Ubiquitous Memory Augmentation via Mobile Multimodal Embedding System

Dongqi Cai[1,2], Shangguang Wang[1*], Chen Peng[1], Zeling Zhang[1], Zhenyan Lu[1,3], Tao Qi[1], Nicholas D. Lane[2,4*], Mengwei Xu[1*]
[1]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China
[2]Department of Computer Science and Technology, University of Cambridge, Cambridge, UK
[3]Pengcheng Laboratory, Shenzhen, China
[4]Flower Labs, London, UK
[*]Correspondance Emails: sgwang@bupt.edu.cn, ndl32@cam.ac.uk, mwx@bupt.edu.cn

Abstract

Forgetting is inevitable in human memory. Recently, multimodal embedding models have been proposed to vectorize multimodal reality into a unified embedding space. Once generated, these embeddings allow mobile users to quickly retrieve relevant information, effectively augmenting their memory. 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, an efficient on-device multimodal embedding system that enables high-throughput embedding 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.

1 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 24×7, or local network-attached storage 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, 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 exciting types of mobile applications as shown in 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 had been eavesdropping on uploaded user conversations to enhance their Siri model [13]. With cloud-based MEMs, users risk comprehensive life surveillance, with no way to verify.

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 consumption. The slow inference speed, combined with the immense computing power required, results in 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.

Reminisce is an efficient on-device multimodal embedding system. Its key idea is *coarse-grained embedding*, built upon the early-exiting technique. It draws inspiration from the top-down predictions of cognitive brain [17]. Embeddings from early-exited MEMs serve as coarse-grained representations to filter likely candidates during retrieval. These candidates are then refined by the remaining layers at query time for final selection. While early exiting avoids full model execution during memorization, three key system challenges remain on mobile devices (§4.1): low parallelism, limited exiting benefits, and performance degradation. To further promote the practical deployment of Reminisce, we propose three software-hardware co-designs: (1) Data-aware pre-exit predictor (§4.2) is a unified, lightweight early-exit predictor model applicable across all modalities. It facilitates efficient batching and pipeline execution, improving encoding throughput; (2) Progressive LoRA healing (§4.3) retrofits low-rank adaptation (LoRA) [18], a popular parameter-efficient fine-tuning method, to ensure high retrieval performance with earlier exits by progressively increasing shared bottom layers. This enables intermediate results to be cached and reused; (3) Speculative fine-grained retrieval (§4.4). Query embeddings from different exits are used for speculative filtering, with top candidates from each granularity undergoing a second matching round for accurate final retrieval.

Our extensive experiments demonstrate that, with these designs, Reminisce accelerates the multimodal embedding process while ensuring accurate retrieval. We evaluate Reminisce on multiple mobile devices, achieving an average 12.4× improvement in throughput compared to the original MEM. We further conduct a case study using recent Twitter data and a user study based on mobile application traces collected from eight users over one week, demonstrating the practicality of Reminisce in real-world scenarios.

2 Results

2.1 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 [19].

From the device perspective, the service has two runtimes:

- Embedding runtime (Offline remembering in the background). 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, as deep learning models are often too large for mobile devices. This can lead the OS to terminate inference processes. Current mobile inference engines support layerwise execution to accommodate large models [20, 21]. A 1024-dimensional embedding is generated for each item in a unified space.
- Query runtime (Online recall in the foreground). 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.

2.2 Preliminary Measurements

First, we present a preliminary study to demonstrate the utility and efficiency of on-device multimodal embedding 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 [22] to detect and store newly appeared visual content. Images are hashed to include only new content. Images smaller than 100KB are excluded to avoid capturing icons and minor system elements. One collected trace is illustrated in Figure 2a.

MEMs are observed to be contextually expressive. All images and corresponding texts are collected and embedded using ImageBind [6]. By aligning multimodal embeddings into a unifed space, ImageBind can effectively retrieve semantically relevant content from different modalities using human-friendly inputs (Supplementary Figure 2). A numerical analysis of various tasks will be presented in §2.5.

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

Despite their contextually expressive capabilities, the embedding speed is too slow to keep pace with the figures

generated by applications. As shown in Figure 2b, 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 [23] struggles to handle an entertainment task generating 36.3 images per minute. The only exception is the NVIDIA ORIN [24], which performs comparably to a cloud server using an NVIDIA A40 [25]. However, continuously running the CPU or GPU on mobile devices is impractical due to battery depletion.

The heavy embedding workloads and low throughput 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 2c, 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. While GPUs process data faster, they consume more energy than CPUs, making them unsuitable for long-term embedding in the current MEM design.

2.3 System Designs

As shown in Figure 3a, the core design of Reminisce is the coarse-grained embedding, built upon the early-exit mechanism. This approach offloads the computation of the full embedding to the less frequent, intent-specific query phase. Specifically, embeddings generated by early-exited MEMs serve as coarse-grained embeddings to filter the most likely candidates during retrieval queries. These candidates are further refined by the remaining layers of the exited MEMs at query time to ensure accurate retrieval. We propose and prototype this mobile-friendly early-exit system for efficient multimodal embedding. Three hardware-software co-design optimizations further enhance the performance of Reminisce, making it practical for mobile devices.

The first optimization is data-aware pre-exit prediction (§4.2). Traditional early-exit methods determine exits at the end of each branch computation, causing inconsistent workloads and memory fragmentation [26], and existing predictive models for CNNs cannot effectively scale to multimodal embedding models due to their convolution-specific design [27, 28]. Our observation is that different data inherently carry varying amounts of information (Supplementary Figure 4a), and intermediate multimodal embeddings provide effective cues for determining optimal exit points (Supplementary Figure 4b). Based on this unique observation, we propose a unified, lightweight early-exit predictor that leverages these intermediate embeddings to preemptively determine the exit layer, enabling batch scheduling for improved parallelism and amortizing loading times (Figure 3b).

The second optimization is progressive LoRA healing (§4.3). Previous early-exit healing approaches [29] utilize LoRA [18] to fine-tune NLP models for earlier exits. However, these methods fine-tune separate LoRA modules for each exit, preventing the reuse of intermediate results and thereby negating early-exit benefits on mobile devices. As illustrated in Figure 3c, we propose sharing previously tuned parameters, reducing the number of layers required per token and enabling reuse of intermediate activations. Based on our observation that sharing LoRA weights at top layers is more effective (Supplementary Figure 5), we propose a progressive LoRA healing method that incrementally increases tuning depth (number of shared layers) at later exits to minimize performance degradation from shared LoRA weights.

The third optimizations is speculative fine-grained retrieval (§4.4). Using a full-capacity encoder to generate query embeddings leads to unbalanced retrieval performance when matched with coarse-grained embeddings, resulting in poor top-1 retrieval accuracy (Supplementary Figure 6). To address this issue, we introduce a speculative fine-grained retrieval mechanism (shown in Figure 3d) to balance the retrieval process. It first performs speculative filtering using query embeddings at all granularities and then refines the selection through a second, fine-grained matching stage.

2.4 Experimental Setup

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 [30].

We compare Reminisce to the following alternatives: (1) Multimodal Embedding Model (MEM) without any optimization. (2) BranchyNet [26], using a traditional early-exit mechanism. (3) Fluid Batching [31], an early-exit-aware batching algorithm that allows sample preemption at runtime. For completeness, we also include a naive baseline using monolithic model, i.e., without layer-wise execution, though it incurs nearly unaffordable memory footprint on certain mobile devices. For a fair comparison, all baselines are equipped with ImageBind fine-tuned for the downstream task.

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 phase. (5) Memory Usage: Peak memory footprint during the embedding phase.

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 fine-tuned 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 fine-tuned 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.

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 [32], 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.

We evaluate Reminisce on the NVIDIA ORIN (ORIN) [24], Jetson TX2 (TX2) [33], Raspberry Pi 4B (RPI4B) [34], and a flagship smartphone with Qualcomm Snapdragon 8Gen3 (8GEN3) [35]. The default operating mode for ORIN is MAXQ, which is the most cost-effective mode with four large cores disabled. For the Jetson TX2, we select the MAXN mode, the most powerful mode available, to fully utilize GPU computing power. To reduce memory consumption, we quantize the model to INT4 precision for the 8GEN3 smartphone and INT8 precision for ORIN, TX2, and RPI4B. Please refer to Supplementary §G for more implementation details about hardware specification, executing mode specifications and quantization. Reminisce runs on the GPU for the ORIN and TX2 boards. For the RPI4B and the 9GEN3 smartphone, Reminisce runs on the CPU due to the lack of CUDA support. Current mobile inference engines cannot effectively utilize GPUs for MEM execution [9, 20, 36].

2.5 Evaluation Statement

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?

2.6 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.

Reminisce achieves an order of magnitude improvement in throughput. Figure 4 shows that Reminisce can achieve a 12.4× 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-audio retrieval task CLOTO on Jetson ORIN, Reminisce achieves a 45× throughput improvement with less than 3% relative accuracy loss under the default query latency budget of 1.5s (further analyzed in §2.8).

Regarding stronger baselines, Fluid Batching introduces a early-exit-aware batching mechanism, achieving a 3× throughput improvement over the naive early-exiting baseline BranchyNet and 6× over MEM under the layer-wise inference setting. However, Reminisce still outperforms Fluid Batching across all datasets, providing up to 2.4× 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.

2.7 Significance of Key Designs

As illustrated in Figure 5a, 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 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.

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 5a, compared to exiting all samples at a fixed layer, the data-aware pre-exit mechanism improves retrieval accuracy by up to 19.8%. The higher coarse-grained retrieval performance is crucial for final fine-grained retrieval.

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.

2.8 Impact of Query Latency Tolerance

Although query costs are negligible compared to embedding costs in the long run—since queries occur less frequently than continuous daily embeddings—they are immediately noticeable to users. Thus, we illustrate Reminisce's performance under different query latency tolerance in Figure 5b. During queries, the device holds the entire quantized model in memory without layer-by-layer loading. Given the infrequency of queries, the temporary memory increase is acceptable. Query latency comprises three components: query embedding, matching, and fine-grained embedding. Baseline methods with memory encoders require only the first two steps, typically taking around 1.2 seconds. Reminisce takes less than 1.5 seconds (the default latency budget used in §2.6) to achieve acceptable query accuracy. As shown, if the system tolerates higher query delays, performance can be further enhanced. For example, on the FLICKR dataset, the relative retrieval accuracy of Reminisce improves from 92% to 99% after refining an additional 10 candidates (≈0.2s).

Additionally, similar to web cookies [37], the query process can skip the complex fine-grained embedding when repeated, improving efficiency in multi-query scenarios where frequently queried items are retrieved faster. Once a local embedding is queried, its embedding is permanently upgraded. Under these conditions, the system becomes more efficient by skipping the fine-grained embedding process for frequently queried items.

2.9 System Cost

Figure 6 shows the normalized energy consumption of Reminisce and various baselines. Reminisce reduces energy consumption by up to 29× and 20× on average compared to layerwise-executed baselines. Even compared to naive MEM without layerwise execution, Reminisce still achieves up to 7× 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.

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 §2.1, 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 [38] consumes 14GB of storage per month on average, as officially reported [39].

2.10 Case Study: Twitter Meme Retrieval

To demonstrate the practicality of Reminisce in real-world scenarios, we conducted a case study using daily surfing images and captions collected from Twitter memes. End users filtered the data to ensure privacy, and a total of 805 figures were collected to simulate 30 minutes of surfing. Our evaluation compares multiple methods—including Naive MEM without layer-wise execution, the MEM baseline, BranchyNet, Fluid Batching, and our Reminisce —in terms of throughput, energy, memory, and retrieval accuracy.

As shown in Figure 7, all baseline methods take over 80 minutes to complete the retrieval task on a fully utilized CPU. Naive MEM incurs a large memory footprint by loading the entire model at once, even with INT4 quantization. Its layer-wise execution counterpart (MEM baseline) reduces memory usage but decreases throughput due to frequent layer-switching overhead. BranchyNet improves throughput by skipping layers but at the expense of lower accuracy. In contrast, Reminisce completes the same task in 28 minutes—achieving a 3× throughput improvement compared to even the strong baseline Fluid Batching, due to our mobile-friendly optimizations.

Our approach reduces peak memory usage by 7× compared to Naive MEM, lowering the footprint below 200MB. This includes a small buffer (under 50MB) for pipelined execution and temporary activations—a reasonable tradeoff for performance gains. Energy consumption is reduced by up to 4×, enabled by fewer layer computations and more efficient batching. The system also achieves higher retrieval accuracy than naive early-exit methods while maintaining an acceptable query latency of just 0.5 seconds. The additional memory overhead from batching parallelism is justified by the substantial performance improvements.

These quantitative improvements—from faster processing and lower resource consumption to robust retrieval performance—demonstrate that Reminisce is highly practical for deployment in mobile scenarios, where computational efficiency and low-latency requirements are critical.

2.11 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. 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. As shown in Figure 8, without Reminisce, the naive MEM system (in 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.

3 Discussion

In this work, we develop Reminisce, an efficient on-device multimodal embedding system to function as a memory augmenting service. Extensive experiments and case studies demonstrate that Reminisce 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 [40]. Different from advanced sparsification or quantization optimizations, which provides little to no benefit during inference due to the limited support of mobile hardware [41–45], 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 mobile 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 mitigates risks associated with data breaches and unauthorized access, addressing a critical concern in modern 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 Supplementary Figure 3. Fortunately, it is still within a practical range, e.g., 82M for embedding IMU information, which is below the average Android application memory consumption of 100M as reported in 2020 [19, 46]. After 5 years, the mobile RAM capacity has increased significantly, with up to 24GB available on high-end devices [47]. Less than 200MB 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 the challenges of low embedding throughput and high energy consumption.
- We introduce Reminisce, an efficient on-device multimodal embedding system that addresses these challenges. Reminisce incorporates three 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 mobile devices.

4 Methods

4.1 Reminisce Overview

In this work, we develop Reminisce, an efficient on-device multimodal embedding system to address the challenges outlined above. Reminisce is designed to minimize embedding energy costs and query latency while maximizing throughput and achieving near state-of-the-art retrieval accuracy. Additionally, Reminisce shall integrate easily into off-the-shelf mobile applications to enhance user experience without requiring complex hardware modifications. Lastly, Reminisce aims to be both versatile and transferable across a wide range of tasks. To achieve these goals, we leverage early exit, a widely studied optimization technique, as the backbone of our system.

Early Exiting is the key building block. It terminates the computation of a deep neural network at an intermediate layer based on prediction confidence. Typically, a prediction head is introduced at the end of each layer to serve as a separate exit branch, allowing samples to be correctly classified at the earliest possible layer.

We choose early exit as the backbone of Reminisce because it aligns with our design principles: (1) Early exit is mobile hardware-friendly: it requires no sparsification kernel compilation and integrates easily into existing multimodal embedding applications. Most mobile devices do not fully support advanced sparsification or quantization optimizations, providing little to no benefit during inference [41–45]. (2) Early exit preserves the raw structure of MEMs, maintaining their generalization capacity while bypassing only downstream alignment. Additionally, early exit is caching-friendly, as the top layers share the same bottom weights with the exited layers, allowing intermediate activations to be reused and reducing duplicated computations. Other techniques like pruning and quantization cannot fully leverage the intermediate computation of coarse-grained embeddings. This reduction is crucial for Reminisce, as it eliminates redundant forward passes, accelerating both embedding and query phases, which we discuss in §4.5. (3) Compared to quantization, early exit offers a broader trade-off space. As shown in our experiments (Supplementary Figure 4a), easy inputs require only one layer (just 3% of total computation)

to achieve accurate results. Such a large reduction in cost is not possible with quantization.

As shown in Figure 3a, 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 (§4.3). 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 (§4.2). After pre-exit, samples with the same exits are batched and processed layer by layer 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 (§4.4).

In short, we offload the full-sized embedding cost to the query phase, which is infrequent and carries precise retrieval information [2]. This mirrors the human brain, which retains key information in long-term memory and recalls details only when necessary [40]. Retrieval accuracy and latency are sacrificed within acceptable limits to significantly reduce embedding costs, as demonstrated in §2.5.

While early exit reduces computational load, its application in mobile MEMs introduces several unique challenges: (1) *Low parallelism:* Early exit is incompatible with batching, as all samples in a batch must exit before processing the next [26]. This reduces throughput on mobile devices with limited computational resources. Without batching, it is also harder to amortize loading costs, further slowing layer-wise inference. (2) *Limited benefits:* MEMs are not naturally designed for early prediction and tend to distribute computation across all layers. For instance, ImageBind's 32-layer vision module requires an average of 21.4 layers to process data, limiting computation savings to 33.1%. MEMs need to reduce the layers required for token prediction and minimize computational resources spent on hesitant or fluctuating predictions. (3) *Performance degradation:* Despite thorough training of exit branches and predictors, some samples may exit too early, leading to degraded search performance. This is especially problematic in MEMs, where incorrect embeddings can disrupt the unified embedding space, causing unbalanced distributions and inaccurate retrieval.

4.2 Design 1: Data-aware Pre-exit Predictor

Traditionally, most early-exit methods decide whether to exit at the end of each branch computation [26, 48, 49]. This approach limits hardware acceleration and batching, as exit points vary by data, leading to inconsistent workloads within batches and memory fragmentation [26–28]. Although some predictive models for CNNs [27] 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.

Different data contains varying amounts of information content (Supplementary Figure 4). 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 $\mathtt{F}_x$ for each data point $x \in \mathtt{X}$ 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 $\mathtt{C}_x^i$ 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 $\mathtt{F}_x$ and $\mathtt{C}_x^i$ becomes the largest among $\mathtt{F}_x$ and $\mathtt{C}_X^i$. query retrieves $\mathtt{C}_x^i$ from $\mathtt{C}_X^i$ 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 shown in Figure 5a.

As shown in Figure 3b, 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.

4.3 Design 2: 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.

Previous early-exit healing approaches [29] use the parameter-efficient fine-tuning method, LoRA [18], 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 drawback: the embedding from layer n cannot be reused to compute the embedding for layer $n+1$. As illustrated in Figure 3c, this occurs because LoRA $l_n^{1,\ldots,n}$ for layer n is not the same as the first n layers of LoRA $l_{n+1}^{1,\ldots,n+1}$. 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.

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 (Supplementary Figure 5b). 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.

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 size (Supplementary Figure 5a). 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 (Supplementary Figure 5b).

4.4 Design 3: 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 (Supplementary Figure 6a). 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 Supplementary Figure 6a. Thus, this challenge can be alleviated by refining the coarse-grained embeddings filtered with query information.

Challenge 2: Unbalanced embedding distribution. 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. 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 (Supplementary Figure 6).

Inspired by speculative decoding [50], a popular acceleration technique for language models, we propose feeding the query embedding at different granularities to achieve balanced filtering, as shown in Figure 3d. (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.

4.5 Cache Reuse and Invalidation

As shown in Figure 3, coarse-grained embeddings can be reused for fine-grained embeddings. However, due to the down-sampling structure in the output head, it cannot be reused directly. To address this, we store intermediate activations prior to each down-sampling layer. This approach allows coarse-grained embeddings to be reused without recomputation, reducing query latency by up to 70%. We also reuse superficial embeddings to lower the cost of data-aware coarse-grained embedding, improving embedding throughput by up to 30%.

To efficiently manage intermediate activations and avoid resource waste from stale data, we adopt a cache invalidation strategy as shown in Figure 9. During offline embedding phase, intermediate activations from superficial embeddings are temporarily stored in RAM to compute coarse-grained embeddings. After each batch, these cached activations are sequentially invalidated from RAM. Coarse-grained intermediate activations are subsequently stored on disk, which has fewer constraints compared to RAM (see Supplementary §F for details). At query phase, cached embeddings matching the incoming query are loaded to compute fine-grained embeddings and are promptly invalidated afterward.

Data Availability

The datasets involved in this study are all publicly available and can be accessed as follows: The COCO dataset used in this study are available in the COCO database under accession code https://cocodataset.org/#download. The FLICKR dataset used in this study are available on Kaggle under accession code https://www.kaggle .com/datasets/adityajn105/flickr8k. The CLOTHO dataset used in this study are available on Zenodo under accession code https://zenodo.org/records/349 0684. The HARSMART dataset used in this study are available in the UCI database under accession code

https://archive.ics.uci.edu/ml/machine-learning-databases/00364/dataset_u ci.zip. The collected Twitter meme dataset have been deposited on Kaggle under accession code https://www.kaggle.com/datasets/penguin0211/twitter-dataset-for-mobile-search.

The collected traces in this study have been deposited on Kaggle under accession code https://www.kaggle. com/datasets/dongqicai/mobile-trace-of-viewed-images. All user data used in this study were anonymized prior to analysis. Personally identifiable information such as names and device identifiers were removed following standard anonymization protocols. The resulting dataset contains only abstracted behavioral features (e.g., app usage timestamps, total ImageView count, and image view throughput per app) that cannot be linked back to individuals. All participants provided informed consent prior to data collection. Each participant was informed about the purpose of the study, the type of data collected, the anonymization procedure, and their rights to withdraw at any time.

Furthermore, the open-sourced multimodal embedding models utilized in this paper can be accessed via the following links: ImageBind (https://dl.fbaipublicfiles.com/imagebind/imagebind_huge.pth) and CLIP-b/16 (https://huggingface.co/openai/clip-vit-base-patch16).

Code Availability

Codes for this work are available at [51]: https://github.com/caidongqi/Mobile-Search-Engine/tree/pc. We also provide sufficient details in the methods section and supplementary information for replicating 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)*, pages 459–472. USENIX Association, July 2021.

[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*, pages 7329–7342. PMLR, 2023.

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

[4] Tianshi Wang, Fengling Li, Lei Zhu, Jingjing Li, Zheng Zhang, and Heng Tao Shen. Cross-modal retrieval: A systematic review of methods and future directions. *Proceedings of the IEEE*, 112(11):1716–1754, 2024.

[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*, pages 8748–8763. PMLR, 2021.

[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*, pages 15180–15190, 2023.

[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*, 30, 2017.

[8] Daliang Xu, Hao Zhang, Liming Yang, Ruiqi Liu, Gang Huang, Mengwei Xu, and Xuanzhe Liu. Fast on-device llm inference with npus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, page 445–462, New York, NY, USA, 2025. Association for Computing Machinery.

[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*, pages 1–6, 2024.

[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*, pages 279–295, 2024.

[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 Computing Surveys*, 57(5):1–39, 2025.

[12] Eric Fassbender and Wolfgang Heiden. The virtual memory palace. *Journal of Computational Information Systems*, 2(1):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*, 13(1):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*, 16(1-2):1–214, 2024.

[16] Chameleon Team. Chameleon: Mixed-modal early-fusion foundation models. *arXiv preprint arXiv:2405.09818*, 2024.

[17] Kestutis Kveraga, Avniel S Ghuman, and Moshe Bar. Top-down predictions in the cognitive brain. *Brain and cognition*, 65(2):145–168, 2007.

[18] 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.

[19] Android: Low memory killer daemon. https://source.android.com/docs/core/perf/lmkd, 2022.

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

[21] NCNN authors. NCNN. https://github.com/Tencent/ncnn, 2024.

[22] Android Developers. Accessibility Services. https://developer.android.com/guide/topics/ui/accessibility/service, 2024. Accessed: 2024-09-06.

[23] Agus Kurniawan and Agus Kurniawan. Introduction to nvidia jetson nano. *IoT Projects with NVIDIA Jetson Nano: AI-Enabled Internet of Things Projects for Beginners*, pages 1–6, 2021.

[24] NVIDIA Corporation. Jetson Orin NX 16GB. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, 2022. Accessed: 2024-09-06.

[25] 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.

[26] 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)*, pages 2464–2469. IEEE, 2016.

[27] 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)*, pages 178–183. IEEE, 2019.

[28] 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*, volume 37, pages 8657–8665, 2023.

[29] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Dan Roberts. The unreasonable ineffectiveness of the deeper layers. In *The Thirteenth International Conference on Learning Representations*, 2025.

[30] 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.

[31] 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.

[32] 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*, volume 14, pages 153–164, 2020.

[33] NVIDIA Corporation. Jetson TX2. https://developer.nvidia.com/embedded/jetson-tx2, 2017. Accessed: 2024-09-06.

[34] Raspberry Pi Foundation. Raspberry Pi 4 Model B. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, 2019. Accessed: 2024-09-06.

[35] Xiaomi. Redmi turbo 3 specifications. https://www.mi.com/prod/redmi-turbo-3, 2023. Accessed: 2025-03-10.

[36] 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*, pages 31–36, 2021.

[37] 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*, pages 891–901, 2016.

[38] Rewind AI. Rewind AI. https://www.rewind.ai, 2023. Accessed: 2024-09-06.

[39] Rewind. How does rewind compression work? https://help.rewind.ai/en/articles/6706118-how-does-rewind-compression-work, 2022. Accessed: 2024-09-06.

[40] Alison K Banikowski and Teresa A Mehring. Strategies to enhance memory based on brain-research. *Focus on Exceptional Children*, 32(2):1–16, 1999.

[41] 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*, pages 977–991, 2021.

[42] 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.

[43] 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)*, pages 180–185. IEEE, 2022.

[44] Giorgos Armeniakos, Georgios Zervakis, Dimitrios Soudris, and Jörg Henkel. Hardware approximate techniques for deep neural network accelerators: A survey. *ACM Computing Surveys*, 55(4):1–36, 2022.

[45] 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)*, pages 97–110. IEEE, 2021.

[46] 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)*, pages 873–887, 2020.

[47] ASUS. ROG Phone 9 Pro. https://rog.asus.com/phones/rog-phone-9-pro/, 2024. Accessed: 2024-12-20.

[48] 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*, pages 1–6, 2021.

[49] 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.

[50] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[51] Dongqi Cai, Shangguang Wang, Chen Peng, Zeling Zhang, Zhenyan Lu, Tao Qi, Nicholas D. Lane, and Mengwei Xu. Ubiquitous memory augmentation via mobile multimodal embedding system. GitHub Repository: https://github.com/caidongqi/Mobile-Search-Engine/tree/pc, DOI: https://doi.org/10.5281/zenodo.15379675, 2025.

[52] 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*, pages 740–755. Springer, 2014.

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

[54] 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)*, pages 736–740. IEEE, 2020.

[55] Erhan Bulbul, Aydin Cetin, and Ibrahim Alper Dogru. Human activity recognition using smartphones. In *2018 2nd international symposium on multidisciplinary studies and innovative technologies (ismsit)*, pages 1–6. IEEE, 2018.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under grant numbers 62425203 (S.W.) and 62032003 (S.W.); the Royal Academy of Engineering via DANTE (N.D.L.); the European Research Council through the REDIAL project (N.D.L.); SPRIND under the Composite Learning Challenge (N.D.L.); the Google Academic Research

Award (N.D.L.); and the CCF-Sangfor “Yuanwang” Research Fund (M.X.).

Author Contributions Statement

D.C. conceived the idea, designed the system, and led the implementation and evaluation. S.W., M.X., and N.D.L. jointly supervised the project and provided high-level guidance. C.P., Z.Z. contributed to system development and conducted comprehensive experiments. Z.L. contributed to the quantization experiments. T.Q. supported the revision experimental designs. All authors discussed the results and contributed to writing and revising the manuscript.

Competing Interests Statement

The authors declare no competing interests.

Dataset	Modality	Size	Metric
COCO [52]	Text-**Image**	123,287	R@5
FLICKR [53]	Text-**Image**	8,091	R@1
CLOTHO [54]	Text-**Audio**	3,938	R@10
HARSMART [55]	**IMU**	10,299	Acc.

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

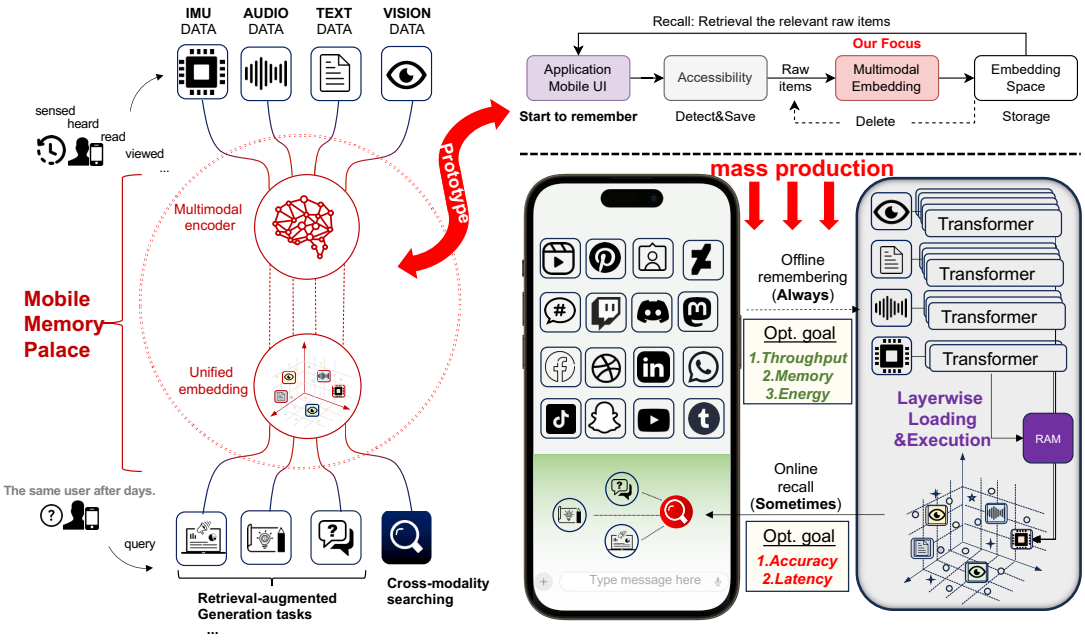

Figure 1. MEM-based ubiquitous memory palace workflow and its instantiation on mobile devices.

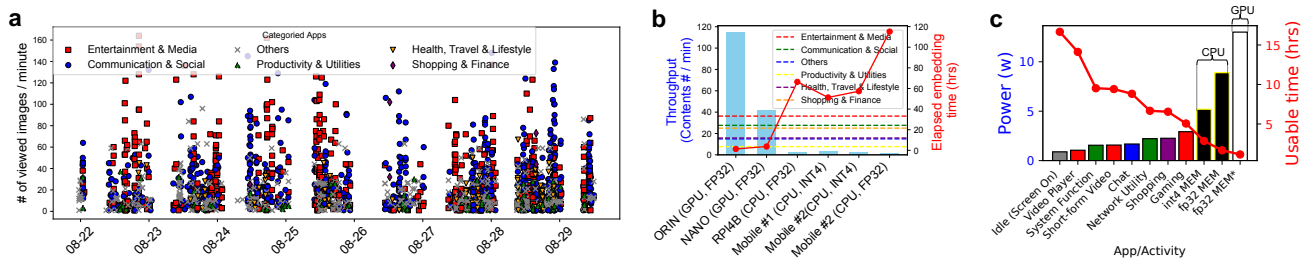

Figure 2. Motivations and challenges of multimodal embedding on mobile devices. (a) Viewed-image traces from one mobile user. (b) MEM inference speeds across different devices, compared to the average image viewing rates of common mobile applications. (c) MEMs rapidly drain mobile batteries. * indicates testing performed on the GPU of the Jetson ORIN.

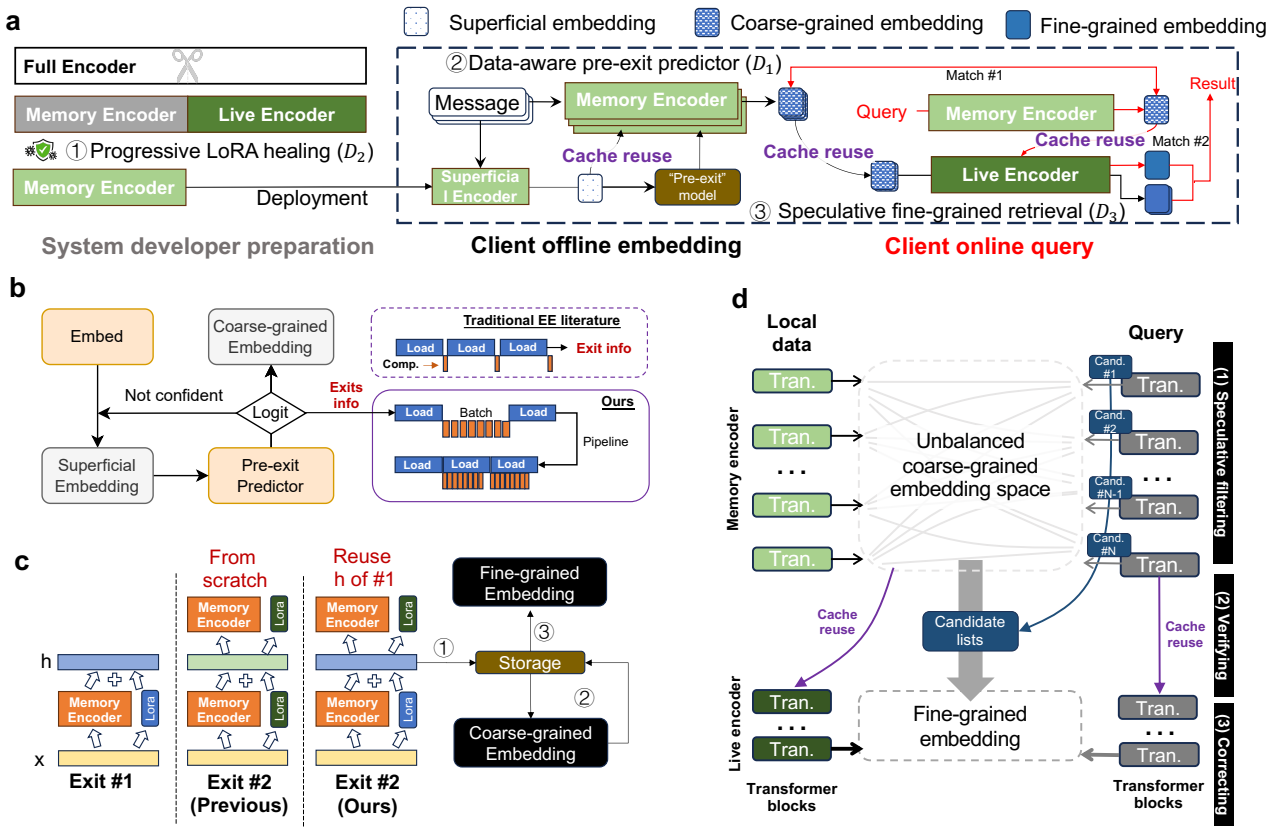

Figure 3. Illustrations of the proposed Reminisce. (a) Detailed workflow of Reminisce with system Designs$_{1,2,3}$. (b) Illustration of Design 1: Data-aware pre-exit predictor and its advantages over traditional early-exit approaches. (c) Illustration of Design 2: Comparison of our progressive LoRA approach to previous methods. (d) Illustration of Design 3: Coarse-grained embeddings are speculatively filtered, and top-ranking candidates are refined into fine-grained embeddings for final retrieval.

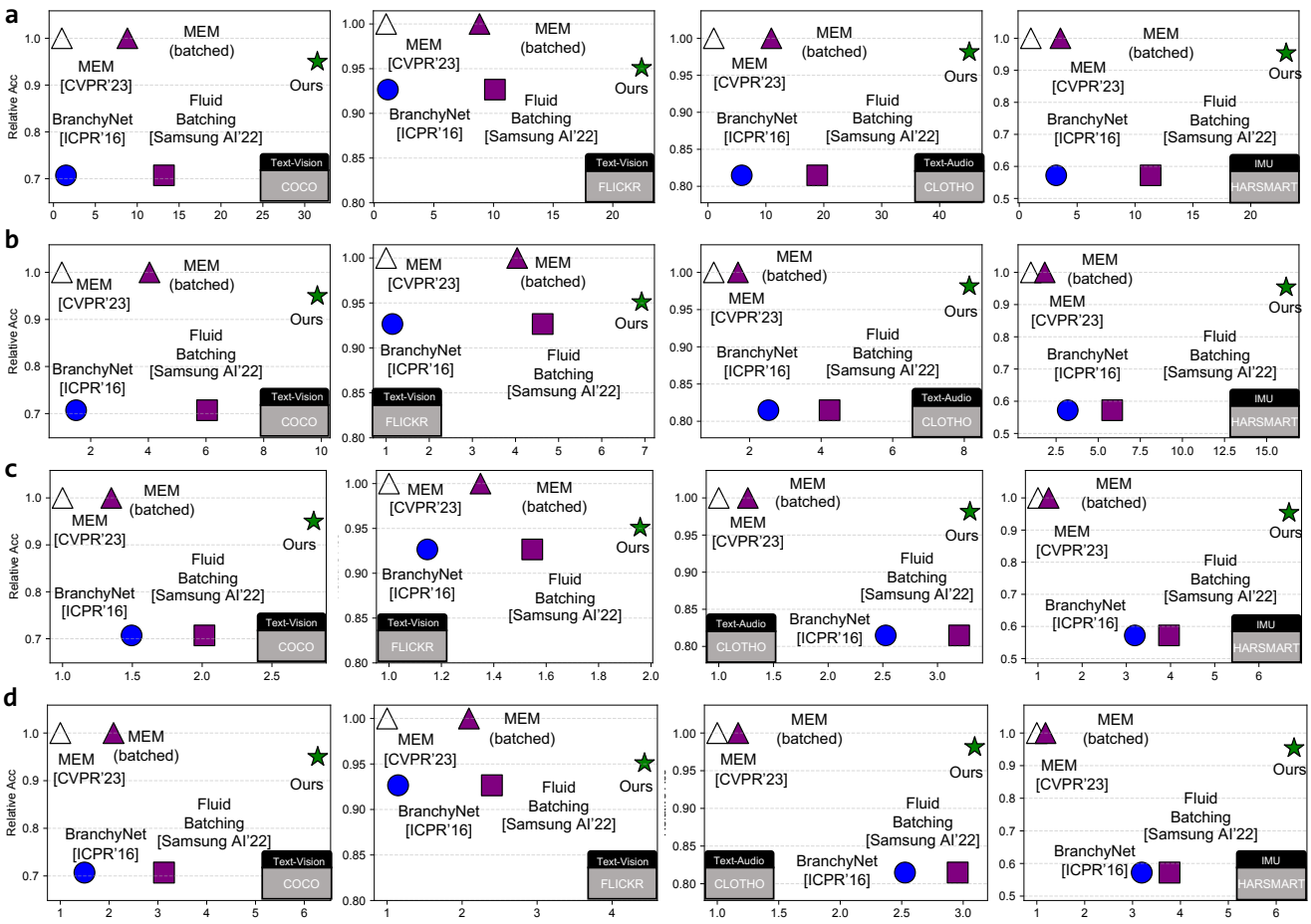

Figure 4. Illustrations of throughput versus accuracy across different methods and devices. (a) Jetson Orin (INT8). (b) Jetson TX2 (INT8). (c) Raspberry Pi 4B (INT8). (d) 8Gen3 Smartphone (INT4). For fairness, only layerwise baselines are included.

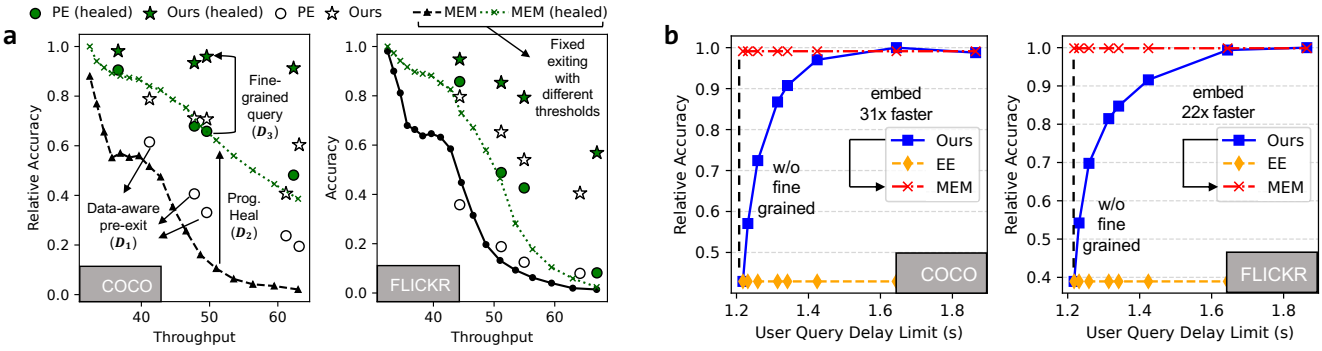

Figure 5. Performance analysis of Reminisce's key designs and query latency impact on ORIN (INT8). (a) Throughput-to-accuracy trade-off with and without Reminisce's key Designs$_{(1,2,3)}$. PE refers to pre-exited coarse-grained embeddings without fine-grained upgrading during the query phase. (b) Performance under different query latency tolerance.

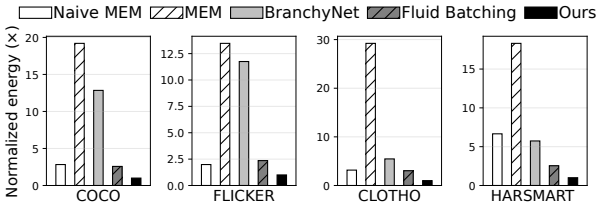

Figure 6. Comparison of energy consumption for different methods. Device: ORIN (INT8).

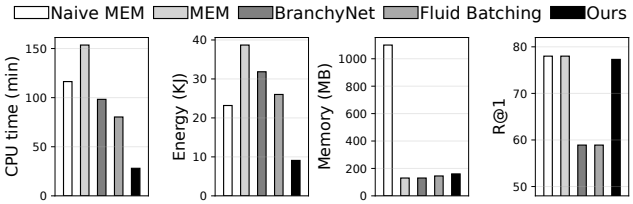

Figure 7. Performance analysis during 30 minutes of Twitter browsing. Device: 8GEN3 (INT4).

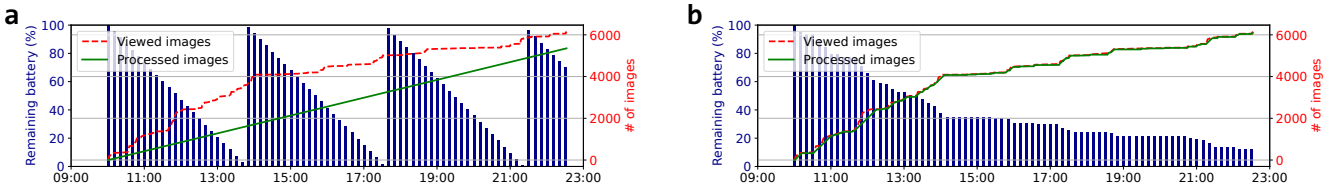

Figure 8. Energy and throughput comparison of embedding images viewed under real mobile traces. (a) Naive MEM. (b) Ours. Device: 8GEN3 (INT4).

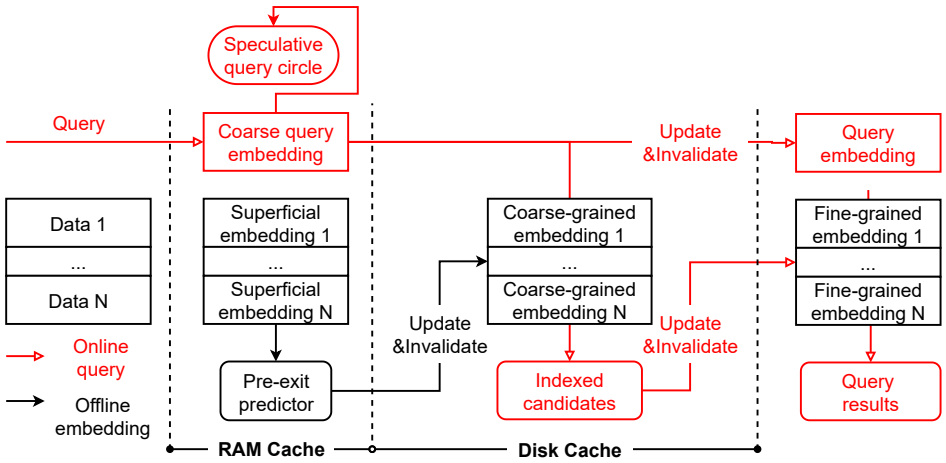

Figure 9. Invalidation strategy of Reminisce.