Ubiquitous Memory Augmentation via Mobile Multimodal Embedding System

Dongqi Cai
dc912@cam.ac.uk
University of Cambridge https://orcid.org/0000-0003-2751-2500
Shangguang Wang
Beijing University of Posts and Telecommunications
Chen Peng
Beijing University of Posts and Telecommunications
Zeling Zhang
Beijing University of Posts and Telecommunications
Nicholas Lane
University of Cambridge
Mengwei Xu
Beijing University of Posts and Telecommunications

Article

Keywords:

Posted Date: January 16th, 2025

DOI: https://doi.org/10.21203/rs.3.rs-5686668/v1

Additional Declarations: There is **NO** Competing Interest.

Ubiquitous Memory Augmentation via Mobile Multimodal Embedding System

Dongqi Cai
University of Cambridge
Cambridge, United Kingdom

Shangguang Wang
Beijing University of Posts and Telecommunications
Beijing, China

Chen Peng
Beijing University of Posts and Telecommunications
Beijing, China

Zeling Zhang
Beijing University of Posts and Telecommunications
Beijing, China

Nicholas D. Lane
University of Cambridge
Cambridge, United Kingdom

Mengwei Xu
Beijing University of Posts and Telecommunications
Beijing, China

Abstract

Forgetting is inevitable in human memory. Recently, multimodal embedding models have been proposed to vectorize multimodal reality into a unified embedding space. The generated embeddings can be easily retrieved to help mobile users remember and recall information when needed. However, as the model's capacity increases, its resource consumption also rises. The resulting slow throughput and significant computational resource requirements hinder its deployment on mobile devices. In this paper, we present `Reminisce`, the first efficient on-device multimodal embedding system that enables high-throughput and precise retrieval on resource-constrained mobile devices. The core design draws inspiration from the memory functions of the human brain, utilizing coarse-grained embeddings to identify likely candidates, which are then refined through query-driven fine-grained retrieval. A series of algorithm-hardware orchestrated optimizations automatically navigates this process and strenghen the embedding quality. Experiments show that `Reminisce` provides high-quality embedding representation with high throughput while operating silently in the background with negligible memory usage and reduced energy consumption.

Introduction

Mobile devices are ubiquitous nowadays. They capture lots of data in users' daily usage, digitally chronicling every aspect of a person's life. However, such data has not been fully utilized, attributed not to how to *store* them, but how to accurately *retrieve* them [1]. Specifically, smartphones have abundant storage (up to 1TB for iPhone 15 Pro) to host the information captured at 24x7, or local network-attached storage (NAS) can help accommodate those data as well; yet there has been a lack of method to efficiently locate the data intended at query time [2, 3]. The fundamental challenge is that data generated on devices is multimodal by nature (e.g., text, image, audio, IMU, etc), which are hard to be accurately retrieved in a user-friendly manner, e.g., through natural language [4].

Fortunately, the recent development of multimodal embedding models (MEM) has shed light on multimodal data retrieval. For example, CLIP unifies text and image modalities into one embedding space [5]. ImageBind further extends the functionality to 6 modalities through contrastive learning [6]. At architecture level, those models primarily consist of multi-layer transformer encoders [7]. In general, MEMs will catelyze two novel, exciting types of mobile applications as shown on the left of Figure 1: (1) *cross-modality searching*, which allows users to retrieve data in any modality with user-friendly interface; (2) *retrieval-augmented LLM generation*, which first identifies the relevant multimodal data (e.g., a picture) in a historical database with user prompt, and uses it to enhance the LLM generation quality, e.g., "in the picture I took for my kid yesterday, is she wearing a blue skirt or yellow?".

This work addresses the emerging scenario of *on-device multimodal embedding*, where MEMs operate as a system service on local devices to embed continuous data streams [8–11], functioning like a memory palace [12]. The local generation of embeddings is motivated by user privacy concerns, since MEMs can greatly expand the usage of device data, including screen UIs, recorded voices, etc. Offloading such information to the cloud may expose it to unauthorized access. For instance, it was revealed that Apple Siri had been eavesdropping on uploaded user conversations to enhance their public voice assistant model [13]. With cloud-based MEMs, users risk comprehensive life surveillance, with no way to verify.

Cost of on-device MEMs. Despite on-device MEM is private and generalizable to various downstream tasks [6, 14–16], it comes at a cost of resource intensity. Specifically, our pilot experiments identify two key obstacles towards on-device multimodal embedding: (1) *Low embedding throughput.* It takes dozens of seconds for billion-sized MEMs to embed a single image, which is significantly slower than the rate at which mobile devices generate data. As a result, even if the device runs continuously throughout the day, only 20% of daily information can be embedded. (2) *High energy*

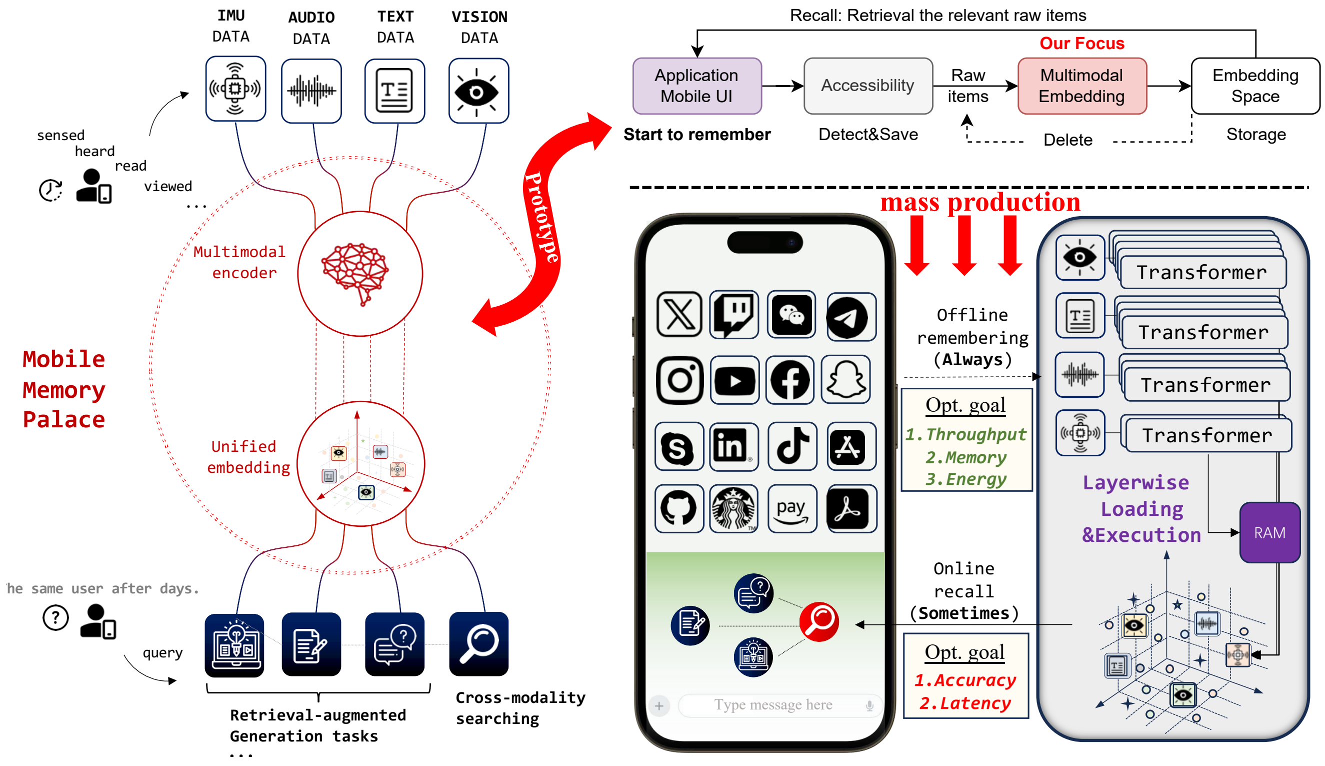


Figure 1. MEM-based ubiquitous memory palace workflow and its instantiation on mobile devices.

consumption. The slow inference speed, combined with the immense computing power required, results in extremely high energy consumption. Embedding data from applications consumes even more energy than running the applications themselves. As a result, the battery life of mobile devices is significantly reduced, often to less than 2 hours. Even if the embedding process is batched and executed offline (e.g., when the device is idle), its substantial resource demands still hinder practical deployment.

Our response: Reminisce. Reminisce is the first-of-its-kind efficient on-device multimodal embedding system. The key idea behind Reminisce is *coarse-grained embedding*, built upon the early-exiting technique. It draws inspiration from the memory functions of the human brain. The embeddings generated by the exited MEMs work as coarse-grained embeddings, which are used to filter out the most likely candidates during retrieval queries. These candidates are then further refined at query time to finalize accurate retrieval.

Furthermore, Reminisce introduces three key optimizations: predicting exits, healing exited branches, and speculative fine-grained retrieval. These enhancements adapt traditional early-exit methods for mobile MEMs, resulting in higher embedding throughput, improved embedding quality, and better retrieval precision. Our extensive experiments demonstrate that Reminisce significantly accelerates the multimodal embedding process while ensuring accurate searches.

We evaluate Reminisce on multiple mobile devices, delivering an average 14.9× improvement in throughput and 13.1× reduction in energy consumption compared to the original MEM. We further conduct a case study using recent Twitter data and a user study on mobile application traces collected from eight users for one week, demonstrating the practicality of Reminisce in real-world scenarios.

Results

Overall framework

As shown in right side of Figure 1 , we prototype an on-device MEM-powered search service to embed multimodal streaming data for future retrieval, functioning like a memory palace [12]. We specifically target mobile devices, including smartphones and IoT devices with similar computing capabilities. These devices have usable but weaker processing units compared to cloud servers, with limited battery and memory available for long-term background processes [17].

From the device perspective, the service consists of two runtimes:

- **Embedding runtime (Offline remembering in the backend)** continuously detects and stores newly generated multimodal content, such as downloaded images, scanned texts, listened-to audio, and logged IMU sensor data. Each item is processed layer by layer through MEMs[1], generating 1024-dimensional embeddings in a unified space.

- **Query runtime (Online recall in the frontend)** is triggered when the user searches for a specific item or performs other tasks based on search results. To retrieve relevant items, the query embedding is compared with stored embeddings to find the most similar matches. If the raw data corresponding to the matched embeddings aligns with the query intent, the query is tagged as successful.

System developers prepare the embedding model offline, typically by fine-tuning with powerful cloud GPUs, using widely-used pretrained multimodal embedding models [5, 6]. They define the expected offline costs and online performance for each application by configuring system hyperparameters before deployment.

Experimental Setup

Models The default MEM model is pretrained ImageBind (huge version) [6]. ImageBind extends the visual and textual pretrained encoder of CLIP [5] with additional capacity that embeds 6 modalities into a shared space. To demonstrate the scalability and versatility of `Reminisce`, we also evaluate it on CLIP. Over 80% (35 out of 43) of recent multimodal foundation models are based on those two MEM models [20].

Baselines We compare `Reminisce` to the following alternatives: (1) `Multimodal Embedding Model (MEM)` without any optimization. (2) `BranchyNet` [21], using a traditional early-exit mechanism. (3) `Fluid Batching` [22], a novel early-exit-aware batching algorithm that allows sample preemption at runtime. For completeness, we also include a naive baseline without layer-wise execution, though it incurs an unaffordable memory footprint on certain mobile devices. For a fair comparison, all baselines are equipped with ImageBind fine-tuned for the downstream task.

Metrics We evaluate the performance of `Reminisce` using the following metrics: (1) *Accuracy*: Retrieval accuracy for each task, with relative accuracy compared to the full-sized MEM model finetuned on the corresponding dataset. (2) *Latency*: Query latency on mobile devices, defined as the time from query initiation to completion. (3) *Throughput*: The amount of content processed per second or minute, assuming all samples are buffered in storage. (4) *Energy Consumption*: Energy consumed during the embedding process. (5) *Memory Usage*: Peak memory footprint during the embedding process.

Dataset As summarized in Table 1, we use four publicly available datasets across four modalities to demonstrate the effectiveness of `Reminisce`: (1) `COCO` dataset: Used for text-image retrieval, it contains 123k images, each paired with five captions. We use the validation subset of `COCO` to evaluate inference performance, with each caption retrieving its corresponding image. For example, given a caption, 75% of the relevant images are successfully retrieved within the top five results (R@5), based on the full-sized MEM model finetuned on the `COCO` dataset. (2) `FLICKR` dataset: Used for image-text retrieval, it consists of images paired with textual descriptions. Abosulte retrieval accuracy is 70% for the finetuned full-sized MEM model. (3) `CLOTHO` dataset: Used for text-audio retrieval, it contains audio clips paired with textual descriptions, enabling evaluation across audio and text modalities. Full-sized MEM model achieves 30% retrieval accuracy. (4) `HARSMART` dataset: Used for IMU retrieval, it employs fine-grained embeddings as queries to assess performance in retrieving IMU data based on embeddings. The MEM model achieves 78% retrieval accuracy.

Dataset	Modality	Size	Metric
COCO [23]	Text-**Image**	123,287	R@5
FLICKR [24]	Text-**Image**	8,091	R@1
CLOTHO [25]	Text-**Audio**	3,938	R@10
HARSMART [26]	**IMU**	10,299	Acc.

Table 1. Description of the datasets used. The embedded modality is highlighted. The performance metric is obtained from the full-sized ImageBind finetuned for the downstream tasks.

Additionally, to demonstrate the effectiveness of `Reminisce` in real-world scenarios, we conduct a case study using recent internet data that was not seen by the model during pretraining. Following prior empirical literature on Twitter analysis [27], we collect a recent publicly available dataset of Twitter memes, referred to as `TWITTER`. The `TWITTER` dataset contains 803 images and their corresponding meme descriptions across various up-to-date topics.

Hardware and Quantization We test `Reminisce` on the NVIDIA ORIN (ORIN) [28], Raspberry Pi 4B (RPI4B) [29], and one flagship smartphones with Qualcomm Snapdragon 8Gen3 (8GEN3) [30]. For the 8GEN3 smartphone, `Reminisce` runs on the CPU with the model quantized to INT4 precision to reduce memory consumption. To ensure the continuity of experiments, an external battery of equivalent capacity is connected. Since ORIN's GPU does not support INT4 computation, we load the raw model with FP32 precision. `Reminisce` runs on the RPI4B's CPU due to the lack of CUDA support.

Preliminary Measurements

First, we present a preliminary study to demonstrate the utility and efficiency of on-device multimodal embedding

[1]Deep learning models are often too large for mobile devices, leading to inference processes being terminated by the OS. Current mobile inference engines provide layerwise execution to support large models [18, 19].

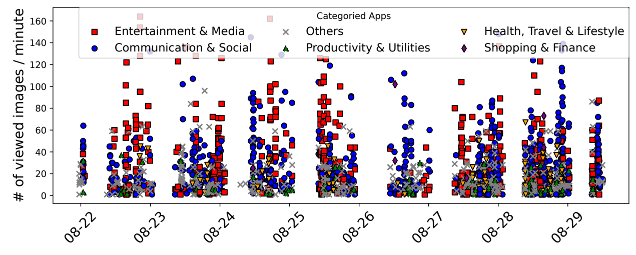
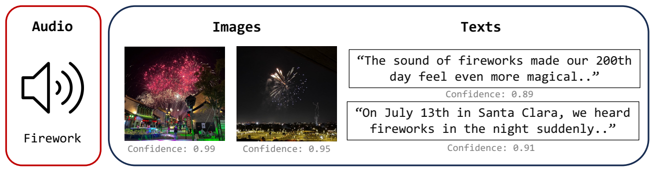
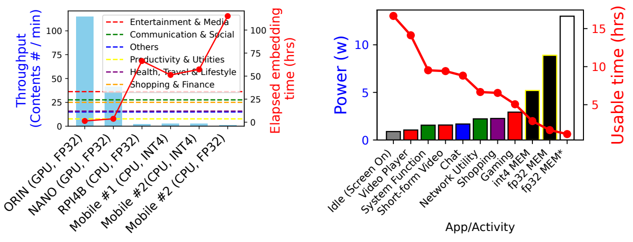


Figure 2. Viewed image trace of mobile users.



"The sound of fireworks made our 200th day feel even more magical.."
Confidence: 0.89
"On July 13th in Santa Clara, we heard fireworks in the night suddenly.."
Confidence: 0.91

Figure 3. Demo of cross-modal retrieval using MEMs.



(a) Throughput **(b)** Battery

Figure 4. Throughput and energy issues. (a) MEM inference speed across different devices compared to the average image generation speed of various mobile applications. (b) MEMs rapidly drain the battery of the test mobile phone. * indicates GPU inference power consumption of the Jetson ORIN.

in real-world scenarios. We conducted a user study to collect viewed images from daily mobile applications used by 8 volunteers, aged 20 to 52, over the course of a week. To achieve this, we developed an Android application with accessibility services [31] to detect and store newly appeared visual content[2]. One collected trace is shown as an example in Figure 2.

Observation: MEMs are contextually expressive. All images and corresponding texts are collected and embedded using ImageBind [6]. By aligning multimodal embeddings into a common space, ImageBind can effectively retrieve semantically relevant content from different modalities using human-friendly input formats. For example, as shown in Figure 3, the sound of fireworks retrieves images of fireworks from the albums and their corresponding textual notes with high confidence. A rigorous numerical analysis across various tasks will be presented in the evaluation section.

Challenge: MEMs are resource-intensive. To assess the cost of on-device embedding, we ran ImageBind inference on four different mobile devices, ranging from development boards to commodity smartphones.

Huge workloads and low throughput. Despite their contextually expressive capabilities, the embedding speed is too slow to keep pace with the figures generated by applications. As shown in Figure 4a, on all CPU-based devices, the encoding speed is insufficient for real-time application use. Over a full day of usage, the speed is only sufficient to embed 20% of the figures generated by applications, requiring more than 100 hours to process all figures from a single day. Even with a GPU, Jetson NANO [32] struggles to handle an entertainment task generating 36.3 images per minute. The only exception is the NVIDIA ORIN [28], which performs comparably to a cloud server using an NVIDIA A40 [33]. However, continuously running the CPU or GPU on mobile devices is impractical due to battery depletion.

Battery depletion. The heavy embedding workloads and low throughput significantly strain battery life. Continuous embedding drains the battery even faster than running the app itself. To illustrate, we used ImageBind to continuously embed figures from daily apps. As shown in Figure 4b, the embedding process consumes more energy than the apps themselves. For example, even when quantized to INT4, MEMs consume 1.8× more energy than gaming. We also measured GPU energy consumption on an NVIDIA ORIN[3]. While GPUs process data faster, they consume more energy than CPUs, making them unsuitable for long-term embedding in the current MEM design.

The above challenge has been addressed by the proposed optimized system Reminisce, which significantly improves throughput and reduces energy consumption, as shown in the following evaluation section.

Evaluation

We evaluate Reminisce to address the following key questions: (1) How much improvement does Reminisce achieve in terms of embedding throughput and relative retrieval accuracy under different memory budgets across various devices? (2) How much performance improvement does each component contribute? (3) What is Reminisce's performance under different query latency budgets? (4) What is the system cost of Reminisce? (5) How does Reminisce perform on commodity mobile phones in daily usage scenarios?

End-to-end Performance

First, we present the end-to-end embedding throughput performance under the layer-wise inference setting, a more user-friendly approach for always-on daily applications due to its low memory footprint.

[2]Images are hashed to include only new content. Images smaller than 100KB are excluded to avoid capturing icons and minor system elements.

[3]Current mobile inference engines cannot effectively utilize GPUs for MEM execution [9, 18, 34].

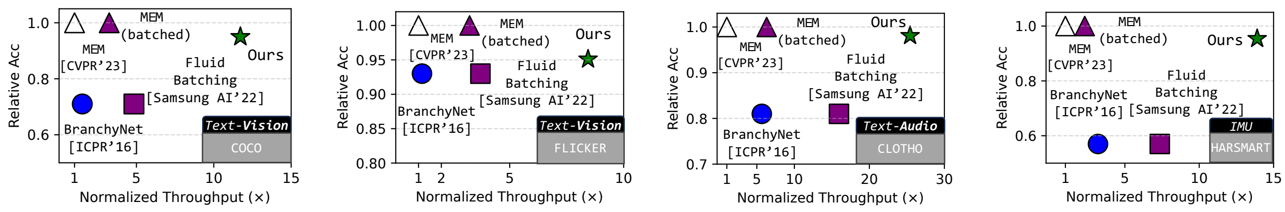


Figure 5. Illustration of throughput-to-accuracy on Jetson TX2. For fairness, only layerwise baselines are included.

Dataset	COCO				FLICKER				CLOTHO				SENSOR			
Throughput (Contents / s)	**Relative Accuracy**	**ORIN (FP32)**	**RPI4B (FP32)**	**8GEN3 (INT4)**	**Relative Accuracy**	**ORIN (FP32)**	**RPI4B (FP32)**	**8GEN3 (INT4)**	**Relative Accuracy**	**ORIN (FP32)**	**RPI4B (FP32)**	**8GEN3 (INT4)**	**Relative Accuracy**	**ORIN (FP32)**	**RPI4B (FP32)**	**8GEN3 (INT4)**
MEM (w/o layerwise)	100.0%	OOM	OOM	0.17	100.0%	OOM	OOM	0.16	100%	83.3	0.26	0.34	100%	127	0.88	/
MEM		1.92	0.04	0.05		1.92	0.04	0.05		5.23	0.22	0.27		31.3	0.74	/
MEM (batched)		6.22	0.05	0.10		6.22	0.05	0.10		33	0.25	0.32		72	0.84	/
BranchyNet (w/o layerwise)	71.0%	OOM	OOM	0.25	92.7%	OOM	OOM	0.19	81%	211	0.66	0.85	57%	405	2.81	/
BranchyNet		2.88	0.06	0.07		2.21	0.04	0.06		29.6	0.55	0.69		99.9	2.36	/
Fluid Batch		9.29	0.07	0.16		7.13	0.05	0.12		83.4	0.63	0.80		230	2.68	/
Ours	**95.0%**	**22.5**	**0.10**	0.31	**95.1%**	**16.2**	**0.07**	0.22	**98.1%**	**133**	**0.66**	0.84	**95.4%**	**435**	**4.52**	/
Ours (w/o layerwise)		**47.9**	**0.10**	0.33		**33.5**	**0.07**	0.23		**211**	**0.66**	0.85		**680**	**4.71**	/

Table 2. Throughput vs. relative retrieval accuracy. '/' means not supported. 'OOM' means out of device memory.

Reminisce achieves an order of magnitude improvement in throughput. Table 2 summarizes the embedding throughput comparison, while Figure 5 shows that Reminisce can achieve a 14.9× average throughput improvement compared to MEM. This gain is primarily driven by the early-exit mechanism, which allows the model to exit early when the embedding is sufficiently accurate, avoiding unnecessary computations. Additionally, after parameter-efficient healing, the coarse-grained embeddings can convey similar semantics to fine-grained embeddings. For instance, in the text-image retrieval task on the COCO dataset, Reminisce delivers an 11.7× throughput improvement with less than 3% accuracy loss.

Regarding stronger baselines, Fluid Batch introduces a early-exit-aware batching mechanism, achieving a 3× throughput improvement over the naive early-exit baseline BranchyNet and 5× over MEM under the layer-wise inference setting. However, Reminisce still outperforms Fluid Batch across all datasets, providing up to a 3× speedup in throughput. The advantages of Reminisce arise not only from the early-exit mechanism but also from the pre-exit strategy, which predictively adjusts the embedding granularity based on the sample's characteristics.

Although Reminisce primarily targets layer-wise scenarios, we also evaluated throughput performance when loading all encoders simultaneously. While this approach can provide significant throughput gains, it presents challenges such as out-of-memory errors on ORIN and RPI, especially for larger models like vision encoders. Reminisce maintains high throughput in a layer-wise setting, making it a more practical solution for resource-constrained devices. For instance, on the 8GEN3 mobile, Reminisce can process data up to 2.5× faster than the naive MEM without loading layers sequentially, while reducing memory footprint by up to 3.3×.

Interestingly, we find that healing the exited larger MEMs is more effective than using a smaller-sized foundation model.

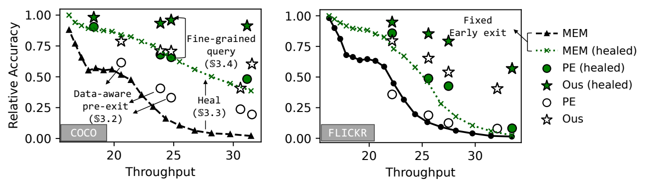


Figure 6. Throughput-to-accuracy trade-off with and without Reminisce's key designs, demonstrating their significance. PE refers to pre-exited coarse-grained embeddings without fine-grained upgrading during the query phase.

For example, using CLIP-b/16 with 85.6M parameters results in embeddings that are 2.7× faster than ImageBind but significantly reduces the ability to embed different modalities concisely, leading to up to 39.8% accuracy loss. Though with our system design, the accuracy loss is mitigated to 3.1% with a 2.7× throughput improvement, its performance is still inferior to the healed ImageBind. Fortunately, Reminisce enables narrows the cost gap between the two models, while achieving much higher retrieval performance.

Significance of Key Designs

Effect of Exit Healing As illustrated in Figure 6, while the zero-shot embedding of ImageBind has the generalization ability across different datasets, the exit healing mechanism is crucial for enhancing Reminisce's performance. As shown by the green dotted lines, retrieval accuracy significantly improves after healing the exited branches. For instance, compared to zero-shot MEM, exit healing boosts retrieval accuracy by 37.8% and 13.2% on average for the COCO and FLICKR datasets, respectively.

Effect of Data-aware Pre-exit After healing, Reminisce leverages the pre-exit mechanism to dynamically adjust embedding granularity based on each sample's characteristics. It can predictively exit at the optimal layer to balance the trade-off between accuracy and throughput. As shown in Figure 6, compared to exiting all samples at a fixed layer,

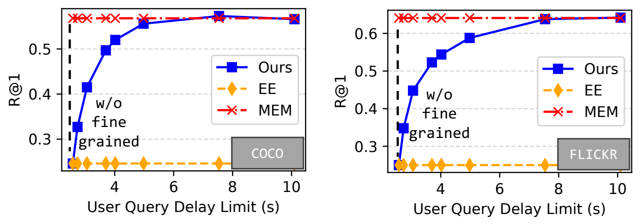


Figure 7. Performance under different query latency budgets.

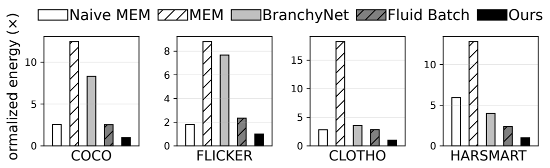


Figure 8. Energy consumption.

the data-aware pre-exit mechanism improves retrieval accuracy by up to 19.8%. The higher coarse-grained retrieval performance is crucial for achieving optimal fine-grained retrieval.

Effect of Speculative Fine-grained Query With a default query candidate pool size of 10, retrieval accuracy using filtered fine-grained embeddings is, on average, 35.5% higher than the previous coarse-grained retrieval accuracy. This improvement is due to the fact that over 95% of the targets retrievable by full-sized MEMs are successfully retrieved from the toplist of coarse-grained embeddings. As a result, the embedding accuracy of Reminisce is comparable to that of the full-sized MEM.

Impact of Query Latency Budget

Although query cost is negligible compared to embedding cost in the long term—since queries occur less frequently than daily always-on embeddings—it is immediately noticeable to the user. Thus, we show Reminisce's performance under different query latency budgets in Figure 7. Query latency consists of three components: text embedding, matching, and fine-grained embedding. Baseline methods with memory encoders require only the first two steps, typically taking 2 seconds. With a higher query latency budget, we can improve fine-grained embedding accuracy from 27% to 55%.

Additionally, similar to web cookies [35], the query process can skip the complex fine-grained embedding when it is repeated, making it more efficient for multi-query scenarios where frequently queried items are retrieved faster. Once a local embedding is queried, its embedding is permanently upgraded.

System Cost

Energy Consumption Figure 8 shows the normalized energy consumption of Reminisce and various baselines.

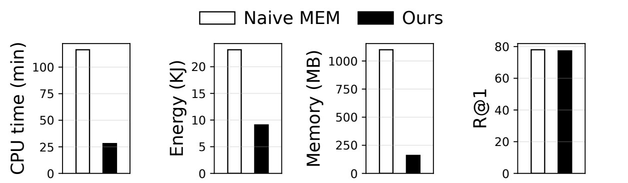


Figure 9. Performance analysis during 30 minutes of Twitter browsing. Device: 8GEN3 [30].

Reminisce reduces energy consumption by up to 18.2× and 13.1× on average compared to layerwise-executed baselines. Even compared to naive MEM without layerwise execution, Reminisce still achieves up to 3.3× energy savings on average. This is due to Reminisce's ability to determine the optimal number of layers for embedding and offload embedding computation to the less frequent querying process.

Storage Cost We store the embeddings of the items in INT4 precision. Each embedding is 1024-dimensional, resulting in a storage cost of approximately 5KB per item. Based on the trace statistics in §, typical users encounter around 6000 images daily. Thus, the storage cost for image embeddings is roughly 29.3MB per day. Annually, this amounts to about 10.4GB, which is comparable to the storage required for a high-quality movie. In contrast, the current off-the-shelf solution Rewind [36] consumes 14GB of storage per month on average, as officially reported [37].

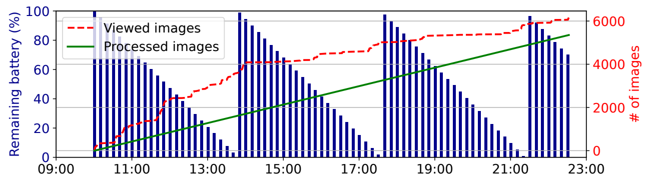
Case Study: Twitter Meme Retrieval

As with the previous datasets, we evaluated the performance of Reminisce on the TWITTER dataset. A total of 828 figures were embedded according to the trace data collected. Naive MEM takes over an hour to complete the retrieval task on a fully utilized CPU. In comparison, Reminisce achieves a 4× throughput improvement, completing the task within 27 minutes. Moreover, Reminisce demonstrates significant resource savings, using 5× less memory and 10× less energy than the baseline. This is due to sequentially loading layers and reducing the total number of layers executed. Storing these figure embeddings requires approximately 3MB, which is comparable to the size of a single raw image. During the query phase, the query latency is 0.5s, which is acceptable for daily use, as surveyed. Our case study demonstrates that Reminisce can provide high-quality embedding representation with highly optimized system performance in real-world usage scenarios.

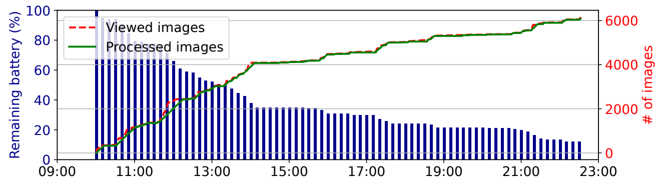
User Study: Mobile Application Trace

To further validate Reminisce, we conducted a user study by collecting real user data and simulating the system's performance in embedding images generated during daily mobile app usage[4]. Without Reminisce, the naive MEM system (in

[4]We do not account for charging time or the energy used by the applications themselves to provide a more straightforward comparison between naive MEM and Reminisce.



(a) Naive MEM.



(b) Ours

Figure 10. Energy and throughput comparison of embedding images viewed under real mobile traces.

INT4 precision) would require more than 3 battery charges per day, and over 20% of the images would remain unembedded due to time constraints. In contrast, Reminisce reduces the number of required charges by 3×, allowing all daily generated data to be embedded. This user study highlights Reminisce's ability to efficiently manage and embed large volumes of data, reducing the burden on battery life and ensuring that the vast majority of daily usage data is preserved and embedded in real-time.

Discussion

In this work, we develop Reminisce, an efficient on-device multimodal embedding system to function as a memory augmenting service. Extensive experiments and user studies demonstrate that Reminisce significantly improves embedding throughput and reduces energy consumption while maintaining high retrieval accuracy, making it practical for modern mobile devices.

We offload the full-sized embedding cost to the query phase, which is infrequent and carries precise retrieval information [2]. Only coarse-grained key information is preserved using exited embedding models. This mirrors the human brain, which retains key information in long-term memory and recalls details only when necessary [38]. Different from advanced sparsification or quantization optimizations, which provides little to no benefit during inference due to the limited support of mobile hardware [39–43], Reminisce can be seamlessly integrated into off-the-shelf mobile applications to enhance user experience without requiring complex hardware modifications.

The ability of Reminisce to operate within the constraints of devices such as smartphones and Raspberry Pi 4B, while maintaining high-quality embeddings, highlights its practicality for real-world applications. For instance, mobile users can now efficiently index and recall multimedia content, fostering new use cases in personal assistants, health tracking, etc.

A pivotal advantage of Reminisce lies in its on-device processing capability, which eliminates the need to offload sensitive data to cloud services. This significantly mitigates risks associated with data breaches and unauthorized access, addressing a critical concern in privacy-preserving AI systems.

However, due to the extra memory overhead of batching parallelism, Reminisce has a slightly higher peak memory footprint compared to the naive layer-wise baseline. Detailed information is provided in the Appendix. Fortunately, it is still within a practical range, e.g., 92M for embedding IMU information, which is below the average Android application memory consumption of 100M as reported in 2019 [17, 44]. After 5 years, the mobile RAM capacity has increased significantly, with up to 24GB available on high-end devices [45]. Less than 300MB of peaky memory usage is affordable for most modern mobile devices.

This study provides the following takeaway messages:

- We prototype the first MEM-empowered mobile search service architecture. Through user studies and pilot experiments, we identify challenges related to low embedding throughput and high energy consumption[5].
- We introduce Reminisce, an efficient on-device multimodal embedding system that addresses these challenges. Reminisce incorporates three novel techniques: preemptive exit for dynamic execution scheduling, progressive model healing for cache optimization, and speculative retrieval to correct premature exits.
- Extensive experiments demonstrate that Reminisce significantly improves throughput and reduces energy consumption while maintaining search performance, making it practical for modern mobile devices.

Methods

Reminisce Overview

As shown in Figure 11, Reminisce provides a memory encoder for clients to build coarse-grained embeddings offline, while the rest of the model functions as a live encoder for precise online retrieval. (1) *System developer preparation:* Developers first refine widely-used pretrained multimodal models to reduce the number of layers needed for token prediction. The refined model is then deployed to mobile devices for offline embedding. (2) *Client offline embedding:* Users employ part of the memory encoder to build superficial embeddings for pre-exit prediction. After pre-exit, samples with the same exits are batched and processed layer by layer

[5]Codebases and collected trace data have been made public.

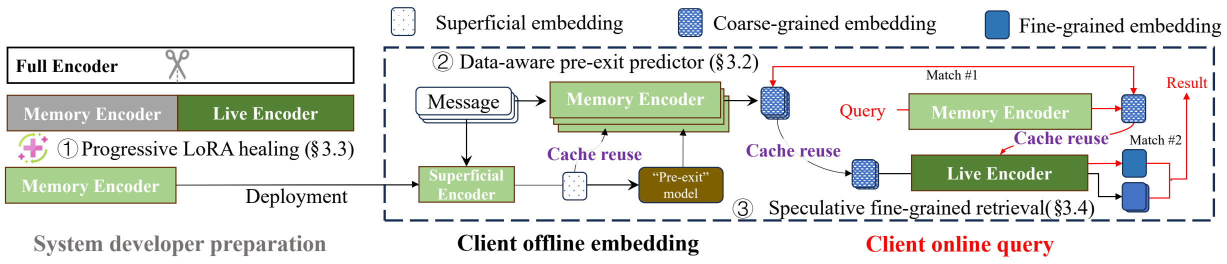


Figure 11. Overview of Reminisce.

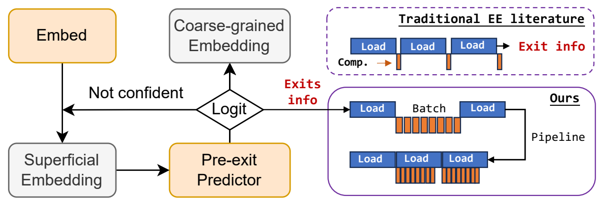
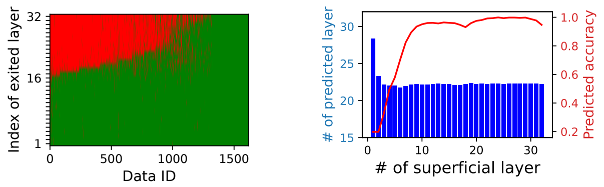


Figure 12. Data-aware pre-exit workflow.



(a) Exit stastics. **(b)** Predictor performance.

Figure 13. (a) Each dataset has a different optimal exit point. (b) The data-aware predictor is well-trained to assign the appropriate exit for each sample.

through pipeline scheduling to generate coarse-grained embeddings. (3) *Client online query:* During the query phase, the query is embedded for matching. Likely candidates are filtered and refined from the coarse-grained embeddings, which are then matched with the query embedding to finalize retrieval.

Data-aware Pre-exit Predictor

Traditionally, most early-exit methods decide whether to exit at the end of each branch computation [21, 46, 47]. This approach limits hardware acceleration and batching, as exit points vary by data, leading to inconsistent workloads within batches and memory fragmentation [21, 48, 49]. Although some predictive models for CNNs [49] predict exit values in advance, they cannot scale to MEMs due to their convolution-specific design. In this work, we propose a unified, lightweight early-exit predictor model for all modalities, derived from intermediate data embeddings. The data-aware pre-exit predictor preemptively decides the exit point for MEMs, enabling batch scheduling for better parallelism and helping to amortize and hide loading time.

Algorithm 1: Our Pre-exit Predictor

```
input  :Superficial Embedding Layer $N$;
        Predict model $\phi_S$;
        Burst-in Streaming Input, $\mathbb{X}$.
output :Embedding, $\mathbb{E}$.
1  Function Data-aware_Coarse-grained_Embedding($N$, $\phi_S$, $\mathbb{X}$):
2      Embedding ← Batched_Layerwise_Encoding(0, $N$, $\mathbb{X}$);
3      Predicted Exit $e \leftarrow \phi_S(\mathbb{E})$;
4      Group $\mathbb{X}$ into $\mathbb{X}^e$ with the same exit seperately;
5      forall $\mathbb{X}^e$ do
6          Embedding ← Batched_Layerwise_Encoding($N$, $e$, $\mathbb{X}^e$);
7      Store Embedding $\mathbb{E}$ in the disk.
8  Function Batched_Layerwise_Encoding($i$, $j$, $\mathbb{X}$):
9      $\mathbb{X}_B$ ← Batching $\mathbb{X}$;
10     forall $\mathbb{X}_B$ do
11         while $i<j$ do
12             Encode $\mathbb{X}_b^i$; load layer $i$+1 concurrently;
13     Embedding ← Postprocessor(Intermediate results);
14     return Embedding.
```

Data-aware coarse-grained embedding granularity Different data contains varying amounts of information content. Unlike previous work that defines predictive models manually, we propose using intermediate embeddings to predict the exit value without supervision. First, we build the fine-grained embedding F_x for each data point $x \in \mathsf{X}$ as a proxy query label. Next, we feed the input into the pre-trained MEM layer by layer, obtaining a set of coarse-grained embeddings C_x^i at different granularities $i \in \text{range(layers)}$. We then measure the similarity between the fine-grained and coarse-grained embeddings. When the similarity between F_x and C_x^i becomes the largest among F_x and C_X^i, query retrieves C_x^i from C_X^i successfully. We mark it as a valid embedding exit. The intermediate embeddings are fed into the predictor model, and an MLP model is trained to predict its exit value. This method outperforms fixed early-exit baselines, as will be shown in §.

Batch-friendly and pipeline execution As shown in Figure 12, with the data-aware pre-exit predictor, we can predict the exit value before embedding, enabling efficient batching of input data. In addition to early-exit-specific batching, we propose pipelining the layer-by-layer encoding process, where loading and embedding are conducted simultaneously.

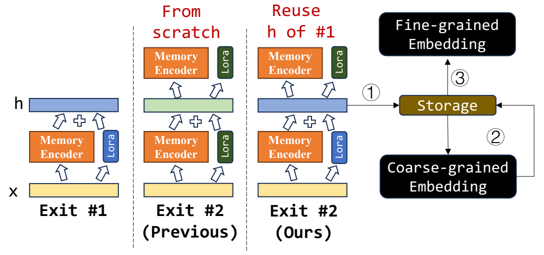


Figure 14. Comparison between Progressive LoRA and previous methods.

Pre-exit Predictor in detail We summarize the use of the pre-exit predictor in Algorithm 1. First, we load Layer$_i$ and encode all input data as a batch, while Layer$_{i+1}$ is loaded concurrently to minimize loading time. This process iterates until all N layers are loaded. Next, we feed the intermediate embeddings (i.e., superficial embeddings) to the predictor model. Data are then batched according to the predicted exit values. These steps are repeated for each batch until all data reach their predicted exits.

Pre-exit predictor cost Training the predictor is efficient, requiring only tens of iterations on hundreds of samples, taking just a few minutes on a single GPU. The trained predictor is lightweight, with a memory footprint of around 1MB. The main concern is the cost of computing the superficial embedding. Fortunately, this embedding can be reused for subsequent coarse-grained embeddings.

Micro Experiments are conducted to demonstrate the effectiveness of the pre-exit predictor. As shown in Figure 13b, prediction accuracy improves with the increase of superficial embedding layers. As indicated by Figure 13a, most samples require the complexity of more than 7 layers. With $N = 7$, the predicted accuracy is 85%, the average predicted layer is 15.5, and the average actual layer is 16.5. An interesting finding is that as the intermediate embeddings are fed layer by layer, the deeper the layers, the more accurately the predictor model can determine the exit value. This improvement occurs because deeper layer embeddings are more discriminative and better suited for predicting the final embedding.

Progressive LoRA Healing

Original MEMs are not designed for early exit, as they tend to distribute computation across all layers. As a result, most data requires many layers before exiting. We propose a progressive LoRA approach to heal the model, reducing the number of layers needed for each token.

Parameter-efficient LoRA Healing Previous early-exit healing approaches [50] use the parameter-efficient fine-tuning method, LoRA [51], to distill knowledge into lower layers, reducing the number of layers required for each token. Naive LoRA tuning fine-tunes a separate LoRA suite for each early-exit layer. For instance, with 32 exits, 32 LoRA suites are required. While this ensures good performance, it has a

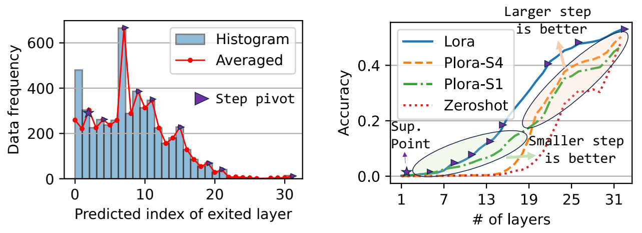


Figure 15. The progressive steps affect tuning performance.

significant drawback: the embedding from layer n cannot be reused to compute the embedding for layer $n+1$. As illustrated in Figure 14, this occurs because LoRA $l_n^{1,\dots,n}$ for layer n is not the same as the first n layers of LoRA $l_{n+1}^{1,\dots,n+1}$. Unlike standard embeddings, which complete all layers sequentially, early-exit methods must check whether each layer is the final one. If layer n's embedding is incompatible with layer $n+1$, the early-exit method must recompute the embedding for layer $n+1$ from scratch, negating many of the benefits of early exit.

On cloud servers, computation is not a major issue due to their high processing power, and reducing model weights to alleviate I/O pressure is the primary concern. However, for mobile devices with limited computational power, I/O pressure is less of a concern since they typically serve only one user at a time.

Progressive LoRA healing (P-LoRA) Reminisce proposes a progressive LoRA healing method to address this issue, aiming to use a single LoRA suite for all exits. To achieve this, we tune the LoRA layer by layer. For each exit, we tune only the LoRA for the current exit while keeping the previous exits' LoRA fixed. Since the tunable parameters are fewer than the fixed ones, the healing capacity is weaker compared to using separate LoRA suites, which negatively impacts convergence (i.e., fine-grained embedding) performance, as shown in Figure 15. To mitigate this, instead of tuning one LoRA layer at a time, we progressively tune more LoRA layers at later exits. Similar to the window size in convolutional layers, we define the number of tuned LoRA layers as the LoRA step.

P-LoRA step decision As shown in Figure 15, the optimal healing step varies across exit layers. In general, the larger the n, the greater the per-step healing capacity, due to the increased number of tunable parameters. However, if step 4 is applied to all exits, exits 2 and 3 will miss opportunities for healing. This is acceptable for the top layers, as they already have a strong feature representation from earlier healing. Larger steps benefit later layers by improving convergence performance. For smaller exits, earlier features are still weak and require healing at each exit.

To determine the optimal step during training, we use information from the predicted exit statistics. We set the training step at the pivot of the predicted exit statistics, ensuring that most exits are healed with an appropriate step

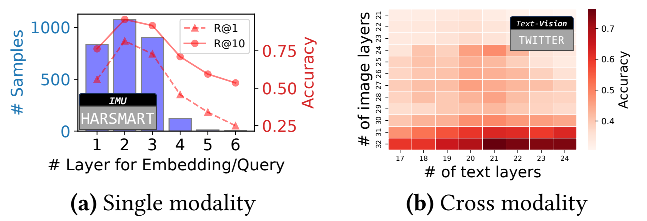


Figure 16. Retrieval accuracy across different embedding granularities, i.e., embeddings generated by different MEM layer exits.

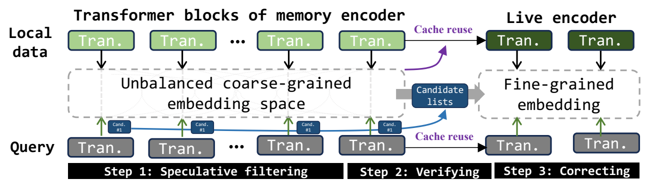


Figure 17. Coarse-grained embeddings are speculatively filtered. The highest-ranking embedding candidates are refined to fine-grained embeddings for final retrieval.

size. This approach prioritizes smaller exits, aligning with the heuristic that most data exits occur at earlier layers, which require more focused healing. At later stages, larger steps enhance fine-grained performance during queries without significantly affecting exit flexibility.

Training Details The healing P-LoRA is designed to be parameter-efficient and highly transferable. Application developers can customize the personalized healing adapter during the testing phase. During deployment, healing occurs iteratively, and embedding granularity can be updated in real time to better fit the data and synchronize with their representations. In this work, for simplicity, the embedding granularity predictor was trained on zero-shot embeddings. The training objective is the fine-grained embedding, not the query embedding. We leave the output layer untuned to mitigate the dynamic embedding mismatch issue. Even without the healing P-LoRA, we demonstrate that Reminisce can still achieve usable retrieval performance.

Speculative Fine-grained Retrieval

With coarse-grained embeddings, we can filter out potential candidates. Further fine-grained embeddings are then processed on these filtered candidates to complete the final retrieval. However, using the default query embedding with a full-capacity encoder does not achieve precise top-1 retrieval (R@1), as shown in Figure 16. This poor performance stems from two unique challenges.

# Challenge 1: Reduced embedding capacity. Even if we modify the model to predict early and align it with the full embedding, exiting early during inference inevitably reduces accuracy compared to full-capacity embedding. Fortunately, while coarse-grained embeddings may not achieve precise top-1 retrieval, they can filter out the most likely candidates when expanding the retrieval range to top-10 as shown in Figure 16a. Thus, this challenge can be alleviated by refining the coarse-grained embeddings filtered with query information.

# Challenge 2: Unbalanced embedding distribution. As described in §, different data exits at different layers, leading to unbalanced embeddings in storage. Although each embedding is fine-tuned to approximate the full embedding, embeddings from different exit layers retain unique characteristics. For example, samples from similar exit layers tend to have similar embedding distributions. As a result, a query embedding from a full-capacity encoder cannot retrieve these embeddings precisely. This phenomenon is shown in Figure 16. For single-modality retrieval on the HARSMART dataset, using the full-capacity MEM to retrieve filtered embeddings results in a top-1 accuracy of only 24.9%, 56.6% lower than using a 2-layer query embedding, since over 99% of samples exit before 3 layers during local embedding. The same phenomenon occurs in the cross-modal TWITTER dataset.

Speculative retrieval Inspired by speculative decoding [52], a popular acceleration technique for language models, we propose feeding the query embedding at different granularities to achieve balanced filtering, as shown in Figure 17. (1) Speculative filtering: The top k candidates at each query granularity are preserved for the second round of filtering. (2) Global verifying: The second round selects the final top k candidates from all granularities. If a sample ID is duplicated, the candidate with the next highest score is preserved. (3) Fine-grained correcting: Finally, the coarse-grained embeddings are refined using the rest of the model to generate fine-grained embeddings, which are then matched with the query for more precise retrieval.

Intermediate results reuse As shown in Figure 14, the coarse-grained embedding can be reused for fine-grained embedding. However, due to the down-sampling structure of the output head, the coarse-grained embedding cannot be directly used for fine-grained embedding. To simplify this, we store the intermediate activations before each down-sample layer. This approach allows reusing the superficial embedding to reduce the cost of data-aware coarse-grained embedding, improving embedding throughput by up to 30%. It also extends the coarse-grained embedding to fine-grained embedding without encoding from scratch, accelerating query latency by up to 70%.

Cache analysis The drawback of this approach is the need to cache intermediate activations. Fortunately, we can quantize them to INT4 and de-quantize them during reuse, which takes significantly less time than re-computation (around 10 ms per embedding). During prediction, the activations can remain in RAM. Once coarse-grained embedding begins, these cached activations replace the intermediate variables typically stored in RAM during embedding, so no additional peak memory footprint is required. After the process ends, the activations are released sequentially. For cache reuse in the fine-grained embedding procedure, the activations are temporarily stored in storage, which is less constrained than RAM, until the query occurs. The loading time is approximately 1 ms for 10 activations. Once an image is queried, it is updated to the fine-grained embedding, and its storage can be freed.

Data Availability

The datasets involved in this study are all publicly available ones. The collected traces are available at https://www.kaggle.com/datasets/dongqicai/mobile-trace-of-viewed-images. The access to other datasets can be found in the Result Experiment Setup section.

Code Availability

Codes for this work are available on a public repository https://github.com/caidongqi/Mobile-Search-Engine/tree/pc. We also provide sufficient details in the Methods and Supplementary Information for implementing experiments in this work.

References

[1] Mengwei Xu, Tiantu Xu, Yunxin Liu, and Felix Xiaozhu Lin, "Video analytics with zero-streaming cameras," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. July 2021, pp. 459–472, USENIX Association.

[2] Michiel De Jong, Yury Zemlyanskiy, Nicholas FitzGerald, Joshua Ainslie, Sumit Sanghai, Fei Sha, and William W Cohen, "Pre-computed memory or on-the-fly encoding? a hybrid approach to retrieval augmentation makes the most of your compute," in *International Conference on Machine Learning*. PMLR, 2023, pp. 7329–7342.

[3] Gautier Izacard and Edouard Grave, "Leveraging passage retrieval with generative models for open domain question answering," in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty, Eds., Online, Apr. 2021, pp. 874–880, Association for Computational Linguistics.

[4] Fengling Li, Lei Zhu, Tianshi Wang, Jingjing Li, Zheng Zhang, and Heng Tao Shen, "Cross-modal retrieval: a systematic review of methods and future directions," *arXiv preprint arXiv:2308.14263*, 2023.

[5] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al., "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[6] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra, "Imagebind: One embedding space to bind them all," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 15180–15190.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] Daliang Xu, Hao Zhang, Liming Yang, Ruiqi Liu, Gang Huang, Mengwei Xu, and Xuanzhe Liu, "Empowering 1000 tokens/second on-device llm prefilling with mllm-npu," *arXiv preprint arXiv:2407.05858*, 2024.

[9] Xiang Li, Zhenyan Lu, Dongqi Cai, Xiao Ma, and Mengwei Xu, "Large language models on mobile devices: Measurements, analysis, and insights," in *Proceedings of the Workshop on Edge and Mobile Foundation Models*, 2024, pp. 1–6.

[10] Jinliang Yuan, Chen Yang, Dongqi Cai, Shihe Wang, Xin Yuan, Zeling Zhang, Xiang Li, Dingge Zhang, Hanzi Mei, Xianqing Jia, et al., "Mobile foundation model as firmware," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2024, pp. 279–295.

[11] Mengwei Xu, Dongqi Cai, Wangsong Yin, Shangguang Wang, Xin Jin, and Xuanzhe Liu, "Resource-efficient algorithms and systems of foundation models: A survey," *ACM Comput. Surv.*, Nov. 2024, Just Accepted.

[12] Eric Fassbender and Wolfgang Heiden, "The virtual memory palace," *Journal of Computational Information Systems*, vol. 2, no. 1, pp. 457–464, 2006.

[13] CNBC, "Apple apologizes for listening to Siri conversations," https://www.cnbc.com/2019/08/28/apple-apologizes-for-listening-to-siri-conversations.html, 2019, Accessed: 2024-09-06.

[14] Nanyi Fei, Zhiwu Lu, Yizhao Gao, Guoxing Yang, Yuqi Huo, Jingyuan Wen, Haoyu Lu, Ruihua Song, Xin Gao, Tao Xiang, et al., "Towards artificial general intelligence via a multimodal foundation model," *Nature Communications*, vol. 13, no. 1, pp. 3094, 2022.

[15] Chunyuan Li, Zhe Gan, Zhengyuan Yang, Jianwei Yang, Linjie Li, Lijuan Wang, Jianfeng Gao, et al., "Multimodal foundation models: From specialists to general-purpose assistants," *Foundations and Trends® in Computer Graphics and Vision*, vol. 16, no. 1-2, pp. 1–214, 2024.

[16] Chameleon Team, "Chameleon: Mixed-modal early-fusion foundation models," 2024.

[17] "Android: Low memory killer dae- mon," https://source.android.com/docs/core/perf/lmkd, 2022.

[18] Rongjie Yi, Xiang Li, and Mengwei Xu, "mllm," https://github.com/UbiquitousLearning/mllm, 2024.

[19] Hui Ni and The ncnn contributors, "ncnn," June 2017.

[20] Duzhen Zhang, Yahan Yu, Chenxing Li, Jiahua Dong, Dan Su, Chenhui Chu, and Dong Yu, "Mm-llms: Recent advances in multimodal large language models," *arXiv preprint arXiv:2401.13601*, 2024.

[21] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[22] Alexandros Kouris, Stylianos I Venieris, Stefanos Laskaridis, and Nicholas D Lane, "Fluid batching: Exit-aware preemptive serving of early-exit neural networks on edge npus," *arXiv preprint arXiv:2209.13443*, 2022.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[24] Aditya Joshi, "Flickr 8k Dataset for Image Captioning," https://www.kaggle.com/datasets/adityajn105/flickr8k, 2020, Accessed: 2024-09-06.

[25] Konstantinos Drossos, Samuel Lipping, and Tuomas Virtanen, "Clotho: An audio captioning dataset," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 736–740.

[26] Mayur Sonawane, Sahil Rajesh Dhayalkar, Siddesh Waje, Soyal Markhelkar, Akshay Wattamwar, and Seema C. Shrawne, "Human activity recognition using smartphones," 2024.

[27] Yuhao Du, Muhammad Aamir Masood, and Kenneth Joseph, "Understanding visual memes: An empirical analysis of text superimposed on memes shared on twitter," in *Proceedings of the International AAAI Conference on Web and Social Media*, 2020, vol. 14, pp. 153–164.

[28] NVIDIA Corporation, "Jetson Orin," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, 2022, Accessed: 2024-09-06.
[29] Raspberry Pi Foundation, "Raspberry Pi 4 Model B," https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, 2019, Accessed: 2024-09-06.
[30] Xiaomi, "Redmi note 12 turbo specifications," https://www.mi.com/redmi-note-12-turbo, 2023, Accessed: 2024-09-06.
[31] Android Developers, "Accessibility Services," https://developer.android.com/guide/topics/ui/accessibility/service, 2024, Accessed: 2024-09-06.
[32] Agus Kurniawan and Agus Kurniawan, "Introduction to nvidia jetson nano," *IoT Projects with NVIDIA Jetson Nano: AI-Enabled Internet of Things Projects for Beginners*, pp. 1–6, 2021.
[33] Edge AI and Vision Alliance, "Is the new nvidia jetson agx orin really a game-changer? we benchmarked it," https://www.edge-ai-vision.com/2022/04/is-the-new-nvidia-jetson-agx-orin-really-a-game-changer-we-benchmarked-it/, 2022, Accessed: 2024-09-06.
[34] Dongqi Cai, Qipeng Wang, Yuanqiang Liu, Yunxin Liu, Shangguang Wang, and Mengwei Xu, "Towards ubiquitous learning: A first measurement of on-device training performance," in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, 2021, pp. 31–36.
[35] Aaron Cahn, Scott Alfeld, Paul Barford, and Shanmugavelayutham Muthukrishnan, "An empirical study of web cookies," in *Proceedings of the 25th international conference on world wide web*, 2016, pp. 891–901.
[36] Rewind AI, "Rewind AI," https://www.rewind.ai, 2023, Accessed: 2024-09-06.
[37] Rewind, "How does rewind compression work?," https://help.rewind.ai/en/articles/6706118-how-does-rewind-compression-work, 2022, Accessed: 2024-09-06.
[38] Alison K Banikowski and Teresa A Mehring, "Strategies to enhance memory based on brain-research," *Focus on Exceptional Children*, vol. 32, no. 2, pp. 1–16, 1999.
[39] Liqiang Lu, Yicheng Jin, Hangrui Bi, Zizhang Luo, Peng Li, Tao Wang, and Yun Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 977–991.
[40] Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W Mahoney, et al., "Full stack optimization of transformer inference: a survey," *arXiv preprint arXiv:2302.14017*, 2023.
[41] Yang Sun, Wei Hu, Fang Liu, Min Jiang, FeiHu Huang, and Dian Xu, "Speformer: An efficient hardware-software cooperative solution for sparse spectral transformer," in *2022 IEEE 9th International Conference on Cyber Security and Cloud Computing (CSCloud)/2022 IEEE 8th International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE, 2022, pp. 180–185.
[42] Giorgos Armeniakos, Georgios Zervakis, Dimitrios Soudris, and Jörg Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–36, 2022.
[43] Hanrui Wang, Zhekai Zhang, and Song Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 97–110.
[44] Niel Lebeck, Arvind Krishnamurthy, Henry M Levy, and Irene Zhang, "End the senseless killing: Improving memory management for mobile operating systems," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 873–887.
[45] ASUS, "ROG Phone 9 Pro," https://rog.asus.com/phones/rog-phone-9-pro/, 2024, Accessed: 2024-12-20.
[46] Stefanos Laskaridis, Alexandros Kouris, and Nicholas D Lane, "Adaptive inference through early-exit networks: Design, challenges and directions," in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, 2021, pp. 1–6.
[47] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al., "Layer skip: Enabling early exit inference and self-speculative decoding," *arXiv preprint arXiv:2404.16710*, 2024.
[48] Xiangjie Li, Chenfei Lou, Yuchi Chen, Zhengping Zhu, Yingtao Shen, Yehan Ma, and An Zou, "Predictive exit: Prediction of fine-grained early exits for computation-and energy-efficient inference," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023, vol. 37, pp. 8657–8665.
[49] Meiqi Wang, Jianqiao Mo, Jun Lin, Zhongfeng Wang, and Li Du, "Dynexit: A dynamic early-exit strategy for deep residual networks," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2019, pp. 178–183.
[50] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts, "The unreasonable ineffectiveness of the deeper layers," *arXiv preprint arXiv:2403.17887*, 2024.
[51] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.
[52] Yaniv Leviathan, Matan Kalman, and Yossi Matias, "Fast inference from transformers via speculative decoding," in *International Conference on Machine Learning*. PMLR, 2023, pp. 19274–19286.
[53] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.
[54] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
[55] Alexey DOSOVITSKIY, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.
[56] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever, "Robust speech recognition via large-scale weak supervision," in *International Conference on Machine Learning*. PMLR, 2023, pp. 28492–28518.
[57] Danfeng Hong, Bing Zhang, Xuyang Li, Yuxuan Li, Chenyu Li, Jing Yao, Naoto Yokoya, Hao Li, Pedram Ghamisi, Xiuping Jia, et al., "Spectralgpt: Spectral remote sensing foundation model," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
[58] Shengkun Tang, Yaqing Wang, Zhenglun Kong, Tianchi Zhang, Yao Li, Caiwen Ding, Yanzhi Wang, Yi Liang, and Dongkuan Xu, "You need multiple exiting: Dynamic early exiting for accelerating unified vision language model," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 10781–10791.
[59] Omar Hamed, Souhail Bakkali, Marie-Francine Moens, Matthew Blaschko, and Jordy Van Landeghem, "Multimodal adaptive inference for document image classification with anytime early exiting," *arXiv preprint arXiv:2405.12705*, 2024.
[60] Soheil Hor and Amin Arbabian, "Sense, predict, adapt, repeat: A blueprint for design of new adaptive ai-centric sensing systems," *arXiv preprint arXiv:2312.07602*, 2023.
[61] Jan-Paul Huttner, Kathrin Robbert, and Susanne Robra-Bissantz, "Immersive ars memoria: evaluating the usefulness of a virtual memory palace," 2019.
[62] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al., "Personal llm agents: Insights and survey about the capability, efficiency and security," *arXiv preprint arXiv:2401.05459*, 2024.

[63] Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu, “Appagent: Multimodal agents as smartphone users,” *arXiv preprint arXiv:2312.13771*, 2023.

[64] Dongfang Liu, Yiming Cui, Zhiwen Cao, and Yingjie Chen, “Indoor navigation for mobile agents: A multimodal vision fusion model,” in *2020 international joint conference on neural networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[65] “Microsoft recall,” https://support.microsoft.com/en-us/windows/retrace-your-steps-with-recall-aa03f8a0-a78b-4b3e-b0a1-2eb8ac48701c, 2024.

[66] “Apple,” https://www.apple.com/newsroom/2023/12/report-2-point-6-billion-records-compromised-by-data-breaches-in-past-two-years/, 2022.

[67] Zhengcong Fei, Xu Yan, Shuhui Wang, and Qi Tian, “Deecap: Dynamic early exiting for efficient image captioning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12216–12226.

[68] Weihao Cui, Han Zhao, Quan Chen, Hao Wei, Zirui Li, Deze Zeng, Chao Li, and Minyi Guo, “{DVABatch}: Diversity-aware {Multi-Entry}{Multi-Exit} batching for efficient processing of {DNN} services on {GPUs},” in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 183–198.

[69] Rongkang Dong, Yuyi Mao, and Jun Zhang, “Resource-constrained edge ai with early exit prediction,” *Journal of Communications and Information Networks*, vol. 7, no. 2, pp. 122–134, 2022.

[70] Fabawi, “Imagebind-lora: Fine-tuning "imagebind one embedding space to bind them all" with lora,” 2023, Accessed: 2024-12-06.

[71] Chunyuan Qin, Chuan Deng, Jiashun Huang, Kunxian Shu, and Mingze Bai, “An efficient faiss-based search method for mass spectral library searching,” in *2020 3rd International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)*. IEEE, 2020, pp. 513–518.