Appendix

User study on viewed images

We collect viewed images of daily mobile applications from 8 volunteers, ranging from 20 to 52 years old. Uiautomator was executed at the backward to dump daily android UI page (stored in xml file format) for final analysis.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<hierarchy rotation="0">
    <node>
        ...
        <node
            index="1"
            text="Brita Kettle..."
            resource-id="com.xingin.xhs:id/cy7"
            class="android.widget.ImageView"
            package="com.xingin.xhs"
            content-desc="product photo"
            checkable="false"
            checked="false"
            clickable="true"
            enabled="true"
            focusable="true"
            focused="false"
            scrollable="false"
            long-clickable="false"
            password="false"
            selected="false"
            bounds="[0,114][78,213]"
        />
        ...
    </node>
</hierarchy>
```

Top-10 applications generating the most viewed images are displayed to show the user trace.

The 'ImageView' elements are detected to monitor whether there is the new figures. Relevant images and their description are saved in storage for future retrieval. Each imageview xml element group is hashed to only include newly appeared images. More collected traces are illustrated in Figure 18.

Memory issue and loading latency

ImageBind model weights are around 1.5GB, which is too large for mobile devices. This causes the application to be easily killed by the OS due to memory pressure [17]. as shown in Figure 19. Storing the complete model weights for each modality is memory-intensive. For example, the ImageBind model weights are around 1.5GB, which is too large for mobile devices. This causes the application to be easily killed by the OS due to memory pressure. Even after the model is quantized to INT4, it still occupies more than 0.2GB of memory for the weights, not to mention the intermediate activations. This causes the application to be easily killed by the OS due to memory pressure.

One common solution is to load the model layer by layer and sequentially remove the weights from the memory after finishing relevant operations. It can save memory by up to 10× for executing MFM inference. However, the loading latency for each transformer block is around 0.27s, which is 7× larger than the encoding time of each block (around 0.04s for one image). This leads to a significant increase in the encoding time, as shown in Figure 19b.

`Reminisce` is designed to address these issues by predicting the optimal number of layers to load and embedding for each sample, reducing memory usage and optimizing the encoding pipeline to hide the loading latency. Compared to naive `MEM`[6], `Reminisce` can reduce memory usage by up to 7.7×.

Background, Applications and Related Works

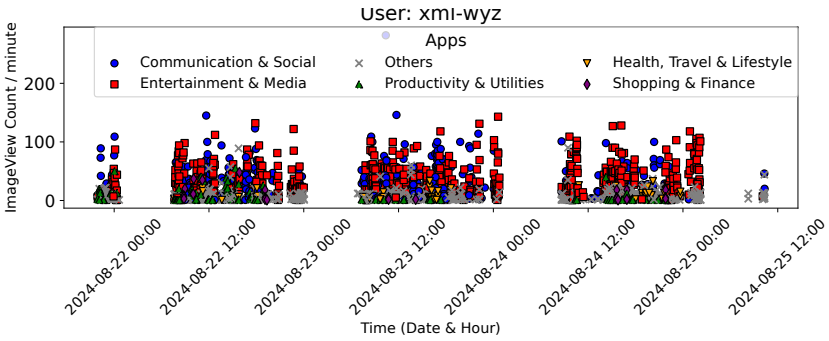
Unified multimodal embedding Embedding was initially proposed to vectorize text data for understanding similarities between different texts [53]. Large language models use embedding layers to generate text embeddings [7, 54]. Similarly, vision, audio, and sensor data can also be transformed into vectorized embeddings [55–57]. However, embedding methods focused on a single modality cannot access information across different modalities due to the gap between their embedding spaces.

To bridge this gap, multimodal embedding models (MEMs) have been developed to unify different modalities into a single embedding space, enhancing the model's ability to understand and bind multimodal inputs. CLIP [5] aligns text and vision by jointly training on image-text pairs, using contrastive learning to map both modalities into a shared space while maintaining their distinction through a dual-tower architecture. ImageBind [6] extends this to align six modalities, including text, vision, audio, depth, thermal, and IMU readings. Each modality is processed by a separate encoder, and the embeddings are fused in a multimodal head to generate a unified embedding. ImageBind demonstrates strong zero-shot classification and retrieval performance across these modalities, matching or outperforming single-modality models. This is achieved through training on large-scale multimodal data.

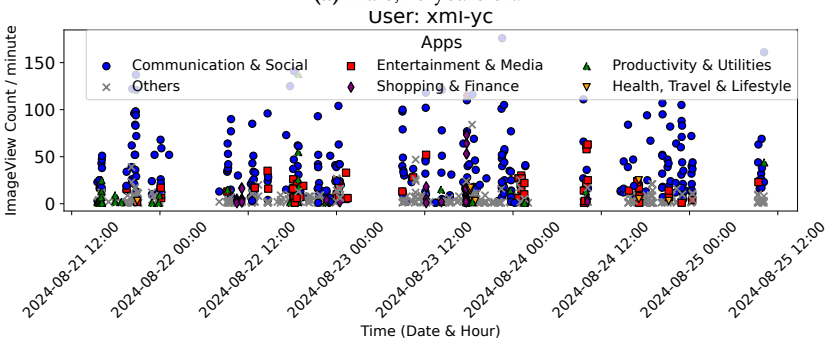
According to Zhang et al. [20], over 80% (35 out of 43) of multimodal foundation models utilize ImageBind or its subset, CLIP, as their modality encoder. This widespread adoption highlights the efficiency and effectiveness of ImageBind in managing multimodal data. Further optimizations have enhanced the fusion of vision and text [58, 59], as well as the fusion of dynamic sensing [60]. However, these methods do not address new modalities in open-set recognition, as handled by ImageBind. For the first time, we enable efficient multimodal embedding in a unified space with usable search accuracy on mobile devices.

Multimodal mobile applications MEMs optimize alignment between high-quality representations across modalities. As such, multimodal information can be composed to enable a rich variety of mobile context-aware applications. For example, MEMs could embed visual, audio, text and sensor data experienced on a mobile device into a personalized memory palace [12, 61]. Whenever users want to recall a specific

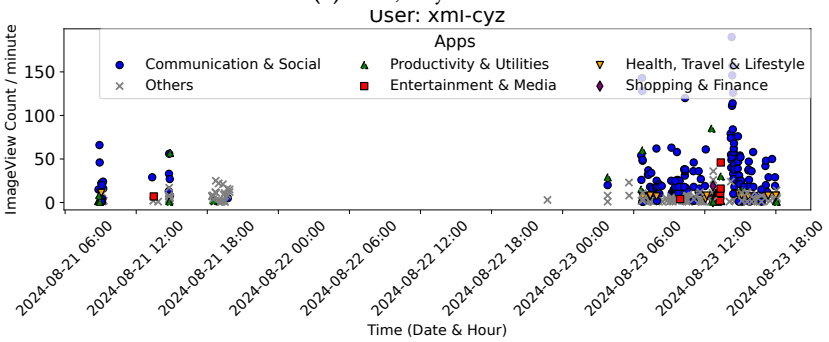
[6]Memory footprint of naive `MEM` is tested in a memory-unlimited server environment.



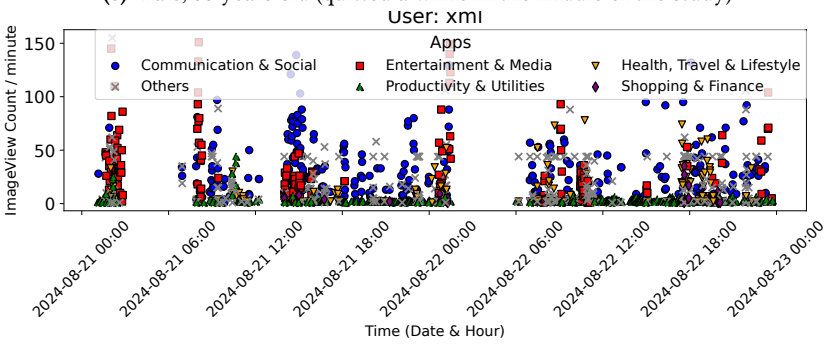
(a) Male, 23 years old



(b) Male, 25 years old

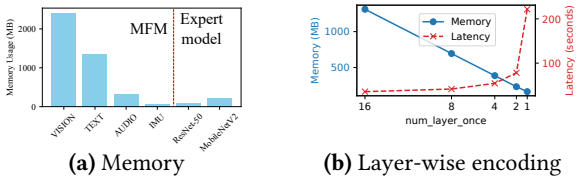


(c) Male, 55 years old (quitted a while in the middle of the study)



(d) Female, 24 years old.

Figure 18. Visualization for part of collected traces of viewed images of daily mobile applications.



(a) Memory **(b)** Layer-wise encoding

Figure 19. Memory issue. (a) It is highly likely to be a victim to OS memory killer; (b) Merits and drawbacks of layer-by-layer encoding. BS=1, # of encoded figures=50..

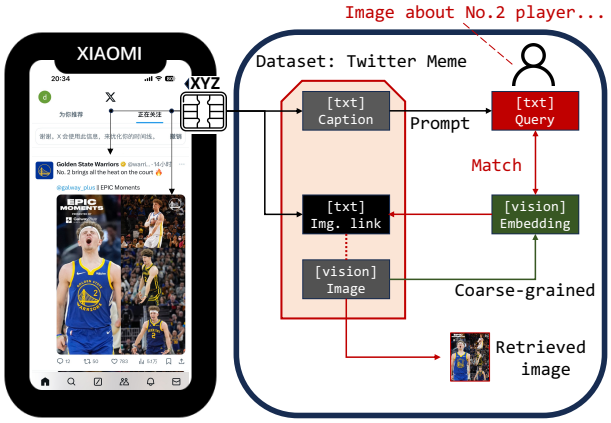


Figure 20. Case study: Daily personal application retrieval.

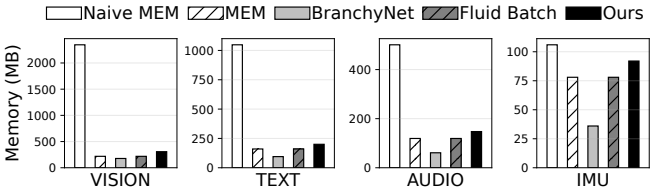


Figure 21. Memory footprint.

moment or items, they can query the memory palace with a multimodal query, and the system will retrieve the most relevant items. MEMs can also facilitate mobile agents to iteract with users in a more human-like manner [62–64]. Recently, Microsoft launches a project called Recall that makes a note of everything ever displayed on personal computer for AI-empowered retrospective search [65].

On-device Multimodal Embedding Data for embedding is continuously sourced from end users and is often private and sensitive. Evidence suggests that cloud service providers may be curious about uploaded data to improve their services [13], and database leaks and breaches pose significant threats [66]. Conducting embedding locally prevents the need to upload daily viewed, sensed, or heard data to the cloud, offering strong privacy protection. From the cloud perspective, a single user views over 6,000 images per day, according to our user study, requiring approximately 1065.6KJ of energy and 0.8 GPU hours. For 1 billion daily active users, cloud providers would need 1.1 TWh of energy and 0.8M GPU hours daily, costing over $100 million per day. On-device multimodal embedding shifts this cost to end users, making the service more practical to deploy.

Early-Exiting While early exiting has been a known technique in both traditional CNNs and recent language models [21, 48, 50, 67], it is rarely integrated with MEMs for mobile devices. BranchyNet [21] showed that features learned in early layers are often sufficient for classification, with only a few difficult samples requiring deeper layers. DynExits [49] introduced learnable early-exit branch weights to avoid the pitfalls of manually defined loss weights, while DVABatch [68] dynamically adjusted batch sizes to allow independent query exits, though with limited improvement. Layer Skip [47] and DeeCap [67] utilized early exits for tasks like text decoding and image captioning. Gromov et al. [50] demonstrated that removing deeper layers often does not degrade performance due to the similarity between adjacent deep layers. In this work, we propose the first early-exiting system for on-device MEMs, providing a lightweight and efficient solution for mobile devices.

Predictive Early Exit Predictive Exit [48] designed a low-cost prediction engine for CNNs, using zero padding, filter generation, and one-dimensional convolution to predict exit points in computer vision tasks. Dong et al. [69] introduced an exit predictor that uses depthwise separable convolutions to generate scores for deciding whether to skip certain exits, reverting to confidence-based decisions when necessary. However, these methods are complex and not suited for attention-based transformers, where matrix multiplication dominates. In summary, while effective for CNNs, these approaches do not scale well to transformer-based models due to their convolution-specific designs. Hamed et al. [59] integrated early exits at the end of encoders to balance predictive performance and computational efficiency, but they did not address exit timing or hardware compatibility. Our system introduces a hardware-friendly, lightweight predictor for efficient early exits in transformers, tailored for mobile devices, ensuring both performance and accuracy.

Implementation Details

`Reminisce` is built on ImageBind-LoRA [70], an open-source multimodal embedding model fine-tuning framework. For matching, we use matrix multiplication, as it is not the primary bottleneck in query cost. Further vector database optimizations, such as FAISS [71], are orthogonal to `Reminisce`. LoRA tuning and embedding accuracy evaluations are emulated on a GPU server to enable faster iterations and energy savings. Embedding inference latency, power consumption, and memory usage are directly measured by running `Reminisce` on a mobile device using the open-source on-device multimodal inference engine mllm [18]. Since mllm [18] currently does not support the IMU modality and lacks GPU

optimizations, we also use PyTorch on a development board for broader dataset comparisons.

Case Study: Twitter Meme Retrieval

To reveal the practicality of `Reminisce` in real world scenarios, we collected daily surfing images and relevant captions from Twitter memes from the Internet as shown in Figure 20. The data were filtered by end users to avoid privacy issues. These figures are not stored locally; only their links are saved to conserve storage.