Appendix A. Related Work

Below, we review closely related work on motion extraction, attention mechanisms, prompts, and adapter layers for video processing. We also highlight the significant differences between our work and these studies.

Motion. Optical flow computed between consecutive video frames is a widely used secondary input for video processing (Simonyan and Zisserman, 2014; Carreira and Zisserman, 2018; Wang and Koniusz, 2024), complementing the use of conventional videos (Wang et al., 2018, 2021). Dynamic images, which record spatio-temporal information in a single frame, were introduced in (Bilen et al., 2016). A channel sampling method that combines the R, G, or B channels of consecutive frames into a single frame for better motion capture was proposed in (Kim et al., 2022). Recently, Taylor videos have been introduced to capture dominant motions in videos for action recognition (Wang et al., 2024).

Unlike these approaches, our motion prompts are (i) lightweight, with only two extra learnable parameters, (ii) dependent on motions guided by frame differencing maps, and (iii) driven by attention mechanisms that highlight the spatio-temporal motion regions over time.

Attention. To improve feature representations, attention mechanisms capture the relationships between tokens. Attention mechanisms have been efficiently incorporated into transformers, including self-attention (Vaswani et al., 2017; Dosovitskiy et al., 2021) and cross-attention (Lin et al., 2021; Hashiguchi and Tamaki, 2022; Chen et al., 2021; Wei et al., 2020; Wang and Koniusz, 2023), among others.

Our attention mechanism, however, differs from traditional approaches. It is lightweight and does not require the learning of attention matrices. We use a sequence of frame differencing maps to record video dynamics, along with a newly introduced regularization term, to learn spatio-temporally smooth attention maps. Consequently, our attention mechanism is motion-dependent rather than dataset-dependent. Furthermore, compared to existing works, we use an activation function as power normalization to modulate motions, with learnable slope and shift parameters for controlling motion strengths and thresholding. This makes our attention mechanism more transparent and interpretable.

Prompt. Prompt engineering has gained significant interest with advancements in image and video processing. Representative works include the use of prompt templates (Radford et al., 2021), textual prompts for video content description (Hu et al., 2022), incorporating video features into language models as prompts (Yang et al., 2022), and learnable continuous prompt vectors as virtual tokens (Ju et al., 2022). Visual prompts include methods such as visual prompt tuning (Jia et al., 2022), fine-grained visual prompting (Yang et al., 2024), among others.

To the best of our knowledge, none of these works consider using a sequence of refined motions as motion prompts to enhance video processing tasks. Our motion prompts are defined as a sequence of video frames with highlighted spatio-temporally smooth motion regions per frame. Our motion prompts are learnable and form a plug-and-play motion prompt layer. They are optimized using the original loss function.

Adapter. There are several layers and mechanisms in neural networks that can be considered as adapters between the data input and the model itself. Embedding layers, commonly used in NLP and recommendation systems, convert categorical data or tokens into dense

vectors of fixed size (Hrinchuk et al., 2019; Wu et al., 2019). Positional encoding, used in transformer models, adds positional information to input embeddings to retain the order of elements in sequences (Vaswani et al., 2017; Dosovitskiy et al., 2021). Normalization layers, such as Batch Normalization and Layer Normalization, stabilize and accelerate training by normalizing input data (Ioffe and Szegedy, 2015; Ba et al., 2016). Adapter layers, often used in fine-tuning pre-trained models, add task-specific layers between existing layers to adapt pre-trained models to new tasks (Lee et al., 2020; Ding et al., 2022). These adapter layers and mechanisms serve as intermediaries that process and transform input data, making it compatible with the model architecture. This enhances the model's ability to learn from and make predictions on the input data effectively.

Our motion prompt layer functions as an adapter as well, bridging the gap between 'blind motion extraction' and motion prompt-guided motion extraction.

Appendix B. Preliminary

Below, we refer to the preliminary works used in the paper.

Activation function. Activation functions such as Sigmoid, Tanh, ReLU, and Softmax are among the most popular and commonly used in neural networks. These functions can be either linear or non-linear, depending on their formulation and the context in which they are applied. A review of activation functions in deep learning can be found in (Nwankpa et al., 2018; Dubey et al., 2022). A logistic function is a common S-shaped (sigmoid) curve with the equation:

$$f(x) = \frac{L}{1 + e^{-a(x-b)}} \tag{8}$$

where L is the carrying capacity, a is the logistic growth rate (the steepness of the curve), and b is the point at which the output transitions from below $L/2$ to above $L/2$. For values of x much less than b, $f(x)$ is close to 0 (or the lower bound), and for values of x much greater than b, $f(x)$ is close to L (or the upper bound). This characteristic makes b act as a kind of threshold in the Sigmoid function. The standard logistic function is when $L = 1$, $a = 1$, and $b = 0$.

Both the Sigmoid and Softmax functions introduce non-linearity; however, the Softmax function additionally provides a means to interpret the neural network's output as probabilities. In attention mechanisms, the Softmax function is applied to a set of scores (often called attention scores or logits) to produce a probability distribution over the elements in the sequence. This ensures that the importance (attention) weights sum to 1.

A mathematical view of attention. The Vision Transformer (ViT) applies the attention mechanism to image processing by first dividing an image into n patches and then treating these patches as input tokens $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. The self-attention mechanism involves three main components: query $\boldsymbol{Q}$, key $\boldsymbol{K}$ and value $\boldsymbol{V}$ matrices, which are computed as linear projections of the input matrix $\boldsymbol{X}$. The self-attention scores are then computed as the scaled dot-product of the query and key matrices, and the resulting attention matrix is

then used to weight the value matrix. The mechanism can be described as follows:

$$\boldsymbol{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right), \tag{9}$$

$$\boldsymbol{Z} = \boldsymbol{A}\boldsymbol{V}, \tag{10}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is the attention matrix representing the attention weights, and $\boldsymbol{Z} \in \mathbb{R}^{n\times d_k}$ is the output of the self-attention layer. $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q$, $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K$, and $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$. $\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V \in \mathbb{R}^{d\times d_k}$ are learned projection matrices. In practice, multi-head attention is used to allow the model to jointly attend to information from different representation subspaces at different positions. This mechanism enables the ViT to effectively capture the relationships between different parts of an image (Dosovitskiy et al., 2021). In our work, we design a lightweight attention mechanism to highlight the motions of interest from a sequence of frame differencing maps.

Power normalization family. Power Normalization (PN) is used to adjust the power or amplitude of signals to a standard or desired level. It is commonly used in signal and image processing, as well as in statistical methods such as non-linear pooling of features (Jégou et al., 2009; Koniusz and Zhang, 2021) and optical flow correction (Wang and Koniusz, 2024). We apply power normalizing functions to enhance or reduce motions in a sequence of frame differencing maps. A review of well-behaved PN functions such as Gamma, MaxExp, AsinhE (Arcsin hyperbolic function) and SigmE (Logistic a.k.a. Sigmoid functions) can be found in (Koniusz and Zhang, 2021).

Appendix C. Boundedness and Differentiability

Below we proof the boundedness and differentiability for both $a(m)$ and $b(n)$.

Boundedness. The hyperbolic tangent function $\tanh(m)$ has a range of $(-1, 1)$. Therefore, $\beta\,|\tanh(m)|$ has a range of $(0, \beta)$. Adding a positive constant $\epsilon > 0$, the range of $\beta\,|\tanh(m)| + \epsilon$ is $(\epsilon, \beta + \epsilon)$. Thus, $a(m)$, which is $\frac{\alpha}{\beta|\tanh(m)|+\epsilon}$, ranges from $\frac{\alpha}{\beta+\epsilon}$ to $\frac{\alpha}{\epsilon}$. This means $a(m)$ is bounded between $\frac{\alpha}{\beta+\epsilon}$ and $\frac{\alpha}{\epsilon}$.
Similarly, $\tanh(n)$ has a range of $(-1, 1)$ and $b(n) = \gamma\tanh(n)$ has a range of $(-\gamma, \gamma)$. This means $b(n)$ is bounded.
Differentiability. To show differentiability, we compute the derivative of $a(m)$ with respect to m: $a'(m) = -\frac{\alpha g'(m)}{[g(m)]^2}$, where $g(m) = \beta\,|\tanh(m)| + \epsilon$. The derivative of the hyperbolic tangent function is $\tanh'(m) = 1 - \tanh^2(m)$. The derivative of $|\tanh(m)|$ is generally given by $\frac{d}{dm}|\tanh(m)| = \text{sgn}(\tanh(m)) \cdot \tanh'(m) = \text{sgn}(\tanh(m)) \cdot (1 - \tanh^2(m))$. Therefore, $g'(m)$ is: $g'(m) = \beta \cdot \text{sgn}(\tanh(m)) \cdot (1 - \tanh^2(m))$. Thus, the derivative of $a(m)$ is: $a'(m) = -\frac{\alpha\cdot\beta\cdot\text{sgn}(\tanh(m))\cdot(1-\tanh^2(m))}{(\beta|\tanh(m)|+\epsilon)^2}$, where $\text{sgn}(\tanh(m))$ is the sign function, which is 1 for $\tanh(m) \geq 0$ and -1 for $\tanh(m) < 0$.
Similarly, we compute the derivative of $b(n)$ with respect to n: $b'(n) = \gamma \cdot \tanh'(n) = \gamma \cdot (1 - \tanh^2(n))$. Since $\tanh(n)$ is differentiable, $b(n)$ is also differentiable, and its derivative is given by: $b'(n) = \gamma \cdot (1 - \tanh^2(n))$.

Appendix D. Parameter Constraints and Sensitivity Analysis

In this section, we explore the relationships among α, β and γ. We then present a sensitivity analysis of how these parameters affect our power normalization function, and illustrate the selection process for these parameters.

Constraints on α, β, and γ. Given the function $f(\boldsymbol{D}) = \frac{1}{1+e^{-\left(\frac{\alpha}{\beta|\tanh(m)|+\epsilon}\right)(\boldsymbol{D}-\gamma\tanh(n))}}$, we need to ensure that $f(\boldsymbol{D})$ has a minimum value of 0 and a maximum value of 1 over the interval $[-1, 1]$. For simplicity, assume $0 < \gamma < 1$, $\alpha > 0$ and $\beta > 0$.

To satisfy $f(1) \approx 1$:

$$\begin{aligned}\left(\frac{\alpha}{\beta|\tanh(m)|+\epsilon}\right)(1-\gamma\tanh(n)) &\to +\infty \\ \left(\frac{\alpha}{\beta|\tanh(m)|+\epsilon}\right)(1-\gamma\tanh(n)) &\gg 1\end{aligned} \tag{11}$$

In the worst-case scenario where $|\tanh(m)| = 1$ and $\tanh(n) = 1$, it follows:

$$\begin{aligned}\left(\frac{\alpha}{\beta+\epsilon}\right)(1-\gamma) &\gg 1 \\ \alpha(1-\gamma) &\gg \beta+\epsilon \\ \alpha &\gg \frac{\beta+\epsilon}{1-\gamma}\end{aligned} \tag{12}$$

To ensure $f(-1) \approx 0$:

$$\begin{aligned}\left(\frac{\alpha}{\beta|\tanh(m)|+\epsilon}\right)(-1-\gamma\tanh(n)) &\to -\infty \\ \left(\frac{\alpha}{\beta|\tanh(m)|+\epsilon}\right)(-1-\gamma\tanh(n)) &\ll -1\end{aligned} \tag{13}$$

In the worst-case scenario where $|\tanh(m)| = 0$ and $\tanh(n) = -1$, it follows:

$$\begin{aligned}\frac{\alpha}{\epsilon}\cdot(-1+\gamma) &\ll -1 \\ \alpha(-1+\gamma) &\ll -\epsilon \\ \alpha &\gg \frac{-\epsilon}{-1+\gamma}\end{aligned} \tag{14}$$

From Eq. (12) and (14), to ensure a practical and feasible relationship between α, β, and γ, we derive:

$$\begin{aligned}\alpha &\gg \max\left\{\frac{\beta+\epsilon}{1-\gamma}, \frac{-\epsilon}{-1+\gamma}\right\} \\ \alpha &= k\cdot\max\left\{\frac{\beta+\epsilon}{1-\gamma}, \frac{-\epsilon}{-1+\gamma}\right\}\end{aligned} \tag{15}$$

where $k \gg 1$ is a constant ensuring that α is sufficiently large.

Sensitivity analysis on α, β, and γ. In this section, we analyze the sensitivity of the function $f(\boldsymbol{D})$ with respect to the parameters α, β, and γ. Given the function $f(\boldsymbol{D}) = \frac{1}{1+e^{-\left(\frac{\alpha}{\beta|\tanh(m)|+\epsilon}\right)(\boldsymbol{D}-\gamma\tanh(n))}}$, we compute the partial derivatives to understand how small changes in these parameters affect the value of $f(\boldsymbol{D})$. First, let $g(\boldsymbol{D}) = \left(\frac{\alpha}{\beta|\tanh(m)|+\epsilon}\right)(\boldsymbol{D} - \gamma\tanh(n))$, we calculate the partial derivative of $f(\boldsymbol{D})$ with respect to α:

$$\frac{\partial f(\boldsymbol{D})}{\partial \alpha} = \frac{\partial}{\partial \alpha}\left(\frac{1}{1+e^{-g(\boldsymbol{D})}}\right),$$
$$\frac{\partial f(\boldsymbol{D})}{\partial \alpha} = \frac{e^{-g(\boldsymbol{D})}}{\left(1+e^{-g(\boldsymbol{D})}\right)^2} \cdot \frac{\partial g(\boldsymbol{D})}{\partial \alpha}$$

Now, calculate $\frac{\partial g(\boldsymbol{D})}{\partial \alpha}$:

$$\frac{\partial g(\boldsymbol{D})}{\partial \alpha} = \frac{\boldsymbol{D} - \gamma\tanh(n)}{\beta|\tanh(m)| + \epsilon}$$

Thus,

$$\frac{\partial f(\boldsymbol{D})}{\partial \alpha} = \frac{e^{-g(\boldsymbol{D})}}{\left(1+e^{-g(\boldsymbol{D})}\right)^2} \cdot \frac{\boldsymbol{D} - \gamma\tanh(n)}{\beta|\tanh(m)| + \epsilon} \tag{16}$$

Similarly, we calculate the partial derivative of $f(\boldsymbol{D})$ with respect to β and γ:

$$\frac{\partial f(\boldsymbol{D})}{\partial \beta} = \frac{e^{-g(\boldsymbol{D})}}{\left(1+e^{-g(\boldsymbol{D})}\right)^2} \cdot \left(-\frac{\alpha(\boldsymbol{D} - \gamma\tanh(n))|\tanh(m)|}{(\beta|\tanh(m)| + \epsilon)^2}\right) \tag{17}$$

$$\frac{\partial f(\boldsymbol{D})}{\partial \gamma} = \frac{e^{-g(\boldsymbol{D})}}{\left(1+e^{-g(\boldsymbol{D})}\right)^2} \cdot \left(-\frac{\alpha\tanh(n)}{\beta|\tanh(m)| + \epsilon}\right) \tag{18}$$

The parameter α scales the exponent inside the Sigmoid function of $f(\boldsymbol{D})$, defined as $g(\boldsymbol{D})$. Increasing α steepens the curve of $f(\boldsymbol{D})$, causing it to approach 1 more rapidly for positive $\boldsymbol{D} - \gamma\tanh(n)$, and approach 0 more rapidly for negative $\boldsymbol{D} - \gamma\tanh(n)$. Conversely, α makes $f(\boldsymbol{D})$ change more gradually. Note that while α affects how quickly $f(\boldsymbol{D})$ transitions from 0 to 1, it does not directly alter the learning process of m and n. However, the steepness can indirectly affect the gradients during optimization. The parameter β affects the sensitivity of $f(\boldsymbol{D})$ by influencing the denominator inside $g(\boldsymbol{D})$. As β increases, the term $\beta|\tanh(m)| + \epsilon$ in the denominator increases, making $g(\boldsymbol{D})$ smaller. Consequently, $f(\boldsymbol{D})$ changes more gradually. Decreasing β makes the denominator smaller, causing $g(\boldsymbol{D})$ to increase and thereby making $f(\boldsymbol{D})$ transition more sharply. Similar to α, β affects the sensitivity of $f(\boldsymbol{D})$ but does not directly control the learning of m and n. The parameter γ controls the horizontal shift of $f(\boldsymbol{D})$. Increasing γ shifts $g(\boldsymbol{D})$ to the right by increasing $\gamma\tanh(n)$, thereby reducing $f(\boldsymbol{D})$ for a given $\boldsymbol{D}$. Conversely, decreasing γ shifts $g(\boldsymbol{D})$ to the left, increasing $f(\boldsymbol{D})$ for a given $\boldsymbol{D}$. Therefore, γ adjusts the position at which $f(\boldsymbol{D})$ transitions from 0 to 1, affecting the predictions but not directly influencing the learning process of m and n.

Below we show the derivatives with respect to m and n:

$$\frac{\partial f(\boldsymbol{D})}{\partial m} = \frac{e^{-g(\boldsymbol{D})}}{\left(1+e^{-g(\boldsymbol{D})}\right)^2} \cdot \frac{\alpha\beta \tanh'(m) \cdot (\boldsymbol{D} - \gamma \tanh(n))}{(\beta|\tanh(m)| + \epsilon)^2} \tag{19}$$

$$\frac{\partial f(\boldsymbol{D})}{\partial n} = \frac{e^{-g(\boldsymbol{D})}}{\left(1+e^{-g(\boldsymbol{D})}\right)^2} \cdot \frac{-\alpha\gamma \cdot \tanh'(n)}{\beta|\tanh(m)| + \epsilon} \tag{20}$$

While α, β and γ play crucial roles in shaping $f(\boldsymbol{D})$ and its behavior, their impact on the learning of m and n is indirect. The learning process for m and n depends more directly on how well $f(\boldsymbol{D})$ fits the video data and the gradients of m and n with respect to the loss function. Therefore, while tuning α, β and γ is important for optimizing $f(\boldsymbol{D})$, their influence on the learnability of m and n is secondary to the fitting and optimization process itself.

Parameter selection and analysis. Considering the constraints (Eq. (15)), sensitivity analysis (Eq. (16) – (20)) and optimisation (Eq. (7) as the loss function), we simply choose $\beta = 0.45$ and $\gamma = 0.6$ (small constant $\epsilon = 0.1$ as in the main paper):

$$\begin{aligned}\alpha &= k \cdot \max\left\{\frac{0.45+0.1}{1-0.6}, \frac{-0.1}{-1+0.6}\right\} \\ &= k \cdot \max\{1.375, 0.25\} \\ &= 1.375k\end{aligned} \tag{21}$$

We set $\alpha = 5$ ($k \approx 3.6$).

The roles of α, β, and γ. We now provide a detailed analysis of the selected α, β, and γ values. With $\alpha = 5$ and $\beta = 0.45$ in our power normalization function (Eq. (3)): if the learnable $|\tanh(m)| = 0$, the slope $a(m)$ (Eq. (2)) becomes the steepest, leading to significant changes around the learnable shift $b(n)$; conversely, if $|\tanh(m)| = 1$, the slope is the most gentle. This pair of parameters allows flexible adjustment of the slope, which defines the strength of motion modulation, through the learnable and well-bounded $|\tanh(m)|$, resulting in a wide slope range from 9.09 to 50 (see Fig. 3 (b)).

On the other hand, parameter $\gamma = 0.6$ controls the maximum shift for the threshold (Eq. (2), and also see Fig. 3 (c)). The learnable $\tanh(n)$, ranging from -1 to 1, determines the degree (*e.g.*, $\tanh(n) = 1$ indicates a maximum rightward shift of 0.6) and the direction (*e.g.*, positive for right, negative for left) of shift.

The roles of the tanh **function.** Therefore, the $\tanh(\cdot)$ function in our power normalization achieves several objectives: (i) it bounds the parameter search space of slope and shift between -1 and 1, allowing for free learning of m and n, (ii) it effectively adjusts the strengths of motion modulation in slope, where $|\tanh(\cdot)| = 0$ indicates the steepest slope and 1 indicates the gentlest slope, (iii) it controls the direction of shift based on the sign of $\tanh(\cdot)$, where a negative value indicates a left shift and a positive value indicates a right shift, and (iv) it adapts the threshold for motion modulation based on data properties, *e.g.*, frame differencing maps, are also range from -1 and 1, allowing the shift to consider both positive and negative motions.

Table 2: Variant study of fine-tuning on MPII Cooking 2 using a TimeSformer model pretrained on Kinetics-600. We evaluate model variants, including pretrained and baseline (finetuned without VMPs) models, with and without the use of VMPs. This setup explores all potential fine-tuning combinations. ✹ and ❄ denote fine-tuning and frozen states, respectively. '✹Baseline' denotes a second round of fine-tuning. We highlight improvements in red. Fig. 7 shows insightful comparisons of VMPs-based finetuning models across various layers of TimeSformer.

	Pretrained	**VMPs +** ❄Pretrained	Baseline	**VMPs +** ✹Pretrained	**VMPs +** ❄Baseline	**VMPs +** ✹Baseline
Top-1	36.6	**37.1**$^{\uparrow 0.5}$	50.6	**56.6**$^{\uparrow 6.0}$	**56.2**$^{\uparrow 5.6}$	**57.1**$^{\uparrow 6.5}$
Top-5	**66.9**	66.2$^{\downarrow 0.7}$	81.8	**84.4**$^{\uparrow 2.6}$	**84.3**$^{\uparrow 2.5}$	**83.7**$^{\uparrow 1.9}$

We notice that this set of parameters gives reasonably good performance on both generic and fine-grained action recognition datasets, so we use it for evaluations across various action recognition datasets and model architectures.

Appendix E. Additional Visualisations & Discussions

E.1. Exploring the Impact of Video Motion Prompts (VMPs)

The impact of VMPs on fine-tuning. We explore four sets of experiments using Eq. (7) as the loss function: (i) [VMPs+❄Pretrained] freezing the Kinetics-600 pretrained model and training only our motion prompt layer, (ii) [VMPs+✹Pretrained] finetuning the Kinetics-600 pretrained model, including the motion prompt layer, (iii) [VMPs+❄Baseline] freezing the finetuned baseline model and training only our motion prompt layer, and (iv) [VMPs+✹Baseline] end-to-end finetuning of the finetuned baseline model, including the motion prompt layer. We choose MPII Cooking 2 and use TimeSformer as the backbone.

The evaluations are summarized in Table 2. We observe that using VMPs outperforms [Pretrained] by 0.5% in terms of Top-1 accuracy. We also observe that fine-tuning the Kinetics-600 pretrained model with VMPs ([VMPs+✹Pretrained]), freezing the baseline model weights while training only the VMPs ([VMPs+❄Baseline]), and performing a second round of fine-tuning on the baseline model with VMPs ([VMPs+✹Baseline]) outperformed the baseline model (finetuned on MPII Cooking 2 without VMPs) by 6%, 5.6%, and 6.5%, respectively. We include an intriguing plot (Fig. 7) that compares per-layer weight similarity among these models, showing the effects of VMP-based finetuning across various layers and/or blocks of TimeSformer.

VMPs in the context of various pretrained models. Table 3 presents the results. The SSv2 pretrained model achieved the best results after being fine-tuned on the MPII Cooking 2 dataset. However, fine-tuning with the motion prompt layer significantly boosts performance for both Top-1 and Top-5 accuracy, particularly for models with lower baseline performance. Notably, the model pretrained on the Kinetics-600 dataset exhibits the most significant improvement (a 6% increase in top-1 accuracy) when fine-tuning with the motion

Table 3: Evaluations of TimeSformer pretrained on different datasets for fine-tuning, with and without the motion prompt layer, on MPII Cooking 2. We highlight improvements in red.

Model	HowTo100M		SSv2		Kinetics-400		Kinetics-600	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
TimeSformer	50.4	82.2	53.7	81.4	50.0	79.4	50.6	81.8
+**VMPs**	**54.5** $^{\uparrow 4.1}$	**82.5** $^{\uparrow 0.3}$	**55.6** $^{\uparrow 1.9}$	**83.7** $^{\uparrow 2.3}$	**55.2** $^{\uparrow 5.2}$	**82.5** $^{\uparrow 3.1}$	**56.6** $^{\uparrow 6.0}$	**84.4** $^{\uparrow 2.6}$

Table 4: Cross-dataset evaluation with and without video motion prompts. Both top-1 and top-5 performance are reported, with improvements highlighted in red.

	FineGym to MPII		MPII to FineGym	
	Baseline	+**VMPs**	Baseline	+**VMPs**
Top-1	34.1	**34.3** $^{\uparrow 0.2}$	**47.2**	46.5 $^{\downarrow 0.7}$
Top-5	63.0	**64.1** $^{\uparrow 1.1}$	**84.3**	82.6 $^{\downarrow 1.7}$

prompt layer. Overall, consistent improvements are observed across all pretrained models with the addition of the motion prompt layer.

Evaluating VMPs across different datasets. We evaluate the generalizability of fine-tuned TimeSformer+VMPs models on MPII-cooking 2 and FineGym datasets using unseen videos in Table 4. For the model finetuned on MPII-cooking 2, we use FineGym as the test set, training only the final fully connected (FC) layer for 5 epochs while keeping the other layer weights frozen. We also perform the evaluation in the reverse scenario.

As shown in the table, the model trained on a moving camera scene and then adapted to a static camera scene (FineGym to MPII) demonstrates improved performance, with Top-1 and Top-5 accuracy being 0.24% and 1.15% higher, respectively.

Interestingly, when the model is trained on a static camera scene and then adapted to an unseen moving camera scene (MPII to FineGym), performance slightly decreases, with Top-1 and Top-5 accuracy being 0.71% and 1.78% lower, respectively. This is likely due to the difficulty in adapting learned attention mechanisms from static camera videos to moving camera scenarios.

Analyzing VMPs in model fine-tuning through per-layer weight similarity. We choose TimeSformer pretrained on Kinetics-600 as our backbone. We use cosine similarity to compare the per-layer weights between the baseline model (pretrained on Kinetics-600 and then finetuned on MPII Cooking 2) and each model variant finetuned with our VMPs. The purpose of this comparison is to assess the effects of our VMPs on model finetuning across different layers of the backbone network.

We conduct a detailed analysis on four pairs of experiments: (i) pretrained on Kinetics-600 *vs*. model finetuned on MPII Cooking 2 (baseline, no VMPs), (ii) baseline *vs*. finetuned with our VMPs, (iii) baseline *vs*. end-to-end finetuning of the finetuned baseline model with our VMPs (a second round of VMP-based finetuning), and (iv) finetuned with our VMPs

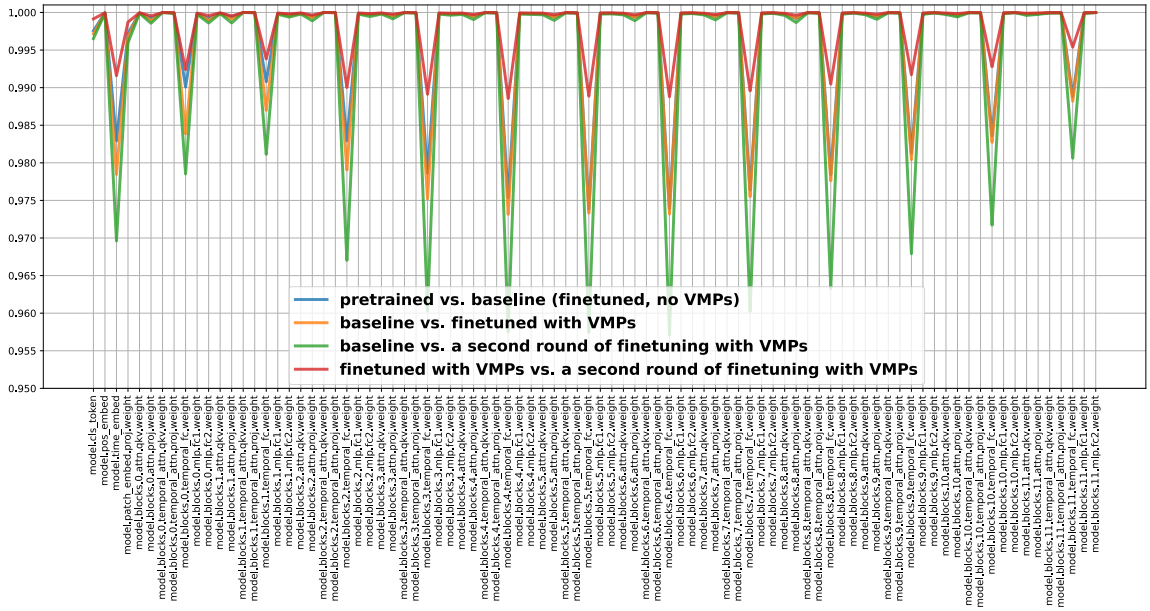


Figure 7: Roles of VMPs in model finetuning via per-layer weight similarity comparison. We use TimeSformer pretrained on Kinetics-600 as the backbone, and finetuned on MPII Cooking 2 with or without VMPs. The vertical axis shows the similarity scores, ranging from 0 to 1, with higher scores indicating greater similarity between model weights. The horizontal axis displays the names of the backbone's layers. Note that the final projection layer has been removed to better display the other layers. We use cosine similarity to evaluate the vectorized per-layer weights across different experimental setups: (i) (blue color) pretrained on Kinetics-600 *vs*. MPII Cooking 2 finetuned (baseline, no VMPs), (ii) (orange color) baseline *vs*. finetuned with VMPs, (iii) (green color) baseline *vs*. end-to-end VMP-based finetuning of the baseline model, and (iv) (red color) finetuned with VMPs *vs*. second-round VMP-based finetuning. Our VMP-based finetuning demonstrates significant effects across different aspects of model architecture. Firstly, it affects the weights of attention projection layers, using only two learnable parameters to subtly enhance focus. Secondly, VMPs impact FC layers dedicated to temporal information embedding, enriching the video processing task with nuanced motion cues. Thirdly, they affect initial layers such as tokens and time embeddings, acting as a adapter to refine the focus on motion and enhance the embeddings of motion-guided tokens and temporal concepts. These findings show the effectiveness of VMPs in optimizing model for action recognition tasks.

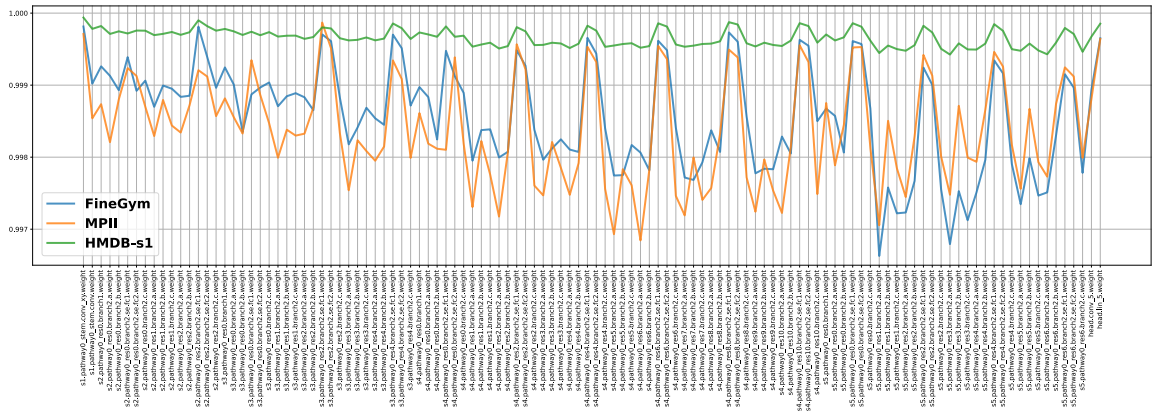


Figure 8: Roles of VMPs in model finetuning via per-layer weight similarity comparison. We use X3D pretrained on Kinetics-600 as the backbone and finetune it on FineGym, MPII Cooking 2 (MPII), and HMDB-51 split 1 (HMDB-s1). We measure the per-layer weight similarity between baseline model (no VMPs) and model finetuned with our VMPs per dataset. The vertical axis depicts similarity scores ranging from 0 to 1, where higher scores indicate greater similarity between model weights. The horizontal axis lists the names of X3D's layers. Note that the final projection layer has been removed to better display the other layers. Our VMP-based finetuning demonstrates significant effects across different layers of X3D. We observe that on HMDB-s1, both the baseline and the models finetuned with VMPs show high similarities in weights between pairs of layers compared to FineGym and MPII. This difference arises because FineGym and MPII are designed for fine-grained action recognition, whereas Kinetics-600 shares similar action types with HMDB-s1, hence minor adjustments in weights are reasonable. Interestingly, our VMPs notably influence the temporal modeling blocks of X3D. These findings highlight the effectiveness of VMPs in optimizing model for 3D CNN-based action recognition.

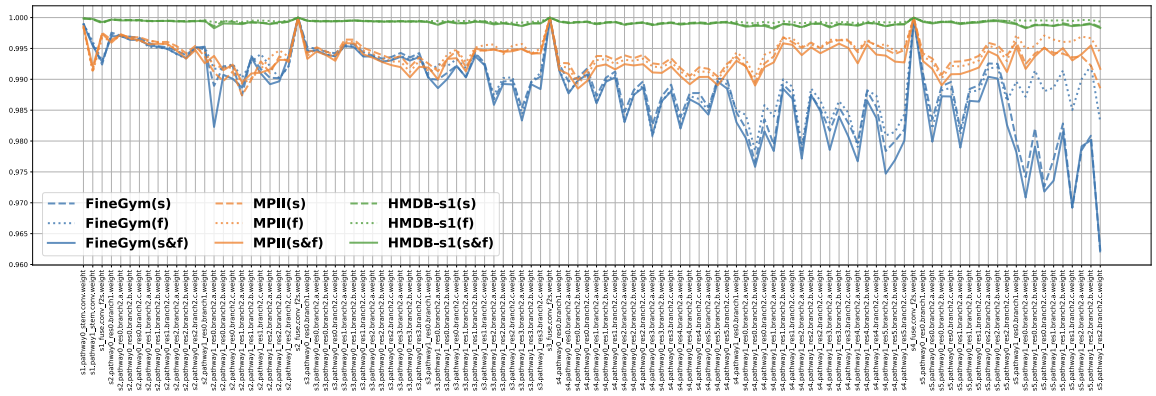


Figure 9: Roles of VMPs in model finetuning via per-layer weight similarity comparison. We use SlowFast pretrained on Kinetics-600 as the backbone and finetune it on FineGym, MPII Cooking 2 (MPII), and HMDB-51 split 1 (HMDB-s1). We measure the per-layer weight similarity between baseline model (no VMPs) and model finetuned with our VMPs per dataset. We explore three variants: adding VMPs to (i) the slow stream only (s), (ii) the fast stream only (f), and (iii) both the slow and fast streams (s&f). The vertical axis depicts similarity scores ranging from 0 to 1, where higher scores indicate greater similarity between model weights. The horizontal axis lists the names of SlowFast's layers. Note that the final projection layer has been removed to better display the other layers. Our VMP-based finetuning demonstrates significant effects across different layers of SlowFast. We observe that on HMDB-s1, both the baseline and the models finetuned with VMPs show high similarities in weights between pairs of layers compared to FineGym and MPII. This difference arises because FineGym and MPII are designed for fine-grained action recognition, whereas Kinetics-600 shares similar action types with HMDB-s1, hence minor adjustments in weights are reasonable. Interestingly, our VMPs notably influence the temporal modeling blocks of SlowFast. These findings highlight the effectiveness of VMPs in optimizing model for 3D CNN-based action recognition.

Table 5: Evaluation of the VMPs layer's effectiveness under various video degradation conditions, including salt-and-pepper noise and reduced resolution from compression. This table presents the performance of the TimeSformer model pretrained on Kinetics-600, comparing scenarios with and without the motion prompt layer in both static and moving camera settings. The results show the improvements in action recognition performance facilitated by our motion prompts in the presence of degraded video quality.

		Origin		Pepper-salt noise		Compression	
		Baseline	**+VMPs**	Baseline	**+VMPs**	Baseline	**+VMPs**
FineGym	Top-1	83.6	**84.4**$^{\uparrow 0.8}$	**83.1**	**83.1**	79.7	**80.5**$^{\uparrow 0.8}$
	Top-5	**98.7**	98.5$^{\downarrow 0.2}$	**98.4**	98.3$^{\downarrow 0.1}$	**97.7**	97.6$^{\downarrow 0.1}$
MPII	Top-1	50.6	**56.6**$^{\uparrow 6.0}$	51.2	**53.2**$^{\uparrow 2.0}$	48.1	**52.8**$^{\uparrow 4.7}$
	Top-5	81.8	**84.4**$^{\uparrow 2.6}$	82.0	**83.5**$^{\uparrow 1.5}$	78.6	**80.1**$^{\uparrow 1.5}$

vs. a second round of finetuning with VMPs. Fig. 7 summarizes our results. Table 2 in the main paper presents the quantitative results on variant study of finetuning.

Interestingly, we observe that our VMP-based finetuning: (i) influences the weights of each attention projection layer. It is worth noting that our VMPs only consist of two learnable parameters and subtly adjust the attention mechanism to enhance focus. (ii) impacts the weights of FC layers used for embedding temporal information. This highlights how our VMPs enrich the video processing task with nuanced motion cues. (iii) affects initial layers such as tokens and time embeddings. This demonstrates that our VMPs act as an adapter, refining not only the input video data's focus on motion but also the embeddings of motion-guided tokens and temporal concepts.

We also observe that the similarity measures for the layer weights between the model finetuned with VMPs and the variant of a second round of finetuning with VMPs are higher than those of the other three experiments. Notably, both the model finetuned with VMPs and the second round of finetuning with VMPs achieve superior performance. This indicates that models tuned with VMPs tend to show very similar learned weights per layer, as evidenced by significantly higher similarities compared to the other three experiments.

Fig. 8 and 9 present additional plots on all three action recognition datasets using X3D and SlowFast backbones. We also observed that our VMPs influence the temporal modeling blocks in these 3D CNN-based architectures.

VMPs in addressing extreme variations in video quality and camera movements. We evaluate the effectiveness of the VMPs layer under various conditions by synthesizing low-quality videos in two distinct ways. First, we introduce salt-and-pepper noise, with a probability of 0.01 for each pixel to display either salt or pepper characteristics. Second, we apply a compression algorithm that reduces the original video quality by half, then upsample it back to the original input size to reduce resolution. This dual approach allows us to assess the performance enhancements provided by the VMPs layer in handling different types of video degradation.

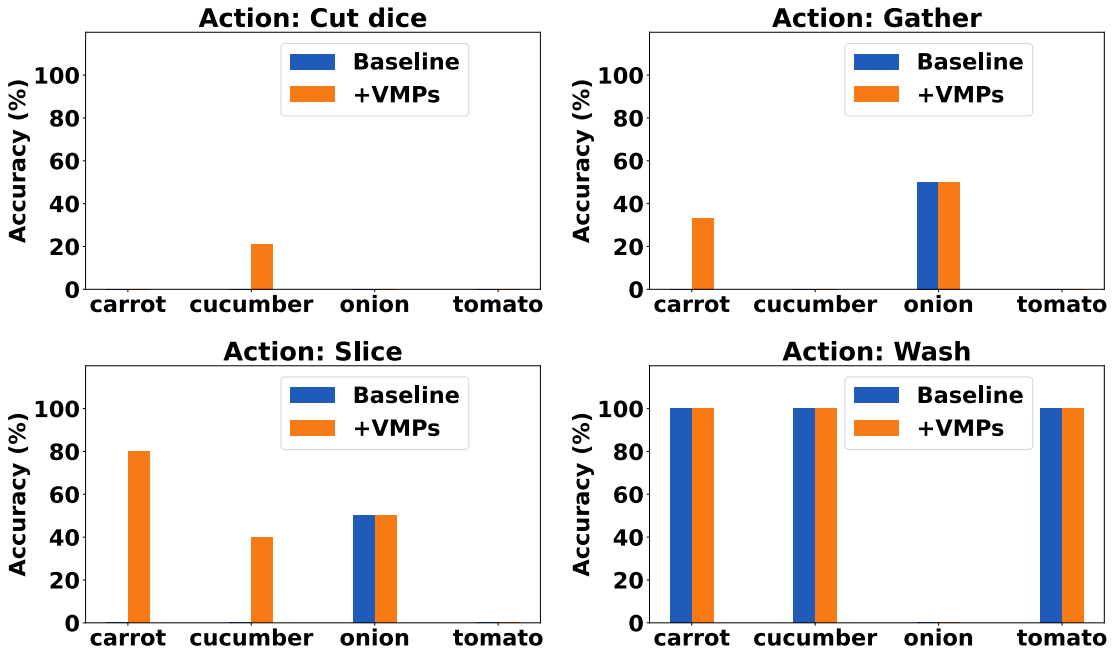


Figure 10: Action recognition performance comparison on selected object-centric actions from the MPII Cooking 2 dataset, with and without the use of video motion prompts (VMPs). The figure illustrates the improvement in accuracy for actions such as *cut dice*, *gather*, and *slice*, particularly when combining motion information with object silhouettes and appearances.

We finetune the TimeSformer pretrained on Kinetics-600 with and without our motion prompt layer. The model without the motion prompt layer serves as the baseline. Below, we present experimental results and analysis regarding video quality and camera movements. Table 5 summarizes the results. We observe that our video motion prompts lead to improved performance in both static and moving camera scenes, even in the presence of low-resolution and noisy videos.

Our video motion prompts are computed based on the frame differencing maps; therefore, (i) color variations, such as dark and bright pixels, (ii) video quality, and (iii) camera movements can affect the quality of the motion prompts. Our future work will focus on exploring the use of motion prompts at the patch level. For instance, we plan to divide video frames into equal-sized image patches and apply motion prompts to each patch individually. Additionally, we will investigate integrating our motion prompts as intermediate layers within the network to modulate feature maps or feature vectors. This approach aims to address issues related to extreme variations in videos and camera movements.

VMPs in understanding visual and motion concepts. Video models rely on both spatial information, such as textures and foreground objects, and temporal information, including motion and evolving dynamics over time. Motion alone is often insufficient for

Table 6: Performance evaluation of action recognition tasks involving specific objects from the MPII Cooking 2 dataset. This table presents the accuracy of the TimeSformer model pretrained on Kinetics-600 and fine-tuned on MPII Cooking 2, comparing results with and without the integration of VMPs. The accuracy in the third column reflects the proportion of correctly classified videos per action class, while the last column summarizes overall performance across the entire test set. Results indicate that the combination of motion and object information enhances action recognition for actions such as *cut dice*, *gather*, and *slice*.

Action	Model	Sample acc.	Total acc.
cut dice	Baseline	0.0	18.9
	+VMPs	**6.0**	**27.3**
gather	Baseline	11.1	29.6
	+VMPs	**33.3**	**44.0**
slice	Baseline	7.7	6.9
	+VMPs	**53.9**	**17.2**
wash	Baseline	**100.0**	71.7
	+VMPs	**100.0**	**78.7**

tasks such as action recognition and is unlikely to perform well independently. Typically, motion occurs alongside object silhouettes and appearances; thus, combining both motion and object information can enhance performance.

To validate the correlation between motion and object, we select a subset of four actions involving specific objects from the MPII Cooking 2 dataset. We use the TimeSformer backbone pretrained on Kinetics-600 and fine-tuned on MPII Cooking 2, evaluating both with and without VMPs. Each subset focuses on a specific object across the selected actions. Below, we present our experimental results. The accuracy in the third column is calculated as the ratio of the total number of correctly classified videos (across all selected object categories) to the total number of sampled videos per action class. The last column displays the overall performance on the entire test set for these actions.

Fig. 10 shows the performance on a subset of test samples. We observe that, for the selected objects, using VMPs generally improves performance on actions such as *cut dice*, *gather* and *slice*.

Table 6 shows that combining both motion and object information leads to improved action recognition performance.

E.2. Analysis of Regularization, Complexity, and Motion Modulation

The role of regularization in static and moving camera scenarios. We examine the effects of applying a regularization term on models fine-tuned on both static and moving camera datasets in Table 7. In static camera settings (MPII Cooking 2), smaller regularization values (λ) generally result in better performance, with the highest accuracy observed when $\lambda = 0$. However, for moving camera scenarios (FineGym), larger λ values improve

Table 7: Effects of λ in the regularization term on (*left*) MPII Cooking 2 (static camera) and (*right*) FineGym (moving camera). We experiment with two backbones (TimeSformer and SlowFast), with the top performance highlighted in bold.

	0	0.1	0.5	1	2
TimeSformer	**56.6**	56.2	55.7	55.6	55.6
SlowFast (slow-only)	**55.5**	53.2	53.5	52.9	-
SlowFast (fast-only)	53.1	53.1	**55.2**	52.8	-
SlowFast (slow&fast)	54.4	54.3	**56.8**	53.6	-

	0	0.5	2.0	2.5	5
TimeSformer	81.9	83.4	84.1	**84.4**	83.3
SlowFast (slow-only)	89.3	88.2	88.3	88.6	**89.7**
SlowFast (fast-only)	89.6	**90.3**	90.3	90.0	90.2
SlowFast (slow&fast)	89.7	88.5	**90.1**	88.6	88.7

performance by smoothing attention maps, addressing temporal discontinuities caused by continuous changes in camera viewpoint while tracking moving subjects.

In the static camera scenario (MPII Cooking 2), the regularization term appears to have a limited impact, with the models achieving optimal performance at smaller λ values. Specifically, TimeSformer performs best when no regularization is applied ($\lambda = 0$), and for the SlowFast variants, moderate values of λ tend to decrease performance.

Conversely, in the moving camera scenario (FineGym), the regularization term becomes more important. Larger λ values contribute to smoothing the attention maps, which is crucial in videos where the camera continuously changes its viewpoint. This helps mitigate temporal discontinuity, leading to improved performance. For instance, TimeSformer reaches its peak performance at $\lambda = 2.5$, while SlowFast (slow-only) and SlowFast (fast-only) also benefit from higher regularization values, showing the importance of adjusting the regularization term according to the nature of the video data.

Computational complexity analysis. The computational cost for generating the frame differencing map is $\mathcal{O}(H \times W)$, where H and W represent the frame height and width, respectively. The learnable PN acts element-wise on the frame differencing map, resulting in a computational complexity of $\mathcal{O}(H \times W)$ as well. The element-wise multiplication between the generated attention map and the original frame also incurs a cost of $\mathcal{O}(H \times W)$. Therefore, the total computational cost of the motion prompt layer remains $\mathcal{O}(H \times W)$. Table 8 summarizes our results.

Attention maps and motion prompts. Below we show more visualisations of motion prompts and attention maps on MPII Cooking 2 and HMDB-51.

We notice that in MPII Cooking 2, the background is lighter orange, indicating lower attention scores (Fig. 12, 13 and 14). In contrast, in HMDB-51 (Fig. 15 and 16), the background is much darker red, indicating higher attention scores due to the videos being captured by moving cameras.

Additionally, we observe that in HMDB-51, the attention maps and motion prompts with and without the regularization term are very similar (Fig. 15 and 16). This behavior is likely because (i) the learned slope and shift parameters for both cases are very close, (ii) the optimal regularization penalty parameter λ is small (0 *vs*. 0.001), and (iii) HMDB-51 is a noisy dataset where camera motions are often more significant than human motions.

Table 8: Computational costs with and without the use of the motion prompt layer (evaluated on HMDB-51 split 1). The experiments are conducted with a batch size of 8, and the reported times (in seconds) reflect the processing time per batch. Each experiment is run 10 times, and we report the mean and standard deviation. The results show that the motion prompt layer adds a negligible computational overhead, as seen in the 'Extra' columns.

Model	**Training**				**Testing**	
	Forward	**Extra**	**Backward**	**Extra**	**Forward**	**Extra**
SlowFast	0.028±0.023		0.269±0.027		0.005±0.027	
+VMPs (slow-only)	0.029±0.037	+0.001	0.271±0.017	+0.002	0.005±0.019	+0.000
+VMPs (fast-only)	0.030±0.022	+0.002	0.290±0.023	+0.021	0.009±0.018	+0.004
+VMPs (slow&fast)	0.030±0.023	+0.002	0.293±0.017	+0.024	0.011±0.021	+0.006
C2D	0.007±0.009		0.076±0.005		0.003±0.016	
+VMPs	0.008±0.011	+0.001	0.079±0.007	+0.003	0.004±0.012	+0.001
I3D	0.007±0.010		0.106±0.004		0.003±0.019	
+VMPs	0.008±0.011	+0.001	0.121±0.008	+0.015	0.005±0.013	+0.002
X3D	0.019±0.015		0.178±0.005		0.023±0.009	
+VMPs	0.020±0.014	+0.001	0.179±0.005	+0.001	0.024±0.009	+0.001
TimeSformer	0.141±0.011		0.253±0.001		0.010±0.020	
+VMPs	0.161±0.011	+0.020	0.255±0.001	+0.002	0.011±0.019	+0.001

We also observe that the attention maps show several interesting patterns: they (i) highlight the motion regions in the current frame, (ii) capture potential movements from the previous frame, and (iii) attend to background scenes affected by camera motions. These observations indicate that our attention maps, guided by only two learnable parameters, effectively highlight visual contents of interest while capturing dynamics over short periods of time.
In static camera scenes, such as MPII Cooking 2, the attention mechanism highlights motion regions, particularly the hand regions, which are central to cooking activities. In contrast, for moving camera scenes where the viewpoint continuously changes to track subjects, as in HMDB-51 and FineGym, the attention tends to highlight the background. Interestingly, we observe that our motion prompt layer also focuses on the silhouettes of human subjects and the boundaries of objects in these dynamic camera scenes.

Per-class accuracy on MPII Cooking 2. Fig. 11 shows the per-class accuracy comparison between the baseline model (pre-trained on Kinetics-600 and then finetuned on MPII Cooking 2 without VMPs) and our VMP-enhanced model on MPII Cooking 2. We use TimeSformer as the backbone. As shown in the figure, integrating our VMPs results in improvements in the accuracy of 34 fine-grained actions (out of a total of 67 actions). The model enhanced with VMPs is able to classify actions that are previously challenging for the baseline model (*e.g.*, with 0 accuracy), such as *read*, *rip open*, and *test temperature*. Furthermore, The model finetuned with VMPs also outperforms the baseline model on actions like *open*, *peel*, *push down*, *slice*, *stir*, *take apart*, and *taste* by a large margin.

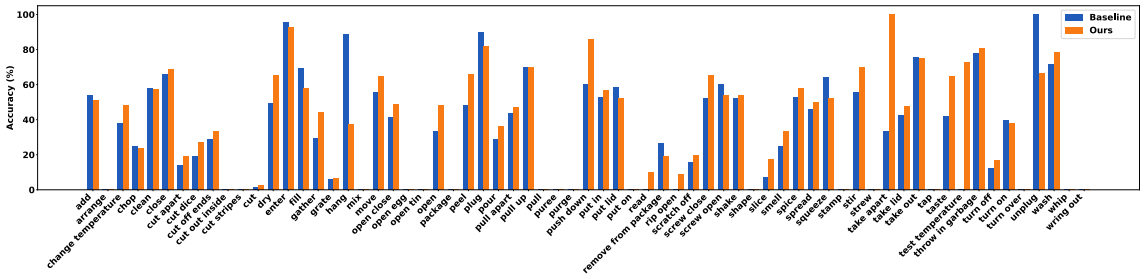


Figure 11: Per-class accuracy comparison is conducted between the baseline model (pre-trained on Kinetics-600 and then finetuned on MPII Cooking 2, without VMPs) and our VMP-enhanced model on MPII Cooking 2, using TimeSformer as the backbone. The integration of our VMPs results in improvements across 34 fine-grained actions out of a total of 67. Notably, with our VMPs, the model achieves accuracies for actions previously classified with 0 accuracy in the baseline model, such as *read*, *rip open*, and *test temperature*.

Figure 12: Effects of the regularization term. We use the *arrange* action from MPII Cooking 2 for visualization. The first two rows show motion prompts and attention maps without regularization ($\lambda = 0$, learned $a = 23.45$ and $b = 0.03$). The last two rows show results with regularization ($\lambda = 2$, learned $a = 10.90$ and $b = 0.08$). Regularization removes unnecessary motion details, resulting in smoother and cleaner attention maps.

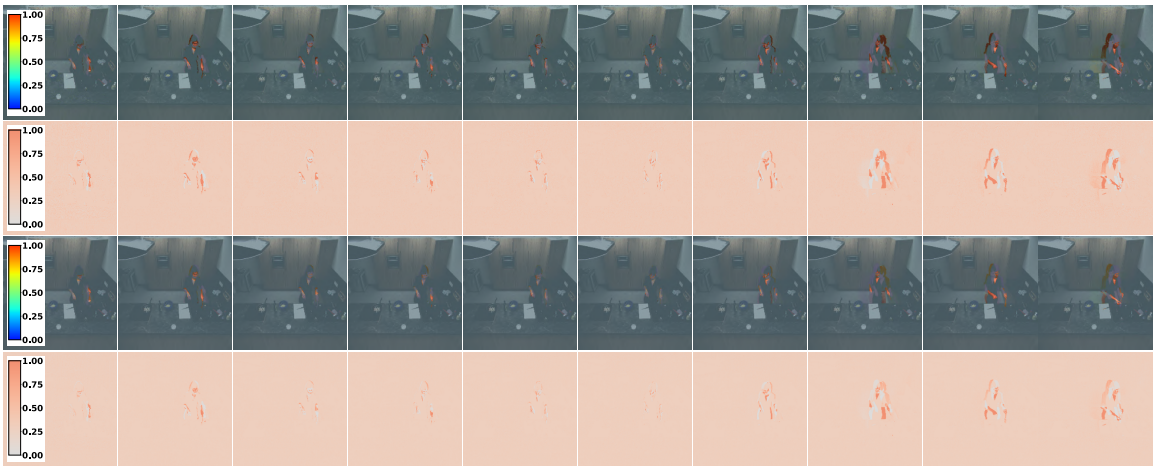


Figure 13: Effects of the regularization term. We use the *shake* action from MPII Cooking 2 for visualization. The first two rows show motion prompts and attention maps without regularization. The last two rows show results with regularization.

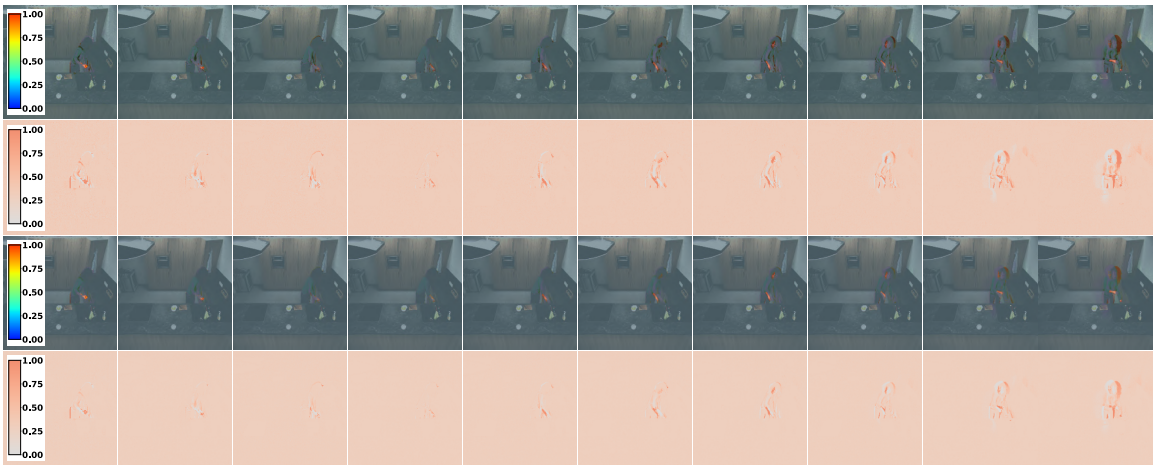


Figure 14: Effects of the regularization term. We use the *throw in garbage* action from MPII Cooking 2 for visualization. The first two rows show motion prompts and attention maps without regularization. The last two rows show results with regularization.

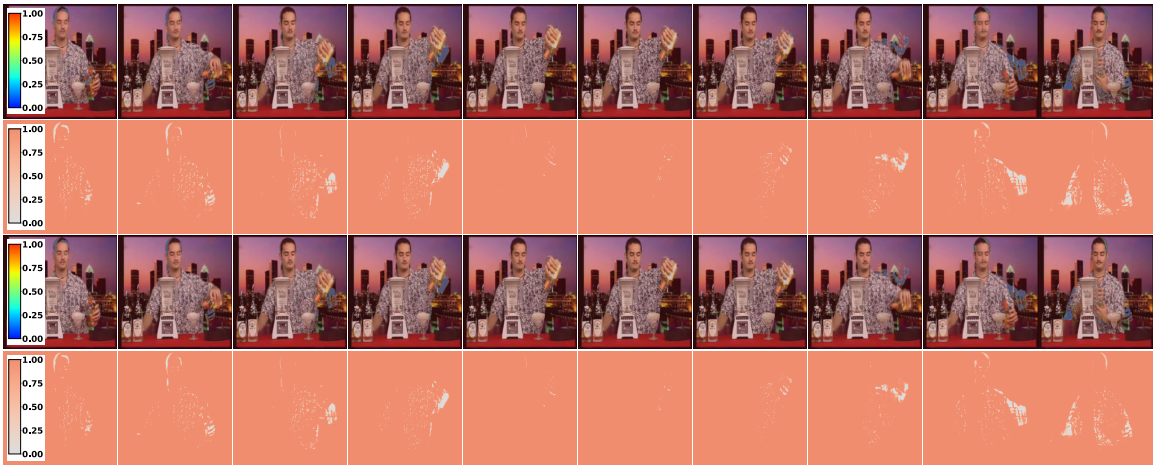


Figure 15: Effects of the regularization term. We use the *pour* action from HMDB-51 for visualization. The first two rows show motion prompts and attention maps without regularization ($\lambda = 0$, learned $a = 20.32$ and $b = -0.36$). The last two rows show results with regularization ($\lambda = 0.001$, learned $a = 21.70$ and $b = -0.39$). We observe that in HMDB-51, the attention maps and motion prompts with and without the regularization term are very similar. This behavior is likely because (i) the learned slope and shift parameters for both cases are very close, (ii) the optimal penalty parameter λ is small (0 *vs*. 0.001), and (iii) HMDB-51 is a noisy dataset where camera motions are often more significant than human motions.

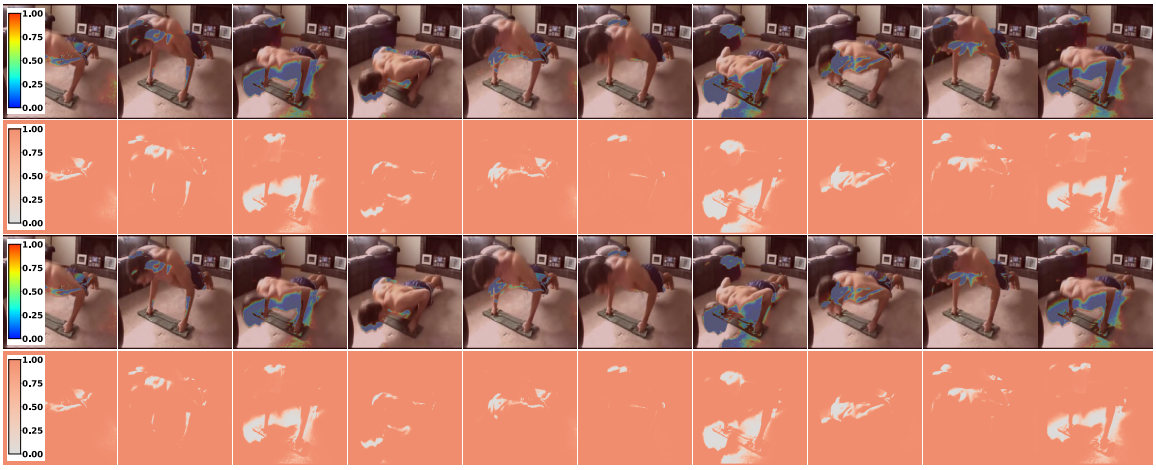
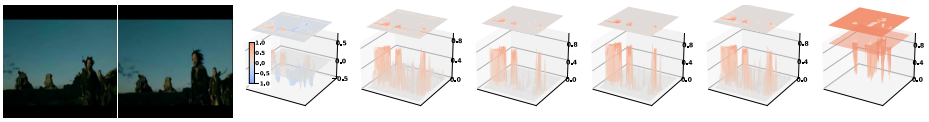
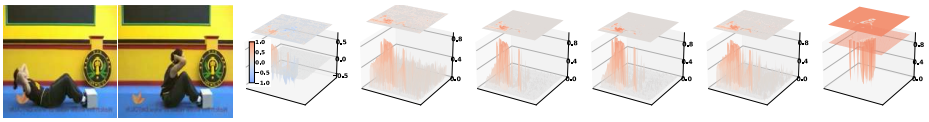


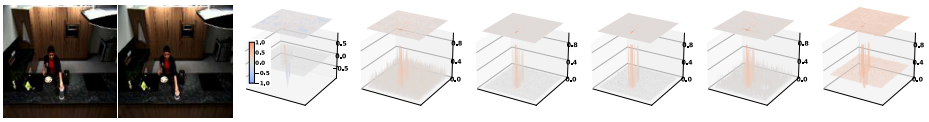
Figure 16: Effects of the regularization term. We use the *pushup* action from HMDB-51 for visualization. The first two rows show motion prompts and attention maps without regularization. The last two rows show results with regularization.



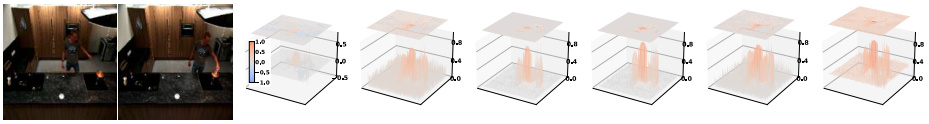
(*a*) Action *run* from HMDB-51.



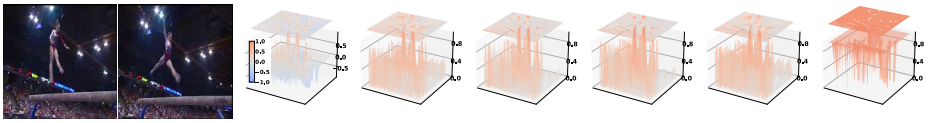
(*b*) Action *situp* from HMDB-51.



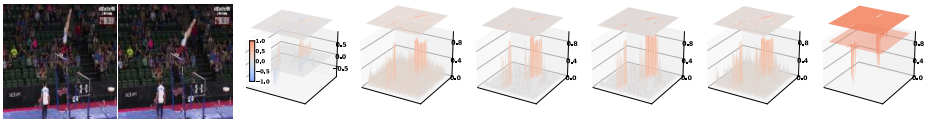
(*c*) Action *push down* from MPII Cooking 2.



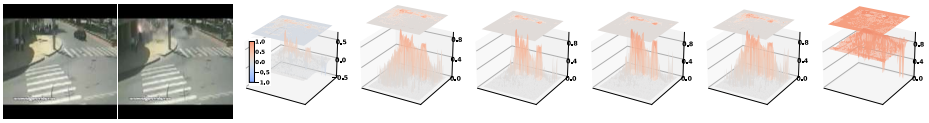
(*d*) Action *whip* from MPII Cooking 2.



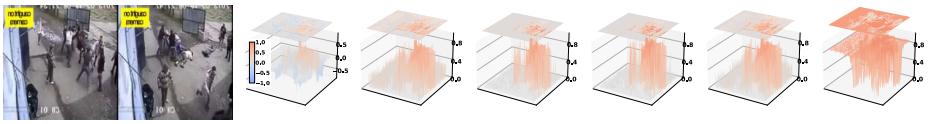
(*e*) Action (*Balance Beam*) *leap forward with leg change* from FineGym.



(*f*) Action (*Uneven Bar*) *giant circle backward* from FineGym.



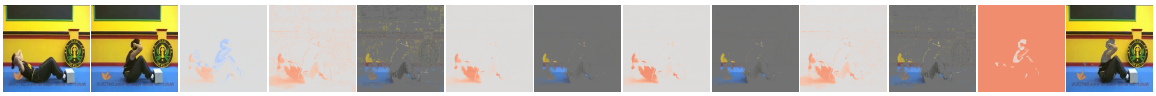
(*g*) Anomaly *explosion* from UCF-Crime.



(*h*) Anomaly *fighting* from UCF-Crime.

Figure 17: We compare existing PN with our PN on motion modulation. The first two columns show consecutive video frames, and the third column displays 3D surface plots of frame differencing maps. Columns 4-7 show attention outputs in both attention maps and 3D surface plots for Gamma, MaxExp, SigmE, and AsinhE. The final column shows our PN's outputs, which capture various motions across different video types. For UCF-Crime, we apply the slope and shift learned from MPII Cooking 2, as both are captured by static cameras.

(*a*) Action *run* from HMDB-51.

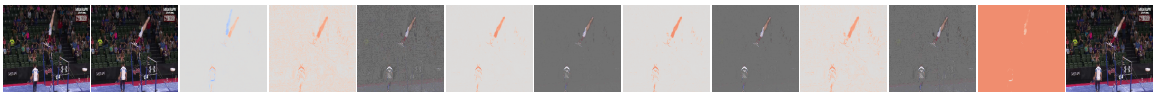


(*b*) Action *situp* from HMDB-51.

(*c*) Action *push down* from MPII Cooking 2.

(*d*) Action *whip* from MPII Cooking 2.

(*e*) Action *(Balance Beam) leap forward with leg change* from FineGym.



(*f*) Action *(Uneven Bar) giant circle backward* from FineGym.

(*g*) Anomaly *explosion* from UCF-Crime.

(*h*) Anomaly *fighting* from UCF-Crime.

Figure 18: Visualizations include original consecutive frames (first two columns), frame differencing maps (third column), pairs of attention maps and motion prompts for Gamma, MaxExp, SigmE, and AsinhE (fourth to eleventh columns). The last two columns display our attention maps and motion prompts. Our attention maps (i) depict clear motion regions, (ii) highlight motions of interest and/or contextual environments relevant to the motions, and our motion prompts capture rich motion patterns. Existing PN functions only focus on motions, often capture noisy patterns and without emphasizing contexts.

Comparison of PN functions on motion modulation. Our qualitative results in Fig. 17. We notice that for static cameras, such as in MPII Cooking 2, our PN function focuses on motion regions, and captures more fine-grained motion patterns compared to other PN functions, which tend to capture more noisy motions.

For moving cameras, such as those in HMDB-51 and FineGym, our PN function captures the background context while the bright regions highlight the motions of interest. This is reasonable as most motions are relevant to surrounding objects and contexts. We apply our learned slope and shift parameters from MPII Cooking 2 to the UCF-Crime dataset, as both are captured by static cameras. Interestingly, our PN function is able to highlight the contextual environment while emphasizing the anomaly motion regions. This demonstrates that our motion prompt layer, equipped with our new PN function, is motion-dependent, attention-driven and generalizable to different video types, including anomaly detection videos.

Fig. 18 compares visualizations of attention maps and motion prompts generated by various PN functions, including ours. As shown in the figure, existing PN functions typically focus on motions, often capturing noisy patterns and frequently disregarding the contextual environment in which the motion occurs. In contrast, our PN function captures both motions of interest and relevant contexts. For instance, anomalies are largely contextual, and our PN function effectively captures both the motions and their surrounding environments.