SimCache: Similarity Caching for Efficient VLM-based Scene Understanding

Surya Selvam[1] Ravi K. Rajendran[2] Murugan Sankaradas[2]
Anand Raghunathan[1] Srimat T. Chakradhar[2]
[1]Purdue University [2]NEC Laboratories America
{selvams,raghunathan}@purdue.edu, {rarajendran,murugs,chak}@nec-labs.com

Abstract

Scene understanding systems analyze visual contexts by detecting objects, their attributes, and the interactions among them to provide a holistic interpretation. Understanding a scene requires analyzing multiple salient regions within a single video frame. Recently, Vision-Language Models (VLMs) have emerged as powerful tools for scene understanding, leveraging learned world knowledge to enable deployment without specialized training or fine-tuning. However, deploying VLMs in real-time applications is challenging due to their high computational and memory requirements, which limit processing throughput.

We propose SimCache, a novel software-based caching mechanism that optimizes VLM-based scene understanding systems by reducing redundant computations. SimCache stores the embedding representation of a salient region and its detected activity, enabling reuse of VLM computations for similar regions in future frames. Specifically, SimCache exploits two types of redundancy: (1) temporal locality, reusing computations for similar regions across adjacent frames, and (2) semantic locality, reusing computations for visually distinct regions that represent the same activity at different times. SimCache includes a multi-tier cache architecture with specialized cache search and refinement policies to exploit redundancy efficiently and accurately. Experiments on action recognition datasets demonstrate that SimCache improves system throughput by up to 9.4× and reduces VLM computations by up to 24.4× with minimal accuracy loss.

1. Introduction

Scene understanding requires analyzing visual scenes to identify objects, their attributes, and the interactions between them. It is a critical component of real-world applications such as autonomous driving, robotics, surveillance, and industrial automation, where real-time decision-making is essential. For instance, autonomous vehicles must detect and react to hazards in real time to ensure safe navigation, while surveillance systems continuously analyze scenes to monitor activities and detect anomalies. However, achieving real-time scene understanding is inherently challenging, as it involves processing multiple objects and their interactions within each frame. For instance, a single surveillance video frame might include several people and the objects they carry, each requiring separate analysis to determine their interactions. Therefore, constructing an accurate and structured representation of the scene requires decomposing the frame into multiple salient regions and processing them efficiently to meet real-time constraints.

In recent years, Vision-Language Models (VLMs) have been increasingly deployed in scene understanding systems [16, 18, 21, 22, 27]. Unlike traditional deep learning methods that require domain-specific training, VLMs generalize across diverse scenarios by leveraging pre-trained world knowledge. Despite these advantages, deploying VLMs for real-time scene understanding remains challenging due to their high computational cost. Large VLMs (e.g., 13B or 34B parameters) exceed the memory capacity of a single workstation GPU with 24 GB memory, making local deployment impractical. Even smaller VLMs (fewer than 7B parameters) exhibit very low throughput, thereby limiting the number of frames that can be processed in real time.

Figure 1 illustrates a typical VLM-based scene understanding system, which processes a video stream frame by frame to generate insights in the form of a scene graph. The system follows a three-stage pipeline. First, an object detector identifies objects within a video frame. Next, a salient region identifier dynamically selects regions in a frame that may contain significant interactions between objects. Finally, these salient regions are fed into a VLM to identify the interaction between object pairs, producing structured triplets in the form of *(subject, relationship, object)*. Despite its advantages, the system's main bottleneck is the high computational cost of VLMs, making it difficult to meet real-time constraints. For instance, LLaVA-1.5-7B [14], a state-of-the-art VLM, can process only four salient regions per frame (with batching) at a throughput of just 0.4 FPS on an NVIDIA RTX 3090 Ti GPU. This constraint severely limits the deployment of VLM-based scene

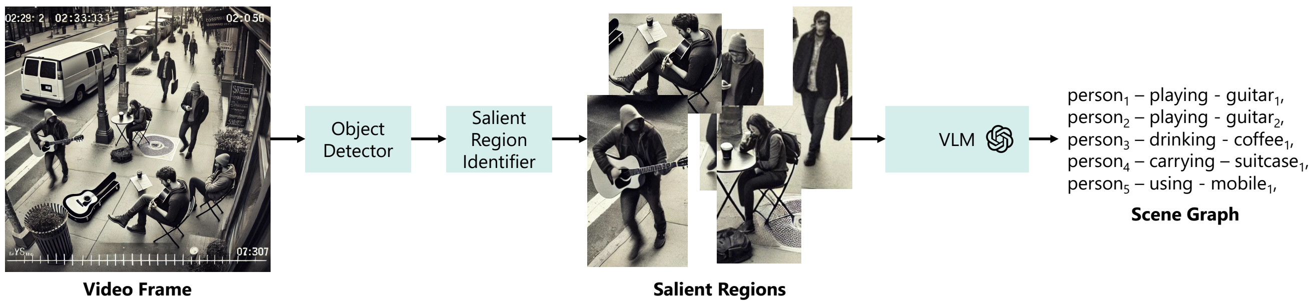
Video Frame
Object Detector
Salient Region Identifier
Salient Regions
VLM
person1 – playing - guitar1,
person2 – playing - guitar2,
person3 – drinking - coffee1,
person4 – carrying – suitcase1,
person5 – using - mobile1,
Scene Graph

Figure 1. An overview of a VLM-based scene understanding system, which consists of three key steps: (1) object detection, (2) salient region identification, and (3) relationship extraction using a VLM.

understanding in real-time applications such as autonomous driving and surveillance, where analyzing every salient region is essential.

A common approach to improving VLM processing throughput is to use uniform [22] or adaptive [2] frame-skipping strategies to avoid processing similar frames. While these methods improve throughput, they introduce accuracy trade-offs since skipping frames can lead to information loss. Moreover, frame-skipping strategies reduce the number of processed frames, leaving open the challenge of efficiently processing frames that are not skipped. Complementary to previous work, our work exploits two key forms of redundancy in video streams at the region level: (1) *temporal locality*, where adjacent frames often contain minimally changed regions, and (2) *semantic locality*, where certain regions reappear across time, describing the same activity despite visual variations. Processing these redundant regions results in unnecessary computational overhead, significantly impacting system efficiency.

In this paper, we introduce SimCache, a caching mechanism designed to optimize VLM-based scene understanding systems by reducing redundant computations. SimCache minimizes redundant VLM computations on salient regions by caching their outputs and reusing them for temporally and semantically similar regions in future frames. To achieve this, SimCache computes an embedding representation for each salient region and uses it as a key for cache lookup and updates. Our approach features a multi-tier cache architecture consisting of: (1) a temporal cache, which stores region embeddings and VLM outputs from the last N frames to exploit temporal locality, and (2) a semantic cache, which maintains embeddings and the corresponding outputs over longer intervals to exploit semantic locality. Additionally, we propose specialized cache search and refinement policies to further enhance the efficiency and accuracy of cache operations. Our experiments demonstrate that SimCache improves throughput by up to 9.4× and reduces VLM computations by up to 24.4×, while maintaining accuracy on the UCF-101 action recognition dataset.

2. Scene Understanding Systems: Challenges and Opportunities

This section provides an overview of conventional VLM-based scene understanding systems, discusses key challenges in their deployment, and highlights opportunities for improving their efficiency.

2.1. VLM-based scene understanding

A typical VLM-based scene understanding system (Figure 1) consists of three main components - an object detector, a salient region identifier, and a VLM - which we describe below.

Object Detector. Each video frame is first processed by an open-vocabulary object detector to obtain bounding boxes for objects of interest. These detectors can identify objects based on arbitrary textual queries, unlike traditional closed-vocabulary object detectors, which can recognize only a fixed set of object classes. Additionally, the list of target objects can be dynamically configured depending on the deployment scenario. Although open-vocabulary object detection provides flexibility, it is not the primary bottleneck in a scene understanding system. For instance, YOLO-World [5], a lightweight open-vocabulary object detector used in VisionGPT [22], processes frames at 66.67 FPS with a latency of 15 ms per frame on an NVIDIA RTX 3090 Ti GPU, demonstrating that object detection can be performed in real time.

Salient Region Identifier. After detecting objects, the salient region identifier selects object pairs that are likely to interact, as processing all $\binom{n}{2}$ possible pairs would be computationally expensive. The system prioritizes regions where interactions are more likely to occur by computing a score based on spatial attributes such as position, distance, relative size, and intersection area, following techniques like RECODE [12]. If a pair's score exceeds a predefined threshold, a proposal region is formed by merging their bounding box coordinates. This region is then analyzed by the VLM to determine the relationship between the objects. Since these computations primarily involve basic geometric operations, the salient region identifier adds

Model	Batch Size	GPU Memory (GB)	Throughput (FPS)
Moondream-2 [1]	1	4.29	3.04
	2	4.83	2.28
	4	5.89	1.53
	8	8.01	0.90
	16	12.23	0.49
	32	20.74	0.26
	64	OOM	–
LLaVA-1.5-7B [14]	1	15.81	1.42
	2	17.50	0.77
	4	20.86	0.41
	8	OOM	–

Table 1. Comparison of GPU memory usage and throughput (FPS) for different batch sizes on the Moondream-2 and LLaVA-1.5-7B.

negligible computational overhead.

VLM. The final step involves using the VLM to infer relationships between objects within a salient region. The VLM processes each salient region alongside a textual prompt, such as: *"What is the [subject] doing with the [object]? Possible answers include [relationship 1], [relationship 2], ... and [relationship n]. Please choose one."* The VLM response is then used to construct a scene graph for downstream vision applications, including action recognition and question answering.

Notwithstanding their advantages, VLMs present two significant computational challenges in scene understanding systems. First, they must process all salient regions in a frame, but their batch processing capacity is limited by memory constraints. Table 1 shows that even lightweight models like Moondream-2 (a 1.87B parameter model[1]) can process at most 32 regions per frame, while LLaVA-1.5-7B can handle only four regions per frame on an NVIDIA RTX 3090 Ti GPU. This restriction severely limits the number of relationships that can be extracted per frame. Second, scene understanding systems must operate in real-time, but VLM throughput is too low to meet these requirements. Even with a batch size of 1 (Table 1), Moondream-2 and LLaVA-1.5-7B achieve only 3.1 and 1.4 FPS, respectively—far below standard video frame rates (e.g., 30 FPS). These challenges necessitate novel approaches to optimize VLM-based scene understanding.

2.2. Opportunities: Localities in Video Streams

Streaming videos exhibit inherent redundancies, where many regions remain unchanged over time or reappear periodically. For instance, in surveillance systems with stationary cameras, the background and static objects, such as fire hydrants, remain the same over time. Additionally, certain events, such as a person carrying a briefcase, occur repeatedly. These redundant regions do not need to be processed by VLMs multiple times in a scene understanding system. In this work, we identify two key localities in video streams – temporal locality and semantic locality – that can be exploited to reduce redundant computation and alleviate computational bottlenecks.

[1]Evaluated using model revision dated 2024-05-20

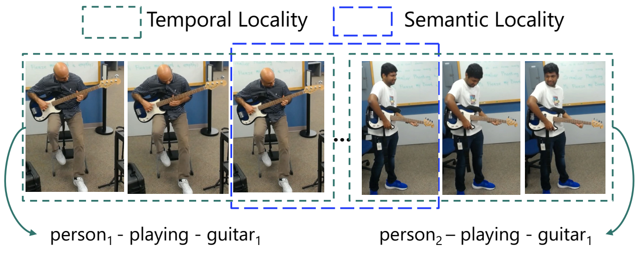
Temporal Locality
Semantic Locality
...
person_1 - playing - guitar_1
person_2 – playing - guitar_1

Figure 2. Illustration of key locality types in video streams: (1) temporal locality, and (2) semantic locality.

Temporal Locality. Temporal locality refers to the scene invariance across consecutive frames in a video. Since the scene does not change instantaneously, adjacent frames often contain minimal variation. This property has been widely exploited in video compression [20, 24]. Figure 2 illustrates two instances of temporal locality in video streams, where consecutive frames depict the same activity. Skipping computations on temporally similar frames enhances system efficiency while preserving accuracy.

Semantic Locality. Semantic locality refers to the recurrence of visually distinct yet semantically equivalent scenes describing the same activity or event. Unlike temporal locality, which occurs only between adjacent frames, semantic locality can occur at any point in a video. As shown in Figure 2, two non-consecutive frames are semantically similar because they contain two different individuals performing the same activity. These regions convey identical information, despite their differences in appearance, position, or lighting. By recognizing and reusing computations on semantically similar regions, the system can skip additional computations, significantly improving efficiency.

3. SimCache

To optimize VLM-based scene understanding, we introduce SimCache, a caching mechanism that reduces redundant computations by leveraging both temporal and semantic locality in video streams. SimCache minimizes the number of regions processed by compute-heavy VLMs by storing and retrieving previously computed outputs for similar regions. SimCache consists of three key components: (1) an embedding generator that computes feature representations for salient regions, (2) a cache manager that handles retrieval and updates, and (3) a multi-tier cache architecture with novel policies for efficient storage, lookup, and replacement. Figure 3 provides an overview of SimCache within the VLM-based scene understanding pipeline.

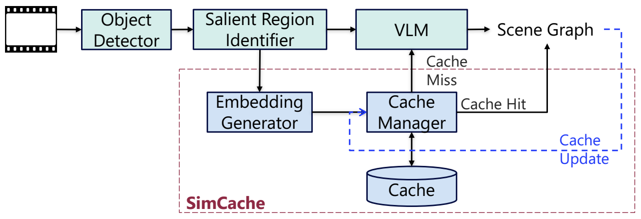

Figure 3. Integration of SimCache into the VLM-based scene understanding system.

3.1. Embedding Generator

The overall goal of the embedding generator is to obtain a feature representation of salient regions that enables efficient comparison and facilitates skipping computations for similar regions. Raw pixel-based comparisons are highly sensitive to minor variations in appearance, such as lighting changes. Instead, embeddings capture the underlying activity or interaction within a salient region rather than its exact pixels. This enables SimCache to detect and reuse cached outputs for visually distinct but semantically similar regions, improving efficiency. Given a salient region SR^t at time step t, we compute its embedding e^t using a neural network feature extractor f such that $e^t = f(SR^t), \quad f : \mathcal{R}^{H\times W\times 3} \rightarrow \mathcal{R}^d$, where e^t is the d-dimensional vector that captures the contextual interaction between the objects in a salient region. These embeddings are then used as keys for cache lookups and updates, enabling efficient reuse of prior VLM outputs for similar regions.

3.2. Cache Manager

The cache manager is responsible for retrieving and updating cache entries. Given a salient region embedding, the cache manager searches for similar embeddings in the cache by computing a similarity metric such as cosine similarity or L1/L2 distance between the query embedding and all entries in the cache. It determines whether a cache hit occurs by checking if the similarity score exceeds a predefined threshold or by performing majority voting among the Top-k selected embeddings. Upon a cache hit, the system retrieves the previously computed VLM output, reducing redundant computation. If not, the salient region is processed by the VLM. After obtaining the relationship between objects in a salient region, either through a cache hit or from the VLM, the cache manager uses this result to update and refine the cache entries.

3.3. Cache Architecture

Figure 4(a) illustrates the cache architecture of SimCache, which employs a multi-tier structure comprising a temporal cache and a semantic cache to efficiently retrieve relationships given an embedding.

Temporal Cache. The first-tier cache, the temporal cache, stores embeddings and their corresponding VLM outputs

Algorithm 1 Cache Search

```
Require: embedding $e$, subject $s$, object $o$, parameter $k$
Ensure: $cache\_hit$, relationship $r$
 1: $cache\_hit \leftarrow False, r \leftarrow None$
 2: $top_k\_scores, top_k\_rels \leftarrow$ search($cache[s][o], e, k$)
 3: $rels \leftarrow$ unique($top_k\_rels$), $\mu \leftarrow \{\}$
 4: for $j \in rels$ do
 5:     $\mu[j] \leftarrow$ mean($\{top_k\_scores[i] | top_k\_rels[i] = j\}$)
 6: end for
 7: $\mu \leftarrow \{i : \mu[i] \mid \mu[i] > \tau[i]\}$
 8: if $\mu = \varnothing$ then
 9:     return $cache\_hit, r$
10: end if
11: sort_by_descending_scores($\mu$)
12: $r, score \leftarrow \mu[0]$
13: $valid \leftarrow$ all($|score - \mu[i]| \geq \delta \ \forall i \neq r$)
14: $cache\_hit \leftarrow valid$
15: return $cache\_hit, r$
```

for a fixed window of the most recent N frames. It is designed to exploit temporal locality in videos, where adjacent frames contain minimal variation. Our analysis shows that approximately 85% of frames exhibit high similarity to the immediately preceding frame. By explicitly designing the cache to exploit temporal locality, we reduce the search space and enable efficient retrieval.

Semantic Cache. The second-tier cache, the semantic cache, stores embeddings and outputs from a longer historical window to exploit semantic locality. Unlike the temporal cache, which focuses on short-term redundancy, the semantic cache captures repeated activities or interactions that occur throughout the video. The semantic cache stores the entries in a partitioned structure with a unique cache for each subject-object pair, as shown in Figure 4(a). This hierarchical structure enables targeted searches, as each pair of objects is identified at the object detector stage. Consequently, it eliminates the need for filtering based on subject-object pairs, thereby reducing the search complexity.

3.4. Cache Policies

Search Policy. Our cache search policy aims to enhance retrieval accuracy while minimizing negative cache hits. A negative cache hit occurs when the ground-truth relationship or action of the query embedding differs from the relationship retrieved from the cache. Traditional image retrieval methods often use k-nearest neighbors (kNN) search with majority voting, but this approach is prone to negative cache hits. While a static similarity threshold can reduce negative cache hits, it is often very restrictive, thereby limiting potential cache hits. To address this, we propose a search policy as described in Algorithm 1. Given a query, our approach retrieves the Top-k most similar embeddings,

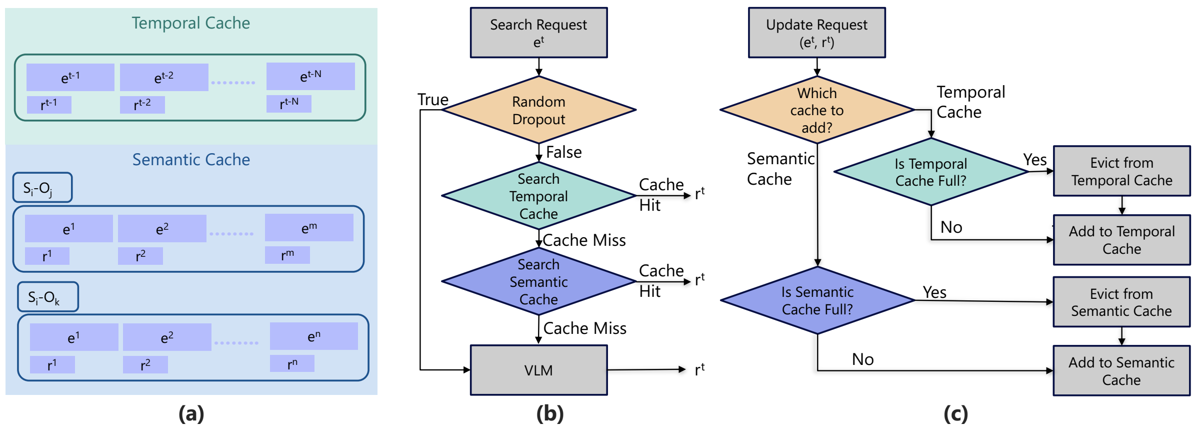

Figure 4. Overview of SimCache's multi-tier cache architecture and workflow. (a) The multi-tier cache consists of a temporal cache and a semantic cache. (b) Search workflow. (c) Update workflow.

groups them by relationship type, and discards groups with an average similarity below a dynamic threshold τ. To further distinguish between filtered relationships that are close in similarity, we apply a secondary static threshold δ. This two-step filtering search policy jointly optimizes for high cache hit rates while minimizing negative cache hits.

Dynamic Similarity Threshold Determination. A static similarity threshold fails to adapt to evolving video scenes, leading to poor cache hit rates. To mitigate this, we use a dynamically computed threshold τ (Algorithm 2) that adjusts periodically based on the entries in the cache. Specifically, we compute a separate threshold for each relationship by taking the mean of the pairwise similarity matrix of the corresponding cached embeddings. Each row of this matrix contains similarity scores between a single embedding and all other embeddings within the same relationship category, thereby capturing intra-class similarity distribution. This algorithm is invoked whenever the cache is refined, enabling the system to adapt to scene variations and maintain high cache hit rates.

Refinement Policy. To accommodate new entries when the cache reaches full capacity, we propose a cache refinement algorithm that enforces both intra- and inter-relationship diversity. Specifically, it prevents the cache from being dominated by highly similar entries within a single relationship category or overrepresented by one particular relationship. Our approach (Algorithm 3) utilizes k-means clustering to selectively evict redundant embeddings for each relationship type. Entries for a given relationship are considered for eviction only if their count exceeds a minimum threshold m. If this condition is met, similar embeddings within that relationship are clustered, and redundant ones are evicted. The number of clusters is determined by the cache retention rate α. We then form k clusters and retain the entries closest to the cluster centroids, thereby maintaining a diverse and representative set of cache entries.

3.5. Overall Workflow

Figures 4(b) and (c) illustrate the overall workflow of the proposed multi-tier cache architecture, which handles both search and update requests. For a search request, SimCache employs a random dropout mechanism, introducing a forced cache miss with probability p. This dropout mechanism ensures periodic cache updates with entries validated by the VLM. The query is then searched hierarchically—first in the temporal cache using a Top-k similarity search with a static threshold, and then in the semantic cache using Algorithm 1. If a cache hit occurs, the retrieved relationship is returned, bypassing VLM computations. Otherwise, salient regions corresponding to the cache misses are forwarded to the VLM for inference. Each search request triggers a corresponding update. By default, new embedding-relationship pairs are stored in the temporal cache to retain recent frame embeddings. Only VLM-validated entries are promoted to the semantic cache, ensuring high-quality entries and preventing cache pollution. The temporal cache follows a FIFO eviction policy, while the semantic cache is managed using Algorithm 3. Algorithm 2 recalibrates similarity thresholds each time the semantic cache is refined for adaptive cache management.

4. Experimental Results

In this section, we first describe the experimental setup, then present our main results, perform qualitative analysis, and finally conduct ablation studies.

4.1. Experimental Setup

Datasets. We evaluate SimCache on two action recognition datasets: UCF-101 [19] and an in-house human-object interaction dataset. UCF-101 consists of 101 categories of human-object interactions, where each video primarily depicts a single activity. Each clip contains only one possible human-object relationship. For instance, a video labeled

Algorithm 2 Dynamic Threshold τ Determination

```
Require: subject $s$, object $o$
Ensure: thresholds $\tau$
1: $\tau \leftarrow \{\}$
2: $rels \leftarrow$ get_relationships_from_cache$(s, o)$
3: for $r \in rels$ do
4:     $embs \leftarrow$ get_embeddings_from_cache$(r)$
5:     $scores \leftarrow$ compute_pairwise_similarity$(embs)$
6:     $\tau[r] \leftarrow$ mean$(scores)$
7: end for
8: return $\tau$
```

Algorithm 3 Cache Refinement with K-Means Clustering

```
Require: cache_retention_rate $\alpha$
Require: min_entries_per_relationship $m$
1: for all $s, o \in$ get_subject_object_pairs() do
2:     $rels \leftarrow$ get_relationships_from_cache$(s, o)$
3:     for $r \in rels$ do
4:         $embs \leftarrow$ get_embeddings_from_cache$(r)$
5:         $n_r \leftarrow embs.size()$
6:         $k \leftarrow \max\big(\min(n_r, m), \lfloor \alpha\, n_r \rfloor\big)$
7:         $labels, centroids \leftarrow$ KMeans$(embs, k)$
8:         $rep\_idx \leftarrow \cup_{i=1}^{k}\{\arg\min_{j:\, labels[j]=i} \|embs[j] - centroids[i]\|_2^2\}$
9:         $evict\_idx \leftarrow \{0, 1, 2, \ldots, n_r - 1\} \setminus rep\_idx$
10:        evict_entries$(evict\_idx)$
11:    end for
12: end for
```

person playing guitar exclusively depicts that action, without instances of a person near a guitar but not playing it. In contrast, our internal dataset captures more complex interactions in which multiple relationships are possible between the same object pair. For example, it includes both playing and non-playing scenarios for a person and a guitar, requiring fine-grained recognition of activity changes.

Implementation Details. We implement SimCache using FAISS [7] for efficient similarity search and storage of embeddings. We use cosine similarity to compare feature embeddings. The VLM-based scene understanding system is built with PyTorch [17] and Hugging Face [25], using YOLO-World [5] for open-vocabulary object detection and either Moondream-2 [1] or LLaVA-1.5-7B [14] for relationship identification. SimCache utilizes embeddings generated from MobileNet-V3-Large [11] or ViT-Base/Large [6], depending on the scenario and deployment constraints. All experiments are conducted on an NVIDIA RTX 3090 Ti GPU (24GB memory).

SimCache Cache Configurations. Unless stated otherwise, we use the following cache settings for all experiments. The temporal cache operates with a window size of $N = 5$ and a static similarity threshold $\tau = 0.9$. The semantic cache stores 30 entries per subject-object pair and retrieves the Top-k ($k = 5$) similar embeddings using a dynamic threshold with $\delta = 0.05$. For cache refinement, we use clustering with a cache retention rate of $\alpha = 0.5$ and a minimum eviction threshold of $m = cache\_size/2$. On UCF-101, we disable the random dropout rate ($p = 0$), whereas on our internal dataset, we set $p = 0.10$.

VLM	Method	Throughput (FPS)	# Salient regions processed by VLM
Moondream-2	Baseline	10.82	244
	SimCache	48.32 (4.5× ↑)	10 (24.4× ↓)
LLaVA-1.5-7B	Baseline	5.02	244
	SimCache	47.23 (9.4× ↑)	10 (24.4× ↓)

Table 2. Comparison of throughput and VLM computations on the UCF-101 dataset across two methods using two VLMs.

Evaluation Metrics. We report the following metrics: (1) throughput, measured as the number of frames processed per second by the VLM-based scene understanding system; (2) number of salient regions processed by the VLMs; (3) cache hit rate, defined as the fraction of queries that successfully retrieved a relationship from the cache; and (4) negative hit rate, defined as the fraction of cache hits that were retrieved incorrectly.

4.2. Main Results

We first evaluate SimCache on UCF-101, presenting the results in Table 2. As a baseline, we use dynamic frame skipping to remove visually redundant frames, demonstrating how SimCache complements such strategies for additional gains. We select a subset of 10 classes from UCF-101 (25 videos per class, totaling 250 videos). We apply Pyscenedetect [4] to perform dynamic frame skipping, resulting in one representative frame per video. These frames are then processed through the VLM-based scene understanding system, followed by SimCache. Our results show that the baseline system processes 244 salient regions (out of 250) using the VLM, while failing to detect objects in six frames. SimCache recognizes all actions while processing only a single frame per class, reducing VLM queries to 10 salient regions and incurring no negative cache hits. This result demonstrates the ability of SimCache to generalize across visually diverse frames depicting the same activity, enabling efficient scene understanding with minimal VLM queries. Overall, SimCache achieves a 4.5× and 9.4× throughput improvement on UCF-101, with a 24.4× reduction in VLM computations. Additionally, it enhances frame skipping by further minimizing redundant computations on semantically similar regions.

Next, we evaluate SimCache on our internal dataset with the Moondream-2 VLM (Table 3). In contrast to UCF-101, we process all frames in the baseline method without

Method	**Embedding Model**	**Throughput (FPS)**	**# Salient regions processed by VLM**	**Runtime Breakdown (seconds)**				**Cache Hit Rate(%)**
				Object Detector	VLM	Embedding Generator	Cache Operations	Overall/Temporal /Semantic
Baseline		9.71	54968	578 (10.9%)	4719 (89%)	0	0	-
SimCache	MB-V3-L	36.2 (3.7× ↑)	5808 (9.5× ↓)	579 (40.7%)	547 (38.4%)	286 (20.1%)	6.47 (0.5%)	89.4/90.3/93.9
	ViT-B	33.2 (3.4× ↑)	5711 (9.6× ↓)	581 (37.5%)	539 (34.8%)	422 (27.2%)	5.22 (0.3%)	89.6/94.7/92.4
	ViT-L	26.96 (2.8× ↑)	5701 (9.6× ↓)	584 (30.6%)	542 (28.4%)	772 (40.5%)	5.85 (0.3%)	89.6/95.9/90.6

Table 3. Performance comparison of SimCache and the baseline system on the internal dataset (using Moondream-2), reported in terms of throughput, number of VLM computations, cache hit rate, and runtime breakdown across different embedding generator models.

Action	**# Correct Predictions**			
	Baseline	SimCache		
		MB-V3-L	ViT-B	ViT-L
Playing guitar	27	25	26	26
Climbing ladder	23	22	21	24
Sitting chair	30	25	30	28
Writing whiteboard	28	26	27	28
Lifting suitcase	26	24	27	29
Total	134	122	133	135

Table 4. Action recognition accuracy comparison for each method across a selected subset of actions.

frame skipping. We assess different embedding generators (MobileNet-V3 Large, ViT-Base, and ViT-Large) to analyze their impact on SimCache's performance. We observe that larger embedding models increase the temporal cache hit rates due to their ability to capture fine-grained details. However, they slightly reduce overall throughput due to the increased computational effort required for embedding generation. To evaluate cache hit accuracy, we reprocess all cache hits using the VLM to determine whether they are positive or negative hits. We find that the negative cache hit rates are 14%, 10%, and 7% for MobileNet-V3 Large, ViT-Base, and ViT-Large, respectively. However, this evaluation assumes VLM predictions are perfectly accurate. To ensure a fair comparison, we randomly sample 30 frames from five selected actions and manually annotate them. Table 4 shows that SimCache with ViT-Base and ViT-Large performs comparably to the baseline. Furthermore, compared to the baseline, where nearly 90% of processing time is spent on VLM computations, SimCache reduces this to 30%-40%, significantly alleviating the computational bottlenecks in scene understanding. Notably, cache search and update operations account for fewer than 0.5% of the total processing time.

4.3. Qualitative Results

For qualitative evaluation, we visualize the Top-k retrieved salient regions for selected queries from UCF-101 and our internal dataset. First, we demonstrate the ability of SimCache to exploit semantic locality. Figure 5 presents a query image and its top-5 most similar images retrieved from the semantic cache. Despite visual variations in appearance and background, all retrieved images depict the same relationship, allowing the query image to bypass redundant VLM computations by reusing this relationship. Second, we highlight the advantages of SimCache's proposed search policy over naive Top-k majority voting. Figure 6 presents a query image depicting the relational triplet *person-standing near-chair*, along with its top-5 retrieved images. Among these, four images correspond to *person-sitting-chair*, while only one correctly represents the ground truth. Majority voting would therefore misclassify the query image as *person-sitting-chair*. To address this, our search algorithm (Algorithm 1) computes the average similarity score for each relationship category, yielding $\mu[sitting] = 0.61$ and $\mu[standing\ near] = 0.67$. Since the difference exceeds the predefined threshold δ ($|\mu[standing\ near] - \mu[sitting]| = 0.06 > \delta$, where $\delta = 0.05$), SimCache correctly classifies the query as *person - standing near - chair*, demonstrating its robustness in distinguishing fine-grained relationships.

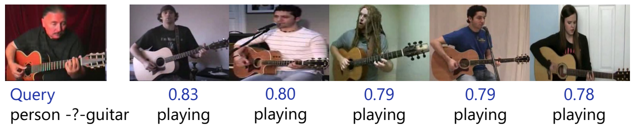

Figure 5. Qualitative analysis of SimCache exploiting semantic locality in the UCF-101 dataset.

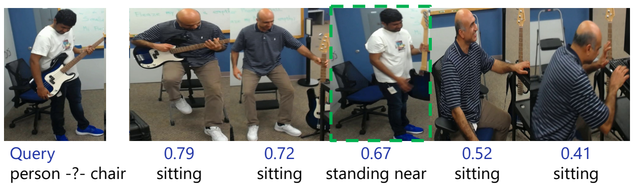

Figure 6. Qualitative analysis of SimCache on an internal dataset, illustrating its ability to differentiate between fine-grained relationships where existing methods fail.

4.4. Ablation Results

To analyze the individual components of SimCache, we conduct ablations on cache design choices, including architecture and policies. For these experiments, we use ViT-Base as the embedding generator model, Top-k majority voting as the default search policy, and LRU as the default replacement policy. First, we examine the effectiveness of the multi-tier cache architecture, as shown in Table 5. The

Cache Configuration	Hit Rate (%)	Negative Hits (%)	Avg. Query Search Time (ns)
Only Temporal	82.82	1.37	5.95 (1×)
Only Semantic	88.2	9.74	26.06 (4.4×)
SimCache	89.0	3.09	7.45 (1.25×)

Table 5. Comparison of different cache architectures.

Search Policy		Hit Rate (%)	Negative Hits (%)
Top-k Majority Voting	$k = 3$	80	57
	$k = 5$	82	58
	$k = 7$	82	59
Static Threshold	$\tau = 0.6$	74	26
	$\tau = 0.7$	63	18
	$\tau = 0.8$	45	5
Dynamic Threshold	$k = 5$	63	10

Table 6. Ablation study on cache search policies.

results indicate that using only the temporal cache leads to a lower cache hit rate, whereas relying on the semantic cache alone results in a higher rate of negative cache hits. Our multi-tier cache architecture effectively combines the strength of both approaches. Furthermore, in our multi-tier cache architecture, fewer than 7% of requests are directed to the semantic cache, resulting in only a 25% increase in average search time per query compared to the temporal cache. Additionally, as the frame sampling rate increases, the temporal cache hit rate declines, necessitating the use of the semantic cache, as higher sampling reduces temporal redundancy. Next, we evaluate our proposed cache search algorithm. For these experiments, we consider only SimCache with the semantic cache, comparing it with various alternatives, as shown in Table 6. We observe that Top-k majority voting achieve high cache hit rates with high errors, whereas static thresholding is more restrictive but results in fewer negative hits. Our algorithm remains on the Pareto-optimal frontier of cache hit rate and accuracy, demonstrating its ability to effectively balance accuracy and efficiency. To perform a standalone evaluation of our clustering-based cache refinement policy, we compare the t-SNE visualizations of the embeddings in the cache before refinement and after applying different policies, including clustering, LRU, and FIFO (Figure 7). We observe that the cache remains balanced after clustering, whereas with LRU and FIFO, the cache is dominated by entries corresponding to a single relationship. This result demonstrates the effectiveness of our algorithm in maintaining cache diversity, enabling SimCache to adapt quickly to new scenarios.

5. Related Works

Frame Skipping. Existing VLM-based scene understanding systems utilize uniform frame sampling techniques to manage computational complexity by skipping computations on temporally similar frames. For instance, VisionGPT [22] invokes an LLM call only every 30 frames, while another work [21] samples five frames at regular intervals as input for a VLM in robot manipulation tasks. Many video understanding systems [8, 13, 15, 23, 28] also employ uniform frame sampling to reduce computational cost. ViTA [2], a video analysis system, leverages PySceneDetect [4] to adaptively skip frames. While these methods focus entirely on skipping computations at the frame level, SimCache instead reuses computations at a finer granularity (the region level) by leveraging both temporal and semantic locality in videos.

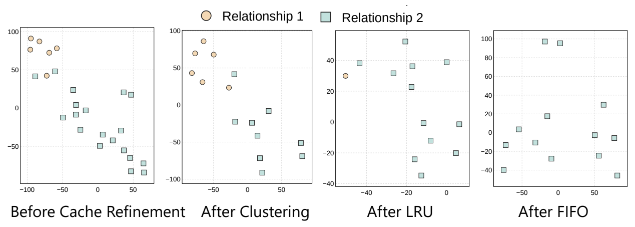

Figure 7. Comparison of t-SNE visualizations of various cache refinement algorithms.

Caching. GPT-Cache [3] introduces a caching mechanism to store and retrieve LLM responses, reducing both cost and response time for repeated queries. While GPT-Cache shares similarities in principle, SimCache additionally proposes efficient search and refinement algorithms to minimize negative cache hits, ensuring more accurate and efficient retrieval. Potluck [9] introduces a caching methodology to reuse computational results across applications, while FoggyCache [10] extends this approach across devices. However, both of these approaches operate at the frame level, whereas SimCache caches at the region level to achieve fine-grained reuse. Additionally, DeepCache [26] explores caching activations across convolutional layers within a DNN, exploiting both temporal locality and model properties for intra-model caching.

6. Conclusion

We introduce SimCache, a caching mechanism designed to optimize VLM-based scene understanding systems. By storing and reusing VLM outputs, we reduce redundant computations and address computational bottlenecks. SimCache generates an embedding for each salient region and stores it alongside the corresponding VLM output. Additionally, SimCache includes a multi-tier cache architecture that exploits two inherent redundancies in video streams: temporal and semantic locality. Experimental results demonstrate that SimCache improves throughput by up to 9.4×, reduces VLM computations by up to 24.4×, and maintains competitive accuracy in action recognition tasks.

Acknowledgments

This work was supported in part by the National Science Foundation under Award 2318101.

References

[1] Moondream, 2024. GitHub repository. 3, 6

[2] Md Adnan Arefeen, Biplob Debnath, Md Yusuf Sarwar Uddin, and Srimat Chakradhar. Vita: An efficient video-to-text algorithm using vlm for rag-based video analysis system. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2266–2274, 2024. 2, 8

[3] Fu Bang. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 212–218, 2023. 8

[4] Brandon Castellano. Pyscenedetect. 6, 8

[5] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16901–16911, 2024. 2, 6

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 6

[7] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024. 6

[8] Yue Fan, Xiaojian Ma, Rujie Wu, Yuntao Du, Jiaqi Li, Zhi Gao, and Qing Li. Videoagent: A memory-augmented multimodal agent for video understanding. In *European Conference on Computer Vision*, pages 75–92. Springer, 2024. 8

[9] Peizhen Guo and Wenjun Hu. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 271–284, 2018. 8

[10] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. Foggycache: Cross-device approximate computation reuse. In *Proceedings of the 24th annual international conference on mobile computing and networking*, pages 19–34, 2018. 8

[11] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019. 6

[12] Lin Li, Jun Xiao, Guikun Chen, Jian Shao, Yueting Zhuang, and Long Chen. Zero-shot visual relation detection via composite visual cues from large language models. *Advances in Neural Information Processing Systems*, 36:50105–50116, 2023. 2

[13] Ji Lin, Hongxu Yin, Wei Ping, Pavlo Molchanov, Mohammad Shoeybi, and Song Han. Vila: On pre-training for visual language models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 26689–26699, 2024. 8

[14] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26296–26306, 2024. 1, 3, 6

[15] Muhammad Maaz, Hanoona Rasheed, Salman Khan, and Fahad Shahbaz Khan. Video-chatgpt: Towards detailed video understanding via large vision and language models. *arXiv preprint arXiv:2306.05424*, 2023. 8

[16] Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, et al. Pivot: Iterative visual prompting elicits actionable knowledge for vlms. *arXiv preprint arXiv:2402.07872*, 2024. 1

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 6

[18] Murugan Sankaradas, Ravi K Rajendran, and Srimat T Chakradhar. Streamingrag: Real-time contextual retrieval and generation framework. *arXiv preprint arXiv:2501.14101*, 2025. 1

[19] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 5

[20] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012. 3

[21] Naoki Wake, Atsushi Kanehira, Kazuhiro Sasabuchi, Jun Takamatsu, and Katsushi Ikeuchi. Gpt-4v (ision) for robotics: Multimodal task planning from human demonstration. *IEEE Robotics and Automation Letters*, 2024. 1, 8

[22] Hao Wang, Jiayou Qin, Ashish Bastola, Xiwen Chen, John Suchanek, Zihao Gong, and Abolfazl Razi. Visiongpt: Llm-assisted real-time anomaly detection for safe visual navigation. *arXiv preprint arXiv:2403.12415*, 2024. 1, 2, 8

[23] Yuetian Weng, Mingfei Han, Haoyu He, Xiaojun Chang, and Bohan Zhuang. Longvlm: Efficient long video understanding via large language models. In *European Conference on Computer Vision*, pages 453–470. Springer, 2024. 8

[24] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003. 3

[25] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers:

State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020. 6

[26] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th annual international conference on mobile computing and networking*, pages 129–144, 2018. 8

[27] Zhiqiang Yuan, Ting Zhang, Jiapei Zhang, Jie Zhou, and Jinchao Zhang. Walkvlm: Aid visually impaired people walking by vision language model. *arXiv preprint arXiv:2412.20903*, 2024. 1

[28] Haoji Zhang, Yiqin Wang, Yansong Tang, Yong Liu, Jiashi Feng, Jifeng Dai, and Xiaojie Jin. Flash-vstream: Memory-based real-time understanding for long video streams. *arXiv preprint arXiv:2406.08085*, 2024. 8