF` models. ORISA then compares the model architecture identifier, quantization layout, number of layers, embedding dimension, head counts, and trained context length in the candidate model file's metadata key-value section with the recovered metadata in the model's `hparams` and context's `cparams` from memory. ORISA finds the exact model file when all metadata matches.

4.2. Batch Tokens Recovery and Detokenization

Utilizing standard pointer-tracing and array-recovery techniques, the next artifact to recover in memory is the batch token sequences used in the previous session. Together with KV cache recovery (§4.3), it represents the

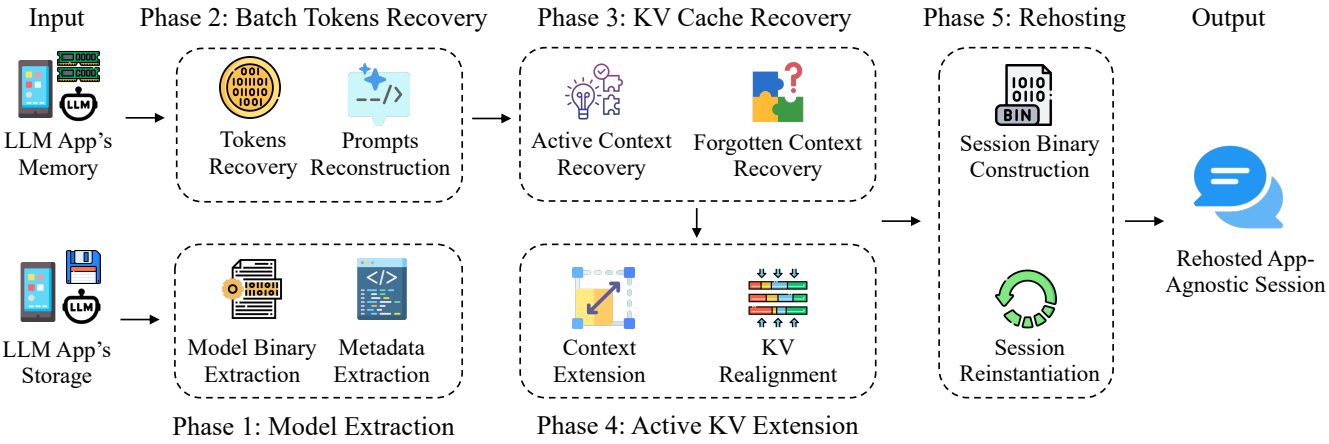

Figure 5: Overview of ORISA's design. ORISA takes a mobile LLM app's memory and app storage as input, conducts model extraction, batch tokens recovery, KV cache recovery, active KV extension, and session rehosting to output an app-agnostic rehosted session representing the previous app session's context.

Algorithm 1: ORISA's recovery of batch tokens from memory.

```
   // Metadata in llama_context manager
      structure
1  n_past = processedTokenCount;
2  Function recoverAndDetokenizeBatchTokens():
       // Use batch_ptr if not cleared
3      if llama_context.batch_ptr.isValid() then
4          A_token = *batch_ptr;
5          for A_i < A_token do
6              if A_i[token] ∈ [0, vocab_size] &
                 A_i_pos.notEmpty() & A_i_seq_id.notEmpty()
                 then
7                  continue;
8              end
9              A_token = A_i;
10             break;
11         end
12     end
       // Else match token vocab size and check
          valid pos and seq_id pointers
13     else
14         Set(A_bases) =
             scanBlocksLargerThan(llama_batch.size +
             n_past * token_type.size);
15         for A ∈ A_bases do
16             if ∀A_n ∈ [0, vocab_size] & A_pos.notEmpty()
                 & A_seq_id.notEmpty() then
17                 A_token = A;
18                 break;
19             end
20         end
21     end
       // Detokenization
22     D = Detokenize(T, A_token, A_token.length);
23     D.partion(T.delimiters);
24 end
```

context state of the session exactly where it was left off. When detokenized, the batch tokens also reveal the previous prompts and responses strings, providing investigators a starting point to conduct live interrogation of the session (§4.5).

However, unlike application-level chat records, the token buffer is encoded and unstructured, containing non-human-readable and interleaved sequences. These tokens are also unlabeled, lacking explicit message boundary markers. ORISA recovers the batch tokens by checking the vocabulary sizes of elements as well as valid positions and unique sequence IDs bound to each token. Algorithm 1 shows ORISA's strategy to recover batch tokens from memory. ORISA first checks whether the *batch_ptr* in the session's context manager (`llama_context` for *llama.cpp*) is still present and points to a valid page. If the region is valid, ORISA then scans backward from the currently active *batch_ptr* to locate additional tokens with valid buffer within the model's vocabulary range and that the position and sequence id buffers are not empty, as shown in Lines 3-12 of Algorithm 1. If the pointer has been cleared, ORISA then scans for contiguous blocks of aligned regions of at least size $llama\_batch.size + n\_past * token\_type.size$, representing previously processed tokens. For each candidate, ORISA validates the token buffer's vocabulary range and its pointers to position and sequence ids are valid, as shown in Lines 13-21 of Algorithm 1. Finally, ORISA detokenizes the verified token sequence back into human-readable prompt and response sequence strings with delimiters by applying the decode function of the exact tokenizer in the recovered model binary file from disk (model binary recovery detailed in §4.1). Each token ID is mapped back deterministically to its string representation, preserving role delimiters and formatting tokens injected by the app layer. To separate the prompts into system prompts, user prompts, and model responses, ORISA partitions the detokenized scripts with delimiters of the session, as shown in Lines 22-24 of Algorithm 1.

Now ORISA has recovered the batch tokens of the previous session and detokenized conversation history,

Algorithm 2: ORISA's recovery of the active and forgotten KV cache.

```
   // Previously recovered token length and
      metadata
1  T_old = tokenBufferLength, H =
     numberOfHeads, D_h = headDim, E =
     elementSize, ctx = originalContextWindow;
2  Function recoverAndAlignKVCache():
       // Per-token kv cache slot size
3      B_token = H * D_h * E * 2;
       // Locate kv cache region
4      Set(A_bases) = scanBlocksLargerThan(B_token * ctx);

5      A_base = null;
6      for A ∈ A_bases do
7          h = cacheHeadOffset(A_base);
8          u = cacheUsedOffset(A_base);
9          if header(A_base).matches(header(KVCache)) &
             h < ctx & u <= T_old then
10             A_base = A
11         end
12     end
       // Determine previous active region
13     A_active_start = A_base + ctx * B_token;
14     if h + u <= ctx then
15         A_ctx = [A_active_start, A_active_start + u * B_token);
16     end
       // Handle wrap around
17     else
18         A_ctx = [A_active_start, A_end) ∪ [A_base, A_base +
             ((h + u)mod(ctx)) * B_token);
19     end
       // Tensor validation with token position
20     for cell[i] ∈ A_ctx do
21         check(cell[i].pos = token[i].pos);
22         check(cell[i].seq_id = token[i].seq_id;
23     end
       // Scan for valid forgotten context
          sensors beyond the context window
24     for cell[j] < A_active_start do
25         if j > T_old then
26             break;
27         end
28         if cell[j].pos = token[j].pos &
             cell[j].seq_id = token[j].seq_id then
29             insert(cell[j], A_ctx);
30             continue;
31         end
32         break;
33     end
34 end
```

which serve as human-readable evidence to investigators for interrogation. To further allow live interrogation of the session, ORISA next recovers active and forgotten contexts from memory (§4.3, §4.4) and rehosts them into a live session (§4.5).

4.3. Active and Forgotten Context Extraction from KV Cache

Bridging the semantic gap and live rehosting requires moving beyond static artifact extraction. Next, ORISA recovers the LLM session's internal attention state after token processing at the time of extraction by retrieving the KV cache. Combined with the batch tokens generated and the model file recovered previously, it allows investigators to resume generation exactly from the point when the scene was seized with rehosting.

However, fully recovering the KV cache with forgotten context is fundamentally challenging for existing tools. Once a session is deleted in the apps' UI, the active KV cache region is dereferenced as we found in our preliminary study (§3), making the recovery from known pointers in `llama-context` infeasible. Because the object pointers are cleared, traditional pointer-tracing techniques (as utilized in §4.2) entirely fail. Additionally, the LLM session only book-keeps positions of KV tensors that are within the currently active context window, making recovery and validation of tensors representing logically evicted tokens challenging. Finally, every tensor recovered needs to be validated and match exactly with previous tokens in the correct sequence in order to represent the correct attention state. ORISA approaches by scanning the memory region with unique KV cache headers and uses previously recovered session metadata and batch token identifiers to guide the recovery and validation of KV cache tensors. Algorithm 2 shows the steps ORISA adopts to recover both the active and forgotten KV cache from memory.

ORISA first uses previously recovered model's number of attention heads (H), head dimension (D_h), element size (E) and the original context window (ctx) as a reference to calculate the per-token KV cache slot size to recover $B_{token} = H * D_h * E * 2$, as shown in Line 3 of Algorithm 2. Then to locate the candidate KV cache ring buffer regions, ORISA scans the heap for a contiguous block of size larger than the minimum size to represent the previous context $B_{token} * ctx$ that are page-aligned and resident. For each candidate base address A, ORISA matches the header with the `llama-kv-cache`, and verifies that the value h at offset to the `cache.head` offset is within the valid region between $[0, ctx)$, and that the value u at offset to the `cache.used` offset is within the recovered token buffer length T_{old}. ORISA accepts the candidate KV cache region when all the above are matched, as shown in Lines 4-12 of Algorithm 2.

Next, ORISA recovers the previously active KV tensors within the context window. ORISA first calculates the base address for the previous active region as $A_{active\_start} = A_{base} + ctx * B_{token}$. If the previously used tokens are within the context window, then the previously active tensors $A_{ctx} = [A_{active\_start}, A_{active\_start} + u * B_{token})$. If they are larger, then there is a wrap-around in the ring buffer so that the part with the most recent tokens' tensors is wrapped to the start of the region. The total active region is thus a union of $[A_{active\_start}, A_{end})$ and $[A_{base}, A_{base} + ((h + u)mod(ctx)) * B_{token})$, where A_{end} is the end of the buffer that have no valid tensors in succeeding slots.

Finally, ORISA validates the KV cache slots recovered by validating their mapping to the batch token positions recovered earlier. From the slot representing the tensors for the most recent tokens, ORISA checks backward in A_{ctx} until it reaches A_{active_start}. For each tensor slot, ORISA

Algorithm 3: ORISA's extension of recovered KV cache tensors with forgotten context and realignment of metadata for rehosting into a larger context window.

```
   // Previously recovered KV cache with
      forgotten context
1  A_ctx = recoveredKVRegion, h =
     originalCacheHeadOffeset(A_base), ctx =
     originalContextWindow, ctx_new =
     newContextWindow;
2  Function extendAndRealignKVCache():
       // Reset KV cache metadata
3      cache.head = 0;
4      cache.used = u_valid = A_ctx.length;
5      cache.size = ctx_new;
       // Remap new kv cache slot index
6      i_new = (i_old − h)mod(ctx) for each i_old ∈ A_ctx.length;
       // Move tensor values and metadata fields
7      for ∀i_new do
8          cells[i_new].pos = cells[i_old].pos;
9          cells[i_new].seq_id = cells[i_old].seq_id;
10         cells[i_new].buf = cells[i_old].buf;
11     end
       // Clear unused cells
12     for i_new ∈ [u_valid, ctx_new] do
13         cells[i_new].pos = −1;
14         cells[i_new].pos = 0;
15     end
       // Rest session past token count
16     runtime_context_set(n_past, u_valid);
17 end
```

validates the $cell[i].pos$ matches with the $token[i].pos$ previously recovered, and that the token sequence identifier stored in the KV cache slot $cell[i].seq\_id$ matches the $token[i].seq\_id$, as shown in Lines 20-23 in Algorithm 2. When it reaches the active region's head A_{active_start}, ORISA continues to scan beyond it for valid tensors logically evicted but still present in the buffer. If any cells' position and sequence ids $cell[j].pos$ and $cell[j].seq\_id$ matches with $token[j].pos$ and $token[i].seq\_id$ of recovered tokens beyond the context window, ORISA marks them as valid tensors representing the forgotten context and inserts them into the recovered KV cache region A_{ctx}. ORISA stops the forgotten context recovery when it reaches the end of the recovered token buffer length or when the new cell position doesn't contain a valid tensor anymore, as shown in Lines 24-33 of Algorithm 2.

4.4. Active KV Extension

After the recovery of the KV cache with logically evicted tensors representing the forgotten context of the session beyond the context window, next ORISA needs to realign the KV cache and enlarge the active region of it so that when rehosted, the new session treats these forgotten contexts as active contexts.

This is challenging because although the recovered forgotten context tensors are physically present, the session still keeps them as invalid and will never incorporate them to continue generation. A key contribution of ORISA over traditional memory forensics techniques is that aside from extracting data, it performs state manipulation to force the inference engine to accept logically evicted memory. ORISA extends the session's perception of the active KV by enlarging the cache size, resetting the cache head and active used section pointer, and realigning the cells with the new positions to the start of the new KV cache region.

Algorithm 3 shows the steps to extend and realign the KV cache after its recovery with forgotten context. ORISA first records the previously recovered KV region A_{ctx}, the original cache head offset with respect to the base h, the previous context window ctx, and the new enlarged context window ctx_{new}. Then, ORISA resets KV cache regions' metadata to represent the new valid region with forgotten context by setting the $cache.head$ pointer to 0, the $cache.used$ count to u_{valid} which is the length of the newly recovered region A_{ctx}, and the $cache.size$ to ctx_{new}, as shown in Lines 3-5 of Algorithm 3. ORISA then calculates the new kv cache slot index $i_{new} = (i_{old} - h)mod(ctx)$ for each $i_{old} \in A_{ctx}.length$ slot for the realignment, as shown in Line 6 of Algorithm 3. For each of the new slots, ORISA then copies the tensor values and metadata fields that match with token positions and token sequence ids from the previous position to the new position. For the regions outside the new active region, ORISA resets the cell position counters and cell sequence ids, as shown in Lines 7-15 of Algorithm 3. Finally, in the rehosted sessions' context manager (`llama-context` for *llama.cpp*), ORISA sets the processed token counters `n_past` to the currently valid context recovered u_{valid} so that the rehosted session can incorporate both the previously active and forgotten context recovered into subsequent generations.

Now ORISA has recovered the batch token sequences and the realigned and extended KV cache with forgotten context from memory for rehosting.

4.5. Session Rehosting

Batch tokens and KV cache need to be loaded into a live LLM session with the original model to continue conversation generation with previous context. After recovering the KV cache with forgotten context and the batch tokens of a conversation, ORISA rehosts them into a new session using the same model configurations with an enlarged context window to enable live interrogation of the previous session for investigators.

Session Binary Construction. To rehost an LLM conversation, we need to represent the previous context as a session state binary that the new session understands and can continue the generation from. This session binary should contain the exact configuration and attention of the model when the previous session stopped. For *llama.cpp* applications, this binary should represent the `llama_context` state that `llama_state_set_data()` takes as input. ORISA constructs the binary by first aligning the recovered context-level pseudo-random generator state, followed by

the most recent output logits kept in the `llama_context` recovered from memory to preserve determinism in subsequent sampling. Then ORISA aligns the KV cache with the extended context to represent the exact attention state of the previous conversation. Finally, ORISA sets the new session binary's context window to the enlarged window, and the bookkeeping parameter of processed tokens `n_past` to the recovered token length. Now the reconstructed session file encodes not only the full attention history of all processed tokens but also the precise sampling state needed to ensure continuity.

Session Reinstantiation. ORISA then loads a fresh session with the exact model binary file recovered in §4.1, the session binary constructed (by calling `llama_state_set_data()` for *llama.cpp*), and the batch tokens recovered with the new processed token counter `n_past`. ORISA now continues the inference from the previous session (by calling `decode()`) and allows investigators to interrogate the new session with the previous context.

5. Evaluation

We deployed ORISA to evaluate its capability to recover batch tokens and KV cache from memory, as well as rehosting them into new sessions for investigators to interrogate using real Android local LLM apps.

Dataset & Implementation. We collected the 10 most popular (according to number of downloads or stars) Android local LLM apps from the Google Play Store [23] and public GitHub repositories that allow users to choose and use their own GGUF LLMs. Notably, they are all powered by llama.cpp [21], which is the most widely adopted mobile LLM inference engine. We installed them and conducted the evaluation on a rooted Google Pixel 6 device running Android 15 [28]. For the local LLM models, we collected 3 different lightweight models from the leading local model repository Hugging Face [26] suitable for local execution on a Google Pixel 6 device, namely Llama 3.2 [29], Qwen 2.5 [30], and Gemma 2 [31]. LLM app processes' memory images are acquired with Frida [32] using Objection [33]. Superuser privilege for Frida is acquired and granted with Magisk [34]. The rehosted sessions are run on an Ubuntu 24.04 LTS system running llama.cpp [21].

5.1. Tokens & KV Cache Recovery

We first evaluated ORISA's recovery of batch tokens and KV cache tensors from within the model's active context window as well as from beyond the context window.

Experiment Setup. For each of the 10 apps, we run 6 sessions using the 3 models mentioned before under 2 different active context windows (256 and 512 tokens). To concretely determine if a response from the LLM is derived from previous chat knowledge rather than from the training knowledge, we randomly created concrete key-value pair information representing a project internal note that the

```
We're keeping some internal project notes.
- Project codename = FERN-9e2f1b5d
- API key hint = live_68cd-PLUM
- Contact = Dana R. <dana@example.org> id=R-77b1
- Ops window: Mondays 09:00–11:00 UTC
- Staging bucket: s3://violet-bridge-14
- On-call PIN = 5421-ALTO
……
```

Figure 6: Concrete key-value pair knowledge fed into the LLMs to simulate session context.

LLMs do not know in advance and fed them into each LLM session to simulate the session context, template as shown in Figure 6. For each session, we continuously fed in the key-value pairs in a batch of 10 until the aggregated token count of the current session from both prompts and LLM responses reached at least 1024 tokens to overfill the 256 and 512-token active context window. This ensures that at least 512 tokens in both sessions are over the model's active context window and are forgotten, enabling ORISA to recover batch tokens and KV cache representing those forgotten context. We then delete the current conversation from each app's UI, dump the app process's memory, and run ORISA to recover the batch tokens and KV cache.

Findings. Table 1 shows the results for batch token and KV cache recovery from each app's memory image running the 3 models under 2 different active context windows. Columns 1-2 of Table 1 show the 10 apps and 3 models loaded in each session. Column 3 shows the initial active context window set in each session for each model. Each model is run with a 256 and 512-token context window in 2 sessions. Column 4 shows the ground truth token count, including both prompt and model responses, that the session contains before the deletion of the chat content from UI and gathering the memory image. Columns 5-9 show the batch tokens, KV cache tensors, and KV cache size within the respective context window recovered by ORISA. Additionally, Columns 10-14 show the logically evicted batch tokens, KV cache tensors, and KV cache size recovered beyond the active context window, representing forgotten context.

Looking at the Total Avg. Row, Column 4 of Table 1, each session had processed an average of 1,056 tokens, overfilling the 256 and 512 active context windows by at least 4 and 2 times. Columns 5-8 show that ORISA recovered 100% of batch tokens and KV tensors from all sessions within the context window, indicating that active tokens and KV cache still fully reside in memory even after the deletion of the chat session from the apps' UI. For KV cache, ORISA recovered 4,194K, 3,670K, and 13,631K tensors from the Llama 3.2, Qwen 2.5, and Gemma 2 models, respectively, for the 256 tokens context window. The recovered tensors from the Gemma 2 models are more than 3 times larger than those recovered from the other 2 models because of the significantly larger number of KV heads and KV head dimensions of the Gemma 2 model. As a result, Column 9

TABLE 1: ORISA's Recovery of Batch Tokens and KV Cache Within and Beyond the Sessions' Active Context Window.

App	Model	Initial Ctx	GT Tokens	Recovery Within Ctx					Recovery Beyond Ctx				
				Tokens		KV Tensors		KV Size	Tokens		KV Tensors		KV Size
				#	%	#(K)	%	(MB)	#	%	#(K)	%	(MB)
Iris	Llama 3.2	256	1,073	256	100.00	4,194	100.00	8.66	138	16.89	2,848	21.28	5.56
		512	1,062	512	100.00	8,389	100.00	16.33	327	59.45	3,756	41.68	7.34
	Qwen 2.5	256	1,048	256	100.00	3,670	100.00	7.56	173	21.84	2,429	21.39	4.74
		512	1,074	512	100.00	7,340	100.00	14.55	265	47.15	3,820	47.41	7.46
	Gemma 2	256	1,044	256	100.00	13,631	100.00	26.85	179	22.72	9,312	22.19	18.19
		512	1,068	512	100.00	27,263	100.00	52.88	308	55.40	14,624	49.40	28.56
Avg.	-	-	1,062	384	100.00	10,748	100.00	21.14	232	37.24	6,132	33.89	11.98
ChatterUI	Llama 3.2	256	1,058	256	100.00	4,194	100.00	8.66	173	21.57	2,343	17.83	4.58
		512	1,067	512	100.00	8,389	100.00	16.33	234	42.16	3,719	40.90	7.26
	Qwen 2.5	256	1,059	256	100.00	3,670	100.00	7.56	149	18.56	2,007	17.43	3.92
		512	1,043	512	100.00	7,340	100.00	14.55	202	38.04	2,652	34.84	5.18
	Gemma 2	256	1,055	256	100.00	13,631	100.00	26.85	184	23.03	9,158	21.53	17.89
		512	1,039	512	100.00	27,263	100.00	52.88	258	48.96	13,418	47.82	26.21
Avg.	-	-	1,054	384	100.00	10,748	100.00	21.14	200	32.05	5,550	30.06	10.84
Layla	Llama 3.2	256	1,071	256	100.00	4,194	100.00	8.66	127	15.58	2,703	20.24	5.28
		512	1,059	512	100.00	8,389	100.00	16.33	392	71.66	3,034	33.85	5.93
	Qwen 2.5	256	1,077	256	100.00	3,670	100.00	7.56	189	23.02	2,235	18.99	4.37
		512	1,033	512	100.00	7,340	100.00	14.55	254	48.75	3,676	49.22	7.18
	Gemma 2	256	1,052	256	100.00	13,631	100.00	26.85	140	17.59	9,815	23.16	19.17
		512	1,066	512	100.00	27,263	100.00	52.88	349	63.00	8,997	30.50	17.57
Avg.	-	-	1,060	384	100.00	10,748	100.00	21.14	242	39.93	5,077	29.33	9.92
Solia	Llama 3.2	256	1,064	256	100.00	4,194	100.00	8.66	176	21.78	2,889	21.82	5.64
		512	1,056	512	100.00	8,389	100.00	16.33	341	62.68	3,560	39.94	6.95
	Qwen 2.5	256	1,070	256	100.00	3,670	100.00	7.56	155	19.04	2,597	22.25	5.07
		512	1,069	512	100.00	7,340	100.00	14.55	386	69.30	2,404	30.11	4.70
	Gemma 2	256	1,049	256	100.00	13,631	100.00	26.85	188	23.71	9,067	21.47	17.71
		512	1,067	512	100.00	27,263	100.00	52.88	224	40.36	12,585	42.58	24.58
Avg.	-	-	1,063	384	100.00	10,748	100.00	21.14	245	39.48	5,517	29.70	10.78
Maid	Llama 3.2	256	1,046	256	100.00	4,194	100.00	8.66	147	18.61	2,534	19.58	4.95
		512	1,076	512	100.00	8,389	100.00	16.33	373	66.13	4,414	47.77	8.62
	Qwen 2.5	256	1,039	256	100.00	3,670	100.00	7.56	180	22.99	2,129	18.97	4.16
		512	1,060	512	100.00	7,340	100.00	14.55	297	54.20	3,053	38.86	5.96
	Gemma 2	256	1,045	256	100.00	13,631	100.00	26.85	163	20.66	10,045	23.91	19.62
		512	1,038	512	100.00	27,263	100.00	52.88	333	63.31	15,221	54.34	29.73
Avg.	-	-	1,051	384	100.00	10,748	100.00	21.14	249	40.98	6,233	33.90	12.17
LLMHub	Llama 3.2	256	1,041	256	100.00	4,194	100.00	8.66	135	17.20	3,122	24.28	6.10
		512	1,063	512	100.00	8,389	100.00	16.33	205	37.21	3,156	34.96	6.16
	Qwen 2.5	256	1,072	256	100.00	3,670	100.00	7.56	190	23.28	2,524	21.58	4.93
		512	1,075	512	100.00	7,340	100.00	14.55	362	64.30	4,267	52.87	8.33
	Gemma 2	256	1,040	256	100.00	13,631	100.00	26.85	122	15.56	8,928	21.39	17.44
		512	1,057	512	100.00	27,263	100.00	52.88	256	46.97	10,428	35.93	20.37
Avg.	-	-	1,058	384	100.00	10,748	100.00	21.14	212	34.09	5,404	31.83	10.56
PocketPal	Llama 3.2	256	1,050	256	100.00	4,194	100.00	8.66	171	21.54	3,070	23.60	6.00
		512	1,043	512	100.00	8,389	100.00	16.33	391	73.63	4,797	55.14	9.37
	Qwen 2.5	256	1,036	256	100.00	3,670	100.00	7.56	143	18.33	2,418	21.62	4.72
		512	1,042	512	100.00	7,340	100.00	14.55	217	40.94	4,042	53.20	7.89
	Gemma 2	256	1,078	256	100.00	13,631	100.00	26.85	175	21.29	9,813	22.42	19.17
		512	1,053	512	100.00	27,263	100.00	52.88	344	63.59	16,011	55.58	31.27
Avg.	-	-	1,050	384	100.00	10,748	100.00	21.14	240	39.89	6,692	38.59	13.07
Local AI	Llama 3.2	256	1,037	256	100.00	4,194	100.00	8.66	186	23.82	2,745	21.45	5.36
		512	1,055	512	100.00	8,389	100.00	16.33	323	59.48	3,083	34.65	6.02
	Qwen 2.5	256	1,035	256	100.00	3,670	100.00	7.56	177	22.72	2,148	19.23	4.20
		512	1,051	512	100.00	7,340	100.00	14.55	398	73.84	3,928	50.83	7.67
	Gemma 2	256	1,065	256	100.00	13,631	100.00	26.85	169	20.89	10,290	23.89	20.10
		512	1,034	512	100.00	27,263	100.00	52.88	278	53.26	14,293	51.42	27.92
Avg.	-	-	1,046	384	100.00	10,748	100.00	21.14	255	42.33	6,081	33.58	11.88
AnythingLLM	Llama 3.2	256	1,077	256	100.00	4,194	100.00	8.66	192	23.39	2,936	21.83	5.73
		512	1,061	512	100.00	8,389	100.00	16.33	209	38.07	3,686	40.98	7.20
	Qwen 2.5	256	1,054	256	100.00	3,670	100.00	7.56	133	16.67	2,344	20.49	4.58
		512	1,058	512	100.00	7,340	100.00	14.55	357	65.38	3,155	40.31	6.16
	Gemma 2	256	1,073	256	100.00	13,631	100.00	26.85	162	19.83	9,252	21.27	18.07
		512	1,047	512	100.00	27,263	100.00	52.88	232	43.36	15,872	55.72	31.00
Avg.	-	-	1,062	384	100.00	10,748	100.00	21.14	214	34.45	6,208	33.43	12.12
LM Playground	Llama 3.2	256	1,032	256	100.00	4,194	100.00	8.66	151	19.46	2,811	22.11	5.49
		512	1,031	512	100.00	8,389	100.00	16.33	319	61.46	4,009	47.14	7.83
	Qwen 2.5	256	1,068	256	100.00	3,670	100.00	7.56	181	22.29	2,718	23.35	5.31
		512	1,070	512	100.00	7,340	100.00	14.55	370	66.31	2,699	33.74	5.27
	Gemma 2	256	1,072	256	100.00	13,631	100.00	26.85	157	19.24	9,787	22.53	19.12
		512	1,071	512	100.00	27,263	100.00	52.88	288	51.52	12,045	40.47	23.53
Avg.	-	-	1,057	384	100.00	10,748	100.00	21.14	244	40.05	5,678	31.56	11.09
Total Avg.	-	-	1,056	384	100.00	10,748	100.00	21.14	233	38.02	5,860	32.60	11.44

of the Total Row shows that the KV cache size recovered from the Gemma 2 models (26.85 MB) is also 3 times larger than that from the other 2 models (8.66 MB and 7.56 MB, respectively).

Moving on to the Total Avg. Row of Columns 10-11 in Table 1, ORISA recovered an average of 233 extra batch tokens beyond the active context window, representing an extra 38.02% of prior tokens. ORISA recovered the most tokens beyond the context window from the *Local AI* app, an average of 255 tokens from all 6 sessions. We noted that when the sessions' ground truth processed tokens are around the same, ORISA recovers more tokens from the session with a larger initial context window. For example, in the *Iris* app running the Llama 3.2 model, ORISA recovered 138 extra tokens when the initial context window is 256, while it recovered 327 extra tokens when that is 512. For KV cache recovery beyond the context window, Columns 12-14 show that ORISA recovered an average of 5,860K extra tensors (32.60%) across all apps with an average size of 11.44 MB. ORISA recovered the most extra KV tensors from the *PocketPal* app, with an average of 6,692K tensors representing 38.59% of extra context. We observed that ORISA can recover more KV tensors from sessions with a larger initial context window because the allocated KV cache buffer is significantly larger, e.g., 16,011K from the *PocketPal* app running Gemma 2 with a 512 context window compared with 9,813K from the same app and model with a 256 context window.

Takeaway. ORISA recovered 100% batch tokens and KV cache tensors within sessions' active context window, confirming that they still persist in memory even after chat content deletion from apps' UI. ORISA additionally recovered an average of 32.60% and 38.59% of batch tokens and KV tensors beyond the models' active context limit, revealing significant forgotten context retention in the memory image. Recovered tensor count varied with model size and configuration. ORISA recovered over 3 times larger KV cache from Gemma 2 sessions due to its higher head count and dimension, and sessions initialized with larger context windows allowed more tensor recovery.

5.2. Context Retention of Rehosted Sessions

Recall from the previous section §5.1, for each chat session, we keep prompting the LLM key-value pair information representing sample internal project notes (Figure 6) for the model to remember. We next evaluate the retention of this key-value pair knowledge in the rehosted sessions constructed with the token and KV cache recovery.

Experiment Setup. After the recovery of tokens and KV cache from memory, ORISA reconstructs the new session binary and rehosts it into a new *llama.cpp* session with a larger active context window (§4.5). Immediately after rehosting, we interrogate the model with this exact sentence to extract its prior knowledge – "List out all the internal project notes I told you earlier." In the response, we record the number of key-value pairs of knowledge it correctly recalled.

TABLE 2: Rehosted Sessions' Retention of Prompted Knowledge Within and Beyond the Active Context Window.

App	Model	Initial Ctx	Fed Pairs	Pairs In Ctx	Rehost Ctx	Extr[1] Pairs	Recovery Rate[2]
Iris	Llama 3.2	256	100	21	1,024	34	161.90%
		512	100	45	1,024	68	151.11%
	Qwen 2.5	256	110	25	1,024	43	172.00%
		512	110	48	1,024	76	158.33%
	Gemma 2	256	80	15	1,024	28	186.67%
		512	80	34	1,024	55	161.76%
Avg.	-	-	97	31	1,024	51	165.30%
Chatter UI	Llama 3.2	256	100	14	1,024	26	185.71%
		512	100	36	1,024	58	161.11%
	Qwen 2.5	256	110	15	1,024	29	193.33%
		512	100	39	1,024	59	151.28%
	Gemma 2	256	70	10	1,024	23	230.00%
		512	70	30	1,024	49	163.33%
Avg.	-	-	92	24	1,024	41	180.80%
Layla	Llama 3.2	256	100	18	1,024	29	161.11%
		512	100	41	1,024	59	143.90%
	Qwen 2.5	256	110	20	1,024	36	180.00%
		512	110	47	1,024	73	155.32%
	Gemma 2	256	80	13	1,024	23	176.92%
		512	80	31	1,024	43	138.71%
Avg.	-	-	95	28	1,024	45	168.93%
Others	-	-	94	27	1,024	46	168.70%
Total Avg.	-	-	94	27	1,024	46	169.59%

1: Extracted KV pairs from LLMs' responses.
2: Recovery rate is the extracted KV pairs in the rehosted session during interrogation over the number of KV pairs within the initial context window.

We designed this strict instruction-following experiment to ensure our evaluation was as repeatable and measurable as possible. In a real forensic investigation, the investigator has full discretion in how they query the reconstructed sessions. They can also roll back to the recovered context (from the memory image) prior to any interrogation whenever needed.

Findings. Table 2 shows the extracted key-value pair knowledge in the rehosted sessions. Columns 1 and 2 show the apps and models used. Column 3 shows the initial context window before the rehosting. Each model is run with both a 256 and a 512 active context window for each app with 2 separate sessions. Column 4 shows the number of key-value pairs (Figure 6) we prompted each session in a batch of 10 until the processed tokens of each session reached at least 1,024 to overfill the context window. Column 5 shows the number of key-value pairs that are still within each session's context window before the rehosting. Columns 6-8 show the rehosted session's context window, extracted KV pairs from the LLMs' responses during the interrogation, and the recovery rate with respect to the key-value pairs inside the context window before the rehosting.

Looking at the Total Avg. Row, Columns 4 and 5 of Table 2, we prompted an average of 94 key-value pair knowledge into each session before overfilling the sessions with at least 1,024 tokens. We found that an average of 27 key-value pairs still reside in the active context window. We noticed that this number is smaller than the tokens needed to represent those key-value pairs on average because a

portion of the context window is filled with system prompt tokens set up by each app, in addition to the prompted key-value pairs. For example, as shown in the Llama 3.2 Rows of the Iris and the Chatter UI app, although the number of fed key-value pairs is both 100, fewer pairs (14, 36) still reside in the context window of the Chatter UI app than the Iris app (21, 45). This is because the Chatter UI app had an average of 120 tokens to represent system prompts compared to around 42 for the Iris app. In addition to system prompt tokens, we also observed that different models' tokenizers also impacted the number of key-value pairs inside the context window.

Moving to the Total Row, Columns 6 and 7 of Table 2, ORISA rehosted each session with a larger 1,024 tokens context window after the recovery and active KV manipulation §4.5. In the interrogated responses, ORISA extracted an average of 46 key-value pairs contained in the originally fed pairs shown in Column 4, significantly larger than those originally retained before the rehosting. We noted that for each app, the number of extracted pairs for the Llama and Qwen models is larger than that of the Gemma model. For example, as shown in the Layla Row Column 7, sessions with Llama 3.2 (29, 59) and Qwen 2.5 (36, 73) had a similar number of pairs extracted in the rehosted sessions, while Gemma 2 model retained fewer pairs (23, 43). This is because the prior 2 models had a similar tokenization rate of around 9.5 tokens per pair, while the Gemma 2 model had around 12.7. As shown in Column 8 of Table 2, ORISA extracted an average of 169.59% of knowledge in the rehosted session compared with the original session, enabling investigators to interrogate prior sessions with more logically evicted context.

Takeaway. On average, although only 27 key-value pairs of knowledge remained within the original 256–512 token context windows for each app session, ORISA's recovery and rehosting with a 1,024 context window successfully extracted 46 pairs—an average recovery rate of 169.6% compared to the original context. Recovery rate varied by models' tokenization rate and apps' system prompt token length. This shows that ORISA not only preserves but substantially amplifies the retrievable knowledge from deleted sessions, enabling investigators to interrogate prior sessions with more context.

5.3. Semantic Consistency of Rehosted Sessions

When ORISA reconstructs and rehosts a session, the forensic validity of any responses generated during interrogation depends on whether those responses are consistent with what the model would have produced if the original session had continued uninterrupted. To measure this validity, we then evaluate the semantic consistency of responses during interrogation between the original and the rehosted sessions.

Experiment Setup. For each of the 10 apps, we ran 3 sessions with one of each model, similar to §5.1. For each session, we prompt the model with 20 key-value pairs of internal project notes to represent new knowledge the model didn't know beforehand (example shown in Figure 6). We set the context window of each session to 2048 tokens, greatly exceeding the number of tokens needed to represent that knowledge and any follow-up interrogations to avoid the original sessions from losing context. We then dumped the process memory for each app session and let ORISA recover and rehost it into a new *llama.cpp* session with the same 2048 context window. To compare the response behaviors of the rehosted and the original session, we then prompted both sessions with the exact instructions listed as "Rule: You must answer ONLY using information that appears visibly in our prior chat messages. If the requested information does not appear in our prior messages, respond exactly: Not in prior messages." We then prompted each session to recall 10 values of keys that was prompted before in prior messages (e.g. we prompted the session "What is the project codename?" when we previously fed the model with "Project codename = FERN-9e2f1b5d") and 5 values for keys that did not (e.g. we prompted the session "What is the client passphrase?" when this information was never fed into the model before). We used a 2:1 ratio for the recall and non-recall questions to avoid over-inflating the false positive rate due to small denominator effects. Finally, we record the TP, FP, TN, and FN of the models' responses to all 15 questions in both the original and the rehosted sessions.

Findings. Table 3 shows the comparison of models'

TABLE 3: Comparison of Response Behaviors Between Original and Rehosted Sessions Under Forensic Interrogation.

App	Model	Original Session				Rehosted Session			
		TP[1]	FP[2]	TN[3]	FN[4]	TP	FP	TN	FN
Iris	Llama 3.2	10	1	4	0	10	1	4	0
	Qwen 2.5	9	0	5	1	9	0	5	1
	Gemma 2	10	0	5	0	10	0	5	0
Chatter UI	Llama 3.2	10	2	3	0	10	2	3	0
	Qwen 2.5	9	0	5	1	9	0	5	1
	Gemma 2	10	0	5	0	10	0	5	0
Layla	Llama 3.2	10	2	3	0	10	2	3	0
	Qwen 2.5	10	1	4	0	10	1	4	0
	Gemma 2	10	1	4	0	10	1	4	0
Solia	Llama 3.2	9	2	3	1	9	2	3	1
	Qwen 2.5	9	1	4	1	9	1	4	1
	Gemma 2	10	1	4	0	10	1	4	0
Maid	Llama 3.2	10	0	5	0	10	0	5	0
	Qwen 2.5	10	1	4	0	10	1	4	0
	Gemma 2	10	0	5	0	10	0	5	0
Others	-	145	14	61	5	145	14	61	5
Total	-	291	26	124	9	291	26	124	9

1: The model correctly recalls the factual value of a key that was previously fed into the session.
2: The model outputs a value for a key that was never fed but asserts it as if known.
3: The model answers "Not in prior messages" for a key that was never fed.
4: The model fails to recall or incorrectly answers the value of a key that was fed earlier in the session.

responses in the original and the rehosted sessions, given the same prior knowledge, context window, and interrogation prompts. Columns 1 and 2 show the app and model used in each session. Columns 3-6 and 7-10 show the TP (The model correctly recalls the factual value of a key that was previously fed into the session), FP (The model outputs a value for a key that was never fed but asserts it as if known), TN (The model answers "Not in prior messages" for a key that was never fed), and FN (The model fails to recall or incorrectly answers the value of a key that was fed earlier in the session) of the models' responses to the 15 interrogation questions in the original session and the rehosted sessions respectively.

As shown in the Total Row, Column 4 of Table 3, 26 FP responses are recorded during interrogation of the original sessions. For example, in the Iris, Llama 3.2 Row, we recorded a FP response from the model where it answers "DEBUG-ON-2025" to the question "What is the client passphrase?" when the client passphrase was never fed into the session. As shown in the Total Row, Column 6, a total of 9 FN responses were also recorded. For example, in the Chatter UI, Qwen 2.5 Row, we recorded an FN response from the model where it answers "Not in chat" to the question "What tracking label did we use?" when we previously fed the session with the knowledge "Tracking label = TKT-8224-amber".

As shown in the Total Row, Columns 8 and 10 of Table 3, the rehosted sessions recorded the exact same number of FP (26) and FN (9) across all app sessions, indicating that the rehosted sessions preserved the exact context of the original session. This is because ORISA's rehosting not only preserves the original sessions' batch tokens and KV cache, it also preserves the pseudo-random generator state in the original session for subsequent generation. Since the FP and FN responses in the rehosted session are also present in the interrogation of the original session, they stem from the intrinsic behavior of the LLMs used themselves (e.g., FP from hallucination and FN from incomplete retrieval) – not from rehosting inaccuracy. This shows that ORISA enables investigators to treat the rehosted sessions' responses as credible evidence, since they faithfully represent what the original model would have produced had the session continued uninterrupted.

Takeaway. ORISA's rehosted sessions produced identical true and false responses (26 FPs and 9 FNs from 450 questions) to those of the original sessions, confirming that ORISA preserves the model's semantic state and internal reasoning behavior. The matching FP and FN patterns indicate that any inaccuracies arise from the model's inherent hallucination or recall limitations rather than rehosting artifacts. This demonstrates that ORISA allows investigators to treat the rehosted session's interrogation outputs as credible evidence of what the model would have generated if the session had continued.

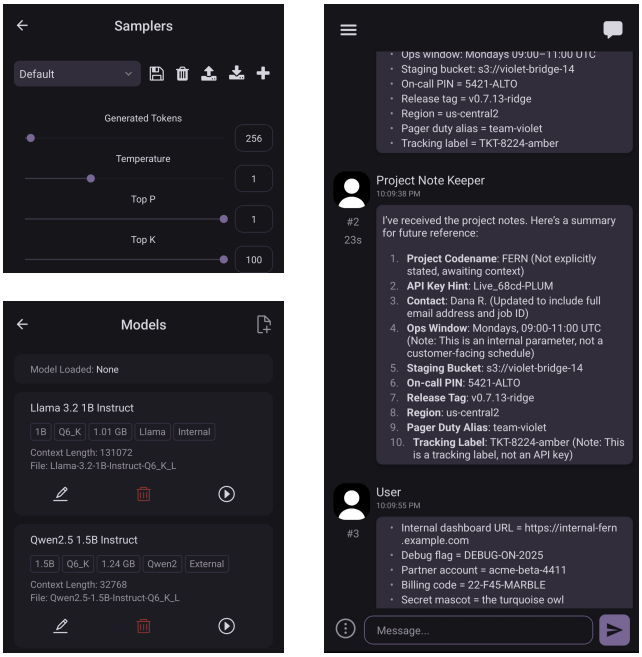

Figure 7: Chatter UI is one of the most popular native mobile LLM chat app. It allows users to customize model sampler parameters (top left) to fine-tune response behaviors, load custom local LLMs (bottom left), and instantiate custom chat characters in chat sessions(right).

6. Case Studies

6.1. Chatter UI

Chatter UI [35] is one of the most popular native mobile frontend apps for hosting on-device LLMs. It is hosted on GitHub and has received over 1.9K stars. Aside from allowing users to load custom models from disk, it also provides users freedom to fine tune the model sampler's parameters, such as temperature, Top P, Top K, etc., for different response behaviors. Users can also build different chat templates for different characters by customizing the system prompt sequence, prefix, and suffix, as shown in Figure 7.

To test ORISA's recovery of tokens and KV tensors from a Chatter UI chat session, we downloaded and loaded a Llama 3.2 1B model and configured it with a 256-token context window. We then created a custom "Project Note Keeper" character with description tokens indicating – "You are a helpful project note-keeping bot. Only remember factual statements that I tell you about the project." in the Chatter UI app and fed it with internal project notes key-value pairs to simulate knowledge not known by the model previously. We instrumented the app and kept prompting the session with key-value pair knowledge until the token count reached past 1024, where 100 pairs had been fed into the session. Notably, after each of the model's responses to our prompts, even after navigating away from the session and back, Chatter UI allows users to continue

Write Project Note Keeper's next reply in a chat between Project Note Keeper and User.You are a helpful project note-keeping bot. Only remember factual statements that I tell you about the project.<|eot_id|><|start_header_id|>assistant<|end_header_id|>

Figure 8: Recovered deserialized tokens show that the Chatter UI app incorporates user-defined character description prompts into system prompts and feeds them to the inference engine as permanent tokens not to be evicted.

generating the response without reprompting it. This implies that Chatter UI keeps the previous conversation turns in memory and does not require a new prompt to restart the conversation.

We then deleted the conversation from the app UI and dumped the app process's memory. After running ORISA, ORISA recovered a total of 429 tokens and 6537K tensors, representing 22% and 18% more from beyond the 256 tokens context window. While deserializing the tokens recovered, we found that the model retains the description tokens we fed into the app during the initiation of the chat session, representing the custom character of the current session, as shown in Figure 8. This indicates that the Chatter UI app stores those character description tokens as system prompts and feeds them to the inference engine as permanent tokens not to be evicted outside the context window (`n_keep` tokens for *llama.cpp*).

After ORISA rehosted the recovered context in a fresh *llama.cpp* session, we resumed the session with the interrogation prompt for it to "List out all the internal project notes I told you earlier". The rehosted session's response correctly outputted 26 values for the keys previously fed in, exceeding the 14 pairs still residing within the context window by 186%, indicating that ORISA's rehosted session successfully retained the forgotten context outside the context window recovered in memory.

6.2. Extending ORISA to the mllm Inference Engine

To test ORISA's adaptability to other inference engines, we extended ORISA to recover and rehost conversations from the mllm [36] Android app. Mllm is a lightweight multimodel LLM inference engine for mobile devices receiving more than 1.4K stars on Github. It diverges from traditional CPU-centric inference engines by providing additional unified support for OpenCL GPUs and Qualcomm NPUs. To extend to mllm, we added key data structure scanning signatures for ORISA. ORISA first scans for mllm `Tensor` object headers instead of `llama_batch` objects in llama.cpp to locate the batch tokens. For KV cache tensors, ORISA is adapted to scan for individual mllm `Tensor` objects for both the Key and Value at each attention layer associated with the `lmcache` class. The tensor validation strategy is similar to that of llama.cpp by checking shape metadata and ids. Forgotten context length is determined by comparing the extracted KV cache tensor size with the active context window size which is recovered as the `seq_dim` of the tensor object instead of the `ctx` parameter in llama.cpp.

To test ORISA's token and KV cache recovery from mllm, we loaded a llama 3.2 model with a 256-token context window in the mllm Android demo app. We kept prompting the session with the simulated internal project note knowledge according to the evaluation setup in §5.1 until the token count was over 1024. After deletion of the conversation from UI, we ran ORISA on the dumped memory. ORISA recovered a total of 493 tokens and 7028K tensors, with 34% and 28% from beyond the active context window respectively. After ORISA rehosted the recovered context in a mllm session, we interrogated the session with the same setup and prompts as discussed in §5.2. The rehosted session correctly responded with 26 key-value pair knowledge over the 18 pairs that were in the previously active context window, demonstrating a knowledge recovery rate of over 144%.

7. Discussion

7.1. Extension to Other Inference Engines

We have designed ORISA to recover and rehost mobile local LLM app sessions' context using the *llama.cpp* inference engine. However, we acknowledge that although *llama.cpp* remains the most widely adopted inference engine for mobile apps [37], [38] due to its ARM CPU optimizations and extensive model support, there are other frameworks for mobile local LLM deployment (mostly supported through each engine's official demo app) such as mllm [36], MediaPipe [39], etc. We have extended ORISA to mllm [36] by adapting to its context management data structure signatures. That said, it is possible to extend ORISA to other inference engines. Since all transformer inference engines require the same core structures – a token buffer that hosts token IDs and positions, a KV cache that stores attention head state, and a runtime state manager that keeps past token count and other metadata – extending ORISA to these engines only requires implementing new frontends to interpret their respective in-memory signatures for each core context-management data structure. The same methodology, namely token recovery, detokenization, active and forgotten KV cache extraction, active KV extension, and session rehosting, still applies.

7.2. Obfuscation and Anti-Forensics Techniques

Adversarial inference frameworks that deviate from the standard, efficient representation of token and KV cache tensors or those that obscure the generic structure of context-management objects can make ORISA's session recovery and rehosting more difficult. Nevertheless, prior work shows that even heavily obfuscated platforms can be analyzed using established memory forensics

techniques [40]–[42]. With additional effort, investigators can apply this perpendicular line of research to deobfuscate and reconstruct the data structures for context management objects [43], [44]. Once these structures are recovered, the investigator can build a corresponding frontend for the adversarial or obfuscated inference engine, similar to a regular and benign engine. Importantly, these frameworks must still retain the minimal information required to represent a session's context to continue the inference, enabling ORISA's recovery and rehosting.

7.3. Prerequisites for Running ORISA

ORISA's input is a memory image and disk image acquired by law enforcement. ORISA does not assume a rooted Android device. In fact in real forensic investigations, law enforcement often use third party tools and zero-day exploits [45]–[47] to acquire memory images. When OS data structure signatures are updated, ORISA can adapt to the changes with orthogonal and automated tools [48], [49].

8. Related Work

LLM Forensics. Prior research has explored forensic techniques to trace the sources of data poisoning in LLMs and neural networks [50]–[53]. Similarly, a growing body of research [54]–[65] surveyed and found LLM security vulnerabilities. For instance, Kim et al. [66] introduced a framework aimed at reducing the leakage of personally identifiable information from LLMs. Greshake et al. [67] explored indirect prompt injection attacks against LM frameworks. Furthermore, several work [58], [68]–[74] evaluated the robustness of current LLM frameworks against jailbreak attacks. However, no prior work explored the forensic investigation and rehosting of deleted local LLM sessions in mobile applications.

Memory Forensics. Memory forensics techniques have been proposed to recover evidence and aid program analysis [12], [13], [75]. Prior work has heavily relied on signature-based heap scanning [40] and value-set analysis (VSA) tools, such as DEEPVSA [76], to trace pointers and recover isolated kernel or application data structures. Oygenblik et al. [14] proposed a technique to accurately recover parameters that define DL models and rehost them for a live investigation. Saltaformaggio et al. [16] developed a spatial-temporal memory forensics technique to recover previous screens of Android apps. While memory forensics techniques [77] have been applied to areas such as GUI reconstruction [15], [78], malware analysis [79], and sequencing user activity [80], no prior work aimed to recover and rehost LLM's context artifacts. While existing VSA methodologies [76] are sufficient for recovering static, isolated artifacts (such as the model binary or raw token integers), they have no semantic understanding of LLM's unique attention mechanisms. Therefore, existing tools cannot physically reconstruct the multi-dimensional alignment required to feed forgotten KV tensors back into a live inference engine, necessitating the novel realignment and state manipulation techniques proposed by ORISA.

9. Conclusion

We presented ORISA, an automated system for recovering mobile LLM sessions' conversations and rehosting their context into a live session for investigators to interact with. ORISA recovers the model binary, reconstructs batch tokens, recovers and extends KV cache tensors, and rehosts both active and forgotten context into a live, app-agnostic session. When evaluated across 60 configurations spanning 10 Android LLM apps and 3 models, ORISA recovered 100% of batch tokens and KV-cache tensors within the active context window and over 30% beyond it, with rehosted sessions retaining an average of 169.6% of the original contextual knowledge. ORISA's rehosted sessions are also shown to preserve the original sessions' response behaviors, allowing investigators to treat the rehosted sessions' interrogation outputs as credible evidence.

10. Ethics Considerations

Continued prompting of the rehosted session may cause it to generate responses influenced by the investigator's prompts rather than the user's prior interactions. Therefore, the forensic investigator must carefully monitor the effects of interrogation prompts to minimize the risks of over-interpreting the session's output and incorrectly using it as evidence against the user. The investigator can always revert to the recovered context prior to any interrogation.

11. LLM Usage Considerations

Originality. LLMs were used for editorial purposes in this manuscript, and all outputs were inspected by the authors to ensure accuracy and originality.

Transparency. LLMs are integral to the methodology and experiments of the paper, as it studies the recovery and rehosting of mobile LLM apps' contexts. However, all LLMs chosen are open-source local models as discussed in §5.

Responsibility. We have chosen LLMs used in the evaluation of the paper, optimized models suitable for execution on resource-constrained mobile devices. Even with local models, we minimized the volume of queries to the ones strictly necessary to execute the experiments for the recovery and rehosting. We have also specified the hardware used to run the experiments in §5.

Acknowledgments

We thank the anonymous reviewers for their constructive comments and feedback. This material was supported in part by the Office of Naval Research (ONR) under grants N00014-23-1-2073, the Advanced Research Projects Agency for Health (ARPA-H) under Other Transaction Agreement No.

140D042590046, the as well as the GTRI Graduate Research Fellowship Program. Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of our sponsors and collaborators.

References

[1] C. Franzen, *Google unveils ultra-small and efficient open source ai model gemma 3 270m that can run on smartphones*. [Online]. Available: https://blog.n8n.io/local-llm/.

[2] A. G., *The 6 best llm tools to run models locally*. [Online]. Available: https://getstream.io/blog/best-local-llm-tools/.

[3] M. Farcas, *How to run a local llm: Complete guide to setup & best models (2025)*. [Online]. Available: https://blog.n8n.io/local-llm/.

[4] K. Anderson, *Create your own local ai assistant for enhanced privacy*. [Online]. Available: https://www.cognativ.com/blogs/post/create-your-own-local-ai-assistant-for-enhanced-privacy/271.

[5] A. Maiya, *Document intelligence with onprem.llm*. [Online]. Available: https://medium.com/data-science-collective/document-intelligence-with-onprem-llm-5ea69e5d8204.

[6] S. Sinha, *Personal ai that runs locally: How small llms are powering privacy-first experiences*. [Online]. Available: https://brimlabs.ai/blog/personal-ai-that-runs-locally-how-small-llms-are-powering-privacy-first-experiences/.

[7] *Android developers. data and file storage overview*. [Online]. Available: https://developer.android.com/training/data-storage.

[8] C. C.-C. Cheng, C. Shi, N. Z. Gong, and Y. Guan, "Evihunter: Identifying digital evidence in the permanent storage of android devices via static analysis," in *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS)*, Toronto, ON, Canada, Oct. 2018.

[9] J. Lee, A. Chen, and D. S. Wallach, "Total recall: Persistence of passwords in android," in *Proceedings of the 2019 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2019.

[10] A. Lyons, J. Gamba, A. Shawaga, J. Reardon, J. E. Tapiador, S. Egelman, and N. Vallina-Rodríguez, "Log: It's big, it's heavy, it's filled with personal data! measuring the logging of sensitive information in the android ecosystem," in *Proceedings of the 32nd USENIX Security Symposium (Security)*, Anaheim, CA, Aug. 2023.

[11] X. Yuan, O. Setayeshfar, H. Yan, P. Panage, X. Wei, and K. H. Lee, "Droidforensics: Accurate reconstruction of android attacks via multi-layer forensic logging," in *Proceedings of the 12th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Abu Dhabi, UAE, Apr. 2017.

[12] M. Andrea, P. Antonio, O. Andrea, A. Simone, B. Davide, and L. Andrea, "Nveiling byovd threats: Malware's use and abuse of kernel drivers," in *Proceedings of the 2026 Annual Network and*

Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2026.

[13] A. Oliveri, M. Dell'Amico, and D. Balzarotti, "An os-agnostic approach to memory forensics," in *Proceedings of the 2023 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2023.

[14] D. Oygenblik, C. Yagemann, J. Zhang, A. Mastali, J. Park, and B. Saltaformaggio, "Ai psychiatry: Forensic investigation of deep learning networks in memory images," in *Proceedings of the 33rd USENIX Security Symposium (Security)*, Philadelphia, PA, Aug. 2024.

[15] B. Saltaformaggio, R. Bhatia, Z. Gu, X. Zhang, and D. Xu, "Guitar: Piecing together android app guis from memory images," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, Denver, CO, Oct. 2015.

[16] B. Saltaformaggio, R. Bhatia, X. Zhang, D. Xu, and G. G. Richard III, "Screen after previous screens: Spatial-temporal recreation of android app displays from memory images.," in *Proceedings of the 25th USENIX Security Symposium (Security)*, Austin, TX, Aug. 2016.

[17] H. Hichri. [Online]. Available: https://huggingface.co/blog/not-lain/kv-caching.

[18] *Revealed: The contentious tool us immigration uses to get your data from tech firms*. [Online]. Available: https://www.theguardian.com/us-news/2023/may/25/us-immigration-surveillance-google-twitter-meta-personal-data.

[19] *Apple, google, meta are sharing more data with the us government than ever, proton finds*. [Online]. Available: https://thenextweb.com/news/big-tech-sharing-more-data-with-us-government.

[20] *Government requests for customer data report*. [Online]. Available: https://www.microsoft.com/en-us/corporate-responsibility/reports/government-requests/customer-data.

[21] *Llama.cpp. llm inference in c/c++*. [Online]. Available: https://github.com/ggml-org/llama.cpp.

[22] *Ggml. tensor library for machine learning*. [Online]. Available: https://github.com/ggml-org/ggml.

[23] *Google play store*. [Online]. Available: https://play.google.com/store/games?hl=en_US.

[24] *Iris (offline gpt)*. [Online]. Available: https://play.google.com/store/apps/details?id=com.nervesparks.irisGPT%5C&hl=en_IN.

[25] *Iris – features*. [Online]. Available: https://github.com/nerve-sparks/iris_android?tab=readme-ov-file#features.

[26] *Hugging face. the platform where the machine learning community collaborates on models, datasets, and applications.* [Online]. Available: https://huggingface.co/.

[27] G. Richard and V. Roussev, "Scalpel: A frugal, high performance file carver," in *Proceedings of the 2005 Digital Forensic Research Conference (DFRWS)*, New Orleans, LA, Aug. 2005.

[28] *Android 15*. [Online]. Available: https://developer.android.com/about/versions/15.

[29] *Llama-3.2-1b-instruct-q6_k_l*. [Online]. Available: https://huggingface.co/bartowski/Llama-3.2-1B-Instruct-GGUF.

[30] *Qwen2.5-1.5b-instruct-q6_k_l*. [Online]. Available: https://huggingface.co/bartowski/Qwen2.5-1.5B-Instruct-GGUF.

[31] *Gemma-2-2b-it-q6_k*. [Online]. Available: https://huggingface.co/roshanai/gemma-2-2b-it-Q6%5C_K-GGUF.

[32] *Frida. dynamic instrumentation toolkit for developers, reverse-engineers and security researchers*. [Online]. Available: https://frida.re/.

[33] *Objection - runtime mobile exploration*. [Online]. Available: https://github.com/sensepost/objection?tab=readme-ov-file.

[34] *Magisk. the magic mask for android*. [Online]. Available: https://github.com/topjohnwu/Magisk.

[35] *Chatterui - a simple app for llms*. [Online]. Available: https://github.com/Vali-98/ChatterUI.

[36] *Mllm. fast and lightweight multimodal llm inference engine for mobile and edge devices*. [Online]. Available: https://github.com/UbiquitousLearning/mllm.

[37] *Running on the edge: What, why and how. alternatives for running on-device ai on android*. [Online]. Available: https://iurysouza.dev/android-on-device-ai/.

[38] *Llama.cpp: The lightweight engine behind local llms*. [Online]. Available: https://www.sandgarden.com/learn/llama-cpp.

[39] *Mediapipe. on-device machine learning for everyone*. [Online]. Available: https://github.com/google-ai-edge/mediapipe?tab=readme-ov-file.

[40] Z. Lin, J. Rhee, X. Zhang, D. Xu, and X. Jiang, "Siggraph: Brute force scanning of kernel data structure instances using graph-based signatures," in *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2011.

[41] M. Carbone, W. Cui, L. Lu, W. Lee, M. Peinado, and X. Jiang, "Mapping kernel objects to enable systematic integrity checking," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, Chicago, Illinois, Nov. 2009.

[42] B. Saltaformaggio, Z. Gu, X. Zhang, and D. Xu, "Dscrete: Automatic rendering of forensic information from memory images via application logic reuse," in *Proceedings of the 23rd USENIX Security Symposium (Security)*, San Diego, CA, Aug. 2014.

[43] Z. Lin, J. Rhee, C. Wu, X. Zhang, and D. Xu, "Dimsum: Discovering semantic data of interest from un-mappable memory with confidence," in *Proceedings of the 19th Annual Network and*

Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2012.

[44] J. Lee, T. Avgerinos, and D. Brumley, "TIE: Principled Reverse Engineering of Types in Binary Programs," in *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2011.

[45] *Cellebrite shatters smartphone security illusion: Everything can be unlocked, researchers warn*. [Online]. Available: https://cybernews.com/security/cellebrite-shatters-smartphone-security-illusion/.

[46] *Novispy spyware installed on journalist's phone after unlocking it with cellebrite tool*. [Online]. Available: https://thehackernews.com/2024/12/novispy-spyware-installed-on.html.

[47] *Serbian police used cellebrite zero-day hack to unlock android phones*. [Online]. Available: https://www.bleepingcomputer.com/news/security/serbian-police-used-cellebrite-zero-day-hack-to-unlock-android-phones/#:~:text=Head%20of%20Security%20Lab%20at,be%20certain%20about%20it%20yet..

[48] Z. Qi, Y. Qu, and H. Yin, "Logicmem: Automatic profile generation for binary-only memory forensics via logic inference," in *Proceedings of the 2022 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Apr. 2022.

[49] B. Dolan-Gavitt, A. Srivastava, P. Traynor, and J. Giffin, "Robust signatures for kernel data structures," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, Chicago, Illinois, Nov. 2009.

[50] W. Zou, R. Geng, B. Wang, and J. Jia, "PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models," in *Proceedings of the 34th USENIX Security Symposium (Security)*, Seattle, WA, Aug. 2025.

[51] K.-H. Hung, C.-Y. Ko, A. Rawat, I. Chung, W. H. Hsu, P.-Y. Chen, *et al.*, "Attention Tracker: Detecting Prompt Injection Attacks in LLMs," in *Proceedings of the 2025 Annual Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics*, Albuquerque, New Mexico, USA, May 2025.

[52] S. Shan, A. N. Bhagoji, H. Zheng, and B. Y. Zhao, "Poison Forensics: Traceback of Data Poisoning Attacks in Neural Networks," in *Proceedings of the 31st USENIX Security Symposium (Security)*, Boston, MA, Aug. 2022.

[53] Z. Chen, Z. Xiang, C. Xiao, D. Song, and B. Li, "AgentPoison: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases," in *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, Dec. 2024.

[54] G. Wu, Z. Zhang, Y. Zhang, W. Wang, J. Niu, Y. Wu, and Y. Zhang, "I know what you asked: Prompt leakage via kv-cache sharing in multi-tenant llm serving," in *Proceedings of the 2025 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2025.

[55] E. Shayegani, M. A. A. Mamun, Y. Fu, P. Zaree, Y. Dong, and N. Abu-Ghazaleh, "Survey of Vulnerabilities in Large Language Models Revealed by Adversarial Attacks," *arXiv preprint arXiv:2310.10844*, 2023. [Online]. Available: http://arxiv.org/abs/2310.10844.

[56] X. Liu, J. Wang, J. Sun, X. Yuan, G. Dong, P. Di, W. Wang, and D. Wang, "Prompting Frameworks for Large Language Models: A Survey," *arXiv preprint arXiv:2311.12785*, 2023. [Online]. Available: http://arxiv.org/abs/2311.12785.

[57] F. Perez and I. Ribeiro, "Ignore Previous Prompt: Attack Techniques For Language Models," *arXiv preprint arXiv:2211.09527*, 2022. [Online]. Available: http://arxiv.org/abs/2211.09527.

[58] A. Wei, N. Haghtalab, and J. Steinhardt, "Jailbroken: How Does LLM Safety Training Fail?" In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, New Orleans, Louisiana, Dec. 2023.

[59] M. Ibrahim, G. S. Tuncay, Z. B. Celik, A. Machiry, and A. Bianchi, "LM-Scout: Analyzing the Security of Language Model Integration in Android Apps," *arXiv preprint arXiv:2505.08204*, 2025. [Online]. Available: http://arxiv.org/abs/2505.08204.

[60] Y. Liu, G. Deng, Y. Li, K. Wang, T. Zhang, Y. Liu, H. Wang, Y. Zheng, and Y. Liu, "Prompt Injection attack against LLM-integrated Applications," *arXiv preprint arXiv:2306.05499*, 2023. [Online]. Available: http://arxiv.org/abs/2306.05499.

[61] Alessandro Brucato, *LLMjacking: Stolen Cloud Credentials Used in New AI Attack*, https://sysdig.com/blog/llmjacking-stolen-cloud-credentials-used-in-new-ai-attack/, [Online; accessed: 28-October-2025], 2024.

[62] T. Liu, Z. Deng, G. Meng, Y. Li, and K. Chen, "Demystifying RCE Vulnerabilities in LLM-Integrated Apps," in *Proceedings of the 31st ACM Conference on Computer and Communications Security (CCS)*, Salt Lake City, UT, Oct. 2024.

[63] W. M. Si, M. Backes, and Y. Zhang, "Mondrian: Prompt Abstraction Attack Against Large Language Models for Cheaper API Pricing," *arXiv preprint arXiv:2308.03558*, 2023. [Online]. Available: http://arxiv.org/abs/2308.03558.

[64] T. Cui, Y. Wang, C. Fu, Y. Xiao, S. Li, X. Deng, Y. Liu, Q. Zhang, Z. Qiu, P. Li, Z. Tan, J. Xiong, X. Kong, Z. Wen, K. Xu, and Q. Li, "Risk Taxonomy, Mitigation, and Assessment Benchmarks of Large Language Model Systems," *arXiv preprint arXiv:2401.05778*, 2024. [Online]. Available: http://arxiv.org/abs/2401.05778.

[65] D. Ersoy, B. Lee, A. Shreekumar, A. Arunasalam, M. Ibrahim, A. Bianchi, and Z. B. Celik,

"Investigating the Impact of Dark Patterns on LLM-Based Web Agents," in *Proceedings of the 47th IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2026.

[66] S. Kim, S. Yun, H. Lee, M. Gubri, S. Yoon, and S. J. Oh, "ProPILE: Probing Privacy Leakage in Large Language Models," in *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, New Orleans, Louisiana, Dec. 2023.

[67] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, Copenhagen, Denmark, Nov. 2023.

[68] G. Deng, Y. Liu, Y. Li, K. Wang, Y. Zhang, Z. Li, H. Wang, T. Zhang, and Y. Liu, "MasterKey: Automated Jailbreak Across Multiple Large Language Model Chatbots," in *Proceedings of the 2024 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2024.

[69] E. Shayegani, Y. Dong, and N. Abu-Ghazaleh, "Jailbreak in Pieces: Compositional Adversarial Attacks on Multi-Modal Language Models," *arXiv preprint arXiv:2307.14539*, 2023. [Online]. Available: http://arxiv.org/abs/2307.14539.

[70] X. Shen, Z. Chen, M. Backes, Y. Shen, and Y. Zhang, "Do Anything Now: Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models," in *Proceedings of the 31st ACM Conference on Computer and Communications Security (CCS)*, Salt Lake City, UT, Oct. 2024.

[71] J. Yu, X. Lin, Z. Yu, and X. Xing, "GPTFUZZER: Red Teaming Large Language Models with Auto-Generated Jailbreak Prompts," *arXiv preprint arXiv:2309.10253*, 2023. [Online]. Available: http://arxiv.org/abs/2309.10253.

[72] F. Jiang, Z. Xu, L. Niu, Z. Xiang, B. Ramasubramanian, B. Li, and R. Poovendran, "ArtPrompt: ASCII Art-based Jailbreak Attacks against Aligned LLMs," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, Bangkok, Thailand, Jun. 2024.

[73] Y. Liu, G. Deng, Z. Xu, Y. Li, Y. Zheng, Y. Zhang, L. Zhao, T. Zhang, and K. Wang, "A Hitchhiker's Guide to Jailbreaking ChatGPT via Prompt Engineering," in *Proceedings of the 4th International Workshop on Software Engineering and AI for Data Quality in Cyber-Physical Systems/Internet of Things*, Porto de Galinhas, Brazil, Jul. 2024.

[74] W. M. Si, M. Backes, J. Blackburn, E. De Cristofaro, G. Stringhini, S. Zannettou, and Y. Zhang, "Why So Toxic? Measuring and Triggering Toxic Behavior in Open-Domain Chatbots," in *Proceedings of the 29th ACM Conference on Computer and Communications Security (CCS)*, Los Angeles, CA, Nov. 2022.

[75] R. Petrik, B. Arik, and J. M. Smith, "Towards architecture and os-independent malware detection via memory forensics," in *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS)*, Toronto, ON, Canada, Oct. 2018.

[76] W. Guo, D. Mu, X. Xing, M. Du, and D. Song, "DEEPVSA: Facilitating value-set analysis with deep learning for postmortem program analysis," in *Proceedings of the 28th USENIX Security Symposium (Security)*, Santa Clara, CA, Aug. 2019.

[77] F. Pagani and D. Balzarotti, "Back to the whiteboard: A principled approach for the assessment and design of memory forensic techniques," in *Proceedings of the 28th USENIX Security Symposium (Security)*, Santa Clara, CA, Aug. 2019.

[78] B. Saltaformaggio, R. Bhatia, Z. Gu, X. Zhang, and D. Xu, "Vcr: App-agnostic recovery of photographic evidence from android device memory images," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, Denver, CO, Oct. 2015.

[79] M. Yao, J. Fuller, R. P. Sridhar, S. Agarwal, A. K. Sikder, and B. Saltaformaggio, "Hiding in plain sight: An empirical study of web application abuse in malware," in *Proceedings of the 32nd USENIX Security Symposium (Security)*, Anaheim, CA, Aug. 2023.

[80] R. Bhatia, B. Saltaformaggio, S. J. Yang, A. I. Ali-Gombe, X. Zhang, D. Xu, and G. G. Richard III, "Tipped off by your memory allocator: Device-wide user activity sequencing from android memory images.," in *Proceedings of the 2018 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2018.

Appendix A.
Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

A.1. Summary

This paper presents ORISA, a memory forensics framework designed to recover deleted conversation histories from local Large Language Model (LLM) applications running on edge devices. By recovering batch tokens and key-value (KV) caches from process memory, ORISA is able to reconstruct logically deleted session contexts and rehost them into a live, app-agnostic session. The authors evaluated ORISA on 10 Android LLM apps, successfully recovering 100% of the active context and significant portions of forgotten context, enabling investigators to dynamically interrogate the model's past state.

A.2. Scientific Contributions

Creates a New Tool to Enable Future Science.

A.3. Reasons for Acceptance

1) The paper creates a new tool to enable future science. It develops the first end-to-end memory forensics framework specifically for recovering and rehosting local LLM conversations, addressing a novel and timely problem that existing memory and disk forensics techniques cannot solve.
2) The recovery of logically evicted but physically resident KV cache tensors beyond the active context window is a technically sophisticated insight.
3) The empirical validations across 10 popular LLM apps provide valuable evidence of ORISA's benefit, demonstrating a 100% active context recovery rate. The ability to realign context and rehost live sessions is an important step in enabling dynamic interrogation of a model's past state.