ESSENTIAL: Episodic and Semantic Memory Integration for Video Class-Incremental Learning

Supplementary Material

In this supplementary material, we provide architecture/implementation/metrics/dataset details, and additional experimental results to complement the main paper. We organize the supplementary material as follows:

1. Architecture Details
2. Complete implementation details
3. Evaluation metric details
4. Additional experimental results
5. Dataset details

1. Architecture Details

In this section, we provide architecture details of `ESSENTIAL`. We employ CLIP [13] as our frame-level visual encoder. We use $L = 8$ of clip length and both MR module and semantic prompt is task-specific by default.

1.1. Details of memory retrieval.

In Table 1, we provide stage-wise details of memory retrieval module in `ESSENTIAL` during the training stage. We omit the task index k for simplicity. Given an input video clip $\mathbf{X}$ with a length of 8, we first pass it through the frozen visual encoder, resulting in frame-level features $\mathbf{S}_{\text{dense}} \in \mathbb{R}^{L \times d}$. Note that we omit the notation 'frame-level' for simplicity. In this case, we set the number of frame features to $l = 2$. Subsequently, to obtain temporally sparse features $\mathbf{S}_{\text{sparse}}$, we temporally sub-sample the $\mathbf{S}_{\text{dense}}$ using randomly selected frame indices. Then, we take semantic prompt, $\mathbf{P}$, and pass both the semantic prompt $\mathbf{P}$ and the temporally sparse frame-level features $\mathbf{S}_{\text{sparse}}$ through the MR module. The multi-head cross-attention (`MHCA`) layer enables the effectively integration of between $\mathbf{P}$ and $\mathbf{S}_{\text{sparse}}$. Afterward, we pass the output tensors from the `MHCA` layer through the feed-forward network (`FFN`), resulting the retrieved dense features $\tilde{\mathbf{S}}_{\text{dense}}$. Please refer to the Section 3.2 of the main paper for the remaining training process.

1.2. Temporal encoder architecture.

As shown in Figure 3 of the main paper, we employ a temporal encoder, denoted as $f_t(\cdot;\theta_t)$, to obtain a clip-level feature vector. The temporal encoder consists of one or more Transformer layers, including attention layers and feed-forward network (`FFN`). Among the two design choices for the attention layer: i) self-attention and ii) cross-attention, we adopt cross-attention. In the *self-attention* architecture, we concatenate $\mathbf{S}_{\text{sparse}}$ and semantic prompt $\mathbf{P}$ and pass it through self-attention layers. In the *cross-attention* architecture, we first initialize a learnable feature vector as a query and pass it through cross-attention layers, using

Table 1. **Stage-wise details of the Memory retrieval.** We provide a detailed description of each operation performed from an input clip to frame-level reconstruction. The input for this example is one video clip consisting of 8 frames (skip batch term). In the description, D and T represent the embedding dimension and temporal sequence length, respectively. We omit the Layer Normalization and activation functions for simplicity.

	Memory Retrieval Module	
Stage	**Remark**	**Output Tensor Shape**
Feed-Forward Network	Linear Down *with ratio* = 0.25	$\tilde{\mathbf{S}}_{\text{dense}} : 8 \times 768$
Feed-Forward Network	Linear Up & GELU *with ratio* = 4.0	$\mathbf{P} : 8 \times 3072$
Cross-Attention	MHCA$(\mathbf{P}, \mathbf{S}_{\text{sparse}})$ window shape: *time*	$\mathbf{P} : 8 \times 768$
Input of MR module	Taking $\mathbf{P}$ from *semantic memory*	$\mathbf{P} : 8 \times 768$ $\mathbf{S}_{\text{sparse}} : 2 \times 768$
Temporal sub-sampling	Random selection from $\mathbf{S}_{\text{dense}}$	$\mathbf{S}_{\text{sparse}} : 2 \times 768$
Extracting frame-level features	Frozen $f_s(\cdot)$	$\mathbf{S}_{\text{dense}} : 8 \times 768$
Input video clip	-	$\mathbf{X} : 8 \times 224 \times 224 \times 3$

$\mathbf{S}_{\text{sparse}}$ as a key and value. In Table 6, we show the experimental results analyzing the effect of different temporal encoder architectures. In the temporal encoder with cross-attention, we initialize a learnable query token and perform the attention operation, where key and value come from input frame-level features. The temporal encoder enhances the temporal understanding of frame-level features obtained from the frozen visual encoder, addressing their limitations in capturing temporal context.

1.3. Architecture details of the baselines in the MR architecture ablation study.

In Figure 1, we visualize the architectures of the baselines used in the MR module architectures ablation study, as presented in Table 3 (b) of the main paper. In Figure 1 (a), we present the *MLP w/o semantic memory* baseline, which includes three fully-connected layer with GELU activation without semantic memory. For the fully-connected layer, we flatten $\mathbf{S}_{\text{sparse}}$, resulting in $\mathbb{R}^{l \cdot d}$. After passing the $\mathbf{S}_{\text{sparse}}$ through *MLP*, we reshape the output tensors, $\tilde{\mathbf{S}}_{\text{dense}} \in \mathbb{R}^{L \cdot d}$ to $\mathbb{R}^{L \times d}$. In Figure 1 (b), we present the architectures of *Add* and *Multiply* baselines, which serves as naive baselines to integrate semantic prompt, $\mathbf{P}$ and temporally sparse features, $\mathbf{S}_{\text{sparse}}$. To match a clip length between $\mathbf{P}$ and $\mathbf{S}_{\text{sparse}}$, we conduct nearest neighbor interpolation $\mathbf{S}_{\text{sparse}}$ along the temporal axis. In Figure 1 (c), we show the *Self-attention* baseline, which replaces the cross-attention layer of MR module with self-attention layer. We append the $\mathbf{S}_{\text{sparse}}$ to the $\mathbf{P}$ and pass them through a self-attention layer and a feed-forward network. Afterward, we detach the $\mathbf{S}_{\text{sparse}}$ from the output tensors to obtain the $\tilde{\mathbf{S}}_{\text{dense}}$.

2. Implementation details

2.1. Training.

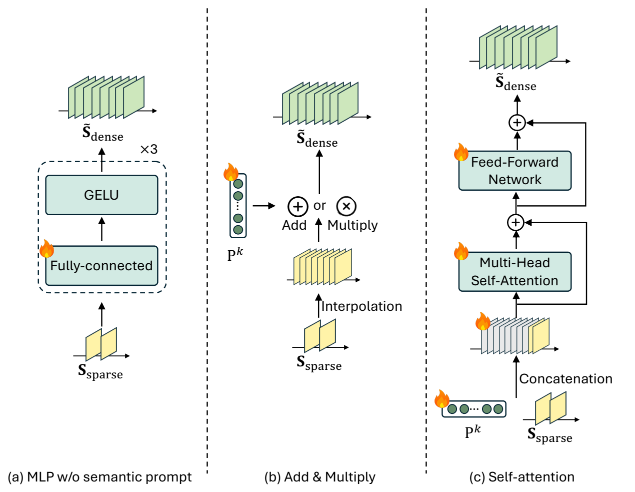
We use CLIP ViT-B/16 [13] as the visual encoder and keep it frozen. The temporal encoder consists of a 3-layer cross-attention architecture, while the memory retrieval module utilizes a 1-layer cross-attention architecture. For SSV2 [6], we attach and learn a lightweight adapter [19] during the base task learning stage and then freeze it in the subsequent incremental learning stages. We conduct our experiments with a batch size of 24 for all datasets except UCF-101 [15], where the batch size is set to 10. The learning rate is set to 0.001, following a cosine scheduling strategy. For the current task training stage, we train for 50 epochs for all datasets except UCF-101, where we train for 30 epochs. During the rehearsal stage, we train for 30 epochs across all datasets. We conduct the experiments with 24 NVIDA GeForce RTX 3090 GPUs for all datasets except UCF-101 and ActivityNet [2], where we use 8 GPUs. We implement ESSENTIAL using PyTorch and build upon the code of VideoMAE [16]. We follow the prior works [5, 20] in adopting the local cross-entropy loss, where we only compute the loss between current task logits and ground truth labels and apply logit masking for the classes belong to the other tasks. We set both the α and β to 1.0 for all experiments.

2.2. Memory update.

After each training stage, given an input clip length of L, we store N_s per class feature vectors with a clip length of l ($l \ll L$) in the episodic memory and a semantic prompt with a clip length of L in the semantic memory as shown in Figure 2. In Tables 1-3 of the main paper, we use the optimal combination of ($N_s \times l$) for each dataset on two benchmarks [11, 18]. In the TCD benchmark, we select ($N_s \times l$) combinations as follows: UCF-101 (10×1), HMDB51 (10×1) and SSV2 (4×4). In the vCLIMB benchmark, we select ($N_s \times l$) combinations as follows: ActivityNet (32×2), Kinetics-400 (32×2) and UCF-101 (16×2). In Table 3, we show the experimental results exploring how to determine the optimal ($N_s \times l$) configuration.

3. Evaluation Metric.

Performance We evaluate the performance using two metrics: Average Accuracy (AA) and Average Incremental Accuracy (AIA). AA on k-th task, AA_k is the average classification accuracy of the model evaluated up to the k-th task. Following vCLIMB [18], we evaluate UCF-101, ActivityNet, and Kinetics-400 using the final average accuracy, AA_K, where K denotes the total number of tasks. Following TCD [11], we evaluate UCF-101 and SSV2 using the Average Incremental Accuracy (AIA), defined as AIA $= \frac{1}{K}\sum_{k=1}^{K} AA_k$, where K denotes the total number of tasks.



$\tilde{\mathbf{S}}_{\text{dense}}$
×3
GELU
Fully-connected
$\mathbf{S}_{\text{sparse}}$
(a) MLP w/o semantic prompt
$\tilde{\mathbf{S}}_{\text{dense}}$
P^k
Add
or
Multiply
Interpolation
$\mathbf{S}_{\text{sparse}}$
(b) Add & Multiply
$\tilde{\mathbf{S}}_{\text{dense}}$
Feed-Forward Network
Multi-Head Self-Attention
Concatenation
P^k
$\mathbf{S}_{\text{sparse}}$
(c) Self-attention

Figure 1. **Architecture visualization of the different memory retrieval methods.** We visualize the various MR architectures presented in Table 3 (b) of the main paper.

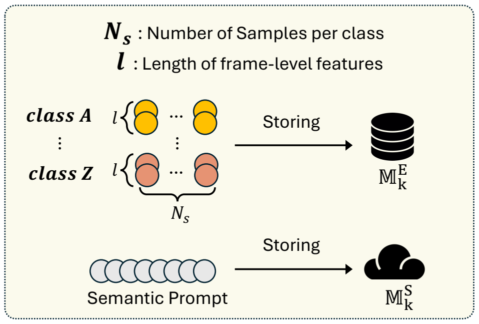


Figure 2. **Memory Update.** After training k-th task, we update episodic memory and semantic memory. For episodic memory, we select N_s samples per class and subsample l frame-level features from each sample among the training samples of the current task. For semantic memory, we store the semantic prompt learned from the current task.

Memory usage. For details on the memory usage of ESSENTIAL, please refer to Section 4.2 of the main paper. We compute the memory usage for storing RGB frames using the formula $N_s \times L \times 224 \times 224 \times 3$ where N_s is the number of samples per class and L is the temporal length. We assume that a single frame has a resolution of 224×224 pixels with 3 color channels. For centroid feature vectors [12], we calculate the memory usage as $N_c \times d_c \times 4$, where N_c is the number of centroids, and d_c is the centroid dimension. For the STSP [3], raw examples or features are not stored in episodic memory. Instead, it stores a covariance matrix. Thus, we assume that the input consists of frames with a resolution of 224×224 pixels and then compute the memory usage of the covariance matrix accordingly. Since there is no publicly available code of ST-Prompt [12] and STSP [3], we calculate the memory usage based on the configuration described in the paper. [1]

Table 2. **Memory usage of VCIL methods, including additional overhead from MR modules in `ESSENTIAL`.**

Method	Params	Mem.	MR module	Acc.
ST-Prompts	151M	80.5MiB	-	85.1
PIVOT (RAW frame)	161M	26.4GiB	-	93.4
PIVOT (JPEG compressed)	161M	7.5GiB	-	93.4
`ESSENTIAL`$_{task}$	162M	9.7MiB	270MiB	**95.8**
`ESSENTIAL`$_{global}$	99M	9.7MiB	27MiB	94.8

Memory consumption by MR module. Since we consider only raw RGB frames and feature vectors extracted from video samples when calculating memory usage, we also report the additional memory overhead introduced by MR module for completeness. Each MR module contains approximately 7M parameters, occupying around 27 MiB in FP32 precision. In the `ESSENTIAL`$_{\text{task}}$ configuration, where a separate MR module is maintained per task, this results in a total of $\sim 27 \times$ #Tasks MiB memory consumption.

As shown in the Table 2, we report the memory usage of each method (Mem) along with the additional memory required by our MR modules (MR) in vCLIMB UCF-101, 10 tasks setting. For a fair comparison, we also include the result that estimates PIVOT's memory usage assuming it stores raw video frames in JPEG-compressed format. `ESSENTIAL` is more memory-efficient than storing raw RGB data—with JPEG-compression— (7.5GiB *vs.* 279.7MiB).

For scenarios where the MR module memory may be a concern, we also provide results of a variant using a single global MR module across tasks (`ESSENTIAL`$_{global}$). As shown in Table below and in Table 3 (a) of the main paper, `ESSENTIAL`$_{global}$ achieves favorable performance (94.8%) with significantly lower memory usage (27MiB). Optimizer state occupies ~54MiB during the incremental training stage. However, since we freeze the MR module during the rehearsal stage, it does not incur any optimizer memory overhead.

[1]ST-Prompt reports the size of the prompts but not the size of the centroids. Additionally, they do not take data types into account.

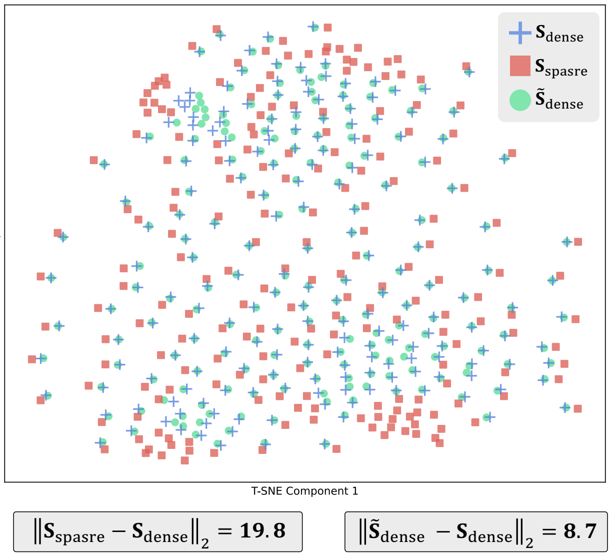


Figure 3. **T-SNE [17] visualization of memory retrieval of the MR module on third task in SSV2 (10×9) tasks.**

4. Additional experimental results.

4.1. Additional analysis.

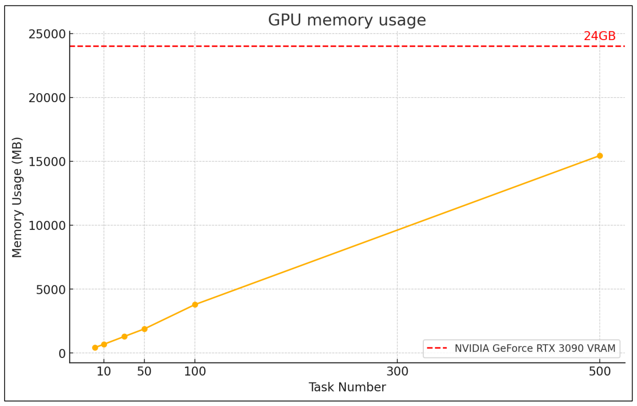


Figure 4. **GPU memory usage of `ESSENTIAL` as the number of tasks increases during training on the SSV2 dataset.** We plot the GPU memory usage required for training as the number of tasks increases up to 500 on a single RTX 3090 GPU. The GPU memory capacity of the RTX 3090 is represented by a red dashed line. Notably, `ESSENTIAL` utilizes only approximately 63% of the available GPU memory even when the number of tasks reaches 500, showcasing its remarkable memory efficiency.

Visualization of memory retrieval. We provide a T-SNE visualization for Task 3 of SSV2 (10×9) in Figure 3. Additionally, for a fair comparison, we use the validation set for this analysis. We compare the L_2 distances from the original temporally dense feature vectors ($\mathbf{S}_{\text{dense}}$) to (i) the temporally sparse feature vectors ($\mathbf{S}_{\text{sparse}}$) and (ii) the retrieved

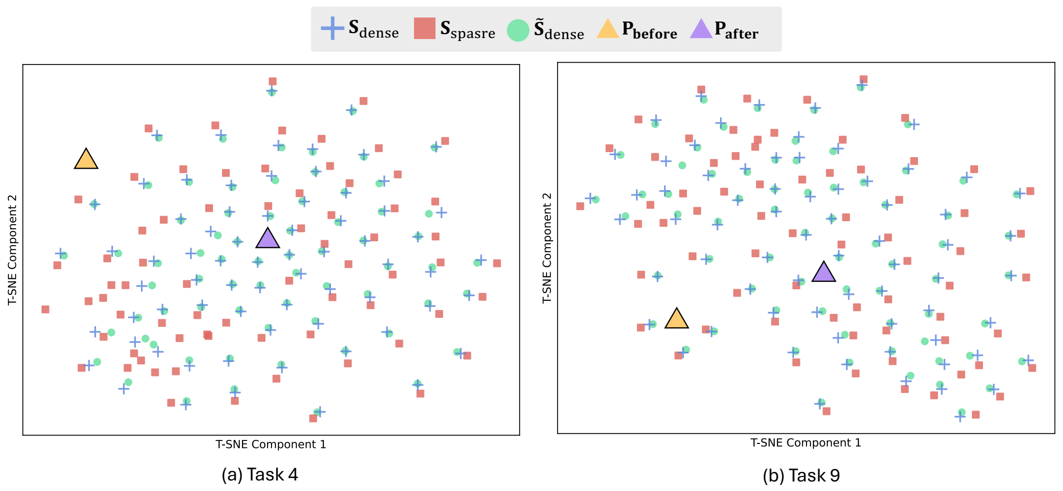


Figure 5. **T-SNE of semantic prompts and task feature vectors in SSV2 (10×9) tasks.** We present the T-SNE [17] visualization of semantic prompt and task feature vectors to observe what the semantic prompt learns. (a) The semantic prompt before training $\mathbf{P}_{\text{before}}$ (triangle) is far from the current data feature vectors ($\mathbf{S}_{\text{dense}}$,$\mathbf{S}_{\text{sparse}}$, and $\tilde{\mathbf{S}}_{\text{dense}}$). After training, we observe that the semantic prompt $\mathbf{P}_{\text{after}}$ (triangle) is located at the center of the current data feature vectors (triangle $\rightarrow$ triangle). (b) Similar to (a), in the ninth task, the semantic prompt also moves toward the center of the task feature vectors after training. This indicates that the semantic prompt captures general knowledge of the current task.

feature vectors ($\tilde{\mathbf{S}}_{\text{dense}}$). As shown in Figure 3, the distance between $\mathbf{S}_{\text{dense}}$ and $\tilde{\mathbf{S}}_{\text{dense}}$ is significantly smaller than the distance between $\mathbf{S}_{\text{dense}}$ and $\mathbf{S}_{\text{sparse}}$ (19.8 *vs*. 8.7). This result demonstrates that the MR module effectively retrieves temporally dense feature vectors from temporally sparse feature vectors.

GPU memory usage of MR module. The MR module consists of cross attention and MLP components. Each MR module occupies approximately 27MiB of GPU memory when using float32 precision. Notably, we train the k-th MR module during the *k*-th task training stage and then freeze it for subsequent use. We plot the GPU memory usage of ESSENTIAL as the number of task increases as shown in Figure 4. The memory usage remains approximately 15GB even when the number of task reaches 500, which is a reasonably high number in continual learning. This demonstrates the memory-efficiency of ESSENTIAL, even in scenarios with a large number of total tasks. Additionally, if GPU memory is somehow very limited, we can use a global MR module as a memory-efficient alternative while still achieving comparable performance, as shown in Table 4 (a) of the main paper.

Visualization of semantic prompt. We present the T-SNE [17] visualization of semantic prompt and task feature vectors to observe what the semantic prompt learns. As shown in Figure 5, the semantic prompt before training $\mathbf{P}_{\text{before}}$ (triangle) before training is far from the current task feature vectors ($\mathbf{S}_{\text{dense}}$, $\mathbf{S}_{\text{sparse}}$, and $\tilde{\mathbf{S}}_{\text{dense}}$). After training, we observe that the semantic prompt (triangle) is located at the center of the current task feature vectors. This indicates that the semantic prompt captures general knowledge of the current task.

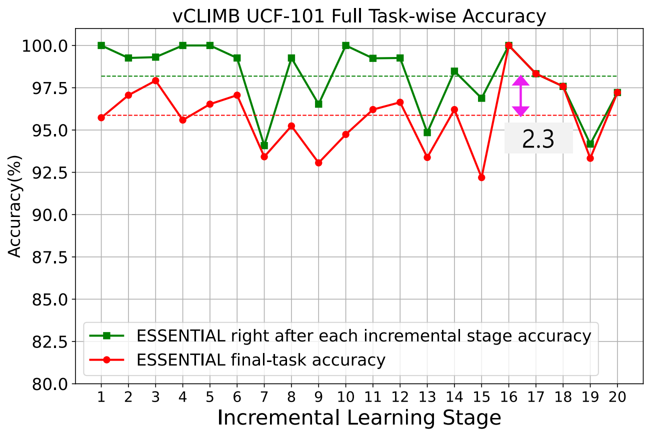
What is the optimal $(N_s \times l)$ configuration for episodic memory? In Table 3, we investigate the optimal configuration of $(N_s \times l)$ for episodic memory, where N_s and l denote the number of samples stored per class and the number of frame features. Since the UCF-101 is a static-biased dataset [4, 10], reducing the number of frames and storing more samples yields the highest performance. On the other hand, we observe that for the SSV2 dataset, ESSENTIAL achieves the highest performance when $N_s = 4$ and $l = 4$. The result indicates that ESSENTIAL needs to store more frames than the UCF-101 setting to capture the essential temporal dynamics since the SSV2 is a temporal-biased dataset [1, 8].

Comparison with rehearsal-free method. We compare ESSENTIAL with rehearsal-free baselines in terms of accuracy and final backward forgetting (BWF) on the vCLIMB UCF-101 dataset with 20 tasks. As shown in Table 4, ESSENTIAL significantly outperforms CODA-Prompt [14], a rehearsal-free method, achieving both higher accuracy (95.9% vs. 74.2%) and lower forgetting (BWF 2.3 vs. 13.3). This indicates that ESSENTIAL effectively mitigates forgetting even without relying on rehearsal buffers. To further illustrate this, we plot full task-wise accuracy curves—i.e., the diagonal and last row of the full accuracy matrix—in Figure 6, showing that ESSENTIAL retains substantially higher accuracy as the number of tasks increases.

Table 3. **Performance comparison with different memory configurations on UCF-101 and SSV2 datasets of TCD benchmark.** We report the Top-1 average incremental accuracy (%) and the total memory usage (MB) for different $(N_s \times l)$ configurations. Here, N_s and l denote the number of samples stored per class and the number of frame features. The **best** results are highlighted.

		UCF-101					
		10 × 5 Tasks		5 × 10 Tasks		2 × 25 Tasks	
Backbone	$(N_s \times l)$	Memory Usage	Acc.	Memory Usage	Acc.	Memory Usage	Acc.
ImageNet pre-trained ViT-B/16	(10×1)		**92.8**		**90.8**		**90.1**
	(5×2)	3.1M	92.1	3.2M	90.2	3.6M	89.1
	(2×5)		90.8		88.2		84.1
CLIP pre-trained ViT-B/16	(10×1)		95.1		**93.9**		**93.3**
	(5×2)	3.1M	**95.3**	3.2M	93.2	3.6M	92.5
	(2×5)		93.0		91.9		89.9

		SSV2			
		10 × 9 Tasks		5 × 18 Tasks	
Backbone	$(N_s \times l)$	Memory Usage	Acc.	Memory Usage	Acc.
ImageNet pre-trained ViT-B/16	(8×2)		42.3		40.8
	(4×4)	8.4M	**44.9**	8.6M	**42.7**
	(2×8)		40.2		38.5
CLIP pre-trained ViT-B/16	(8×2)		47.1		45.8
	(4×4)	8.4M	**48.9**	8.6M	**47.5**
	(2×8)		45.5		44.6



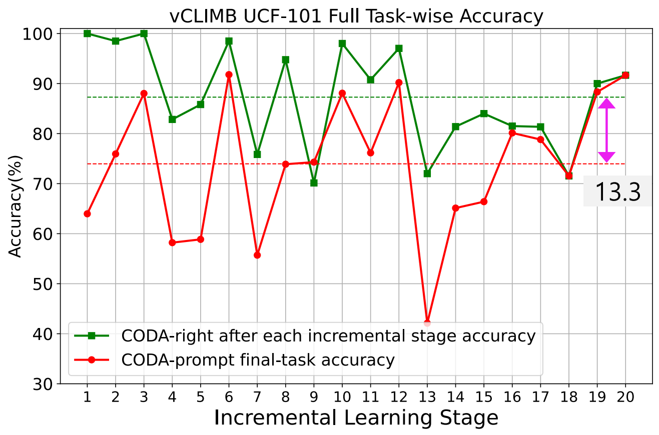


Figure 6. **Task-wise accuracy curves on vCLIMB UCF-101, 20 tasks setting.**

Table 4. **`ESSENTIAL` outperforms rehearsal-free methods in both accuracy and BWF on vCLIMB UCF-101 (20 tasks).**

Method	Rehearsal?	Acc. ↑	BWF ↓
CODA-Prompt [46]	×	74.2	13.3
PIVOT [55]	✓	93.1	3.9
`ESSENTIAL`	✓	**95.9**	**2.3**

4.2. Additional ablation study

To further validate the effect of each component, we present ablation results on the Something-Something-V2 (10 × 9 tasks) dataset. We report the Top-1 average incremental accuracy (%). We use CLIP [13] as a backbone encoder and set $N_s = 4$, and $l = 4$ for episodic memory.

Ablation study on the impact of α and β. `ESSENTIAL` calculates the total loss (L_{total}) by applying a weighted sum of the static matching loss (L_{SM}) and temporal matching loss (L_{TM}) using (α) and β as described in Eq. (10) of the main paper. We show the ablation results on (α) and β in the Table 5. We achieve the highest performance when we set the value of both hyperparameters to 1.

Table 5. **Ablation study on the impact of α and β.**

Loss		Acc.
Static matching (α)	Temporal matching (β)	
0	0	46.3
1	0	46.9
0	1	47.1
0.5	1	47.5
1	0.5	48.1
1	1	**48.9**

Table 6. **Ablation study on temporal encoder architecture.**

Attention strategy	Acc.
Self-attention	48.1
Cross-attention	**48.9**

Effect of temporal encoder architecture. In Table 6, using cross-attention in the temporal encoder, $f_t(\cdot;\theta_t)$, shows favorable performance compared to using self-attention.

Effect of using MR module during inference stage. For each task, we have a dedicated MR module and dedicated

Table 7. **Effect of using MR module during inference stage.**

MR module	Sem. prompt	Inference w/ MR module	Acc.
Global	Global	×	47.4
Global	Global	✓	47.8
Task	Task	×	**48.9**

semantic prompt. Since we do not have task ID information during inference in the class-incremental learning setting, for each test sample, we have to estimate the task ID information to select MR module and semantic prompt. Additionally, we provide experimental results by using a global MR module and global semantic prompts so that we do not need to estimate task ID information during testing. As shown in Table 7, we observe that using the MR module and semantic prompt during inference has little impact on performance (47.4 *vs*. 47.8). Therefore, to improve computation efficiency, we omit the MR module and semantic prompt during testing.

5. Dataset Details

In this section, we provide a detailed description of the datasets.

vCLIMB Benchmark The vCLIMB benchmark [18] includes UCF-101 [15], AcvitiyNet [2], and kinetics-400 [7]. The vCLIMB benchmark provides two experimental settings: a 10 tasks and a 20 tasks configuration. In the 10 tasks setting, the entire set of action classes is partitioned into 10 sequential tasks, whereas in the 20 tasks setting, the classes are split into 20 sequential tasks.

TCD Benchmark The TCD benchmark [11] includes UCF-101 [15], HMDB51 [9], and SSV2 [6]. The UCF-101 in TCD consists of a base task learning 51 classes and provides three settings: 10 classes × 5 tasks, 5 classes × 10 tasks, and 2 classes × 25 tasks. The HMDB51 in TCD consists of a base task learning 26 classes and provides two settings: 5 classes × 5 tasks and 1 class × 25 tasks. The SSV2 in TCD consists of a base task with 84 classes and offers two settings: 10 classes × 9 tasks and 5 classes × 18 tasks.

References

[1] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *ICML*, 2021. 4

[2] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *CVPR*, 2015. 2, 6

[3] Hao Cheng, Siyuan Yang, Chong Wang, Joey Tianyi Zhou, Alex C Kot, and Bihan Wen. Stsp: Spatial-temporal subspace projection for video class-incremental learning. In *ECCV*, 2024. 3

[4] Jinwoo Choi, Chen Gao, Joseph CE Messou, and Jia-Bin Huang. Why can't i dance in the mall? learning to mitigate scene bias in action recognition. In *NeurIPS*, 2019. 4

[5] Qiankun Gao, Chen Zhao, Yifan Sun, Teng Xi, Gang Zhang, Bernard Ghanem, and Jian Zhang. A unified continual learning framework with general parameter-efficient tuning. In *ICCV*, 2023. 2

[6] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. The" something something" video database for learning and evaluating visual common sense. In *ICCV*, 2017. 2, 6

[7] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017. 6

[8] Matthew Kowal, Mennatullah Siam, Md Amirul Islam, Neil DB Bruce, Richard P Wildes, and Konstantinos G Derpanis. A deeper dive into what deep spatiotemporal networks encode: Quantifying static vs. dynamic information. In *CVPR*, 2022. 4

[9] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *ICCV*, 2011. 6

[10] Yingwei Li, Yi Li, and Nuno Vasconcelos. Resound: Towards action recognition without representation bias. In *ECCV*, 2018. 4

[11] Jaeyoo Park, Minsoo Kang, and Bohyung Han. Class-incremental learning for action recognition in videos. In *ICCV*, 2021. 2, 6

[12] Yixuan Pei, Zhiwu Qing, Shiwei Zhang, Xiang Wang, Yingya Zhang, Deli Zhao, and Xueming Qian. Space-time prompting for video class-incremental learning. In *ICCV*, 2023. 3

[13] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 1, 2, 5

[14] James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *CVPR*, 2023. 4

[15] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 2, 6

[16] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training. In *NeurIPS*, 2022. 2

[17] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 9(11):2579–2605, 2008. 3, 4

[18] Andrés Villa, Kumail Alhamoud, Victor Escorcia, Fabian Caba, Juan León Alcázar, and Bernard Ghanem. vclimb: A novel video class incremental learning benchmark. In *CVPR*, 2022. 2, 6

[19] Taojiannan Yang, Yi Zhu, Yusheng Xie, Aston Zhang, Chen Chen, and Mu Li. Aim: Adapting image models for efficient video action recognition. In *ICLR*, 2023. 2

[20] Gengwei Zhang, Liyuan Wang, Guoliang Kang, Ling Chen,

and Yunchao Wei. Slca: Slow learner with classifier alignment for continual learning on a pre-trained model. In *ICCV*, 2023. 2