A2O: Agentic Action-to-Object Feature Learning for Video Action Recognition

Taiki Sekii[1,*] **Fumiaki Sato**[1,*]
[1]CyberAgent
{taiki.sekii, fumiaki.sato.jp}@gmail.com

Abstract

Recent action recognition based on vision–language pretraining and self-supervised video foundation models tends to induce spurious correlations and shortcut learning by relying on action-irrelevant cues, such as backgrounds and object co-occurrences. By contrast, object-detection-based approaches can suppress spurious correlations; however, the loss of input information can limit accuracy. To mitigate this trade-off, we combine these two approaches to learn complementary features that compensate for each other's shortcomings. Specifically, we leverage the commonsense knowledge of large language models (LLMs) regarding human actions and realize a framework in which an LLM agent integrates the two approaches within an agentic learning paradigm to design motion features tailored to the target actions. The LLM agent uses an open-vocabulary object detector to instruct the video foundation model with the target and nontarget objects in the video to make the model attend to objects in a video required for recognizing the target actions. The composition of the detected objects is optimized for the target actions through in-context reinforcement learning (ICRL) using the commonsense knowledge of the LLM. Experiments on multiple public action recognition datasets and an ablation study confirm the robustness of features learned using the proposed method and the effectiveness of ICRL.

1 Introduction

Rapid advances in large language models (LLMs) and vision–language pretraining have culminated in multimodal LLMs (MLLMs) and their numerous useful applications (OpenAI; Stability AI; GitHub, Inc.). MLLMs can now take videos (Team et al., 2025) as input in addition to text and images. Applications that enable humans to communicate with computers via speech and video without text have been realized (OpenAI) by incorporating an automatic speech recognition model (Radford et al., 2022). Consequently, increasing focus has been directed toward the ability of models to recognize human actions in two-dimensional videos, and their importance is increasing across numerous applications, ranging from robots that cooperate with humans (Meng et al., 2024) to sports (Wu et al., 2023) and entertainment (Runway AI).

1.1 Limitations of previous studies

Recent action recognition methods have been able to obtain more generalizable representations from videos through vision–language pretraining (Wang et al., 2025b,c). This progress has benefited from large-scale datasets comprising massive pairs of Internet videos and their text captions. As learned representations are aligned with the space of language embeddings, zero-shot action recognition via prompting has become possible across diverse scenes. Although zero-shot inference enables action recognition to be used in various applications, the learned representations' ability to generalize to motion during supervised fine-tuning (SFT) is limited to what is covered by sparse, low-information pretraining captions. This issue manifests as spurious correlations and shortcut learning, leading the model to overfit cues that are irrelevant to the target action representation, such as foreground/background appearance and object co-occurrence (Steinmann et al., 2025; Käs et al., 2025). In particular, this problem has been observed as a marked drop in the accuracy of benchmarks (Goyal et al., 2017; Li et al., 2021) that require motion understanding instead of appearance.

Approaches that target motion understanding have been actively studied in recent years, including video-based methods (Wang et al., 2023; Bardes et al., 2024; Assran et al., 2025) and object-centric methods (Duan et al., 2022; Hachiuma et al., 2023; Sato et al., 2023; Duan et al., 2023; Qu et al.,

[*] Equal contribution.

Findings of the Association for Computational Linguistics: ACL 2026, pages 36212–36228
July 2-7, 2026 ©2026 Association for Computational Linguistics

2024). These two approaches extract features that are less affected by spurious correlations in different ways. The former learns video foundation models via self-supervised learning, which predicts the motion in videos without using captions. The latter first detects object regions (Duan et al., 2022; Hachiuma et al., 2023) or salient regions (Yu et al., 2024) from videos, and then extracts features from them using supervised learning (Duan et al., 2022; Hachiuma et al., 2023; Duan et al., 2023), unsupervised clustering (Qu et al., 2024), and contrastive learning with captions (Sato et al., 2023).

Both approaches extract features that are useful for understanding motion by focusing on the motion and objects in videos. However, video-based methods, such as vision–language pretraining, use spatiotemporal video volumes as input, and therefore cannot completely eliminate spurious correlations caused by input redundancy. By contrast, object-centric methods remove, in advance, information from nonobject regions that can cause spurious correlations, and by restricting where the model attends, they can recognize the actions of detected objects relatively robustly. However, the loss of information can hinder understanding the context of objects other than the detected targets, creating a trade-off with video-based methods. In this study, we aim to mitigate this trade-off by avoiding information loss.

In a different line of research, leveraging the commonsense knowledge of LLMs for action recognition has recently become popular. For instance, previous studies (Qu et al., 2024; Gong et al., 2024; Lin et al., 2024; Yang et al., 2025b) learned by feeding an LLM with text and visual tokens, where the visual tokens were features obtained via vision–language pretraining (Lin et al., 2024; Yang et al., 2025b) or self-supervised learning (Qu et al., 2024; Gong et al., 2024). The LLM identifies human actions from visual tokens by exploiting knowledge about human actions learned in the language modality. In these studies, the LLM is used as a head (often known as a classifier). In this setting, the knowledge of the LLM is not utilized to train the backbone feature extractor. By contrast, in AutoML, LLMs are used to implement feature extractors; however, because they rely on generic, action-agnostic implementations collected from the Internet, they cannot reflect knowledge about the target actions in feature extraction learning (Tornede et al., 2024; Yao et al., 2025).

1.2 Overview and contributions

Based on the above, this study explores how to learn more robust features for motion understanding, evaluated on supervised benchmarks such as Something–Something v2 (SSv2) (Goyal et al., 2017). In this study, we aim to learn complementary features that compensate for each other's shortcomings by combining video-based and object-centric methods, which have a trade-off relationship. In particular, we realize a framework in which an LLM agent, within an agentic learning paradigm, integrates the two approaches to motion understanding and designs motion features tailored to the target actions by leveraging the commonsense knowledge of the LLM about human actions.

Specifically, we instruct the self-supervised video foundation model on the target and nontarget objects[1] detected by object detection during SFT to make the model attend to the objects in the video that are required for recognizing the target actions. Here, the set of detected objects is derived from the target actions using the commonsense knowledge of the LLM agent, and is optimized for generalization through the framework of in-context reinforcement learning (ICRL) (Moeini et al., 2025). Fig. 1 provides an overview of the proposed ICRL framework. The target object classes are detected in each frame using an open-vocabulary object detector (OVD). By introducing an adapter that converts the detection results, along with the input video, into features and feeds them into the video foundation model, we can make the feature extraction layers of the video foundation model focus only on the target objects during SFT. In the experiments, we evaluate the generalization of the features learned by the proposed method using multiple public action recognition datasets, and verify the effect of agentic feature learning through an ablation study.

The technical contributions of this study are twofold: (1) we demonstrate that the commonsense knowledge of the LLM is effective for designing motion features that attend to objects relevant to the target actions by combining an LLM and an OVD; and (2) we demonstrate that the target and nontarget objects can be automatically optimized using ICRL.

[1] *Nontarget* objects are classes not directly tied to the target actions, used to capture and downweight action-irrelevant cues (*e.g.*, background context).

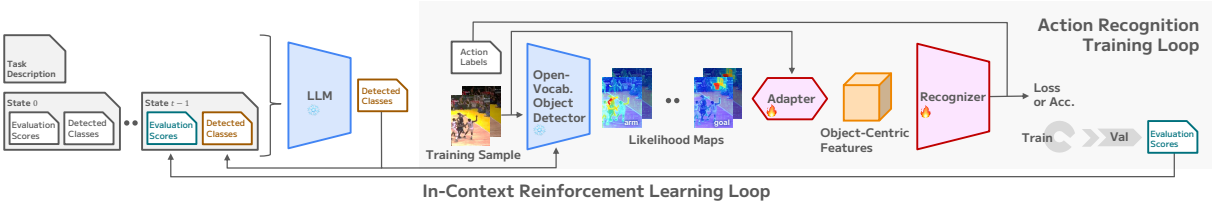

Figure 1: Overview of the proposed ICRL framework (see Sec. 3).

2 Related Work

2.1 Self-supervised/unsupervised representation learning

A representative self-supervised approach for video representation learning masks the input video and reconstructs it with a deep neural network (DNN), for instance, VideoMAE (Wang et al., 2023) masks 90–95% of patches in frames. MotionMAE (Yang et al., 2024) extends VideoMAE by predicting the motion information obtained from the differences between adjacent frames. Focusing on unsupervised approaches, methods have been proposed to discretely cluster features using vector-quantized latent representations (Qu et al., 2024; Gong et al., 2024; Spurio et al., 2025) and to learn feature disentanglement via contrastive learning (Altabrawee and Mohd Noor, 2025). V-JEPA (Bardes et al., 2024; Assran et al., 2025) reconstructs masked parts of videos in latent space, thereby learning representations that focus on the essential semantics of video motion and achieves state-of-the-art (SOTA) performance on numerous supervised benchmarks for motion understanding. When supervised learning is performed, these methods can be adapted to a target dataset via post-training by adding modules or heads (*i.e.*, SFT); however, because they use spatiotemporal video volumes as input, they carry the risk of spurious correlations and shortcut learning. The proposed method differs in that, while building on previous self-supervised learning approaches (Bardes et al., 2024; Assran et al., 2025), it trains the model by making its feature extraction layers attend only to objects relevant to the target actions.

2.2 Object-centric action recognition

Approaches that extract skeletons or object regions from input videos and recognize actions based on these have been actively studied (Duan et al., 2022; Hachiuma et al., 2023; Sato et al., 2023; Duan et al., 2023; Qu et al., 2024). For instance, methods that use only skeleton information (Duan et al., 2022; Sato et al., 2023; Duan et al., 2023) achieve high computational efficiency and robustness against spurious correlations by modeling the spatiotemporal relations among the joint coordinates. Recently, methods that leverage object region information in addition to skeletons (Hachiuma et al., 2023) have also been studied. In contrast to these methods, our key difference is that we automatically optimize the objects to attend to based on the commonsense knowledge of LLMs.

2.3 LLM-based AutoML

Recently, attempts have been made to integrate LLM agents into model training pipelines and to automate the training process within the AutoML community (Tornede et al., 2024; Yao et al., 2025; Zhang et al., 2023; Trirat et al., 2025). For instance, AutoML-Agent (Trirat et al., 2025) is a framework in which multiple LLM agents collaborate to build a pipeline from task specification to deployment. AutoML-GPT (Zhang et al., 2023) automatically generates prompts based on information obtained from data cards, and executes everything from data processing to model design and hyperparameter tuning. All these approaches use LLMs to manage and streamline the entire ML pipeline. Unlike these studies, the present study uses an OVD to exploit the commonsense knowledge of LLMs to design features that attend to objects relevant to target actions. Moreover, by introducing ICRL, we optimized the features to maximize action recognition accuracy.

3 Proposed Method

Overview. Fig. 1 shows the pipeline of the proposed method. In this study, we focus on supervised action recognition and training using labeled videos. The OVD detects the target and nontarget objects required to recognize the target actions in each frame of the input video. Next, the detector outputs and input video are converted by an adapter into features (hereinafter, object-centric features) that are fed into the recognizer. The object classes

are specified by an LLM agent with up to $C_{\max}$ candidates.

We adopt a video foundation model (*e.g.*, V-JEPA2 (Assran et al., 2025)) as the recognizer and transfer the generalization of features obtained from pretraining on large-scale datasets by post-training the recognizer on the target training data. After learning to classify each action using object-centric features from the adapter as input to the recognizer, we evaluate action recognition accuracy. Notably, the object-class selection of the LLM agent is not merely a preprocessing step; it directly affects the learning of feature extraction in the feature extraction layers of the recognizer.

To generalize the object-centric features fed to the recognizer to the target actions, we used an LLM agent to automatically repeat the train–evaluate loop and optimize the detected object classes. The inputs to the LLM agent are instructions for the generation task (a description of the target actions and output format) and the training history of the action recognition model (previously detected object classes and evaluation results). Based on this history, the LLM agent determines the next class of objects. This process can be considered as ICRL, which optimizes the control action of selecting object classes, and the optimal object classes are obtained as byproducts of ICRL.

In the following subsections, we first describe the proposed ICRL framework, and then detail each component in the order of computation.

3.1 Training framework based on ICRL

We formulate the proposed method as ICRL that trains an action recognition model while optimizing the detected objects. The LLM agent generates a set of object classes to be detected from the context containing the trial history and receives the resulting action recognition accuracy as a reward. We summarize the proposed ICRL framework as Algorithm 1.

3.1.1 Problem setup and notation

Let the labeled videos for action recognition be $\mathcal{D} = \{(X_n, y_n)\}_{n=1}^{N}$. We prepare a training set $\mathcal{D}_{\text{train}}$ and a validation set $\mathcal{D}_{\text{val}}$ to compute the rewards and select actions. $\mathcal{D}_{\text{val}}$ is used to evaluate the recognizer at the end of each ICRL iteration.

Let c denote the description of the generation task assigned to the LLM agent. c comprises text describing the target dataset, target actions, and output format, and remains unchanged across iterations. State s_t observed by the agent at iteration t comprises only the history H_{t-1} of the previous control actions a_{t-1} and rewards r_{t-1} of the agent, defined as follows:

$$s_t = H_{t-1} = (a_1, r_1), \ldots, (a_{t-1}, r_{t-1}), \quad (1)$$

where a_t denotes the set of object-class texts provided to the OVD, with a maximum cardinality of $C_{\max}$. The reward r_t is the action recognition accuracy (*e.g.*, classification accuracy) obtained from $\mathcal{D}_{\text{val}}$ after training the action recognition model based on a_t.

Algorithm 1 ICRL framework of the proposed method.

Require: $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$, task description c
Require: number of iterations T, maximum set size $C_{\max}$, policy $\pi_\phi(a \mid s, c)$
Ensure: best object-class set a_{best} and best DNN weights W_{best},

$H \leftarrow \emptyset$ (H_0, hence $s_1 = H_0$)
$a_{\text{best}} \leftarrow \emptyset, r_{\text{best}} \leftarrow -\infty, W_{\text{best}} \leftarrow \emptyset$
for $t = 1, \ldots, T$ **do**
  $s_t \leftarrow H$ ($s_t = H_{t-1}$)
  $a_t \sim \pi_\phi(\cdot \mid s_t, c)$
  $a_t \leftarrow \text{Truncate}(a_t, C_{\max})$
  $W_t \leftarrow \text{TrainDNNs}(\mathcal{D}_{\text{train}}; a_t)$
  $r_t \leftarrow \text{Accuracy}(\mathcal{D}_{\text{val}}; a_t, W_t)$
  $H \leftarrow [H_{t-1}; (a_t, r_t)]$ ($H \leftarrow H_t$)
  if $r_t > r_{\text{best}}$ **then**
    $a_{\text{best}} \leftarrow a_t, r_{\text{best}} \leftarrow r_t, W_{\text{best}} \leftarrow W_t$
  end if
end for

return a_{best} and W_{best}

3.1.2 Policy and environment

Let the LLM agent policy be $\pi_\phi(a \mid s, c)$. At each ICRL iteration t, the agent generates a control action a_t given the concatenated text of the generation task c and history s_t. Given a_t, the environment performs object detection using the OVD, feature extraction using the adapter, and training and evaluation of action recognition using the recognizer and classifier to obtain reward r_t. The state is updated as follows:

$$s_{t+1} = H_t = [H_{t-1}; (a_t, r_t)] \quad (2)$$

Thus, policy optimization proceeds only through the in-context learning of the LLM using history H without parameter updates. Because changes in the policy are reflected in the recognizer weights through newly trained action recognition models at each iteration, the commonsense knowledge of the LLM is indirectly distilled into the feature extractor via ICRL.

3.1.3 Termination condition

Our ultimate goal is to obtain a set of object-class texts proposed by the LLM agent. Therefore, we

do not impose a strict convergence criterion, as in standard RL; instead, following a hyperparameter search, we run a fixed number of iterations up to a predefined maximum T. After all the trials, we select the time step with the highest reward (recognition accuracy) r_t on $\mathcal{D}_{\text{val}}$ among $\{(a_t, r_t)\}_{t=1}^{T}$,

$$\hat{t} = \arg\max_{t} \; r_t, \tag{3}$$

and save the corresponding control action $a_{\text{best}} = a_{\hat{t}}$ and DNN weights W_{best} (adapter, recognizer, and classifier) learned in that trial. Thus, the accuracy does not need to be improved monotonically across iterations.

3.2 Extracting object-centric features

Let the set of object-class texts o generated by the LLM agent be

$$a_t = \{o_i\}_{i=1}^{C_t}, \quad C_t \leq C_{\max}, \tag{4}$$

that we use as detection targets for the OVD. We adopt the Grounded SAM2 (Ren et al., 2024) as the OVD and obtain, for each frame, a likelihood map indicating the presence of each object. Let the map corresponding to class o_i at frame ℓ be $M_\ell^{(i)}(u, v) \in [0, 1]$, where each location (u, v) has the likelihood (score) that the object exists. We denote the vector that stacks the likelihoods over all the object classes within frame ℓ as

$$\mathbf{m}_\ell(u, v) = \left[M_\ell^{(1)}(u, v), \ldots, M_\ell^{(C_t)}(u, v)\right]^\top. \tag{5}$$

To map this vector to a representation that can be fed into the recognizer, we introduce a position-independent adapter. Depending on the dataset and training regime, the adapter is instantiated either as a single 1×1 linear layer or as a shallow stack of 1×1 convolutional blocks with ReLU activations, followed by a final 1×1 linear layer, to produce a feature $\mathbf{f}_\ell(u, v) \in \mathbb{R}^D$ (see Tab. 10). That is, rather than manually constructing inputs for the recognizer, as in conventional visual prompts, we learn optimal weighting and mapping in a data-driven manner. We stack $\mathbf{f}_\ell$ over time to form a video tensor $\mathbf{F} \in \mathbb{R}^{L \times H \times W \times D}$ (L denotes the number of frames) and feed into the recognizer, as described in the next subsection.

3.3 Action classification with a self-supervised video foundation model

We use the video foundation model V-JEPA2 (Assran et al., 2025) as the recognizer. The adapter output $\mathbf{f}_\ell(u, v)$ from the previous subsection is linearly projected onto three channels at each pixel location and added to the original video frame to form the input to the recognizer. Without modifying the layer configuration, V-JEPA2 starts training at the beginning of each ICRL iteration in the proposed method from the weights initialized to the pretrained parameters for all layers. We construct a classifier by attaching an attentive probe (Assran et al., 2025) to the output of the V-JEPA2 architecture. For the action classification loss, we use cross-entropy over the target action labels. After training, we evaluate the action recognition accuracy on $\mathcal{D}_{\text{val}}$ and append the value to the history H as a reward r_t.

4 Experiments

4.1 Datasets

We use SSv2 (Goyal et al., 2017) and MultiSports (Li et al., 2021) (MS) as benchmarks for motion understanding, and Kinetics-400 (K400) (Kay et al., 2017) as a benchmark for appearance-based classification accuracy. SSv2 is a large-scale dataset collected from YouTube with 174 action labels, comprising 220,847 trimmed clips, and is widely used to evaluate action recognition centered on everyday object manipulations. MultiSports contains 66 actions related to four sports (basketball, volleyball, soccer, and aerobics), and provides annotations for 3,200 short clips and 37,701 action instances (over 900k box annotations in total). Kinetics-400 is a large-scale dataset that assigns 400 action labels to short clips trimmed from YouTube. However, because Kinetics-400 tends to allow background and foreground appearances to contribute to discrimination, making it suitable for evaluating the recognition performance based on appearance features (Huang et al., 2018; Shvetsova et al., 2025), we use it to compare the robustness of the baseline methods and that of the proposed method for appearance-biased recognition.

4.2 Experimental setup

We used action classification accuracy as the evaluation metric. In the *few-shot* setting, following previous studies (Kim et al., 2024), we trained and evaluated using only a few labeled training samples per class. In the *full-shot* setting, each model was trained in a supervised manner using the entire training set. We treated the OVD object-class set as a hyperparameter and optimized it on the

held-out, diverse validation split $\mathcal{D}_{\text{val}}$ via ICRL for model selection, because the test split is never used during optimization, and this procedure does not constitute data leakage.

4.3 Implementation details

We used the GPT-5-based ChatGPT (OpenAI) as the LLM agent and ran five ICRL iterations to optimize the target object classes. For the OVD, we used the `small` model of Grounded SAM2 (Ren et al., 2024) and computed a likelihood map of candidate objects for each frame. To train the action recognition model, we used a public implementation of V-JEPA2 (with a `ViT-L` (Dosovitskiy et al., 2021) backbone). Hyperparameters, such as the solver and learning rate, were initialized from the V-JEPA2 settings and optimized for each method and dataset using Optuna (Akiba et al., 2019) (see Appendix B.1). See Secs. 3.2 and 3.3 for the architecture of the action recognition model. Following the baseline (Duan et al., 2022), the proposed method ensembles the recognizer in the same manner as V-JEPA2. The prompt fed to the LLM comprises the task description and trial history up to the previous iteration $t-1$, as shown in Listing 2 (see Appendix B.1). During ICRL, we first tuned the object classes using only one epoch of training data and then trained for all epochs with the obtained optimal classes to produce the final model for evaluation.

4.4 Comparison with previous studies

4.4.1 Evaluation under the few-shot setting

Tab. 1 reports action recognition accuracy of previous methods and that of the proposed method under the few-shot setting. As shown, even V-JEPA2 with simple post-training on a few training samples achieved higher accuracy than other SOTA methods. This can be attributed to (i) V-JEPA2 implemented with a `ViT-L` backbone using a DNN architecture that is more expressive than SOTA methods (CPR-CLIP (Song et al., 2025) uses `ViT-B`) and (ii) self-supervised pretraining on large-scale video datasets for motion understanding. Moreover, post-training the pretrained V-JEPA2 model with the proposed method further improved the accuracy.

4.4.2 Evaluation under the full-shot setting

Tab. 3 reports action recognition accuracy of previous methods and the proposed method under the full-shot setting. In addition to previous methods that perform pretraining in a self-supervised setting, the table includes, for reference, previous methods that either leverage captions or label annotations for videos, as well as larger-scale models. As shown in Tab. 3, the proposed method achieves a higher accuracy than all other approaches based on self-supervised pretraining.

From the results in Tabs. 1 and 3, we found that the proposed method transfers the commonsense knowledge of the LLM to the action recognition model through the OVD, thereby improving robustness.

4.4.3 Evaluation of a spurious-correlation benchmark

We further evaluated robustness on Mimetics (Weinzaepfel and Rogez, 2021), a benchmark designed to assess spurious correlations and shortcut reliance in action recognition models trained on Kinetics-400. We compared the proposed method, structured keypoint pooling (SKP) (Hachiuma et al., 2023), a prior robust action recognition method, and the baseline V-JEPA2, all of which were trained on Kinetics-400. As shown in Tab. 4, the proposed method achieved the highest accuracy on Mimetics. This result demonstrates the effectiveness of our method in mitigating spurious correlations.

4.4.4 Comparison with SOTA MLLMs

To clarify the need for the proposed method, we compared the full-shot version of our method trained on SSv2 with recent SOTA MLLMs that directly process videos end-to-end. As shown in Tab. 5, the proposed method substantially outperforms all MLLM baselines. These results suggest that current end-to-end MLLMs remain limited in the fine-grained motion understanding required for action recognition, whereas the proposed method more effectively learns action-relevant features through supervised training on the target dataset.

4.5 Ablation study

In our next experiment, we evaluated the effectiveness of each component of the proposed method in a unified experimental setting. For accuracy evaluation, we used the MultiSports dataset and trained the action recognition model on a subset of approximately 5,000 training samples. Model ensembling, as described in Sec. 4.3, was not employed in any experiment except for the dedicated ablation study on ensembling. For the experiments related to ICRL, we reported the mean accuracy in

Table 1: Comparison of action recognition accuracy (%) between previous methods and the proposed method under the few-shot setting.

Method	Param.	SSv2			MS			Avg.	ΔAvg.
		2-shot	*4-shot*	*8-shot*	*2-shot*	*4-shot*	*8-shot*		
Vision-Language Pretraining									
TC-CLIP (Kim et al., 2024)	0.15B	7.3	8.6	9.3	-	-	-	-	-
CPR-CLIP (Song et al., 2025)		8.0	9.1	10.2	-	-	-	-	-
Max-Pool-CLIP (Zohra et al., 2025)	0.3B	9.8	11.6	13.8	-	-	-	-	-
Self-Supervised Pretraining									
V-JEPA2 (Assran et al., 2025)	0.3B	10.0	20.5	33.3	22.9	31.9	41.1	26.6	0.0
V-JEPA2 + ours		**11.7**	**22.6**	**36.1**	**24.1**	**33.7**	**43.0**	**28.5**	**+7.1**

Table 2: Effectiveness of ICRL on the MultiSports dataset. †: mean of three ICRL runs.

Method	Acc. (%)
PC3D (Duan et al., 2022)	24.8
PC3D++	56.1
Static prompts†	56.9
Prompt pool†	57.2
Ours†	**59.0**
+OOD-history†	58.4

Table 3: Comparison of action recognition accuracy (%) between previous methods and the proposed method under the full-shot setting. Gray numbers indicate results under a favorable setting that uses captions or labels for pretraining, and/or employs larger-scale models.

Method	Param.	SSv2	MS	K400
		Motion Recog.		*Appearance Recog.*
Self-Supervised Pretraining				
VideoMAEv1 (Tong et al., 2022)		74.3	-	85.2
VideoMAEv2 (Wang et al., 2023)		75.7	-	85.4
MotionMAE (Yang et al., 2024)	0.3B	74.6	-	85.3
V-JEPA2 (Assran et al., 2025)		75.5	83.4	85.9
V-JEPA2 + ours		**76.5**	**84.4**	**86.9**
Pretraining with Captions and/or Massive Models				
MVD (Madan et al., 2024)	0.3B	76.1	-	86.0
InternVideo2 (Wang et al., 2024)	6B	77.5	-	91.9
STAVEQ2 (Rasekh et al., 2025)	2B	78.0	-	-

Table 4: Evaluation on Mimetics, a benchmark for spurious correlations in action recognition.

Method	Acc. (%)	Δ
SKP (Hachiuma et al., 2023)	6.7	0.0
V-JEPA2 (Assran et al., 2025)	6.7	0.0
V-JEPA2 + ours	**8.6**	**+28.4**

over three runs of the ICRL for each method.

4.5.1 Effectiveness of ICRL

Accuracy comparison. First, to validate the contribution of ICRL, we compared the proposed method with the following four baselines:

- **PC3D++**: a re-implementation of the object-centric approach PoseConv3D (PC3D) (Duan et al., 2022) described in Sec. 2.2, updated with a modern video foundation model. In this experiment, we implemented PC3D++ by computing a likelihood map using a skeleton detector (Wang et al., 2019) within our implementation.
- **Static prompts**: a variant without ICRL, in which the LLM generates object classes only from the task description without using the trial history. We run the same number of trials (T) as in the full method.

Table 5: Comparison of action recognition accuracy (%) between state-of-the-art end-to-end multimodal LLMs and the full-shot version of the proposed method trained on SSv2.

Method	Acc. (%)
Gemini 2.5 Flash (Team et al., 2025)	28.2
Gemini 3 Flash (Team et al., 2025)	41.2
Gemini 3 Pro (Team et al., 2025)	31.1
Ours	**76.5**

- **Prompt pool**: a non-ICRL variant that broadens the search space without reward-guided refinement. The LLM first enumerates objects relevant to the dataset and then proposes T plausible combinations of object classes, where T is set to match the number of ICRL trials.
- **OOD-history**: a variant that injects out-of-domain ICRL history, obtained from Kinetics-400, into the in-context conditioning at the start of ICRL on MultiSports. This baseline tests whether the proposed method overfits to the exact trajectory of its own trial history.

Tab. 2 compares these methods on the MultiSports dataset. Compared with PC3D++ and *Static prompts*, the proposed method achieved higher accuracy, showing that trial-history-based refinement is beneficial. The *Prompt pool* baseline slightly improved over *Static prompts*, suggesting that increasing the diversity of candidate object sets broadens the search space to some extent. However, its performance was inferior to that of the proposed method, indicating that the gain of our method does not come merely from wider exploration but from iterative reward-guided trial and error through ICRL. *OOD-history* reached 58.4%, only 0.6 points below the proposed method. This result suggests that the proposed method is not highly sensitive to mismatched out-of-domain history injected at initialization.

Consistency analysis. Embedding the object-class lists from all iterations and runs with ChatGPT yielded a lower variance for the proposed method than for the history-free *Static prompts* baseline (0.07 vs. 0.10), indicating a more stable object-selection policy. We observed the same trend for the LLM's post-hoc rationales. Following ICRL on MultiSports, the LLM was prompted to summarize the reasons for the effectiveness of the final object composition; the variance of the resulting summary embeddings across runs was lower for the proposed method (0.08 vs. 0.13). These results suggest that history conditioning in ICRL converges to a stable and consistent policy rationale rather than yielding arbitrary trial-dependent explanations.

Accuracy trend during ICRL. Fig. 2 shows the evolution of accuracy on the validation split $\mathcal{D}_{\text{val}}$ across ICRL iterations. As the iterations progressed, the object classes were optimized and the accuracy improved.

Object-class composition. Listing 1 shows representative object classes before and after ICRL optimization. Before optimization, the set comprised mainly sports-agnostic objects and body parts, whereas after optimization, fine-grained body parts (`torso`, `forearm`, `head`), and additional scene-level elements (`floor`, `court`, etc.) were incorporated. Some of these added classes correspond to regions that account for parts of the background, which encourages the model to suppress irrelevant background cues. This observation is consistent with the qualitative results in Fig. 3, which visualizes the object-centric features learned by the proposed method in the RGB space and shows that the background is visibly darkened.

To directly test this point, we conducted an ablation study on the MultiSports dataset, in which we removed background-related classes from the final object composition selected by ICRL while keeping the training and evaluation protocol unchanged. The performance decreased substantially from 59.8% to 54.1% (-5.7%) when these classes were removed. This result indicates that nontarget classes are not merely redundant additions to the prompt set. Instead, they serve as minimal scene anchors that help the model retain an action-relevant context while suppressing distracting cues.

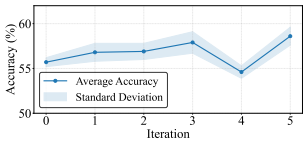

Figure 2: Transition of action recognition accuracy (%) during ICRL iterations (mean of three ICRL runs).

Figure 3: Visualization of object-centric features in the proposed method.

```
# Before ICRL
object_classes_before = [
  "person", "ball",
  "net", "hoop", "goalpost",
  "arm", "leg", "foot"
]
```

```
# After ICRL
object_classes_after = [
  "person", "ball",
  "net", "hoop", "goalpost",
  "arm", "leg", "foot",
  "torso", "forearm",
  "head", "floor", "court",
  "grass", "rim"
]
```

Listing 1: Examples of object classes before and after ICRL optimization on the MultiSports dataset.

4.5.2 Comparison with an AutoML method

We evaluated the effectiveness of the proposed method by leveraging LLMs and comparing the recognition accuracy with that of an AutoML-based learning method that uses LLMs. As a baseline, we used a variant in which only the action recognition model component of our implementation was implemented by an LLM agent, as in a previous study (Trirat et al., 2025). This ensured a fair comparison by sharing implementation details unrelated to the learning capability of the action recognition model, such as video preprocessing and evaluation protocols, across methods. Similar to the proposed method, the baseline iteratively improves the implementation via ICRL, using the implemented model and evaluation results as the state. Tab. 6 lists the recognition accuracy of the proposed method and that of the baseline. The proposed method achieved higher accuracy than the baseline. This result suggests that although the baseline LLM-agent implementation (see Appendix B.2.2) remains a combination of existing knowledge, the proposed method can exploit the commonsense knowledge of the LLM to discover object-centric features.

4.5.3 Component-wise design

Adapter outputs. The detector likelihood maps are converted by the adapter and used as input to the recognizer. Motivated by a prior study (Xie et al., 2022), we tested on the MultiSports dataset whether the three-channel projection causes information loss by comparing it with a no-compression

Table 6: Comparison of action recognition accuracy between an LLM-based AutoML method and the proposed method on the MultiSports dataset (mean of three ICRL runs).

Method	Acc. (%)
AutoML-based method	26.3
Ours	**59.0**

Table 7: Ablation study on LLM selection on the MultiSports dataset (mean of three ICRL runs).

Method	Acc. (%)
ChatGPT-4o	56.2
ChatGPT-5	**59.0**

Table 8: Statistical validation over repeated runs under the full-shot setting. We report the mean $\pm$ standard deviation of accuracy (%), 95% confidence interval (CI; t-distribution), and the p-value of a paired t-test against V-JEPA2.

Dataset	Method	Mean $\pm$ Std	95% CI	p-value
SSv2	**V-JEPA2** (Assran et al., 2025)	75.4 ± 0.16	75.4 ± 0.39	-
	V-JEPA2 + ours	$\mathbf{76.4 \pm 0.12}$	$\mathbf{76.4 \pm 0.30}$	0.0215
MS	**V-JEPA2** (Assran et al., 2025)	83.3 ± 0.06	83.3 ± 0.15	-
	V-JEPA2 + ours	$\mathbf{84.3 \pm 0.07}$	$\mathbf{84.3 \pm 0.18}$	0.0003
K400	**V-JEPA2** (Assran et al., 2025)	85.9 ± 0.02	85.9 ± 0.05	-
	V-JEPA2 + ours	$\mathbf{86.8 \pm 0.13}$	$\mathbf{86.8 \pm 0.33}$	0.0053

variant that directly uses the likelihood maps as input. We observed no significant differences in accuracy, suggesting that the three-channel projection is sufficient in our setting.

LLM selection. Tab. 7 compares accuracy when switching between two LLMs used for ICRL in the proposed method. As the learned action recognition accuracy improves with LLM capability, this indicates that the commonsense knowledge of the LLM directly influences the trial-and-error process of training the action recognition model and improves generalization.

4.5.4 Statistical validation across repeated runs

Because the gains in Tab. 3 are obtained on strong pretrained backbones, their absolute magnitude is modest. To verify that these gains are not due to random variations, we further evaluated V-JEPA2 and the proposed method across repeated runs with matched random seeds. For each dataset, we reported the mean and standard deviation of accuracy, the 95% confidence interval computed from the t-distribution, and the p-value of a paired t-test against V-JEPA2. Tab. 8 shows that the proposed method consistently outperformed V-JEPA2 across repeated runs. Despite the modest absolute gains, the improvements were stable and statistically significant.

4.5.5 Compute analysis

Because the proposed method introduces additional design choices through the OVD and ICRL, we also compared it with a stronger V-JEPA2 baseline, whose number of MoE experts was scaled such that its accuracy approached that of the proposed method. For the baseline, we tuned the standard optimization hyperparameters, namely the learning rate (LR) and weight decay (WD), whereas for the proposed method, we additionally tuned the ICRL object specification. Table 9 reports the resulting accuracy together with the hyperparameter search time and inference time. The proposed method achieved higher accuracy than the scaled V-JEPA2 baseline while requiring less hyperparameter search time and shorter inference time. These results indicate that the proposed method provides a better accuracy–efficiency trade-off in both tuning and inference.

Table 9: Comparison of tuning and inference costs on the MultiSports dataset.

Method	Search space	Acc. (%)	Search time (GPU-hours)	Infer. time (ms)
V-JEPA2 (Assran et al., 2025)	{*LR*,*WD*}	59.2	145	1812
V-JEPA2 + ours	{*LR*,*WD*} + ICRL	**59.8**	**38**	**764**

5 Conclusion

This study focused on the trade-off that action recognition based on video foundation models is prone to spurious correlations/shortcut learning, whereas object-centric approaches suffer from missing information, which can limit accuracy. To mitigate this trade-off, we proposed a framework that leveraged the commonsense knowledge of LLMs regarding human actions and, within an agentic learning paradigm, integrated the two approaches via an LLM agent to design motion features tailored to the target actions. The LLM agent used an OVD to instruct the video foundation model with the target and nontarget objects in the video to make the video foundation model attend to the objects in the video required to recognize the target actions. The composition of the detected objects was optimized for the target actions through ICRL using the commonsense knowledge of the LLM. Experiments on multiple public datasets and an ablation study confirmed the robustness of the features learned by the proposed method and the effectiveness of ICRL.

6 Limitations

The proposed framework combines a video foundation model, an OVD, and an LLM agent to learn object-centric features and thus inherits the limitations of each component and their interactions.

- **Dependence on OVD quality.** Missed detections, imprecise localization, and unstable likelihood maps (*e.g.*, under occlusion, motion blur, or small objects) directly degrade the extracted object-centric features and downstream recognition accuracy.
- **LLM sensitivity and reproducibility.** Performance can vary with the selected LLM, decoding stochasticity, and prompt design; this also introduces additional cost and potential deployment constraints.
- **ICRL stability and efficiency.** ICRL may require multiple iterations to converge and can be sensitive to the trial history quality, thereby increasing the overall training budget.

6.1 Ethical considerations

Data provenance and consent. This work relies exclusively on established, publicly available datasets (SSv2, MultiSports, and Kinetics-400); no new human-subject data were collected. We have complied with each dataset's license and terms of use. We have not attempted to re-identify individuals or infer protected attributes.

Dual-use and misuse. Applications such as activity understanding, rehabilitation, and assistive technologies are socially beneficial; however, the same techniques could be misused for covert surveillance, occupancy tracking, or inference of sensitive behaviors without consent. We oppose and do not condone such uses. To mitigate dual-use risks, we will avoid distributing models tuned for identity attribution.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *SIGKDD*.

Hussein Altabrawee and Mohd Halim Mohd Noor. 2025. Stclr: Sparse temporal contrastive learning for video representation. *Neurocomputing*, 630:129694.

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba, Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, and 11 others. 2025. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*.

Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mahmoud Assran, and Nicolas Ballas. 2024. Revisiting feature prediction for learning visual representations from video. *arXiv preprint 2404.08471*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*.

Haodong Duan, Mingze Xu, Bing Shuai, Davide Modolo, Zhuowen Tu, Joseph Tighe, and Alessandro Bergamo. 2023. SkeleTR: Towards Skeleton-based Action Recognition in the Wild. In *ICCV*.

Haodong Duan, Yue Zhao, Kai Chen, Dahua Lin, and Bo Dai. 2022. Revisiting Skeleton-Based Action Recognition. In *CVPR*.

GitHub, Inc. GitHub Copilot.

Jia Gong, Lin Geng Foo, Yixuan He, Hossein Rahmani, and Jun Liu. 2024. Llms are good sign language translators. In *CVPR*.

Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzyńska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, Florian Hoppe, Christian Thurau, Ingo Bax, and Roland Memisevic. 2017. The "something something" video database for learning and evaluating visual common sense. In *ICCV*.

Ryo Hachiuma, Fumiaki Sato, and Taiki Sekii. 2023. Unified Keypoint-based Action Recognition Framework via Structured Keypoint Pooling. In *CVPR*.

De-An Huang, Vignesh Ramanathan, Dhruv Mahajan, Lorenzo Torresani, Manohar Paluri, Li Fei-Fei, and Juan Carlos Niebles. 2018. What makes a video a video: Analyzing temporal information in video understanding models and datasets. In *CVPR*.

Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. 2017. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*.

Minji Kim, Dongyoon Han, Taekyung Kim, and Bohyung Han. 2024. Leveraging temporal contextualization for video action recognition. In *ECCV*.

Stephanie Käs, Anton Burenko, Louis Markert, Onur Alp Culha, Dennis Mack, Timm Linder, and Bastian Leibe. 2025. How do foundation models compare to skeleton-based approaches for gesture recognition in human-robot interaction? In *ROMAN*.

Yixuan Li, Lei Chen, Runyu He, Zhenzhi Wang, Gangshan Wu, and Limin Wang. 2021. Multisports: A multi-person video dataset of spatio-temporally localized sports actions. In *ICCV*.

Bin Lin, Yang Ye, Bin Zhu, Jiaxi Cui, Munan Ning, Peng Jin, and Li Yuan. 2024. Video-LLaVA: Learning united visual representation by alignment before projection. In *EMNLP*.

Neelu Madan, Andreas Moegelmose, Rajat Modi, Yogesh S. Rawat, and Thomas B. Moeslund. 2024. Foundation models for video understanding: A survey.

Lingyi Meng, Lin Yang, and Enhao Zheng. 2024. Hierarchical human motion intention prediction for increasing efficacy of human-robot collaboration. *IEEE Robotics and Automation Letters*, 9(9):7637–7644.

Amir Moeini, Jiuqi Wang, Jacob Beck, Ethan Blaser, Shimon Whiteson, Rohan Chandra, and Shangtong Zhang. 2025. A survey of in-context reinforcement learning. *arXiv preprint arXiv:2502.07978*.

OpenAI. ChatGPT.

Haoxuan Qu, Yujun Cai, and Jun Liu. 2024. Llms are good action recognizers. In *CVPR*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *ICML*.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.04356*.

Ali Rasekh, Erfan Bagheri Soula, Omid Daliran, Simon Gottschalk, and Mohsen Fayyaz. 2025. Enhancing temporal understanding in video-llms through stacked temporal attention in vision encoders. In *NeurIPS*.

Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, Zhaoyang Zeng, Hao Zhang, Feng Li, Jie Yang, Hongyang Li, Qing Jiang, and Lei Zhang. 2024. Grounded sam: Assembling open-world models for diverse visual tasks. *arXiv preprint arXiv:2401.14159*.

Inc. Runway AI. Gen-4.5.

Fumiaki Sato, Ryo Hachiuma, and Taiki Sekii. 2023. Prompt-guided zero-shot anomaly action recognition using pretrained deep skeleton features. In *CVPR*.

Nina Shvetsova, Arsha Nagrani, Bernt Schiele, Hilde Kuehne, and Christian Rupprecht. 2025. Unbiasing through textual descriptions: Mitigating representation bias in video benchmarks. In *CVPR*.

Hao Song, Yangjun Ou, and Chen Wang. 2025. Cpr-clip: Cross-modal consistent and prompt-diverse regularized clip for action recognition. In *MMAsia*.

Federico Spurio, Emad Bahrami, Gianpiero Francesca, and Juergen Gall. 2025. Hierarchical vector quantization for unsupervised action segmentation. In *AAAI*.

Stability AI. Stable Diffusion.

David Steinmann, Felix Divo, Maurice Kraus, Antonia Wüst, Lukas Struppek, Felix Friedrich, and Kristian Kersting. 2025. Navigating shortcuts, spurious correlations, and confounders: From origins via detection to mitigation.

Gemini Team, Rohan Anil, Sebastian Borgeaud, and 1 others. 2025. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. 2022. VideoMAE: Masked Autoencoders are Data-Efficient Learners for Self-Supervised Video Pre-Training. In *NeurIPS*.

Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, and Marius Lindauer. 2024. Automl in the age of large language models: Current challenges, future opportunities and risks. *TMLR*.

Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. 2025. AutoML-agent: A multi-agent LLM framework for full-pipeline autoML. In *ICML*.

Chenting Wang, Yuhan Zhu, Yicheng Xu, Jiange Yang, Ziang Yan, Yali Wang, Yi Wang, and Limin Wang. 2025a. Internvideo-next: Towards general video foundation models without video-text supervision. *arXiv preprint arXiv:2512.01342*.

Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. 2019. Deep high-resolution representation learning for visual recognition. *PAMI*.

Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang, and Yu Qiao. 2023. Videomae v2: Scaling video masked autoencoders with dual masking. In *CVPR*.

Weiyun Wang, Zhangwei Gao, Lixin Gu, and 1 others. 2025b. Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*.

Yi Wang, Kunchang Li, Xinhao Li, Jiashuo Yu, Yinan He, Chenting Wang, Guo Chen, Baoqi Pei, Rongkun Zheng, Jilan Xu, Zun Wang, and 1 others. 2024. Internvideo2: Scaling video foundation models for multimodal video understanding. In *ECCV*.

Yi Wang, Xinhao Li, Ziang Yan, and 1 others. 2025c. Internvideo2.5: Empowering video mllms with long and rich context modeling. *arXiv preprint arXiv:2501.12386*.

Philippe Weinzaepfel and Grégory Rogez. 2021. Mimetics: Towards Understanding Human Actions out of Context. *IJCV*, 129(5):1675–1690.

Fei Wu, Qingzhong Wang, Jiang Bian, Ning Ding, Feixiang Lu, Jun Cheng, Dejing Dou, and Haoyi Xiong. 2023. A survey on video action recognition in sports: Datasets, methods and applications. *IEEE Transactions on Multimedia*, 25:7943–7966.

Jinheng Xie, Jianfeng Xiang, Junliang Chen, Xianxu Hou, Xiaodong Zhao, and Linlin Shen. 2022. C2am: Contrastive learning of class-agnostic activation map for weakly supervised object localization and semantic segmentation. In *CVPR*.

An Yang, Anfeng Li, Baosong Yang, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Haosen Yang, Deng Huang, Bin Wen, Jiannan Wu, Hongxun Yao, Yi Jiang, Xiatian Zhu, and Zehuan Yuan. 2024. Motionmae: Self-supervised video representation learning with motion-aware masked autoencoders. In *BMVC*.

Zuhao Yang, Yingchen Yu, Yunqing Zhao, Shijian Lu, and Song Bai. 2025b. Timeexpert: An expert-guided video llm for video temporal grounding. In *ICCV*.

Jiapeng Yao, Lantian Zhang, and Jiping Huang. 2025. Evaluation of large language model-driven automl in data and model management from human-centered perspective. *Frontiers in Artificial Intelligence*, 8:1590105.

Runpeng Yu, Weihao Yu, and Xinchao Wang. 2024. Attention prompting on image for large vision-language models. In *ECCV*.

Shujian Zhang, Chengyue Gong, Lemeng Wu, Xingchao Liu, and Mingyuan Zhou. 2023. Automlgpt: Automatic machine learning with gpt. *arXiv preprint arXiv:2305.02499*.

Long Zhao, Nitesh B. Gundavarapu, Liangzhe Yuan, Hao Zhou, Shen Yan, Jennifer J. Sun, Luke Friedman, Rui Qian, Tobias Weyand, Yue Zhao, Rachel Hornung, Florian Schroff, Ming-Hsuan Yang, David A. Ross, Huisheng Wang, Hartwig Adam, Mikhail Sirotenko, Ting Liu, and Boqing Gong. 2024. VideoPrism: A foundational visual encoder for video understanding. In *ICML*.

Fatimah Zohra, Chen Zhao, Shuming Liu, and Bernard Ghanem. 2025. Effectiveness of max-pooling for fine-tuning clip on videos. In *CVPRW*.

A Related Work

A.1 Vision–language pretraining

Breakthrough progress in vision–language pretraining began with CLIP (Radford et al., 2021), which contrastively learns encoded language and vision embeddings. Subsequently, vision–language pretraining research has evolved into multimodal LLMs that combine visual tokens extracted from videos with LLMs. For instance, Qwen (Yang et al., 2025a) and InternVL (Wang et al., 2025b) aligned the outputs of a vision encoder to the inputs of an LLM, achieving strong performance in various vision–language tasks, such as captioning and dialogue. By contrast, InternVideo (Wang et al., 2025c) and VideoPrism (Zhao et al., 2024) combine vision–language pretraining with self-supervised learning to obtain representations that capture temporal motion in videos. These methods are zero-shot approaches that generate responses based on video content for a user query; when supervised learning is performed, they can be adapted to a target dataset via post-training by adding modules or heads. However, captions used in previous studies do not fully describe video content and have relatively little information, whereas the input to the vision encoder is a highly redundant spatiotemporal video volume compared with text. Consequently, spurious correlations and shortcut learning can occur, where the model mistakenly attends to backgrounds or object co-occurrences unrelated to the target actions, thereby reducing the robustness of motion-understanding tasks (Steinmann et al., 2025; Käs et al., 2025). We adopt the motion-understanding-focused approaches (see Sec. 2.1) to avoid these issues arising from the reliance on vision–language pretraining.

B Experiments

B.1 Implementation details

The prompt fed to the LLM comprised the task description and trial history up to the previous iteration, as shown in Listing 2.

Tab. 10 reports the hyperparameters optimized for each dataset in our experiments.

Table 10: Hyperparameters tuned for each dataset.

Parameter	SSv2	MultiSports	Kinetics-400
Epochs (full-shot)	5	20	5
Epochs (few-shot)	25	20	-
Batch Size	4	8	4
Learning Rate (full-shot)	2.0×10^{-5}	7.6×10^{-5}	1.0×10^{-4}
Learning Rate (few-shot)	1.0×10^{-4}	7.6×10^{-5}	-
Weight Decay	1.4×10^{-5}	6.7×10^{-6}	1.4×10^{-5}
Adapter (full-shot)	{1×1 conv. + ReLU} × 4 → 1×1 conv.	1×1 conv.	1×1 conv. + ReLU × 4 → 1×1 conv.
Adapter (few-shot)	1×1 conv.	1×1 conv. + ReLU × 3 → 1×1 conv.	-
Adapter (ablation study)	-	1×1 conv.	-

Table 11: Ablation study with the InternVideo-Next backbone on the MultiSports dataset.

Method	Acc. (%)
InternVideo-Next (Wang et al., 2025a)	59.8
InternVideo-Next + ours	**62.8**

B.2 Ablation study

B.2.1 Component-wise design

InternVideo-Next backbone. Tab. 11 compares action recognition accuracy on the MultiSports dataset when using InternVideo-Next (Wang et al., 2025a) as the recognizer backbone. The proposed method improves the accuracy, demonstrating that it can effectively enhance InternVideo-Next. This result suggests that the proposed object-centric feature learning is also beneficial when combined with a strong video foundation model beyond the default recognizer used in our main experiments.

B.2.2 Comparison with LLM-based AutoML methods

We provide the LLM-generated source code used to implement the final action recognition model under the baseline ICRL setting in Listing 3 (see Sec. 4.5.2).

Listing 2: Example of prompt input to the LLM during ICRL in the proposed method.

```
You are an adaptive reasoning agent collaborating on the
framework. Using the class label set and the iteration
history, propose improved prompts to enhance classification
performance.
Your primary goal is to analyze the iteration history to
identify weaknesses and propose optimized prompt sets that
directly improve overall and class-wise classification
accuracy. Focus on data-driven adjustments that lead to
measurable accuracy gains.
In each new round, adjust gradually with a few edits,
avoiding large jumps to prevent training complexity.

## Output Format Instructions
CRITICAL: Output prompts as a plain text list, one per line,
 following these rules STRICTLY:
- Each line = one single, atomic visual concept (short noun/
noun phrase)
- NO headers, titles, numbering, or explanations
- Do not include any other text before or after the list

Note: open-vocabulary object detector uses Grounded SAM2.
Keep prompts simple and atomic (single concept; avoid commas
, slashes, logic words, or compound phrases), as complex
prompts are not reliably recognized.

## Instructions
* Diagnose likely causes of low accuracy (e.g., missing
contextual cues, insufficient body-part/object coverage,
ambiguous wording).
* Design prompts that better capture discriminative evidence
 for underperforming classes while keeping the vocabulary
concise and generalizable.
* Prefer compact, single-concept phrases suitable for open-
vocabulary object detector (Grounded SAM2-friendly).
* Prompt Count Policy (soft guidance): expand cautiously and
 incrementally; prefer swapping out weak cues over adding
many new ones at once.

## Framework
We propose an agent that turns commonsense into adaptive
feature redesign for action recognition. For each action, a
language model proposes a set of evidence-bearing entities (
objects, targets, body parts). These cues are localized in
video via a text-conditioned open-vocabulary object detector
 (Grounded SAM2), yielding spatio-temporal regions that
guide representation. The agent then rewires the extractor
on the fly via region-conditioned routing/adapters to
emphasize actor-object interactions and suppress background
bias. Learning proceeds in a closed loop: class-level
rewards from recognition performance and attribution
consistency drive in-context reinforcement learning over the
 textual cues, implemented as iterative prompt edits without
 updating the language model's weights, followed by renewed
localization and feature updates. Thus, prompts, detections,
 and features co-evolve as a single trainable pipeline.


## Classification Target Set
["aerobic push up", "aerobic helicopter", "volleyball serve
", "volleyball block", "football shoot", "football dribble",
 "basketball pass", "basketball block"]

## Iteration History
### Iteration 1
Prompts: ["person", "ball", "net", "hoop", "goalpost", "arm
", "leg", "foot"]

Overall Accuracy: 57.900%

Class-wise Accuracy:
aerobic push up: 0.00%
aerobic helicopter: 100.00%
volleyball serve: 82.03%
volleyball block: 96.73%
football shoot: 1.96%
football dribble: 88.57%
basketball pass: 93.91%
basketball block: 0.00%

### Iteration 2
..
```

Listing 3: Source code of action recognition model implemented using LLM-based AutoML method.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from typing import List, Optional


def _pick_num_groups(num_channels: int, max_groups: int =
32) -> int:
    for g in range(min(max_groups, num_channels), 0, -1):
        if num_channels % g == 0:
            return g
    return 1


class DropPath(nn.Module):
    def __init__(self, drop_prob: float = 0.0):
        super().__init__()
        self.drop_prob = float(drop_prob)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        if self.drop_prob == 0.0 or not self.training:
            return x
        keep_prob = 1.0 - self.drop_prob
        shape = (x.shape[0],) + (1,) * (x.ndim - 1)
        random_tensor = keep_prob + torch.rand(shape, dtype=
        x.dtype, device=x.device)
        random_tensor.floor_()
        return x.div(keep_prob) * random_tensor


class SE3D(nn.Module):
    def __init__(self, channels: int, reduction: int = 16):
        super().__init__()
        hidden = max(8, channels // reduction)
        self.fc1 = nn.Conv3d(channels, hidden, kernel_size
        =1, bias=True)
        self.fc2 = nn.Conv3d(hidden, channels, kernel_size
        =1, bias=True)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        y = F.adaptive_avg_pool3d(x, output_size=1)
        y = F.gelu(self.fc1(y))
        y = torch.sigmoid(self.fc2(y))
        return x * y


class BottleneckR2Plus1D(nn.Module):
    expansion = 4

    def __init__(
        self,
        inplanes: int,
        planes: int,
        stride_t: int = 1,
        stride_s: int = 1,
        use_se: bool = False,
        drop_path: float = 0.0,
        layer_scale_init: float = 1e-6,
    ):
        super().__init__()
        mid = planes
        self.conv1 = nn.Conv3d(inplanes, mid, kernel_size=1,
         stride=1, padding=0, bias=False)
        self.gn1 = nn.GroupNorm(_pick_num_groups(mid), mid)

        self.conv2_sp = nn.Conv3d(
            mid, mid, kernel_size=(1, 3, 3),
            stride=(1, stride_s, stride_s),
            padding=(0, 1, 1), bias=False
        )
        self.gn2_sp = nn.GroupNorm(_pick_num_groups(mid),
        mid)

        self.conv2_t = nn.Conv3d(
            mid, mid, kernel_size=(3, 1, 1),
            stride=(stride_t, 1, 1), padding=(1, 0, 0), bias
            =False
        )
        self.gn2_t = nn.GroupNorm(_pick_num_groups(mid), mid
        )

        self.se = SE3D(mid) if use_se else None

        out_channels = planes * self.expansion
        self.conv3 = nn.Conv3d(mid, out_channels,
        kernel_size=1, stride=1, padding=0, bias=False)
        self.gn3 = nn.GroupNorm(_pick_num_groups(
        out_channels), out_channels)
```

```
        self.downsample = None
        if stride_t != 1 or stride_s != 1 or inplanes !=
        out_channels:
            self.downsample = nn.Sequential(
                nn.Conv3d(inplanes, out_channels,
                kernel_size=1, stride=(stride_t, stride_s,
                stride_s), bias=False),
                nn.GroupNorm(_pick_num_groups(out_channels),
                 out_channels),
            )

        self.drop_path = DropPath(drop_path) if drop_path >
        0.0 else nn.Identity()
        self.gamma = nn.Parameter(torch.ones(out_channels) *
         layer_scale_init)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        identity = x

        out = self.conv1(x)
        out = F.gelu(self.gn1(out))

        out = self.conv2_sp(out)
        out = F.gelu(self.gn2_sp(out))

        out = self.conv2_t(out)
        out = F.gelu(self.gn2_t(out))

        if self.se is not None:
            out = self.se(out)

        out = self.conv3(out)
        out = self.gn3(out)

        out = self.drop_path(out) * self.gamma.view(1, -1,
        1, 1, 1)

        if self.downsample is not None:
            identity = self.downsample(x)

        out = F.gelu(out + identity)
        return out


class VideoEncoder3D(nn.Module):
    """
    Hierarchical 3D CNN encoder with R(2+1)D SE bottlenecks.
    Temporal downsampling: 16 -> 8 (stem) -> 4 (stage1),
    then keep T=4
    Spatial downsampling: 256 -> 64 (stem) -> 32 -> 16 -> 8
    -> 4
    Returns multi-scale features:
      - s2: [B, 512, 4, 16, 16]
      - s3: [B, 1024, 4, 8, 8]
      - s4: [B, 1024, 4, 4, 4]
    """
    def __init__(self, in_channels: int = 3, drop_path_rate:
     float = 0.1):
        super().__init__()

        stem_out = 64
        self.stem = nn.Sequential(
            nn.Conv3d(in_channels, stem_out, kernel_size=(3,
             7, 7), stride=(2, 4, 4), padding=(1, 3, 3),
            bias=False),
            nn.GroupNorm(_pick_num_groups(stem_out),
            stem_out),
            nn.GELU(),
        )

        dpr = [x.item() for x in torch.linspace(0,
        drop_path_rate, steps=3 + 4 + 6 + 3)]

        self.layer1 = nn.Sequential(
            BottleneckR2Plus1D(stem_out, 64, stride_t=2,
            stride_s=2, use_se=False, drop_path=dpr[0]),
            BottleneckR2Plus1D(64 * 4, 64, stride_t=1,
            stride_s=1, use_se=False, drop_path=dpr[1]),
            BottleneckR2Plus1D(64 * 4, 64, stride_t=1,
            stride_s=1, use_se=False, drop_path=dpr[2]),
        )  # -> [B, 256, 4, 32, 32]

        self.layer2 = nn.Sequential(
            BottleneckR2Plus1D(64 * 4, 128, stride_t=1,
            stride_s=2, use_se=False, drop_path=dpr[3]),
            BottleneckR2Plus1D(128 * 4, 128, stride_t=1,
            stride_s=1, use_se=False, drop_path=dpr[4]),
            BottleneckR2Plus1D(128 * 4, 128, stride_t=1,
            stride_s=1, use_se=False, drop_path=dpr[5]),
            BottleneckR2Plus1D(128 * 4, 128, stride_t=1,
            stride_s=1, use_se=False, drop_path=dpr[6]),
        )  # -> [B, 512, 4, 16, 16]

        self.layer3 = nn.Sequential(
            BottleneckR2Plus1D(128 * 4, 256, stride_t=1,
            stride_s=2, use_se=True, drop_path=dpr[7]),
            BottleneckR2Plus1D(256 * 4, 256, stride_t=1,
            stride_s=1, use_se=True, drop_path=dpr[8]),
            BottleneckR2Plus1D(256 * 4, 256, stride_t=1,
            stride_s=1, use_se=True, drop_path=dpr[9]),
            BottleneckR2Plus1D(256 * 4, 256, stride_t=1,
            stride_s=1, use_se=True, drop_path=dpr[10]),
            BottleneckR2Plus1D(256 * 4, 256, stride_t=1,
            stride_s=1, use_se=True, drop_path=dpr[11]),
            BottleneckR2Plus1D(256 * 4, 256, stride_t=1,
            stride_s=1, use_se=True, drop_path=dpr[12]),
        )  # -> [B, 1024, 4, 8, 8]

        self.layer4 = nn.Sequential(
            BottleneckR2Plus1D(256 * 4, 256, stride_t=1,
            stride_s=2, use_se=True, drop_path=dpr[13]),
            BottleneckR2Plus1D(256 * 4, 256, stride_t=1,
            stride_s=1, use_se=True, drop_path=dpr[14]),
            BottleneckR2Plus1D(256 * 4, 256, stride_t=1,
            stride_s=1, use_se=True, drop_path=dpr[15]),
        )  # -> [B, 1024, 4, 4, 4]

        self._init_weights()

    def _init_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv3d):
                nn.init.kaiming_normal_(m.weight, mode='
                fan_out', nonlinearity='relu')
                if m.bias is not None:
                    nn.init.zeros_(m.bias)
            elif isinstance(m, nn.GroupNorm):
                nn.init.ones_(m.weight)
                nn.init.zeros_(m.bias)

    def forward(self, x: torch.Tensor):
        x = self.stem(x)        # [B, 64, 8, 64, 64]
        x = self.layer1(x)      # [B, 256, 4, 32, 32]
        s2 = self.layer2(x)     # [B, 512, 4, 16, 16]
        s3 = self.layer3(s2)    # [B, 1024, 4, 8, 8]
        s4 = self.layer4(s3)    # [B, 1024, 4, 4, 4]
        return {"s2": s2, "s3": s3, "s4": s4}


class TransformerBlock(nn.Module):
    def __init__(self, embed_dim: int, num_heads: int,
    mlp_ratio: float = 4.0, dropout: float = 0.1):
        super().__init__()
        self.norm1 = nn.LayerNorm(embed_dim)
        self.attn = nn.MultiheadAttention(embed_dim,
        num_heads, dropout=dropout, batch_first=True)
        self.norm2 = nn.LayerNorm(embed_dim)
        hidden = int(embed_dim * mlp_ratio)
        self.mlp = nn.Sequential(
            nn.Linear(embed_dim, hidden),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(hidden, embed_dim),
            nn.Dropout(dropout),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = x + self.attn(self.norm1(x), self.norm1(x), self
        .norm1(x), need_weights=False)[0]
        x = x + self.mlp(self.norm2(x))
        return x


class TransformerEncoder(nn.Module):
    def __init__(self, depth: int, embed_dim: int, num_heads
    : int, mlp_ratio: float = 4.0, dropout: float = 0.1):
        super().__init__()
        self.blocks = nn.ModuleList([TransformerBlock(
        embed_dim, num_heads, mlp_ratio, dropout) for _ in
        range(depth)])

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        for blk in self.blocks:
            x = blk(x)
        return x


def sinusoidal_positional_encoding_from_positions(positions:
 torch.Tensor, dim: int, base: float = 10000.0) -> torch.
Tensor:
    assert dim % 2 == 0, "Embedding dimension must be even
    for sinusoidal encoding."
    device = positions.device
    inv_freq = 1.0 / (base ** (torch.arange(0, dim, 2,
    device=device).float() / dim))
```

```
    pos = positions.unsqueeze(-1) * inv_freq
    pe = torch.cat([torch.sin(pos), torch.cos(pos)], dim=-1)
    return pe


def fixed_sinusoidal_encoding(length: int, dim: int, device:
 torch.device) -> torch.Tensor:
    assert dim % 2 == 0
    position = torch.arange(length, device=device).float().
    unsqueeze(0)
    return sinusoidal_positional_encoding_from_positions(
    position, dim)


class TemporalMotionGate(nn.Module):
    def __init__(self, alpha: float = 0.5, eps: float = 1e
    -6):
        super().__init__()
        self.alpha = nn.Parameter(torch.tensor(alpha, dtype=
        torch.float32))
        self.eps = eps

    @staticmethod
    def _rgb_to_gray(x: torch.Tensor) -> torch.Tensor:
        r, g, b = x[:, 0], x[:, 1], x[:, 2]
        return 0.2989 * r + 0.5870 * g + 0.1140 * b

    def forward(self, clip: torch.Tensor, T_out: int) ->
    torch.Tensor:
        gray = self._rgb_to_gray(clip)                  # [B
        , 16, H, W]
        dt = torch.abs(gray[:, 1:] - gray[:, :-1])      # [B
        , 15, H, W]
        dt = F.pad(dt, (0, 0, 0, 0, 1, 0))              # [B
        , 16, H, W]
        mean = dt.mean(dim=[1, 2, 3], keepdim=True)
        std = dt.std(dim=[1, 2, 3], keepdim=True) + self.eps
        dt_norm = (dt - mean) / std
        dt_vec = dt_norm.mean(dim=[2, 3], keepdim=False)  #
        [B, 16]
        dt_vec = dt_vec.unsqueeze(1)                     #
        [B, 1, 16]
        dt_resampled = F.interpolate(dt_vec, size=T_out,
        mode='linear', align_corners=True).squeeze(1)  # [B,
         T_out]
        gate = 1.0 + torch.clamp(self.alpha, 0.3, 1.5) *
        torch.tanh(dt_resampled)
        return gate  # [B, T_out]


class CosineClassifier(nn.Module):
    def __init__(self, in_dim: int, num_classes: int, scale:
     float = 20.0):
        super().__init__()
        self.weight = nn.Parameter(torch.randn(num_classes,
        in_dim))
        nn.init.xavier_uniform_(self.weight)
        self.scale = nn.Parameter(torch.tensor(scale, dtype=
        torch.float32))

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x_norm = F.normalize(x, dim=-1)
        w_norm = F.normalize(self.weight, dim=-1)
        logits = self.scale * F.linear(x_norm, w_norm)
        return logits


class VideoActionRecognitionModel(nn.Module):
    """
    High-accuracy action recognition with:
      - Hierarchical 3D CNN encoder (R(2+1)D + SE + DropPath
      )
      - Multi-scale temporal tokens per clip (stages s2/s3/
      s4)
      - Positional encodings: time (from clip_indices),
      segment, view, scale
      - Temporal motion gating
      - Per-clip temporal transformer
      - Global transformer across all clips (segments x
      views x scales)
      - Cosine classifier

    Forward:
      clips: List[num_segments][num_views_per_segment] of [B
      , 3, 16, 256, 256]
      clip_indices: Optional[List[num_segments]] of [B, 16]
    Returns:
      [B, 66] logits
    """
```

```
    def __init__(
        self,
        num_classes: int = 66,
        embed_dim: int = 1024,
        per_clip_depth: int = 2,
        per_clip_heads: int = 16,
        global_depth: int = 6,
        global_heads: int = 16,
        dropout: float = 0.1,
    ):
        super().__init__()
        self.num_classes = num_classes
        self.embed_dim = embed_dim  # REQUIRED attribute

        # Encoder
        self.encoder = VideoEncoder3D(in_channels=3,
        drop_path_rate=0.1)

        # Stage projections to embed_dim
        self.stage2_proj = nn.Sequential(nn.LayerNorm(512),
        nn.Linear(512, embed_dim))
        self.stage3_proj = nn.Sequential(nn.LayerNorm(1024),
         nn.Linear(1024, embed_dim))
        self.stage4_proj = nn.Sequential(nn.LayerNorm(1024),
         nn.Linear(1024, embed_dim))

        # Scale embeddings
        self.scale_embeds = nn.Parameter(torch.randn(3,
        embed_dim))
        nn.init.trunc_normal_(self.scale_embeds, std=0.02)

        # Positional encoding scales: [time, segment, view,
        scale]
        self.pos_scales = nn.Parameter(torch.ones(4))

        # Segment/view id MLPs
        self.segment_mlp = nn.Sequential(
            nn.Linear(1, embed_dim // 2),
            nn.GELU(),
            nn.Linear(embed_dim // 2, embed_dim),
        )
        self.view_mlp = nn.Sequential(
            nn.Linear(1, embed_dim // 2),
            nn.GELU(),
            nn.Linear(embed_dim // 2, embed_dim),
        )

        # Temporal motion gate
        self.motion_gate = TemporalMotionGate(alpha=0.5)

        # Per-clip temporal transformer
        self.per_clip_transformer = TransformerEncoder(depth
        =per_clip_depth, embed_dim=embed_dim, num_heads=
        per_clip_heads, dropout=dropout)

        # Global fusion transformer
        self.cls_token = nn.Parameter(torch.zeros(1, 1,
        embed_dim))
        nn.init.trunc_normal_(self.cls_token, std=0.02)
        self.global_transformer = TransformerEncoder(depth=
        global_depth, embed_dim=embed_dim, num_heads=
        global_heads, dropout=dropout)

        # Final pooling and classifier
        self.final_norm = nn.LayerNorm(embed_dim)
        self.classifier = CosineClassifier(embed_dim,
        num_classes, scale=20.0)

        # Init linear layers
        for m in [self.stage2_proj, self.stage3_proj, self.
        stage4_proj, self.segment_mlp, self.view_mlp]:
            for l in m.modules():
                if isinstance(l, nn.Linear):
                    nn.init.xavier_uniform_(l.weight); nn.
                    init.zeros_(l.bias)

    @staticmethod
    def _spatial_pool(x: torch.Tensor) -> torch.Tensor:
        return x.mean(dim=[-1, -2])  # [B, C, T] -> [B, C, T
        ]

    def _compute_time_pos(self, B: int, T: int, device:
    torch.device, clip_indices_b16: Optional[torch.Tensor])
    -> torch.Tensor:
        if clip_indices_b16 is None:
            time_pe = fixed_sinusoidal_encoding(T, self.
            embed_dim, device=device).expand(B, -1, -1).
            contiguous()
        else:
```

```
            idx = clip_indices_b16.float().unsqueeze(1)  # [
            B, 1, 16]
            idx_down = F.interpolate(idx, size=T, mode='
            linear', align_corners=True).squeeze(1)  # [B, T
            ]
            time_pe =
            sinusoidal_positional_encoding_from_positions(
            idx_down, self.embed_dim)  # [B, T, C]
        return time_pe

    def _segment_pos(self, B: int, T: int, segment_id: int,
    device: torch.device) -> torch.Tensor:
        seg_norm = torch.tensor([[float(segment_id)]],
        device=device)
        seg_pe = self.segment_mlp(seg_norm)  # [1, C]
        seg_pe = seg_pe.unsqueeze(1).expand(B, T, -1).
        contiguous()
        return seg_pe

    def _view_pos(self, B: int, T: int, view_id: int, device
    : torch.device) -> torch.Tensor:
        view_norm = torch.tensor([[float(view_id)]], device=
        device)
        view_pe = self.view_mlp(view_norm)  # [1, C]
        view_pe = view_pe.unsqueeze(1).expand(B, T, -1).
        contiguous()
        return view_pe


    def forward(self, clips: List[List[torch.Tensor]],
    clip_indices: Optional[List[torch.Tensor]] = None) ->
    torch.Tensor:
        assert isinstance(clips, (list, tuple)) and len(
        clips) > 0
        num_segments = len(clips)
        B = clips[0][0].shape[0]
        device = clips[0][0].device

        all_tokens = []

        for s_idx in range(num_segments):
            segment_clips = clips[s_idx]
            num_views = len(segment_clips)

            for v_idx in range(num_views):
                clip = segment_clips[v_idx]  # [B, 3, 16,
                256, 256]
                feats = self.encoder(clip)   # dict of multi
                -scale features
                # Spatial pooling -> [B, C_stage, T]
                s2 = self._spatial_pool(feats["s2"])  # [B,
                512, 4]
                s3 = self._spatial_pool(feats["s3"])  # [B,
                1024, 4]
                s4 = self._spatial_pool(feats["s4"])  # [B,
                1024, 4]

                # Transpose to [B, T, C_stage]
                s2 = s2.transpose(1, 2).contiguous()
                s3 = s3.transpose(1, 2).contiguous()
                s4 = s4.transpose(1, 2).contiguous()

                # Project to embed_dim
                t2 = self.stage2_proj(s2)  # [B, 4, C]
                t3 = self.stage3_proj(s3)  # [B, 4, C]
                t4 = self.stage4_proj(s4)  # [B, 4, C]

                # Motion gating on T=4, replicated across
                scales
                gate = self.motion_gate(clip, T_out=t2.shape
                [1])  # [B, 4]
                gate = gate.unsqueeze(-1)  # [B, 4, 1]
                t2 = t2 * gate
                t3 = t3 * gate
                t4 = t4 * gate

                # Positional encodings
                clip_idx = None
                if clip_indices is not None:
                    assert len(clip_indices) == num_segments
                    clip_idx = clip_indices[s_idx]  # [B,
                    16]
                time_pe = self._compute_time_pos(B, t2.shape
                [1], device, clip_idx)  # [B, 4, C]
                seg_pe = self._segment_pos(B, t2.shape[1],
                s_idx, device)           # [B, 4, C]
                view_pe = self._view_pos(B, t2.shape[1],
                v_idx, device)             # [B, 4, C]
```

```
                # Scale embeddings per scale
                scale2 = self.scale_embeds[0].view(1, 1, -1)
                .expand(B, t2.shape[1], -1)
                scale3 = self.scale_embeds[1].view(1, 1, -1)
                .expand(B, t3.shape[1], -1)
                scale4 = self.scale_embeds[2].view(1, 1, -1)
                .expand(B, t4.shape[1], -1)

                # Add positional signals
                t2 = t2 + self.pos_scales[0] * time_pe +
                self.pos_scales[1] * seg_pe + self.
                pos_scales[2] * view_pe + self.pos_scales[3]
                 * scale2
                t3 = t3 + self.pos_scales[0] * time_pe +
                self.pos_scales[1] * seg_pe + self.
                pos_scales[2] * view_pe + self.pos_scales[3]
                 * scale3
                t4 = t4 + self.pos_scales[0] * time_pe +
                self.pos_scales[1] * seg_pe + self.
                pos_scales[2] * view_pe + self.pos_scales[3]
                 * scale4

                # Per-clip temporal refinement over multi-
                scale tokens combined
                clip_tokens = torch.cat([t2, t3, t4], dim=1)
                  # [B, 12, C]
                clip_tokens = self.per_clip_transformer(
                clip_tokens)  # [B, 12, C]
                all_tokens.append(clip_tokens)

        # Concatenate all clips along sequence
        if len(all_tokens) == 1:
            seq = all_tokens[0]  # [B, L, C]
        else:
            seq = torch.cat(all_tokens, dim=1)  # [B,
            total_L, C]

        # Add CLS
        cls_tok = self.cls_token.expand(B, 1, self.embed_dim
        ).to(device=device, dtype=seq.dtype)
        seq = torch.cat([cls_tok, seq], dim=1)  # [B, 1 + L,
         C]

        # Global fusion
        fused = self.global_transformer(seq)  # [B, 1 + L, C
        ]

        # Pool: combine CLS with mean of non-CLS
        cls_feat = fused[:, 0]
        mean_feat = fused[:, 1:].mean(dim=1) if fused.shape
        [1] > 1 else cls_feat
        feat = self.final_norm(0.5 * (cls_feat + mean_feat))

        logits = self.classifier(feat)  # [B, num_classes]
        return logits
```