LVS: A Learned Video Storage for Fast and Efficient Video Understanding

Yunghee Lee Jongse Park
Korea Advanced Institute of Science and Technology (KAIST)
{yhlee, jspark}@casys.kaist.ac.kr

Abstract

As video understanding (VU) promises unprecedented capabilities in the era of video data explosion, its efficient computation plays a critical role in practicalizing the algorithmic innovations. While VU models often rely on powerful foundational models such as CLIP to understand visual concepts, the massive computational demand hinders their scalable deployment over real-world video data silos. To this end, this paper proposes LVS, a learned video storage system that memoizes feature vectors for the already-seen video clips and reuses them for future VU queries. The key challenge is the video's continuous nature that disallows the naïve computation reuse among VU queries for different video clips. To address this challenge, we identify a unique property in which VU-generated feature vectors form a monoid and leverage the monoid homomorphism using a multilayer perceptron (MLP) model to effectively fuse the disjoint feature vectors. Our evaluation shows that LVS achieves up to 1.59× speedup in VU query processing latency, while experiencing no significant accuracy loss in the UCF101 video classification task.

1. Introduction

As of today, video is the most common data type on the internet. A study reports that video accounts for more than 80% of the global internet traffic [16]. Video sharing platforms such as YouTube [7], Tiktok [2], and Instagram [14] are leading this era of video data explosion. Not only video data is consumed for entertaining purposes, but also it has become the primary information carrier, replacing the conventional alternatives such as text and image.

Video understanding (VU) refers to a group of machine learning algorithms in the field of computer vision that aim to extract high-level insights from raw data and enable semantic capabilities for end applications. While the algorithmic innovations promise new possibilities, practicalizing them poses a wide range of system-level challenges, which is the focus of this work.

Modern VU algorithms leverage foundational models (FM), which are designed to serve various downstream tasks, including but not limited to classification, temporal and/or spatial localization, summarization, and captioning. While there are many variations on how VU algorithms exploit the FMs, there is an algorithmic commonality that the FMs take a video clip (i.e., a collection of video frames) as the input and extract "features" from the clip. These features then become the input for the task-specific "heads" that produce the end application output.

However, the FM-driven feature extraction process requires prohibitive computation power, impracticalizing the use of VU algorithms over long video clips (e.g., minutes-long), which are common in real-world application scenarios. Thus, the common practice in the literature is to evaluate their VU techniques only using a short video clip (e.g., 16 frames). Apparently, this "performance wall" is limiting the usefulness of VU algorithms, necessitating a solution to bridge the gap between algorithmic advances and system realities.

This paper sets out to solve this problem by designing a video storage system customized for FM-based VU model inferencing, namely *Learned Video Storage (LVS)*, where the computation results (i.e., features) from past VU queries are memoized (or cached) and reused for future VU queries. The key challenge to enabling this computation reuse technique is that video is continuous data in the time dimension, and thus, a large number of clips can exist for a given video file. This is because every VU query is associated with an input video clip that has any starting and ending timestamps, engendering it nearly impossible to directly reuse features produced by a query q_1 for the feature computation of another query q_2.

To tackle this challenge, we propose a novel feature fusion technique, identifying a property in which video clips and VU-generated features form their respective monoids. Inspired by this insight, we aim to make the FM a monoid homomorphism by devising a multilayer perceptron (MLP) model that is a monoid operation for the features. Our empirical study demonstrates that the proposed MLP model can serve as a monoid operation. In other words, the MLP model is capable of effectively fusing the disjoint feature

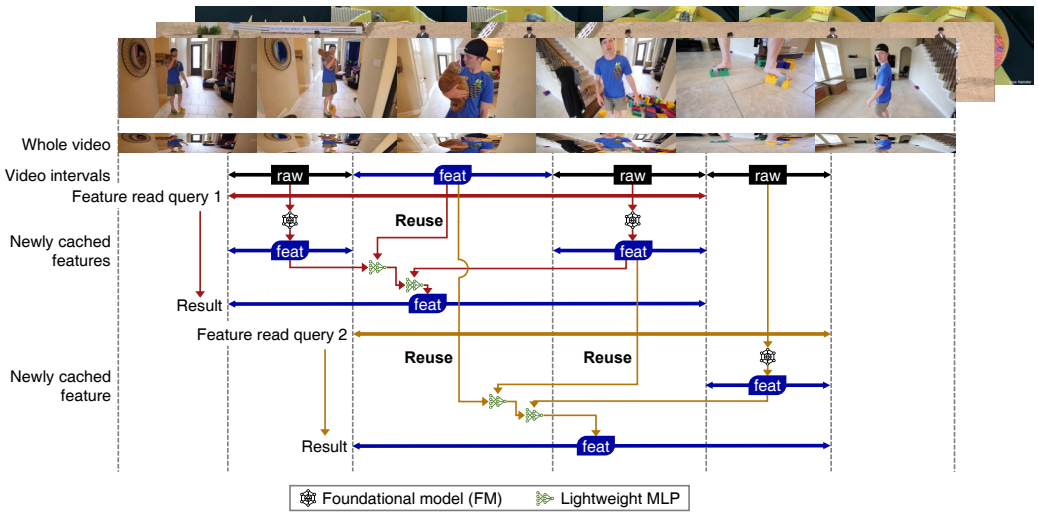

Figure 1. An overview of Learned Video Storage (LVS). When a VU query is given, rather than reading the requested clip and running the foundational model, LVS selects appropriate clips of the target video that already has cached results for previous VU queries. LVS collects the cached features and computes the feature for clips without caches. Then with the help of a lightweight multilayer perceptron (MLP) model, LVS can compute an approximate but accurate feature for the requested clip in a fast and efficient manner by reducing the foundational model usage.

vectors independently produced from different video clips to generate a feature vector for their combined video clip.

Consider the following example. Suppose there are two VU queries, q_1 and q_2, that take the input video clips v_1 and v_2, respectively. The VU system would produce two feature vectors for the queries f_1 and f_2, which are memorized (or cached) in the storage system. Suppose the system receives another query q_3 where its input video clip v_3 is the concatenation of v_1 and v_2. Then, exploiting the proposed feature fusion technique, the proposed system would be able to obtain the feature vector f_3 by *fusing* the memoized feature vectors, f_1 and f_2, through the learned MLP model, instead of inferencing the FM model for the v_3 entirely from scratch.

We evaluate the effectiveness of LVS using four different FMs used for various video understanding tasks. The evaluation utilizes real-world datasets: UCF101, MSR-VTT, and Charades. We implement LVS in Python, building upon PyTorch and SQLite, augmented with the Z3 solver. Our experimental results report that LVS achieves up to 1.59× speedup on the VU query processing time, while not imposing any accuracy loss on the end tasks. These compelling advantages highlight that LVS effectively overcome the limitations of existing systems and take an effective initial step towards the practical use of foundational model based video understanding models for real-world applications.

2. Background

2.1. Video Foundational Models

Foundational models (FMs). Modern video understanding algorithms exploit FMs as their core module, the output of which is subsequently utilized for downstream tasks [1]. As the well-annotated training data is often unavailable, these FMs are usually trained in a self-supervised manner, using multiple multimodal data concurrently. Intuitively, video data is important for VU tasks, while its effective use is relatively more challenging due to the massive scale. According to a recent study on the FMs used for VU models, there are largely two categories: (1) image-based FMs, and (2) video-native FMs. Examples of image-based FMs are CLIP [15] and FLAVA [18], while those of video-native FMs are VideoMAE [21] and InternVideo [23]. While there are discrepancies among these FMs, one algorithmic commonality is that they all rely on the attention mechanism, inspired by Vision Transformer (ViT) [4].

Prohibitive computation cost of transformer-based FMs. While transformer-based FMs show otherwise-unachievable effectiveness in VU tasks, transformers have quadratic time and memory capacity complexities with respect to the number of input tokens. Such large cost complexities become problematic especially for VU tasks, since the number of tokens is ever-increasing. As a result, inferencing FM-based VU models requires massive compute power, producing long response time, even for short video clips and datacenter-scale high-end GPUs. Although there

is various effort from machine learning community such as reformers [11], linear transformers [10], and retentive networks [20], their effectiveness remains to be fully demonstrated, indicating the need for further exploration and experimentation.

FM-produced features. Transformer-based FMs take a video clip as input and return a feature map as output. The feature map is a collection of vectors, each corresponding to a patch of the input. Therefore a feature map F is a tensor in $\mathbb{R}^{l \times c}$ where l is the number of patches and c is the number of channels [25]. In addition, ViT-based FMs produce one more vector corresponding to the class token [4, 15], originating from the natural language processing context. While the FMs are the shared components of VU models, they have task-specific *heads* for different downstream tasks. While simpler tasks such as video classification require only *one feature vector* $\mathbf{f} \in \mathbb{R}^c$, complex tasks require the *whole feature map* $F \in \mathbb{R}^{l \times c}$. The feature vector $\mathbf{f}$ can be obtained in various ways. In ViT-based FMs, the vector corresponding to the class token becomes $\mathbf{f}$. In other FMs, there exist various possible approaches such as (1) the vector of the first token $\mathbf{f} = F[0]$, (2) the average pooled feature map $\mathbf{f} = \frac{1}{l}\sum_{i=0}^{l-1} F[i]$, and (3) an aggregation with a learnable query token τ and cross attention $\mathbf{f} = \textsc{CrossAttention}(\tau, F)$ [25].

2.2. Monoids

Definition. A monoid is a structure in abstract algebra [13], defined with a tuple $(M, *)$ of a set M and a binary operation $*$. A monoid has three requirements. First, the operation must be closed on M. Second, the operation must be associative so that for any $x, y, z \in M$, $(x * y) * z = x * (y * z)$. Finally, there must exist an identity element e such that for any $x \in M$, $x * e = e * x = x$. A representative example of monoids are the set of finite strings A^* with the operation as string concatenation for a given alphabet A. Between two monoids, a function called *monoid homomorphism* preserves the structure of the monoids. Formally, when $h : M \rightarrow M'$ is a monoid homomorphism between $(M, *)$ and $(M', *')$, it satisfies $\forall x, y \in M. h(x * y) = h(x) *' h(y)$. This means the order of applying the homomorphism does not matter to the final result.

Videos and vectors as monoids. Videos and vectors inherently have a monoid structure. Videos can be seen as a finite string where the "alphabet" is the set of frames. Zero or more frames can be stacked to create a video. Video concatenation is the operation for this monoid, and the empty video with zero images is the identity element. Vectors can be considered as a monoid where vector addition is the monoid operation and $\mathbf{0}$ is the identity element.

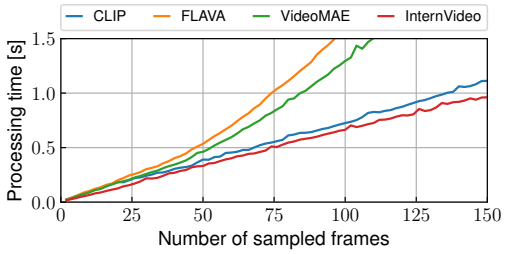

Figure 2. Processing time of FMs on videos of different length. The experiment was run on two Intel Xeon Gold 6326 CPUs and an NVIDIA GeForce RTX 3090 GPU. The results show the average processing time of 5 individual runs.

3. Motivation

Example scenario for a video understanding application. We describe an example application scenario that we assume to support from the system we propose in this work. Consider a video dataset that contains many long videos. The user wants to run downstream tasks in video clips extracted from the long videos. For instance, suppose video-serving platforms might want to classify scenes (video classification) or generate captions in various languages (video captioning). In this scenario, the user must first issue a VU query with the video file path along with start/end timestamps of the input video clip. Then, the system must trim the input video clip from the source video file according to the requested timeframe and run the foundational model to compute the requested feature.

Performance characterization of FMs. As we noted in Section 2, the computation cost for running the FM is high. Figure 2 reports the inference execution time of four different FMs used in diverse VU applications. The results demonstrate that the required processing time rapidly increases as the video clip gets longer. Arithmetically, the increased execution time originates from linear terms (from image resizing and linear layers) and quadratic terms (from attention mechanisms). The inference takes more than half a second to process a 10-second video clip with 75 sampled frames. Such a low performance is attributed to that FMs have an enormous number of parameters and layers (e.g., VideoMAE [21] with 12 transformer layers for its encoder). This high latency of the FM is suboptimal in terms of user experience.

4. Design

Design principle. To address the performance challenge, we design a novel video storage system customized for FM-based VU model inferencing, namely *Learned Video Storage (LVS)*. In designing LVS, the core design principle

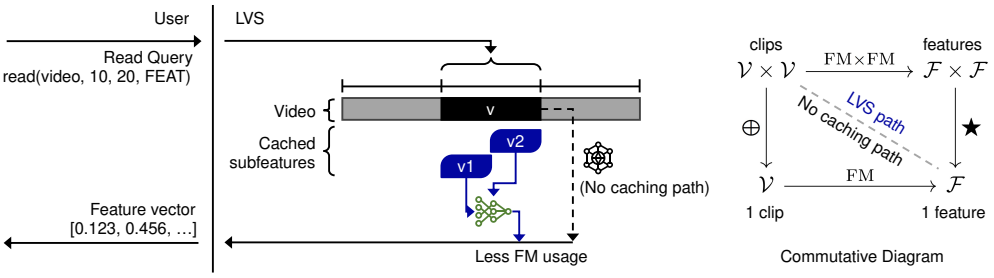

Figure 3. Design of LVS. When LVS receive a query from the user, it selects the total video v and finds out subfeatures for subclips v_1 and v_2. The subfeatures are passed through the feature fusion model to produce the result feature. If there was no such caching mechanism, the foundational model usage must have been increased. The commutative diagram on the right shows that even though the evaluation paths for LVS and the no caching case are different, they must return same results (approximately same in reality since MLPs are used).

is to minimize the number of frames that FMs must scan through and use for the inferencing. This strategy is effective for both image-based and video-native FMs, since their computational cost increases when the number of frames increases. Building upon the principle, LVS caches the already-computed features for partial video clips in the storage system and reuses the features for future queries over video clips that contain the seen video clips as subclips. In this paper, we will refer to the memoized features as "subfeatures". We develop a feature fusion technique to allow computation reuse by combining the subfeatures with the newly computed features for the unseen part of the queried video clip.

Design challenge. The challege in devising LVS is that computing the total feature from subfeatures is not trivial. As transformers compute the self-attention between all tokens, there exist algorithmic dependencies among tokens that are far away from each other, engendering it nontrivial to apply simple fusion techniques concatenation or averaging. Thus, there is a need for a novel feature fusion solution. To this end, we seek an opportunity by employing a multilayer perceptron model as a monoid operation, inspired by the insight that video data inherently have high internal redundancy [21], and therefore, it is likely to be effectively approximable from the subfeatures.

4.1. Feature Fusion Model

Leveraging commutative property. In this section, the proposed feature fusion method is formalized by showing the required property. Let the set of images be I so that $\mathcal{V} = I^*$ is the set of videos. Also, let the set of features be $\mathcal{F}$, either feature vectors $\mathbb{R}^c$ or feature maps $(\mathbb{R}^c)^*$. Here, $(\mathbb{R}^c)^*$ is used instead of $\mathbb{R}^{l\times c}$ to cover cases with different l values. For the simplest case of a clip with two subclips, the task of LVS could be recognized as a function $\mathrm{read} : \mathcal{V} \times \mathcal{V} \to \mathcal{F}$. The naive way to implement this function is to first concatenate the clips and then run the FM: $\mathrm{read}(v_1, v_2) = \textsc{fm}(v_1 \oplus v_2)$ where $\oplus$ denotes concatenation. On the other hand, if there exists an appropriate feature fusion method $\star : \mathcal{F} \times \mathcal{F} \to \mathcal{F}$, we have $\mathrm{read}(v_1, v_2) = \textsc{fm}(v_1) \star \textsc{fm}(v_2)$. To compute accurate feature fusion results, it would desirable for the two to be identical:

$$\textsc{fm}(v_1 \oplus v_2) = \textsc{fm}(v_1) \star \textsc{fm}(v_2). \tag{1}$$

This means that FM should be a monoid homomorphism between the monoids $(\mathcal{V}, \oplus)$ and $(\mathcal{F}, \star)$ so that the commutative diagram illustrated in Figure 3 commutes. Also, $(\mathcal{F}, \star)$ being a monoid should naturally cover the case of multiple subclips by associativity.

Vector space structure. Now, the remained research question would be how actually to develop $\star$. $\star$ is a monoid operation for $\mathcal{F}$, so it is possible to search one from the structure of $\mathcal{F}$. First, the set of feature vectors $\mathbb{R}^c$ is a vector space. Considering that vector space axioms for addition include the monoid axioms, $\mathbb{R}^c$ automatically satisfies the axioms of monoids with vector addition. Thus, $\mathbf{f}_1 \star \mathbf{f}_2 = \mathbf{f}_1 + \mathbf{f}_2$ is a choice that satisfies the monoid axioms. On the other hand, the set of feature maps $(\mathbb{R}^c)^*$ consists of strings, so concatenation is a choice: $F_1 \star F_2 = F_1 \oplus F_2$. However, this simple approach does not work as mentioned. Empirically, just adding or concatenating the features is unable to produce the feature for the query over the entire video.

Multilayer perceptron (MLP) as a monoid operation. As an alternative, we propose using a MLP as the monoid operation for feature vectors. Given a dataset of $\mathbf{f}_1, \mathbf{f}_2$ and the total feature $\mathbf{f}$, it is possible to train the MLP to learn how to run feature fusion, considering the internal semantics of $\mathbf{f}_1$ and $\mathbf{f}_2$. In this case, the FM can approximate a monoid homomorphism between the video monoid and the vector monoid with MLP as the operation, so that

$$\textsc{fm}(v_1 \oplus v_2) \simeq \textsc{mlp}(\textsc{fm}(v_1), \textsc{fm}(v_2)). \tag{2}$$

The MLP will internally concatenate the two vectors into one and push it through a number of fully connected layers and activation functions. Table 1 provides the detailed model specification and training hyperparameters we used for the MLP model.

Activation function	ReLU
Fully connected layer count	2, 3, 4
Dimension of hidden layer(s)	2048
Shape of input feature vectors	$(C), (C)$
Shape of output feature vector	(C)
Optimizer	AdamW
Base learning rate	5e-5
Weight decay	0.0
Optimizer momentum	$\beta_1 = 0.9, \beta_2 = 0.999$
Learning rate scheduler	Linear decay
Warmup epochs	500
Total epochs	5000

Table 1. Model specification of the evaluated MLP. C is the number of the output channels of the FM.

Adding an average term. As v_1 and v_2 are already from a consequent video source and neighboring each other, their semantics are quite close in most cases. Therefore, the concatenated video v will have similar semantics as well. This means $\mathbf{f}_1 \approx \mathbf{f} \approx \mathbf{f}_2$, implying that averaging the input vectors could be potentially helpful for MLP model to be effectively trained. This is because MLP models would be able to focus on learning the additional semantic effects that emerge from concatenation. The new model then becomes

$$\text{MLPWITHAVERAGE}(\mathbf{f}_1, \mathbf{f}_2) = \frac{\mathbf{f}_1 + \mathbf{f}_2}{2} + \text{MLP}(\mathbf{f}_1, \mathbf{f}_2). \quad (3)$$

In this paper, we refer to this variation of the model as "MLP + average".

Adaptation to feature maps. While this MLP-based approach can approximate feature "vectors", approximating feature "maps" requires additional steps since the feature map has more than one vector, each corresponding to a patch of the input video. A simple MLP-based approach to this task first applies average pooling to each input feature map to get

$$\bar{\mathbf{f}}_1 = \frac{1}{l_1} \sum_{i=0}^{l_1-1} F_1[i], \quad \bar{\mathbf{f}}_2 = \frac{1}{l_2} \sum_{i=0}^{l_2-1} F_2[i]. \quad (4)$$

Then, the pooled outputs are passed through the MLP with each of the vectors in the subfeature maps to compute the elements of the total feature map, while using concatenation as a baseline. The resulting feature map becomes

$$F[i] = \begin{cases} F_1[i] + \text{MLP}(F_1[i], \bar{\mathbf{f}}_2), & \text{if } i < l_1 \\ F_2[i - l_1] + \text{MLP}(\bar{\mathbf{f}}_1, F_2[i - l_1]), & \text{otherwise.} \end{cases} \quad (5)$$

In this paper, we refer to this variation of the model as "MLP with pooling".

4.2. Design of Learned Video Storage

Exploiting the afore-described feature fusion method, we build LVS, as depicted in Figure 3. Inspired from VSS [5], LVS can serve read queries by using cached subclips. VU queries can be processed quickly and efficiently by using cached subfeatures. When a user application requests for the feature of a timeframe, LVS processes the query and searches for opportunities in which cached subfeatures can be utilized to serve the feature approximately yet quickly. When multiple possibly overlapping caches exists, it calls an external optimizer to find out which cache it should use for least expected processing time.

Storing features for reuse. When multiple features are required to be fused together, LVS calls the feature fusion method from left to right to reduce the number of features, similar to the behavior of *reduce* in functional programming languages. In this process, intermediate features are created every time the feature fusion method is invoked. Here, as LVS can only benefit from caches of the entirely enclosed subclips, feature caching is more effective and expected to be used for shorter video clips. Therefore, LVS only caches the features directly produced from the FM, while not caching the intermediate feature created from the feature fusion method. The intermediate features requires storage usage quadratic to the number of features to cache and are not frequently usable than the directly produced features.

Cost estimation model. To call the external optimizer library, LVS requires an expecting function that maps the selection of the caches to the expected processing time. The function used by LVS for VU queries is

$$\text{COST}(\{c_1, c_2, \cdots, c_n\}) = rn + \sum_{i=1}^{n} m_i l_i \quad (6)$$

where c_i is the i-th subclip being used,

$$m_i = \begin{cases} 1, & \text{if } c_i \text{ needs decoding and FM} \\ 0, & \text{if } c_i \text{ is already saved as feature,} \end{cases} \quad (7)$$

l_i is the length, or the number of frames in the i-th subclip, and r is a constant factor. This design of the cost function estimates how much frames should be decoded and

processed by the FM. The multiplier m_i removes the cost for cached subclips since LVS does not have to run the FM on it. On the other hand, the constant factor r could be considered as the cost of one run of the feature fusion operator in the units of the processed frame. As the feature fusion operator reduces the number of features by one, the operator must be run for $n-1$ times. Ignoring constants, the rn term gives the total cost of the feature fusion operator. For experiments, LVS uses $r = 10$.

5. Evaluation

5.1. Methodology

We implement LVS using Python, PyTorch, the SQLite database, and the Z3 Solver.

Foundational models. We use CLIP [15], FLAVA [18], VideoMAE [21], and InternVideo [23] as the foundational model. We re-implement these existing FMs to adapt to videos with various lengths, following the early-fusion method of [25] for image-native FMs, which concatenates tokens of all frames just after the projection layer.

Datasets and task. For the video dataset, we use UCF101 [19], MSR-VTT [24], and Charades [17]. To see the effect of LVS on real-world VU application, we use a video classification task. This application classifies UCF101 videos into 101 classes defined in the UCF101 dataset. We run benchmarks on the whole system with the 'blueboy' video from the Long Videos dataset [12].

System specification. The evaluated system is equipped with two Intel Xeon Gold 6326 CPUs and one NVIDIA Geforce RTX 3090 GPU.

5.2. Experimental Results

Our evaluation results are three-fold. (1) We first train and test the feature fusion method (i.e., monoid operation) using the features obtained from the video dataset and the FM. Each video in the dataset is split into clips with random neighboring timeframes, and passed through the FM to obtain a dataset of $\text{FM}(v)$, $\text{FM}(v_1)$, and $\text{FM}(v_2)$. We train the proposed MLP, MLP + Average, and MLP with pooling models upon this feature dataset. We select cosine embedding loss as the loss function for feature approximation, as it is typically used as a loss function for training FMs. (2) We test the train feature fusion method on real-life tasks. In particular, we use a classification task on 101 classes of UCF-101. We train a specialized task head with the whole feature vector $\mathbf{f}$, and compare the accuracies of using the whole feature vector $\mathbf{f}$ to the approximated feature vector $\mathbf{f}_1 \star \mathbf{f}_2$. (3) We run the LVS system to find out how much performance benefit it provides. To see the effects of the cache, we test the performance of LVS using sequences of feature read queries with different spatial locality.

Feature vector approximation. Figure 4 demonstrates the result of the MLP-based feature fusion methods for feature vector approximation. The figure contrasts the cosine embedding losses of three different approaches: average, plain MLP, and MLP + average models. The bars show that for the equivalent FM and dataset, the loss is lower in MLP + average models. However, the results also show that the effect of more layers or parameters is insignificant. The loss does not significantly improve and even suffer from overfitting when we use more layers. In case of InternVideo and MSR-VTT, merely using the average offers the lowest loss value, while the MLP + average model's loss is nearly identical to that of the average. In summary, the MLP + average model provides the optimal approximation for the total feature vector in most cases, while demonstrating the limited effect of larger MLP model size.

Feature map approximation. Figure 5 shows the training loss result of feature fusion methods for feature map approximation. The results emphasize the disparities between the mean cosine embedding losses of two different approaches: plain concatenation, and the MLP with pooling model. The bars exhibit that the loss is the lowest when using the MLP with pooling model with two layers. Similar to feature vectors, while the MLP-based model outperforms compared to the simple operations, the model does not gain significant benefit from adding additional layers.

End task application. Figure 6 shows the accuracy when the approximated feature vectors are used for real tasks. For the UCF-101 classification task, the approximations from the feature fusion methods offer comparable accuracy with the total feature vectors. Especially, for image-based FMs such as CLIP and FLAVA, the use of approximation models provides even higher accuracy in comparison with the total feature vector. As MLP-based models discriminate between the first and second input vector by its position, the resulting feature approximation can possibly include additional temporal context to the output to achieve better accuracy.

End-to-end system evaluation. Figure 7 shows real-world benchmark results using LVS. We train the used feature fusion model using 2-layer MLP + average model from the CLIP FM and the Charades dataset, because it experiences the lowest loss value in the selected FM and dataset. Using the 'blueboy' video re-encoded into 7,218 frames, the simulated user application requests features for 150 frames in random positions for 200 times. We sample the position of the frames from a Gaussian distribution with a fixed

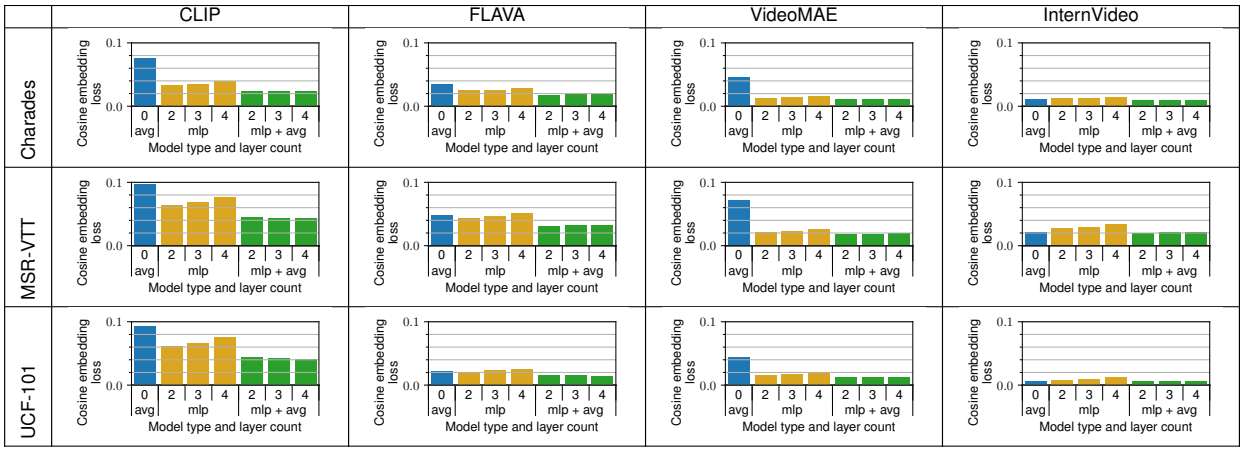

Figure 4. Loss of approximation functions (lower is better) for feature vector approximation. Each bar graph indicates the cosine embedding loss of different types of feature fusion methods that approximate the total feature $\mathbf{x}$ from the subfeatures $\mathbf{x}_1$ and $\mathbf{x}_2$. MLPs and MLP + average models with 2 to 4 layers are tested. For comparison, the most left bar shows the cosine embedding loss when the average function is used instead of a learned model. The graphs are organized with the used FMs and datasets.

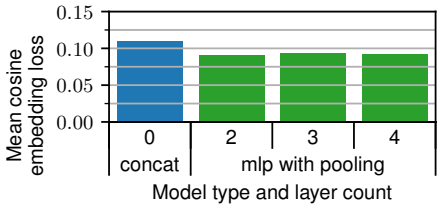

Figure 5. Loss of approximation functions (lower is better) for feature map approximation. Due to the large size of feature maps, we demonstrate the results only for the CLIP foundational model and the Charades dataset. The bars indicate the mean cosine embedding loss for MLP with pooling models with 2 to 4 layers that approximate the total feature map X from the subfeature maps X_1 and X_2. For comparison, the most left bar presents the mean cosine embedding loss when plain concatenation is used instead of a learned model.

σ value or a uniform distribution over the whole video. Small σ values represent the cases with higher chance of queries to nearby clips and high spatial locality. In contrast, large σ values or using the uniform distribution are for the cases with reduced possibilities of such queries and lower spatial locality. The results show that as more caches are accumulated, LVS experience the reduced execution time required for decoding and FM. The average speedup through 200 queries are 1.59×, 1.30×, 1.16× in each experiment, demonstrating that the higher locality offers the higher speedup. On the other hand, when there are insufficient cache features, LVS see marginal speedup gains in contrast to the case without any caches.

The results also exhibit the larger fluctuation of the processing time data that suggest the high locality provides significant speedup for certain queries that the cached subfeatures can entirely cover the query, while LVS never needs to call the FM. We observe that such queries can achieve speedup up to a maximum of 14×, whereas the average speedup is only 1.59× for the case with high locality.

Although the execution time required for decoding and FM keeps decreasing as the number of cached features increases, the latency of the optimization process appears to be increased. When the locality is high, the number of cached features is high, producing the execution time for the optimizer to be increased faster. As a result, the total processing time of LVS can even exceed the processing time of the no-caching case. We expect that limiting the number of candidate features fed to the optimizer or implementing LRU to drop old caches could help reduce the time spent for the optimizer.

6. Related Work and Discussion

Systems for video understanding. As the size of video data has grown exponentially, efficient processing has become an important problem in terms of heat, power usage, and carbon emission. To reduce the required computation for queries on massive size of video data, recent studies have explored aspects of the video analytic systems. NoScope [8] constructs an inference-optimized specialized model to efficiently produce outputs when *a priori* knowledge about the query is given. BlazeIt [9] focuses on aggregation queries and cardinality-limited queries and utilizes neural networks to compute an unbiased estimator of the answer. CoVA [6] captures moving objects from the compressed form of videos to run DNNs only for necessary frames and overcome the decoding bottleneck. TVM [26] splits videos into semantic tiles and constructs an index of

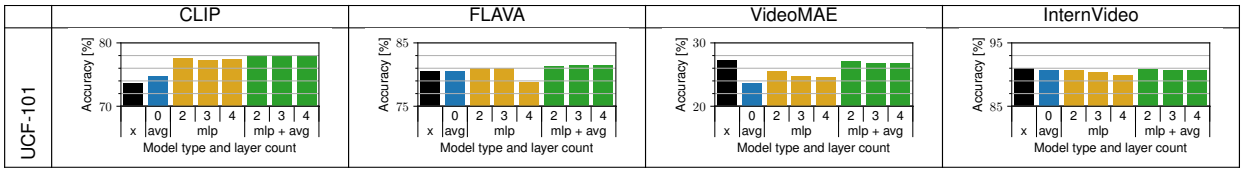

Figure 6. Accuracy (higher is better) for the UCF-101 classification task for different feature fusion methods. Inside each graph, while the leftmost "x" bar indicates the accuracy of using the original total feature vector, other bars show the results when approximation methods are used similar to Fig. 4. The graphs are organized into a table to show which FM was used to produce the feature vectors.

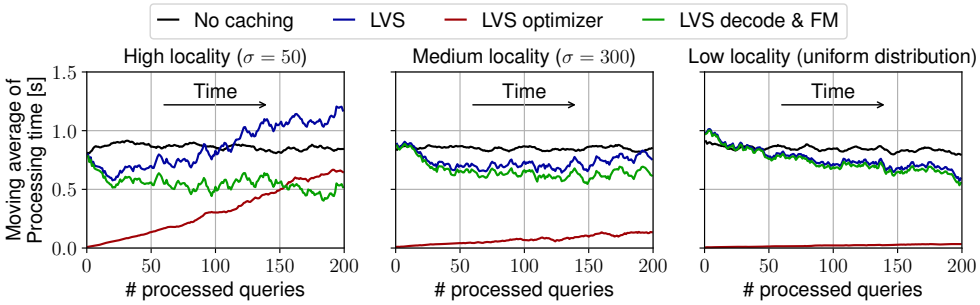

Figure 7. Benchmark result of a LVS usage scenario. Three experiments were run with query sequences with high, medium, and low spatial locality. Each graph show the processing time of each query when LVS is continuously queried by a user application. The moving average of maximum 10 preceding samples are used to hide the fluctuation of processing time from randomly generated queries. The total processing time is the sum of the time spent on the external optimizer (LVS optimizer) and the time spend on decoding and FM (LVS decode & FM). For comparison, the same queries were run for the case without any caching benefit from LVS and the processing time is demonstrated together.

the tiles to reduce decoding and processing of large videos. These recent works focus on a single query and either run on *a priori* known types of queries or remove parts of the video to reduce computation load. In contrast, LVS does not require knowledge about the user application and does not miss frames or tiles that can possibly have additional information. Rather, as LVS focuses on scenarios where multiple queries exist, the proposed technique is complementary to the techniques proposed in prior work to achieve even further acceleration.

Feature fusion. The concept of fusing different feature vectors together is originated from the usage in multi-modal models. Multi-modal models collect data from different dimensions from the same object to achieve better performance. As different data types require different models and give different feature vectors, it is nontrivial to fuse the feature vectors from different sources. Weighted averages or more complicated models [3, 22] can be used. However, the fusion of features from various modalities does not take advantage of the relationships between adjacent videos as in LVS, necessitating the use of more complex models that rely on attention or convolution mechanisms, rather than a straightforward MLP.

7. Conclusion

This paper introduces LVS, a video storage system designed to quickly and efficiently serve video features, using a model that memorizes feature vectors from previous queries for quicker calculations if cached features of the subclips are available. Our experimental findings indicate that the proposed model structure, which treats the feature space as a monoid and uses lightweight multilayer perceptron models for semantics-preserving operations, produces precise feature vectors without the need to run the foundational model.

Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) (No.2024-00396013, No.2022-0-01037, No.2018-0-00503) under the Graduate School of Artificial Intelligence Semiconductor (IITP-2024-RS-2023-00256472), Information Technology Research Center (ITRC) support program (IITP-2024-2020-0-01795), and Artificial Intelligence Graduate School Program (KAIST) (No.2019-0-00075), funded by the Korea government (MSIT).

References

[1] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. 2

[2] ByteDance. Tiktok. https://www.tiktok.com/, 2016. 1

[3] Yimian Dai, Fabian Gieseke, Stefan Oehmcke, Yiquan Wu, and Kobus Barnard. Attentional feature fusion. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 3560–3569, 2021. 8

[4] 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. 2, 3

[5] Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. VSS: A storage system for video analytics. In *Proceedings of the 2021 International Conference on Management of Data*, pages 685–696, 2021. 5

[6] Jinwoo Hwang, Minsu Kim, Daeun Kim, Seungho Nam, Yoonsung Kim, Dohee Kim, Hardik Sharma, and Jongse Park. CoVA: Exploiting Compressed-Domain analysis to accelerate video analytics. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 707–722, 2022. 7

[7] Alphabet Inc. Youtube. https://www.youtube.com/, 2005. 1

[8] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. NoScope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529*, 2017. 7

[9] Daniel Kang, Peter Bailis, and Matei Zaharia. BlazeIt: Optimizing declarative aggregation and limit queries for neural network-based video analytics. *arXiv preprint arXiv:1805.01046*, 2018. 7

[10] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020. 3

[11] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020. 3

[12] Yongqing Liang, Xin Li, Navid Jafari, and Jim Chen. Video object segmentation with adaptive feature bank and uncertain-region refinement. *Advances in Neural Information Processing Systems*, 33:3430–3441, 2020. 6

[13] Rudolf Lidl and Günter Pilz. *Applied Abstract Algebra*, pages 331–378. Springer US, New York, NY, 1984. 3

[14] Meta Platforms. Instagram. https://www.instagram.com/, 2010. 1

[15] 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. 2, 3, 6

[16] Rohit Shewale. Video marketing statistics in 2024 (usage, roi & more). https://www.demandsage.com/video-marketing-statistics, 2024. 1

[17] Gunnar A Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 510–526. Springer, 2016. 6

[18] Amanpreet Singh, Ronghang Hu, Vedanuj Goswami, Guillaume Couairon, Wojciech Galuba, Marcus Rohrbach, and Douwe Kiela. FLAVA: A foundational language and vision alignment model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15638–15650, 2022. 2, 6

[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. 6

[20] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023. 3

[21] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. VideoMAE: Masked autoencoders are data-efficient learners for self-supervised video pre-training. *Advances in neural information processing systems*, 35:10078–10093, 2022. 2, 3, 4, 6

[22] Yikai Wang, Xinghao Chen, Lele Cao, Wenbing Huang, Fuchun Sun, and Yunhe Wang. Multimodal token fusion for vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12186–12195, 2022. 8

[23] Yi Wang, Kunchang Li, Yizhuo Li, Yinan He, Bingkun Huang, Zhiyu Zhao, Hongjie Zhang, Jilan Xu, Yi Liu, Zun Wang, et al. InternVideo: General video foundation models via generative and discriminative learning. *arXiv preprint arXiv:2212.03191*, 2022. 2, 6

[24] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. MSR-VTT: A large video description dataset for bridging video and language. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5288–5296, 2016. 6

[25] Liangzhe Yuan, Nitesh Bharadwaj Gundavarapu, Long Zhao, Hao Zhou, Yin Cui, Lu Jiang, Xuan Yang, Menglin Jia, Tobias Weyand, Luke Friedman, et al. VideoGLUE: Video general understanding evaluation of foundation models. *arXiv preprint arXiv:2307.03166*, 2023. 3, 6

[26] Tianxiong Zhong, Zhiwei Zhang, Guo Lu, Ye Yuan, Yu-Ping Wang, and Guoren Wang. TVM: A tile-based video management framework. *Proceedings of the VLDB Endowment*, 17 (4):671–684, 2023. 7