MeMViT: Memory-Augmented Multiscale Vision Transformer for Efficient Long-Term Video Recognition

Chao-Yuan Wu *, 1 Yanghao Li *, 1 Karttikeya Mangalam 1, 2 Haoqi Fan 1 Bo Xiong 1 Jitendra Malik 1, 2 Christoph Feichtenhofer *, 1

*equal technical contribution

1Facebook AI Research 2UC Berkeley

Abstract

While today’s video recognition systems parse snapshots or short clips accurately, they cannot connect the dots and reason across a longer range of time yet. Most existing video architectures can only process <5 seconds of a video without hitting the computation or memory bottlenecks.

In this paper, we propose a new strategy to overcome this challenge. Instead of trying to process more frames at once like most existing methods, we propose to process videos in an online fashion and cache “memory” at each iteration. Through the memory, the model can reference prior context for long-term modeling, with only a marginal cost. Based on this idea, we build MeMViT, a Memory-augmented Multiscale Vision Transformer, that has a temporal support 30× longer than existing models with only 4.5% more compute; traditional methods need >3,000% more compute to do the same. On a wide range of settings, the increased temporal support enabled by MeMViT brings large gains in recognition accuracy consistently. MeMViT obtains state-of-the-art results on the AVA, EPIC-Kitchens-100 action classification, and action anticipation datasets. Code and models will be made publicly available.

1. Introduction

Our world evolves endlessly over time. The events at different points in time influence each other and all together, they tell the story of our visual world. Computer vision promises to understand this story, but today’s systems are still quite limited. They accurately parse visual content in independent snapshots or short time periods (e.g., 5 seconds), but not beyond that. So, how can we enable accurate long-term visual understanding? There are certainly many challenges ahead, but having a model that practically runs on long videos is arguably an important first step.

In this paper, we propose a memory-based approach for building efficient long-term models. The central idea is that instead of aiming to jointly process or train on the whole long video, we simply maintain “memory” as we process a video in an online fashion. At any point of time, the model has access to prior memory for long-term context. Since the memory is ‘reused’ from the past, the model is highly efficient. To implement this idea, we build a concrete model called MeMViT, a Memory-augmented Multiscale Vision Transformer. MeMViT processes 30× longer input duration than existing models, with only 4.5% more compute. In comparison, a long-term model built by increasing the num-

Figure 1. MeMViT is a class of video models that models long videos efficiently. It has a significantly better trade-off than traditional methods, which increase the temporal support of a video model by increasing the number of frames in model input (Fig. 1a). MeMViT achieves efficient long-term modeling by hierarchically attending the previously cached “memory” of the past (Fig. 1b).
ber of frames will require >3,000% more compute. Fig. 1a presents the trade-off comparison in compute/duration.

More concretely, MeMViT uses the “keys” and “values” of a transformer [68] as memory. When the model runs on one clip, the “queries” attend to an extended set of “keys” and “values”, which come from both the current time and the past. When performing this at multiple layers, each layer attends further down into the past, resulting in a significantly longer receptive field, as illustrated in Fig. 1b.

To further improve the efficiency, we jointly train a memory compression module for reducing the memory footprint. Intuitively, this allows the model to learn which cues are important for future recognition and keeps only those.

Our design is loosely inspired by how humans parse long-term visual signals. Humans do not process all signals over a long period of time at once. Instead, humans process signals in an online fashion, associate what we see to past memory to make sense of it, and also memorize important information for future use.

Our results demonstrate that augmenting video models with memory and enabling long range attention is simple and very beneficial. On the A V A spatiotemporal action localization [31], the EPIC-Kitchens-100 action classification [13,14], and the EPIC-Kitchens-100 action anticipation datasets [13,14], MeMViT obtains large performance gains over its short-term counterpart and achieves state-of-the-art results. We hope these results are helpful for the community and take us one step closer to understanding the interesting long story told by our visual world.

2. Related Work

Video understanding models aim to parse spatiotemporal information in videos. Popular approaches in the past decade include the classic works that use handcrafted features [12,16,20,35,38,52,69–71], recurrent networks [17,33,41,44,49,61], and 2D- [72,73,79] or 3D-CNNs [3,23,24,27,44,44,53,63,66,67,75,80,83]. More recently, methods built upon the Transformer architecture (the vision transformers) have been shown promising results [2,3,22,48,51].

Vision transformers [2, 18, 19, 22, 30, 47, 64, 65, 81] treat an image as a set of patches and model their interactions with transformer-based architectures [68]. Recent works adding vision priors such as multi-scale feature hierarchies [22,30,47,74,81] or local structure modeling [9,18,47] have shown to be effective. They have also been generalized from the image to video domain [3,22,48,51]. In this work, we build our architecture based on the Multi-scale Vision Transformer v2 (MViTv2) architecture [22,43] as a concrete instance, but the general idea can be applied to other ViT-based video models.

Long-term video models aim to capture longer-term patterns in long videos (e.g., >30 seconds). To reduce the high computational cost, one widely studied line of work directly models pre-computed features without jointly training backbones [1,17,29,78,82]. Another potential direction designs efficient models [32,37,45,79,83,85] to make covering more frames feasible. More related to our work is the less-studied middle ground that builds a memory-like design that still allows for end-to-end training but has greatly reduced overhead [8,39,40,77]. For example, “long-term feature bank”-based methods extend standard video backbones to reference long-term supportive context features [50,77]. However, these methods capture only final-layer features and require two backbones, two rounds of training and inference computation. MeMViT flexibly models features at arbitrary layers with minimal changes to standard training methods and only requires one standalone backbone.

Online video modeling arises naturally in applications such as robotics, AR/VR, or video streaming. While one may use an image-based method (e.g., [57]) to parse a video frame-by-frame, to consider longer-term context, most existing works use causal convolutions [6,10,36], RNNs [17,46], or feature fusion [8,84]. In this work, we explore attention-based designs, which directly reference arbitrary points of time in the past, without the need to fight forgetfulness as in RNNs or being constrained by kernel size as in CNNs.

Transformer designs in NLP are also related to our method. MeMViT takes inspiration from long-range language models [11,55,56,59,60], which also cache long-range “memory”. Different from these works, video models process significantly larger tensors (T×W×H), making caching and attending memory expensive if not prohibitive. Prior work in NLP attempts to learn a module to compress memory, but the requirement of backpropagation through time (BPTT) makes it challenging [55]. Rae et al. [55] thus uses autoencoder for memory compression, but that cannot be optimized for the end task. In this paper, we present a “pipelined” memory compression method that is efficient and end-to-end optimizable for the end task, without BPTT.

3. Preliminaries

In this paper, we build MeMViT based on the MViTv2 [22,43] architecture due to its strong performance, but the techniques presented in this paper can be applied to other ViT-based architectures. For completeness, we review ViT and MViT and introduce notations next.
**Vision Transformers (ViT)** first embeds an image into \( N \) non-overlapping patches (using a strided convolution) and packs them into a tensor \( X^0 \in \mathbb{R}^{N \times d} \). A stack of transformer layers then models the interactions among these patches. The central component of a transformer layer is the attention operation, which first linearly projects an input tensor \( X \) to be queries \( Q \), keys \( K \), and values \( V \):\(^{2}\)

\[
Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \quad (1)
\]

and performs a self-attention operation

\[
Z := \text{Attn}(Q, K, V) = \text{Softmax} \left( \frac{QK^T}{\sqrt{d}} \right)V, \quad (2)
\]

to obtain an output tensor \( Z \in \mathbb{R}^{N \times d_w} \).

**Multiscale Vision Transformers (MViT)** improves ViT based on two simple ideas. First, instead of having a fixed resolution of \( N \) throughout the network, MViT learns multiscale representations through multiple stages, starting from fine-grained modeling of smaller patches (with large \( N \) and small \( d \)) to high-level modeling of larger patches in later stages (with small \( N \) and large \( d \)). The transition between stages is done through strided pooling. Second, MViT uses pooling attention (\( P \)) that pools spatiotemporal dimensions of \( Q, K, \) and \( V \) to drastically reduce computational cost of an attention layer, i.e.,

\[
Q = P_Q \left( XW_Q \right), \quad K = P_K \left( XW_K \right), \quad V = P_V \left( XW_V \right). \quad (3)
\]

These two changes significantly improve the model performance and efficiency. In this paper, we build our method based on a slightly modified MViT, where we swap the order of linear layer and pooling:

\[
\hat{Q} = P_Q \left( X \right), \quad \hat{K} = P_K \left( X \right), \quad \hat{V} = P_V \left( X \right) \quad (4)
\]

This allows the linear layer to operate on smaller tensors, reducing the computational cost without affecting accuracy. See Supplementary Material for an ablation study on this change. In the next section, we will see how this change also makes MeMViT more efficient.

To build longer duration video models, most state-of-the-art methods simply increase the number of frames in the input clip [22, 24, 75]. This strategy increases the computational cost significantly. In the next section, we present our method for building more efficient long-term video models.

### 4. MeMViT for Efficient Long-Term Modeling

Our method is simple. We split a video into a sequence of short \( T \times H \times W \) clips and process them sequentially (for both training and inference). Consecutive iterations see consecutive clips. We cache “memory”, some representations of the processed clip, at each iteration. When processing the current clip at time step \( t \), the model has access to previously cached ‘memory’ from earlier iterations \( t' < t \) for long-term context. Fig. 2 shows an overview.
4.1. Memory Attention and Caching

The Basic MeMViT Attention. One simple way to implement this idea is to treat the “keys” $\hat{K}$ and “values” $\hat{V}$ in the transformer architecture as a form of memory, and extend $\hat{K}(t)$ and $\hat{V}(t)$ at current iteration $t$ to include $\hat{K}(t')$ and $\hat{V}(t')$ cached from earlier iterations $t'$ from $t - M$ to $t - 1$, i.e.,

$$
\hat{K}(t) := \left[ \text{sg} \left( \hat{K}^{(t-M)} \right), \ldots, \text{sg} \left( \hat{K}^{(t-1)} \right), \hat{K}(t) \right],$$

$$
\hat{V}(t) := \left[ \text{sg} \left( \hat{V}^{(t-M)} \right), \ldots, \text{sg} \left( \hat{V}^{(t-1)} \right), \hat{V}(t) \right],$$

where the square brackets denote concatenation along the token dimension. With this formulation, the query $Q$ attends not only information about the current time step $t$, but also information from up to $M$ steps before.³ Here, the “stop gradient” operator ($\text{sg}$) breaks further dependency into the past in backpropagation. Note that the memory is built hierarchically over time (see Fig. 1b) and our previous key and value memory holds information stored from prior time-steps.

The additional cost for training and inference encompasses only the GPU memory for memory caching and the extra compute in the extended attention layer. All other parts of the network (MLPs, etc.) remain unchanged. The cost grows with temporal support in $O(M)$, instead of $O(T^2)$ as in traditional scaling methods.

In this basic implementation, we cache the full key and value tensors, which may contain redundant information that is not useful for future recognition. In the next section we will discuss methods to compress memory for keeping only ‘important’ information.

4.2. Memory Compression

Naïve Memory Compression. There are many potential ways to compress the memory, but one intuitive design attempts to jointly train compression modules (e.g., learnable pooling operators), $f_K$ and $f_V$, to reduce the spatiotemporal size of $K$ and $V$ tensors, respectively:

$$
\hat{K}(t) := \left[ f_K \left( \text{sg}(\hat{K}^{(t-M)}) \right), \ldots, f_K \left( \text{sg}(\hat{K}^{(t-1)}) \right), \hat{K}(t) \right],$$

and similarly for $\hat{V}(t)$. With this design, we only need to cache and attend the ‘compressed’ memory, $f_K \left( \hat{K}(t') \right)$ and $f_V \left( \hat{V}(t') \right)$, at inference time, thus reducing the memory footprint and the computational cost. Nonetheless, at training time, it needs to jointly train on all the ‘full’ memory tensor, thus which may actually increase the memory

³Note that we operate on $\bar{K}$ and $\bar{V}$ instead of $K$ and $V$ so that the following linear layer will transform the features before the attention operation. In preliminary experiments we find this to perform better.

Algorithm 1 Pseudocode of MeMViT attention in a PyTorch-like style.

```python
class MemViTAttention():
    # pool_q, pool_k, pool_v: pooling layers
    # lin_q, lin_k, lin_v: linear layers
    # f_k, f_v: compression modules
    self.m_k = []  # cached memory keys
    self.m_v = []  # cached memory values
    self.max_len = max memory length

    def forward(x):
        q, k, v = pool_q(x), pool_k(x), pool_v(x)
        cm_k = f_k(m_k[-1])
        cm_v = f_v(m_v[-1])

        # compress memory
        self.m_k[-1] = cm_k.detach()
        self.m_v[-1] = cm_v.detach()

        # cache current uncompressed memory
        self.m_k.append(k.detach())
        self.m_v.append(v.detach())

        # maintain max length for memory
        if len(self.m_k) > self.max_mem:
            self.m_k = [self.m_k[-1]] + [cm_k, k]
            self.m_v = [self.m_v[-1]] + [cm_v, v]

        # perform attention on augmented keys and values
        z = attn(
            lin_q(q),
            lin_k(cat(self.m_k[-1] + [cm_k, k]),
            lin_v(cat(self.m_v[-1] + [cm_v, v]),
        )

        # cache newly compressed memory
        self.m_k[-1] = cm_k.detach()
        self.m_v[-1] = cm_v.detach()

        # maintain max length for memory
        if len(self.m_k) > self.max_mem:
            self.m_k.pop_first()
            self.m_v.pop_first()
        return z
```

cat: concatenation along token dimension.

consumption and cost, making obtaining such a model expensive. The cost is even higher for models with a larger $M$ for longer-term modeling.⁴

Pipelined Memory Compression. To address this issue, we propose a pipelined compression method. Our insight is that while the compression modules $f_K$ and $f_V$ need to run on uncompressed memory and be jointly optimized, so that the model learns what is important to keep, the learned modules can be shared across all the past memory. Thus, we propose to train to compress memory at only one step at a time, i.e.,

$$
\hat{K}(t) := \left[ \hat{K}^{(t-M)}, \ldots, \hat{K}^{(t-2)}, f_K \left( \text{sg}(\hat{K}^{(t-1)}) \right), \hat{K}(t) \right],$$

and similarly for $\hat{V}(t)$. The right hand side of Fig. 2 illustrates this design. Note that here only the memory $\text{sg}(\hat{K}^{(t-1)})$ from the immediate previous step is cached uncompressed, and to be used to train $f_K$ in the current iteration. $\hat{K}(t') = \text{sg} \left( f_K(\hat{K}(t')) \right)$ for $t'$ from $t - M$ to $t - 2$ are compressed memory cached from earlier iterations. Algorithm 1 presents the pseudo code for this process.

⁴We will present more empirical analysis in §5.2.
In this way, MeMViT adds only a ‘constant’ compression cost over the ‘basic’ MeMViT, since it only runs compression on one single step at a time. But, it reduces the caching and attention cost for all other steps drastically (e.g. 16× by default). In §5, we will show that overall this leads to significant saving, while maintaining high accuracy.

One appealing property of our design is that the receptive field of our video models grows not only with \(M\) but also the number of layers \(L\), since each layer attends further down into the past, therefore hierarchically increasing the temporal receptive field with depth. See Fig. 1b for an illustration.

4.3. Implementation Details

Data Loading. During both training and inference, we perform sequential reading of consecutive chunks of frames (clips) to process videos in an online fashion. This is also the natural setting in a wide range of applications, e.g., robotics or recognition on live streaming video. In our implementation, we simply concatenate all videos and read them sequentially. In cases where the cached memory comes from the previous video (i.e., at the video boundary) we mask the memory to be zero.

Compression Module Design. The compression module can be any function that reduces the number of tokens but maintains the dimensionality \(d\). In our instantiation we choose a learnable pooling [22] due to its simplicity and strong performance, but other choices are possible. We will study the effect of different downsampling factors in §5.2.

Positional Embedding. In the original MViT [22], absolute positional embeddings are added to the input of the network, and each clip uses the same positional embeddings. Thus the positional embeddings can only indicate the positions within a clip, but not the order across multiple clips. Thus, it is important to adopt a relative positional embedding (as used in MViTv2 [43]), so that memory at different points in time has a different relative distance to the queries.

5. Experiments

In this section, we will first compare the scaling behavior of MeMViT with other strategies in §5.1 and then ablate different design choices of MeMViT in §5.2. We perform these experiments on the AVA spatiotemporal action localization dataset [31], which consists of 299 15-minute-long videos sampled from movies. In §5.3, we will study how our method, developed on AVA, generalizes on multiple other tasks and datasets. We will finally compare MeMViT to prior state-of-the-art methods in §5.4.

Implementations. Our default MeMViT model is based on MViTv2-B [43] (16 layers) with 16-frame input clips, sampled at a temporal stride of 4 (denoted ‘16×4’ in model specifications). Following prior work [22–24, 77], all models in this section are pre-trained on Kinetics-400 [34] unless otherwise stated. The AVA models are trained for 30 epochs with SGD using a batch size of 128. We apply random horizontal flipping and random cropping of size 2242 from frames resized such that the short side \(\in [256, 340]\) as data augmentation. We report FLOPs on 2242 crops. We use a cosine learning rate schedule with a base learning rate of 0.6 and weight decay of 10\(^{-8}\). All runtime and memory usages are measured on the same machine with an NVIDIA 16-GB Quadro GP100 GPU with batch size of one. The Kinetics pre-training details, AVA person detector specifications, and additional details are available in Supplementary Material. All methods are implemented using PySlow-Fast [21].

5.1. Scaling Strategies

We first compare the scaling behavior of MeMViT with the widely used “baseline scaling” method [24, 75], which increases the temporal support of a video model by increasing the number of frames \(T\) in input. In Fig. 3, we see that by increasing \(M\), MeMViT scales up to significantly longer temporal support with greatly lower training GPU memory (Fig. 3a), inference GPU memory (3b), training runtime (3c), inference runtime (3d) and FLOPs (3e). Fig. 3f shows that under the same computational costs, our
Ablation Experiments. We conduct detailed ablation on (a): per-layer memory length, (b): compression module downsampling factors, and (c): layers to augment memory. All results are on conducted on the AVA dataset [31] with Kinetics-400 [34] pre-training. We see that MeMViT can increase receptive field, and thus performance, clearly with only small computational cost on a wide range of different design choices. The gray rows denote default choices. (mAP in %).

<table>
<thead>
<tr>
<th>Mem len</th>
<th>Receptive field</th>
<th>GFLOPs</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o mem</td>
<td>1×</td>
<td>57.4</td>
<td>27.0</td>
</tr>
<tr>
<td>1</td>
<td>8×</td>
<td>58.1</td>
<td>28.7</td>
</tr>
<tr>
<td>2</td>
<td>16×</td>
<td>58.7</td>
<td>29.3</td>
</tr>
<tr>
<td>3</td>
<td>24×</td>
<td>59.3</td>
<td>29.2</td>
</tr>
<tr>
<td>4</td>
<td>32×</td>
<td>60.0</td>
<td>28.8</td>
</tr>
</tbody>
</table>

**Table 1a. Per-layer memory length**

<table>
<thead>
<tr>
<th>Compress factor</th>
<th>GFLOPs</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>73.0</td>
<td>28.9</td>
</tr>
<tr>
<td>1×2</td>
<td>62.3</td>
<td>29.0</td>
</tr>
<tr>
<td>2×1</td>
<td>65.3</td>
<td>29.1</td>
</tr>
<tr>
<td>2×2</td>
<td>59.9</td>
<td>29.0</td>
</tr>
<tr>
<td>2×4</td>
<td>58.2</td>
<td>28.3</td>
</tr>
<tr>
<td>4×2×2</td>
<td>58.7</td>
<td>29.3</td>
</tr>
<tr>
<td>4×4</td>
<td>57.8</td>
<td>28.6</td>
</tr>
</tbody>
</table>

**Table 1b. Memory compression factor**

<table>
<thead>
<tr>
<th>Aug layers</th>
<th>GFLOPs</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>60.2</td>
<td>29.1</td>
</tr>
<tr>
<td>75% (uniform)</td>
<td>59.5</td>
<td>29.1</td>
</tr>
<tr>
<td>50% (uniform)</td>
<td>58.7</td>
<td>29.3</td>
</tr>
<tr>
<td>25% (uniform)</td>
<td>58.1</td>
<td>28.7</td>
</tr>
<tr>
<td>early</td>
<td>58.4</td>
<td>28.6</td>
</tr>
<tr>
<td>middle</td>
<td>58.8</td>
<td>28.7</td>
</tr>
<tr>
<td>late</td>
<td>57.8</td>
<td>29.1</td>
</tr>
</tbody>
</table>

**Table 1c. Memory augmentation layers**

The method also obtains clearly better accuracy. We also see that our compression method brings a clear trade-off improvement over the “basic” version that does not compress memory. These results demonstrate that our memory-based design with compression is a promising direction to build practical and strong long-term video models.

5.2. Ablation Experiments

Per-Layer Memory Length. Table 1a compares models with different per-layer memory length (M). We see that all models augmented with memory enjoy clear improvement over the baseline short-term model (1.7-2.3% absolute gain in mAP). Interestingly, the behavior is not very sensitive to the choice of the memory length. Using a per-layer memory length of 2, which corresponds to 16× larger (36-second) receptive field, results in best performance for AVA. We use M=2 as default in the following AVA experiments.

Memory Compression Factor. Table 1b compares compression modules with different downsampling factors. We see that temporal downsampling can be slightly more aggressive (4×) than spatial downsampling (2×) while achieving strong performance. Interestingly, our compression method actually improves the accuracy over the model without compression. This supports our hypothesis that learning “what to keep” in memory can potentially suppress irrelevant noise and help learning. We use downsampling factor of 4×2×2 (for time, height, and width, respectively) as default due to its strong performance.

Memory Augmentation Layers. In Table 1c, we explore if we need to augment memory at all attention layers, and if not, adding memory at which layers is most effective. Interestingly, we see that attending memory at all layers is unnecessary. In fact, augmenting 50% of the layers (i.e., alternating between normal self- and memory-augmented attention) leads to the best performance while saving computation. Furthermore, we observe that putting them uniformly throughout the network works slightly better than concentrating them at early (stage 1&2) layers, middle (stage 3) layers, or late (stage 4) layers.

Compression Strategy. Finally, we compare the scaling behavior of our pipelined compression strategy with that of the basic version without pipeline in Fig. 4. We can see that even with our relatively lightweight pooling-based compression module, the pipelined strategy already shows a significantly better scaling behavior in terms of both GPU memory usage (Fig. 4a) and runtime (Fig. 4b). We thus use it by default in MeMViT. We hope the better scaling behavior will help future research to scale up to even longer-term video models or explore more advanced compression modules more easily.

5.3. Generalization Analysis

So far, we developed and analyzed our method mainly based on an MViTv2-B [22] default backbone on the AVA action localization dataset [31]. Next, we examine MeMViT’s ability to generalize to different settings.
Table 2. Generalization Analysis. We show that our method brings consistent gains with different model sizes and pre-training datasets in Table 2a, and datasets and tasks in Table 2b. Performance measured by mAP (%) for A V A, top-1 (%) for EPIC-Kitchens Classification, and class-mean recall@5 (%) [25] for EPIC-Kitchens Anticipation following standard practice.

<table>
<thead>
<tr>
<th>Model</th>
<th>Pre-train</th>
<th>mAP (%)</th>
<th>GFLOPs</th>
<th>Param</th>
</tr>
</thead>
<tbody>
<tr>
<td>SlowFast, 4×16, R50 [24]</td>
<td>21.9</td>
<td>-</td>
<td>52.6</td>
<td>33.7</td>
</tr>
<tr>
<td>SlowFast, 8×8, R50 [24]</td>
<td>22.7</td>
<td>-</td>
<td>96.9</td>
<td>33.8</td>
</tr>
<tr>
<td>SlowFast, 8×8, R101 [24]</td>
<td>23.8</td>
<td>-</td>
<td>137.7</td>
<td>53.0</td>
</tr>
<tr>
<td>WOO, SFR50 [7]</td>
<td>25.4</td>
<td>-</td>
<td>147.5</td>
<td>-</td>
</tr>
<tr>
<td>MeMViTv1-B, 16×4 [22]</td>
<td>24.5</td>
<td>-</td>
<td>70.5</td>
<td>36.4</td>
</tr>
<tr>
<td>MeMViTv1-B, 32×3 [22]</td>
<td>26.8</td>
<td>-</td>
<td>169.8</td>
<td>36.4</td>
</tr>
<tr>
<td>MeMViTv1-B, 64×3 [22]</td>
<td>27.3</td>
<td>-</td>
<td>345.7</td>
<td>36.4</td>
</tr>
<tr>
<td>MeMViTv1-16, 16×4 [43]</td>
<td>26.2</td>
<td>-</td>
<td>57.4</td>
<td>34.5</td>
</tr>
<tr>
<td>MeMViTv2-16, 16×4</td>
<td>28.5</td>
<td>29.3</td>
<td>58.7</td>
<td>35.4</td>
</tr>
</tbody>
</table>

(b) Additional Datasets & Tasks

Additional Datasets and Task. Table 2b presents results on EPIC-Kitchens-100 egocentric action classification and EPIC-Kitchens-100 action anticipation [13, 14]. The models used here are the same “MeMViTv2-16, 16×4” as the default model used for A V A, except that for EPIC-Kitchens we found that a longer-term model that uses \( M = 4 \) (32×longer-term, or 70.4-second receptive field) to work the best. The model for action anticipation is a causal version to make sure the model output does not see frames beyond the “observed video” [14]. Complete model and training details are available in Supplementary Material. Note that recognition on egocentric videos in the EPIC-Kitchens dataset is quite challenging due to severe motion blur and occlusions on the target action [13, 14]. Also note the large domain difference compared to the videos in A V A [31], which contains stable movie content with different camera motion.

Despite the differences, we see that MeMViT, developed on A V A, works well out-of-the-box on EPIC-Kitchens as well. If we take a closer look at the EPIC classification task, we see that ‘noun’ recognition is a harder task than ‘verb’ recognition, potentially because objects are often occluded by hands, blurred, or even out of the scene. Nonetheless, MeMViT boosts ‘noun’ recognition significantly (+2.4%), supporting our hypothesis that MeMViT may utilize long-term context to disambiguate objects. On the other hand, for action anticipation, predicting the verbs is actually more challenging than predicting the nouns, potentially because nouns are more persistent but verbs can change more frequently (consider ‘washing tomatoes’, followed by ‘cutting tomatoes’, followed by ‘putting tomatoes (into something)’). While with a short-term model, predicting the next ‘verb’ given only the previous one might be challenging, MeMViT sees much more context into the past, bringing large improvement on verbs (+3.5%) and tail verbs (+3.7%).

5.4. State-of-the-Art Comparison

The A V A Dataset. Table 3 compares MeMViT with prior work on the A V A v2.2 dataset [31]. We see that under all pre-training settings, MeMViT obtains a significantly higher accuracy than prior work while having a compara-
MeMViT outperforms MoViNet by a clear margin when being 3× faster and at 2-5×lower GPU memory.

While obtaining high performance, we emphasize that MeMViT uses a simpler and lighter testing procedure, where it simply perform one pass of the videos sequentially, and aggregate all predictions made on target segments by average pooling, without multi-crop testing or over-sampling on testing segments.

The EPIC-Kitchens-100 Action Anticipation Task. Finally, we compare MeMViT with prior work on EPIC-Kitchens-100 Anticipation [13, 14]. Here we use our default model (MeMViT-16, 16×4) pre-trained on Kinetics-400 [34] and also a larger MeMViT-24, 32×3, pre-trained on kinetics-700 [5]. Table 4 shows that MeMViT outperforms all prior works, including those that use multiple modalities, such as optical flow [26], separately trained object feature extractors [28] and large-scale pre-training (IN-21K [15] has ~60× more labels than K400).

The competition winner this year, AVT+ [28], uses a large ViT-based backbone with IN21K pre-training that additionally uses auxiliary losses (e.g., feature regression loss and action recognition loss) and object features. With a simple cross-entropy loss on action labels, our long-term MeMViT outperforms AVT+ by a large margin (action: +1.8%, verb: +4.0%, noun: +5.0%).

6. Conclusion

Long-term video understanding is an important goal for computer vision. To get there, having a practical model for long-term visual modeling is a basic prerequisite. In this paper, we show that extending existing state-of-the-art models to include more input frames does not scale well. Our memory-based approach, MeMViT, scales much more efficiently and achieves better accuracy. The techniques presented in this paper are general and applicable to other transformer-based video models. We hope MeMViT will be useful for future long-term video modeling research.

Table 4. Comparison to prior work on EPIC-Kitchens-100 Action Anticipation [13, 14]. Accuracy measured by class-mean recall@5 (%) [25] following the standard protocol [14]. Gray denotes challenge entries that use additional modalities, such as optical flow or separately extracted object features; MeMViT uses only pixels and still outperforms all of them.

<table>
<thead>
<tr>
<th>Model</th>
<th>Pre-train</th>
<th>Act.</th>
<th>Verb</th>
<th>Noun</th>
<th>Run-time(s)</th>
<th>Mem</th>
<th>FLOPs</th>
<th>Param</th>
</tr>
</thead>
<tbody>
<tr>
<td>TempAgg (RGB + Obj + Flow + ROI) [58]</td>
<td>IN1K + EPIC boxes</td>
<td>-</td>
<td>14.7</td>
<td>23.2</td>
<td>31.4</td>
<td>14.5</td>
<td>28.0</td>
<td>26.2</td>
</tr>
<tr>
<td>RULSTM (RGB + Obj + Flow) [26]</td>
<td>IN1K + EPIC boxes</td>
<td>-</td>
<td>14.0</td>
<td>27.8</td>
<td>30.8</td>
<td>14.2</td>
<td>28.8</td>
<td>27.2</td>
</tr>
<tr>
<td>TSN-AVT+ (RGB + Obj) [28]</td>
<td>IN21K + EPIC boxes</td>
<td>-</td>
<td>14.8</td>
<td>25.5</td>
<td>31.8</td>
<td>11.5</td>
<td>25.5</td>
<td>23.6</td>
</tr>
<tr>
<td>AVT+ (RGB + Obj) [28]</td>
<td>IN21K + EPIC boxes</td>
<td>-</td>
<td>15.9</td>
<td>28.2</td>
<td>32.0</td>
<td>11.9</td>
<td>29.5</td>
<td>23.9</td>
</tr>
<tr>
<td>IPL [76]</td>
<td>-</td>
<td>0.2</td>
<td>6.4</td>
<td>2.0</td>
<td>0.5</td>
<td>14.4</td>
<td>2.9</td>
<td>0.1</td>
</tr>
<tr>
<td>TempAgg (RGB) [58]</td>
<td>IN1K</td>
<td>-</td>
<td>13.0</td>
<td>24.2</td>
<td>29.8</td>
<td>12.2</td>
<td>27.0</td>
<td>23.0</td>
</tr>
<tr>
<td>AVT (RGB) [28]</td>
<td>IN21K</td>
<td>378</td>
<td>59.1</td>
<td>71.4</td>
<td>58.7</td>
<td>35.4</td>
<td>14.5</td>
<td>28.0</td>
</tr>
<tr>
<td>MeMViT, 16×4</td>
<td>K400</td>
<td>59</td>
<td>15.1</td>
<td>28.2</td>
<td>32.0</td>
<td>9.8</td>
<td>27.5</td>
<td>21.7</td>
</tr>
<tr>
<td>MeMViT, 32×3</td>
<td>K700</td>
<td>212</td>
<td>17.7</td>
<td>32.2</td>
<td>37.0</td>
<td>15.2</td>
<td>28.6</td>
<td>27.4</td>
</tr>
</tbody>
</table>

Table 5. Comparison to prior work on EPIC-Kitchens-100 Action Classification [13, 14]. Accuracy measured by top-1 classification accuracy (%).

<table>
<thead>
<tr>
<th>Model</th>
<th>Pre-train</th>
<th>Act.</th>
<th>Verb</th>
<th>Noun</th>
<th>Run-time(s)</th>
<th>Mem</th>
<th>FLOPs</th>
<th>Param</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSN [73]</td>
<td>IN1K</td>
<td>33.2</td>
<td>60.2</td>
<td>46.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TempAgg [58]</td>
<td>IN1K</td>
<td>36.9</td>
<td>59.9</td>
<td>45.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TSM [45]</td>
<td>IN1K</td>
<td>38.3</td>
<td>67.9</td>
<td>50.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SlowFast [24]</td>
<td>K400</td>
<td>38.5</td>
<td>65.6</td>
<td>50.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ego-Exo [42]</td>
<td>K400</td>
<td>-</td>
<td>67.0</td>
<td>52.9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IPL [76]</td>
<td>K400</td>
<td>41.0</td>
<td>68.6</td>
<td>51.2</td>
<td>0.49</td>
<td>8.3</td>
<td>74.9</td>
<td>15.7</td>
</tr>
<tr>
<td>ViViT-L/16×2 [2]</td>
<td>IN21K</td>
<td>44.0</td>
<td>66.4</td>
<td>56.8</td>
<td>-</td>
<td>-</td>
<td>3410</td>
<td>100</td>
</tr>
<tr>
<td>MFormer [51]</td>
<td>IN21K × K400</td>
<td>43.1</td>
<td>66.7</td>
<td>56.5</td>
<td>-</td>
<td>-</td>
<td>370</td>
<td>109</td>
</tr>
<tr>
<td>MFormer-HR [51]</td>
<td>IN21K × K400</td>
<td>44.5</td>
<td>67.0</td>
<td>58.5</td>
<td>-</td>
<td>-</td>
<td>959</td>
<td>382</td>
</tr>
<tr>
<td>MoViNet-AS [36]</td>
<td>N/A</td>
<td>44.5</td>
<td>69.1</td>
<td>55.1</td>
<td>0.49</td>
<td>8.3</td>
<td>74.9</td>
<td>15.7</td>
</tr>
<tr>
<td>MeMViT, 16×4</td>
<td>K400</td>
<td>46.2</td>
<td>70.6</td>
<td>58.5</td>
<td>0.16</td>
<td>1.7</td>
<td>58.7</td>
<td>35.4</td>
</tr>
<tr>
<td>MoViNet-A6 [36]</td>
<td>N/A</td>
<td>47.7</td>
<td>72.2</td>
<td>57.3</td>
<td>0.85</td>
<td>8.3</td>
<td>117.0</td>
<td>31.4</td>
</tr>
<tr>
<td>MeMViT, 32×3</td>
<td>K600</td>
<td>48.4</td>
<td>71.4</td>
<td>60.3</td>
<td>0.35</td>
<td>3.9</td>
<td>211.7</td>
<td>52.6</td>
</tr>
</tbody>
</table>

The EPIC-Kitchens-100 Action Classification Task. We next compare with prior work on EPIC-Kitchens-100 classification [13, 14]. Table 5 shows that MeMViT again outperforms all prior works, including both CNN-based [24, 36, 42, 73] and ViT-based methods [2, 51]. In particular, the previous best method, MoViNet [36], also considers an ‘online’-style model but using causal convolutions, which extend the context only by half of the kernel size (typically one pixel) per layer, thus having a significantly shorter temporal support. MeMViT works significantly better. Also note that MoViNets’ low FLOPs does not translate to efficient runtime on GPUs, in part because MoViNet extensively uses depthwise convolutions, which are known to have low FLOPs, but high runtime in practice [54]. MeMViT outperforms MoViNet by a clear margin while being 3× faster and at 2-5×lower GPU memory.


