LookWhen? Fast Video Recognition by Learning When, Where, and What to Compute

●**Ali Salamatian**[1] ●★**Anthony Fuller**[2,3]
Pritam Sarkar[1,3] **James R. Green**[2]
★**Leonid Sigal**[1,3] ★**Evan Shelhamer**[1,2,3]
[1]University of British Columbia [2]Carleton University [3]Vector Institute
●Co-first author ★Co-advising author

Abstract

Transformers dominate video recognition. They split videos into tokens, and processing them has expensive superlinear computational cost. Yet videos are filled with redundancy, so we can question the need for this expense. We introduce LookWhen, a selector–extractor framework that factorizes video recognition into learning *when*, *where*, and *what* to compute. Our shallow selector gets a scaled-down video and quickly scores all tokens across space-time, while our deep extractor gets the top-K selected tokens to approximate full-video representations without actually processing all the tokens. A key challenge is defining effective supervision for selection and extraction. For selection pre-training, we introduce a score on representations that ranks tokens by uniqueness using a simple nearest-neighbor distance. For extraction pre-training, we distill both a video teacher *and* an image teacher, for which we normalize its frame-wise representations to learn *what changes* within videos. Through these strategies, our selector-extractor learns general and efficient representations for feature extraction or fine-tuning to a task. Through experiments on Kinetics-400, SSv2, Epic-Kitchens, Diving48, Jester, and Charades, we show that LookWhen achieves a better accuracy-computation trade-off than efficient models and upgraded baselines of similar size. LookWhen Pareto-dominates in accuracy-FLOPs on 9 of 12 cases (6 tasks $\times$ 2 settings) and roughly matches on 3. In accuracy-throughput, measuring time in practice, LookWhen is more efficient still at 6.7$\times$ faster than InternVideo2-B at equal accuracy.[1]

1 Introduction: Video computation takes too much time and space

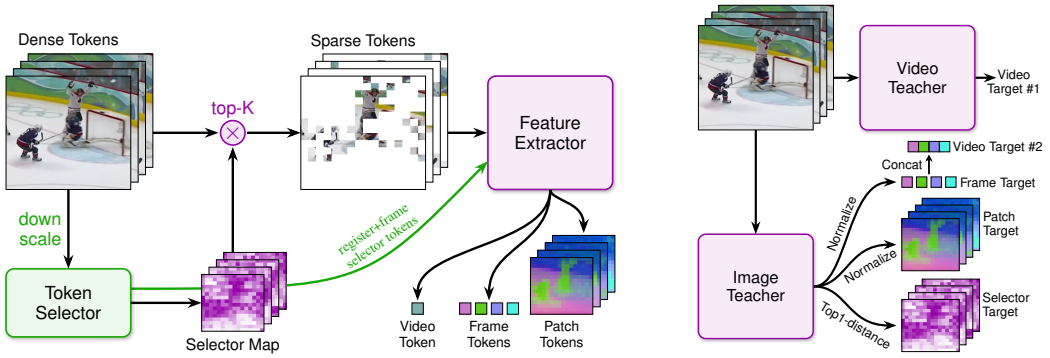
Transformers [1] have revolutionized video modeling [2, 3, 4, 5]. They split videos into several thousand or more tokens for recognition. Computational cost scales superlinearly with the number of tokens [1], which is a drawback for short videos and a real obstacle for long ones. However, not all tokens are needed to compute accurate representations: some tokens are redundant, some can be inferred by their surroundings, and others *capture the scene*. Transformers can choose among tokens, as they naturally handle sparse inputs, and their computation depends only on the number of tokens and not their distribution. Sparsity can thus bring efficiency, but only if we can **select** *when* and *where* to process within each video so that we can **extract** features accurately.

Factorizing *when* (in time), *where* (in space), and *what* (in representation) is not easy. It requires isolating the input tokens that are responsible for the output representation, and quickly, as otherwise it will not boost efficiency. It also requires accurately representing the full video, given only the selected tokens, approximating dense computation with sparse computation.

We divide this approximation into token selection and feature extraction with a model for each. Our efficient selector receives a *downscaled* video, and scores all tokens in the full input. Our

[1]Code and pre-trained models: `https://github.com/alisalamatian1/LookWhen`

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Inference and fine-tuning. (b) Computing targets for pre-training.

Figure 1: **LookWhen's** *shallow* **selector** gets a *downscaled* video and scores tokens on their feature uniqueness (left). Target uniqueness is from our "top1-distance" algorithm, which computes each patch's distance to its nearest neighbor in an image teacher's feature space (bottom right). LookWhen's **extractor** gets the top-K input tokens for *sparse* and *deep* processing. Target features are from a video teacher (top right) and an image teacher (bottom right); we normalize to emphasize within-video *change*. Teachers are only needed during pre-training, so inference and fine-tuning is efficient.

expressive extractor receives *only* the top-K *selected* tokens, yet predicts features of the *full* video. Our **LookWhen** selector-extractor models learn representations of videos that neither model ever fully processes, surpassing the accuracy-compute trade-offs of existing video models.

The pre-training of a general selector-extractor was recently introduced for *images* [6]. Their models learn from a teacher: the selector is trained to predict the teacher's final attention map (*where* to compute) and the extractor is trained to predict the teacher's final representations (*what* to compute). Video offers more redundancy than imagery, thus offering greater opportunity for efficiency via sparsity. Directly learning from a teacher for *video* recognition may not work: it needs a video teacher with relevant [6] and *artifact-free* [7] attention. Attention to irrelevant tokens and artifacts is inefficient and ineffective for learning.

LookWhen is efficient due to how it selects, but selection is not new to video. Prior methods select to minimize *pixel redundancy* [8, 9]. We select to minimize redundancy by learning to *predict token uniqueness*; where a token is unique if no other token in the video has similar features. During pre-training, we measure uniqueness as the distance to a token's nearest neighbor in a *teacher's feature space*. Our "top1-distance" method eliminates the need for artifact-free teacher attention and improves on existing and alternative targets for selection.

LookWhen is efficient because its predictions rely only on standard and highly-optimized operations. Instead of selecting tokens at the input, other adaptive computation methods *merge* tokens at intermediate layers [10, 11]. These methods are more reactive than predictive; they compute all tokens, match, then merge to continue with fewer. They match by clustering and other operations, which can reduce FLOPs, but can fail to save time [6].

We make three main contributions:

- We extend the selector–extractor framework to video with key changes to the architecture and pre-training. We train to **select** unique tokens across space-time rather than the most attended tokens. We train to **extract** video-level tokens supervised by multiple teachers: a video teacher and an image teacher from which we make a target token through concatenating *time-normalized* frame tokens to learn *what changes* within each video for fine-grained classification.
- We show LookWhen improves efficiency over existing video models and our own upgraded baselines. Gains are highest for linear probing and are consistent across tasks: Kinetics-400 (K400) [12], Something-Something-v2 (SSv2) [13], Epic-Kitchens (EK100) [14], Diving48 [15], Jester [16], and Charades [17]. LookWhen can even beat its non-sparse teacher: InternVideo2 (IV2) [18].
- Through ablations on six datasets, we show that our novel methods for selector training (top1-distance) and extractor training (time-normalized frame features) drive LookWhen's performance.

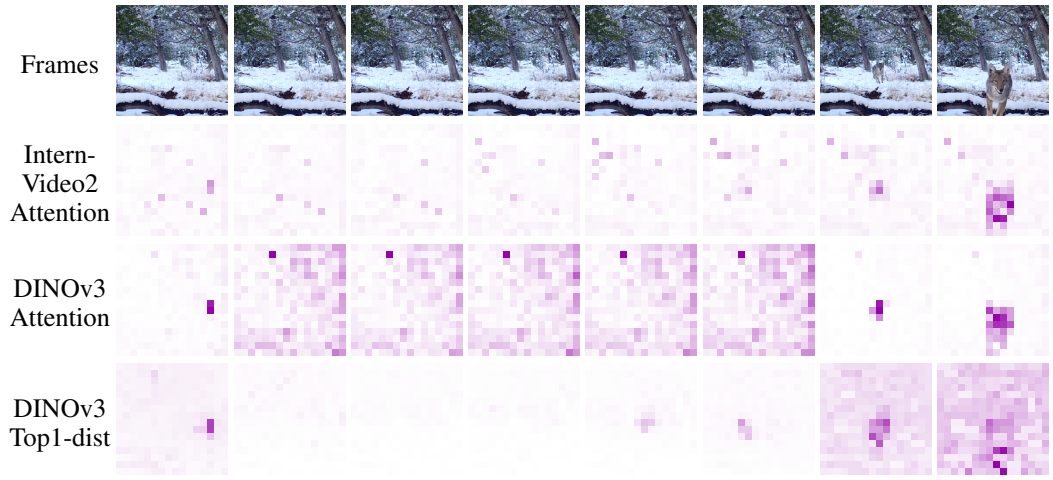


Figure 2: **When and where to compute for efficiency.** InternVideo2 space-time attention maps suffer from *artifacts*. DINOv3 has cleaner attention but is strictly *frame-wise*. "Top1-dist" (each patch's distance to its nearest neighbor in *feature space*) finds the unique patches across all frames; our selector predicts it. The wolf is partly visible in frame 1, runs away, then toward the camera.

2 LookWhen: Selecting across space & time and extracting across teachers

2.1 Preliminaries: Look*Where* and positionally-grounded representations

Look*Where* inspires our Look*When*. LookWhere [6] introduced the selector-extractor framework. The selector gets a downscaled image and outputs a selector map, for which the top-K positions are chosen. The extractor gets the original-resolution image patches at the selected positions to extract features of the full image. During pre-training, a teacher computes targets from *all* original-resolution image patches. The selector trains to predict the teacher's final attention maps. The extractor trains to predict the teacher's final representations. Thus they train in tandem to efficiently approximate the teacher's computation. LookWhere outperforms prior adaptive computation methods at visual recognition; the gains are especially large for large images with more redundancy. Videos have even more redundancy and are more expensive to process in full: this cries out for adaptive methods that choose where—and *when*—to process. However, to realize these potential efficiency gains, we need a teacher for which we can factorize representation (*what*) and location (*when* and *where*).

Positionally-grounded representations. Deep networks, including ViTs [19, 20, 7, 21, 22, 23], can output feature maps that are positionally misaligned from the input. That is, the feature vector at position (x, y, t) may encode information that does not pertain to the input patch at (x, y, t). This misalignment manifests visibly: top-3 PCA projections of such feature maps appear noisy and unstructured. "High-norm" tokens [7] are the most well-known example, where certain patch tokens store global information unrelated to their position; understanding and eliminating high-norm tokens is an active area of research [7, 24]. More broadly, optimization can redistribute features across local or non-local positions—e.g. shifting a patch's representation into a neighboring token—whenever doing so lowers the training loss. This has grave consequences for designing selector-extractor methods, since their success relies on isolating the input tokens responsible for the output tokens.

2.2 Overview and motivation of the architecture and targets

Architecture (Fig. 1). The selector must see the full video to judge redundancy, but it must be fast. So we make it *shallow* and give it a *downscaled* input, substantially reducing its cost. The extractor must be sufficiently expressive to compute rich representations, so we give it full depth. Its efficiency is from input sparsity: it receives only the top-K tokens that the selector deems most informative. These two models approximate dense computation by exploiting the inherent redundancy in video. We define three token types: a video token (1 per video), frame tokens (1 per frame), and patch tokens (1 per spatial position per frame). We supervise them with video-level, frame-level, and



Figure 3: **Example of learned selections.** We pre-train on K400+SSv2 data and our selector generalizes to a video of an author's nephew being thrown in a pool and swimming. More in §A.2.

patch-level features, respectively. After pre-training, the teachers are discarded; only the efficient selector-extractor is needed. The video token is the input to the linear head for downstream tasks.

Selector training targets. Attention maps are a natural choice for selection targets, following LookWhere [6], but the attention of current video models is unsuitable. Fig. 2 shows InternVideo2 [18] attention has artifacts. Other potential video teachers are no better, e.g. V-JEPA-2 [25] attention is noisy and VideoMAE [3] attention does not outperform random selection [26]. DINOv3 [27] has fewer attention artifacts but is an *image* model and thus cannot directly exploit time redundancy. Yet DINOv3 has spatially-grounded features: its output token at position (x, y) represents the input at (x, y) [27]. We achieve alignment in (x, y, t) by processing each frame independently and stacking the resulting feature maps in time. We can then rank tokens by how much unique information they carry: the extractor should get the least redundant tokens to best approximate dense representations. We propose "top1-distance", which ranks each token by its distance to its nearest neighbor in feature space, so we train to select the most isolated tokens. For example, Fig. 3 shows learned selection using this pre-training strategy.

Extractor training targets. We want our extractor to learn general representations so it can be used as a feature extractor or efficiently fine-tuned on downstream tasks. So we choose a SOTA video foundation model, InternVideo2, which is aligned with natural language embeddings. We also choose a SOTA image foundation model, DINOv3, which is fully self-supervised. Since we want our extractor to excel at fine-grained tasks, we normalize DINOv3 features so the targets represent *what changes* throughout the video. We normalize over space-time for patch targets and time for frames.

2.3 Selector-Extractor: Pairing dense low-res computation with sparse high-res computation

We shrink the **selector's** input and model depth for efficiency. We downscale the video by 2× along its time, height, and width dimensions $T_{\mathbf{E}} \times R_{\mathbf{E}} \times R_{\mathbf{E}} \rightarrow T_{\mathbf{S}} \times R_{\mathbf{S}} \times R_{\mathbf{S}}$, resulting in 8× fewer patches. We choose the 2× for simplicity. We use only 3 transformer layers. These two modifications reduce FLOPs by 50× at $T_{\mathbf{E}}$=16, $R_{\mathbf{E}}$=224. The selector patchifies the video into a $T_{\mathbf{S}} \times N_{\mathbf{S}} \times N_{\mathbf{S}}$ grid, and prepends $T_{\mathbf{S}}$ frame tokens and G registers. The selector outputs latents $z_{\mathbf{S}} \in \mathbb{R}^{(T_{\mathbf{S}}+G+T_{\mathbf{S}} \cdot {N_{\mathbf{S}}}^2) \times D}$ and uses an MLP to compute the selector map $\hat{M} \in \mathbb{R}^{T_{\mathbf{E}} \times N_{\mathbf{E}} \times N_{\mathbf{E}}}$ from $T_{\mathbf{S}} \cdot {N_{\mathbf{S}}}^2$ patch tokens.

We shrink the **extractor's** token count for efficiency. The extractor receives *only the top-K patches*—chosen by their selector-map scores—appended to the video and frame tokens. Rather than using learned-embedding frame and register tokens as inputs, the extractor gets the *selector*'s frame and register tokens, which transmits information between the two views. Since the selector has $T_{\mathbf{S}}$ frame tokens but the extractor gets $T_{\mathbf{E}}$ frames, we upscale the selector's frame tokens via linear interpolation in time. We also prepend a learned-embedding video token to the token sequence, which represents the full input. The extractor processes these tokens and outputs features $z_{\mathbf{E}} \in \mathbb{R}^{(1+T_{\mathbf{E}}+G+\mathrm{K}) \times D}$.

2.4 Teachers: Pre-training to select unique patches and extract dense features

We train the **selector** to pick unique patches to drop redundant information in the video for efficiency. To find the unique patches, we first compute features for all $T_{\mathbf{E}} \cdot {N_{\mathbf{E}}}^2$ patches in a video. We extract them using an *image* teacher, DINOv3 [27], which has spatially-grounded representations. We then apply our "top1-distance" algorithm, which computes each patch's cosine similarity to its nearest neighbor—in feature space—then orders all patches by increasing similarity so the most isolated

Model	Params M	Kinetics-400 FLOPs G×T×S	Kinetics-400 Top-1 % acc.	SSv2 FLOPs G×T×S	SSv2 Top-1 % acc.
Larger models for reference only, not for direct comparisons					
V-JEPA-2 [25]	355	935×8×3	85.1	935×2×3	73.7
InternVideo2 [18]	1020	2500×4×3	89.4	2500×2×3	69.7
ViT-B, Swin-B, or Mamba-M models					
UMT-B800e [28]	87	180×4×3	85.7	180×2×3	70.8
VideoMAE [3]	87	180×5×3	81.5	180×2×3	70.8
VideoMAEv2 [5]	87	180×5×3	81.5	180×2×3	71.2
VideoMamba-M800e [29]	74	202×4×3	83.4	202×2×3	71.0
VideoMambaPro [30]	72	392×4×3	84.0	183×4×3	69.4
VideoSwin + STTS ($T_0^{0.6}$) [31]	89	181×4×3	81.4	190×1×3	68.1
VideoMAE + LITE (K=0.3) [26]	87	46×5×3	78.4	46×2×3	68.3
VideoMAE + ToMe (r=64) [10, 8]	87	131×4×3	80.0	131×4×3	69.7
VideoMAE + RLT (τ=0.1) [8]	87	120×4×3	80.1	120×4×3	70.2
VideoMAE + vid-TLDR [11]	87	—	—	57×unk	69.6
LookWhen (90% sparse)	106	40×4×3	82.6	40×2×3	69.3
LookWhen (70% sparse)	106	108×4×3	84.6	108×2×3	72.0

Table 1: **LookWhen achieves a better inference accuracy-computation trade-off for fine-tuning on K400 and SSv2.** LookWhen has more parameters but sparser computation, so it is also *more memory-efficient* (please see Appendix §A.1). In "G×T×S", G is the GFLOPs per view, T is the number of temporal views, S is the number of spatial views; their product is the total GFLOPs per video. "unk" is unknown.

patch is ranked first. Formally, $U_{x,y,t}=1-\max_{(x',y',t')\neq(x,y,t)}\cos\left(z^{\text{DINOv3}}_{x,y,t},\, z^{\text{DINOv3}}_{x',y',t'}\right)$, where U is the uniqueness score and $z^{\text{DINOv3}}_{x,y,t}$ is DINOv3's feature vector at patch position (x,y,t). We then post-process these uniqueness scores so we can train with binary cross-entropy (BCE) loss; specifically, we assign the lowest-ranking patch a 0, the highest a 1, and evenly space scores in between. These steps result in a selector-map target $M\in\mathbb{R}^{T_\mathbf{E}\times N_\mathbf{E}\times N_\mathbf{E}}$, and we compute the loss $\mathcal{L}_{\text{map}}=\text{BCE}(\hat{M},M)$.

We train the **extractor** to represent the full video despite only processing some of it for efficiency. We compute video features from two frozen teachers: a video teacher, InternVideo2, and an image teacher, DINOv3. We choose InternVideo2's attention-pooled token $z^{\text{IV2}}_{\text{video}}\in\mathbb{R}^D$. Since DINOv3 is an image model, it does not have a video-level token. To make one, we first normalize DINOv3's $T_\mathbf{E}$ class tokens over time so each dimension has 0 mean and 1 standard deviation. We then concatenate these normalized class tokens, resulting in the other video-token target $z^{\text{DINOv3}}_{\text{video}}\in\mathbb{R}^{T_\mathbf{E}\cdot D}$. We map from the extractor's video token to these targets with two separate MLPs and compute the mean-squared error (MSE) loss $\mathcal{L}_{\text{video}}=\frac{1}{2}(\text{MSE}(\hat{z}^{\text{IV2}}_{\text{video}},z^{\text{IV2}}_{\text{video}})+\text{MSE}(\hat{z}^{\text{DINOv3}}_{\text{video}},z^{\text{DINOv3}}_{\text{video}}))$.

We also train the **extractor's** *patch* and *frame* tokens so we can approximate all teacher outputs. For frame-token targets, we use DINOv3's class tokens, normalized over time. For patch-token targets, we use DINOv3's patch tokens, normalized over space and time. Since the extractor processes K patches, we upsample with nearest neighbor to make the full feature map $\hat{z}^{\text{DINOv3}}_{\text{patch}}\in\mathbb{R}^{T_\mathbf{E}\times N_\mathbf{E}\times N_\mathbf{E}\times D}$, following LookWhere. Normalizing DINOv3's features emphasizes feature-change *within* each video. We also compute these losses $\mathcal{L}_{\text{frame}}=\text{MSE}(\hat{z}^{\text{DINOv3}}_{\text{frame}},z^{\text{DINOv3}}_{\text{frame}})$ and $\mathcal{L}_{\text{patch}}=\text{MSE}(\hat{z}^{\text{DINOv3}}_{\text{patch}},z^{\text{DINOv3}}_{\text{patch}})$.

Selection and extraction losses. We train the selector and extractor simultaneously to minimize $\mathcal{L}=\mathcal{L}_{\text{map}}+\mathcal{L}_{\text{video}}+\mathcal{L}_{\text{frame}}+\mathcal{L}_{\text{patch}}$. Gradients flow from the extractor to the selector only through the selector's frame and register tokens, which are passed between the models. Gradients do not flow from the extractor to the *selector map*, since top-K patch selection is non-differentiable. We run DINOv3 online, which requires one forward pass per frame to compute all targets for selection and extraction. We pre-compute InternVideo2's video tokens over all pre-training data to save time.

3 Experiments: Accuracy and efficiency in FLOPs and time

Pre-training. We pre-train LookWhen for 20 epochs on the combined Kinetics-400 [12] and SSv2 [13] datasets. We initialize our selector-extractor from DINOv3, and encode time with 1D-sincos embeddings. Since training is bottlenecked by both data loading and online teacher processing, we repeat batches 8× via data augmentation so our selector-extractor sees more data (repeat augmentation is commonly used when training video models [5, 32]). For each batch we uniformly sample the sparsity level $S\in[70\%,95\%]$ so LookWhen supports varied sparsity out of the box; $S\%$ sparsity means $\text{K}=(1-S)\cdot T_\mathbf{E}\cdot {N_\mathbf{E}}^2$. We choose a ViT-B size for our selector-extractor and teachers for feasibility. We use $T_\mathbf{E}$=16 timesteps and $R_\mathbf{E}$=224 image size because that is most common in the literature.

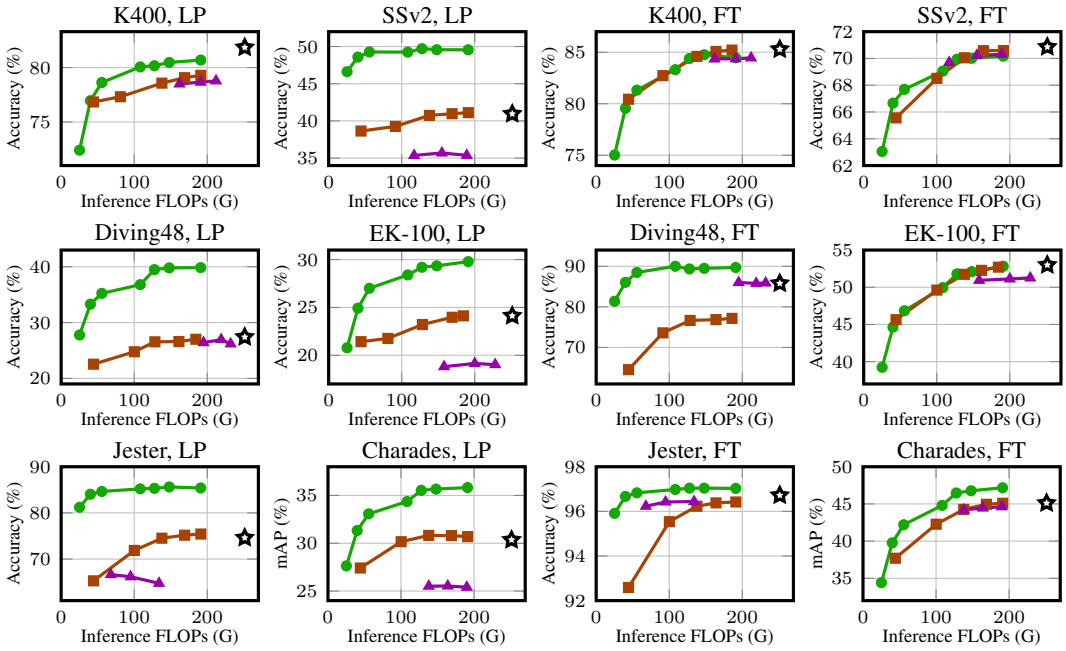


Figure 4: **Linear probing (LP) and fine-tuning (FT) accuracy vs. FLOPs** across six datasets. Our LookWhen (●) mostly outperforms the baselines in controlled settings. Gains are largest for LP, sometimes surpassing the *dense* InternVideo2 (⋆). We make these upgraded baselines by applying the sparsification methods vid-TLDR (■) [11] or RLT (▲) [8] to the SOTA ViT-B InternVideo2 [18].

3.1 Downstream Tasks: Accuracy versus computational cost

Setup: Comparing with existing models on K400 and SSv2. We compare LookWhen to existing ViT-B, Mamba-M, and Swin-B video models (since they are of similar size). We include five adaptive-computation methods: LITE [26], ToMe [10], RLT [8], vid-TLDR [11], and STTS [31]. We include some larger models for reference but not direct comparison. For both K400 and SSv2, we fine-tune our pre-trained LookWhen extractor for 30 epochs with our selector frozen. We sweep 4 learning rates for each sparsity level {70%, 90%} to show performance at two operating points.

Results: LookWhen is more compute-efficient than existing models (Tab. 1). LookWhen achieves 84.6% on K400 at 70% sparsity, which is only 1.1% lower than UMT-B800e [28] but LookWhen achieves it at 40% fewer FLOPs. The next best is VideoMambaPro [30], which scores 84.0%; LookWhen is 0.6% *more* accurate at 72% *fewer* FLOPs. On SSv2, LookWhen also achieves the best accuracy-compute trade-off: LookWhen scores 72.0% at 108 GFLOPs per view, while VideoMAEv2 [5] scores 71.2% at 180 GFLOPs per view.

Since the methods in Table 1 use different pre-training datasets, we also pre-train LookWhen on K400 only to assess the effect of our combined pre-training. After 30 epochs of fine-tuning on the full SSv2 training set, this variant achieves 70.6% at 70% sparsity. This exceeds the sparse baselines and is 0.2 percentage points below VideoMAE, which uses 1.7× as many FLOPs. We further examine pre-training data in Appendix Table 14 and use a common transfer recipe for the controlled comparisons below (Figure 4).

Setup: Controlled and upgraded baselines. InternVideo2-B (IV2 for short) warrants close comparison to LookWhen as our video teacher. However, there are no published results at this scale (results only exist for its much larger variants), so we run it ourselves. We also apply two adaptive computation methods, vid-TLDR [11] and RLT [8], to the IV2 backbone as efficient editions of our teacher. For IV2, IV2+RLT, and our LookWhen, we fine-tune for 10 epochs on K400 [12], SSv2 [13], Jester [16], and Epic-Kitchens-100 [14]; and for 50 epochs for the smaller Diving48 [15] and Charades [17] datasets. Since we pre-train on K400 and SSv2 without labels, evaluations on these two datasets are in-distribution, while the other four datasets test transfer beyond our pre-training data. We equally tune the learning rate for all methods to be fair. Since vid-TLDR sparsifies an already

fine-tuned model, we apply it to our fine-tuned IV2 models. We care about the accuracy-cost *trade-off*, so we measure performance at multiple operating points by varying sparsity. During fine-tuning, we vary RLT's threshold $\tau \in \{0.05, 0.1, 0.2\}$ and LookWhen's sparsity between 50% and 95%. The vid-TLDR paper tries many token-merging schedules for each dataset, so we do the same by trying 4 different schedules per sparsity level and choose the best per level per dataset.

Results: LookWhen is more efficient than upgraded baselines (Figure 4). LookWhen Pareto-dominates on 9 of 12 evaluations and roughly matches on the other 3. LookWhen shines in feature extraction, e.g. >10% more accurate on Diving48 LP. LookWhen sometimes even outperforms its dense video teacher, IV2. We attribute it to our DINOv3 video-token pre-training (see §4).

***Realized* efficiency: LookWhen's efficiency gains increase when measured in practice.** We measure throughput on an NVIDIA L40S GPU to check if theoretical gains (accuracy-FLOPs) translates to practical gains (accuracy-throughput). Our efficiency gains substantially increase; e.g. at equal accuracy, LookWhen is 6.7× faster than IV2 (Fig. 5), where accuracy is the mean across the 6 datasets and 2 settings (LP and FT). IV2+vid-TLDR is less efficient in practice because of Flash Attention [33] incompatibility for token-merging layers (the first 2-4 layers). IV2+RLT is less efficient in practice because it requires a batch size of 1 or masking (where all tokens are processed and the contributions of some tokens are ignored). Our LookWhen uses standard deep learning operations so is efficient in practice using standard libraries, e.g. PyTorch [34]. Because of its selector, LookWhen has more parameters than IV2, yet LookWhen is *more memory-efficient* because it has fewer activations. Please see accuracy-speed plots for all 12 evaluations and memory-use statistics in Appendix §A.1.

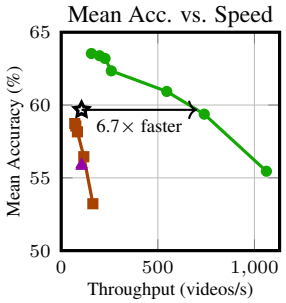


Figure 5: LookWhen (●) dominates baselines in mean accuracy (over 6 datasets and 2 settings) versus measured throughput. Markers: IV2 (⋆), IV2+vid-TLDR (■), and IV2+RLT (▲).

4 Ablations and Analysis: Looking Inside

We analyze the key design choices behind LookWhen. We first describe the ablation dataset and evaluation setup, then study the selection and extraction targets. Full experimental details, including sampling, augmentations, and hyperparameters, are provided in Appendix A.7.

Datasets. We re-use the six tasks to assess general performance. For feasibility, we use 20K training subsets for all except Diving48 and Charades, which have less than 20K, and use all validation points.

Evaluating. To assess both feature quality and fine-tuning ability, we report results for both linear probing (LP) and fine-tuning (FT). For LP, we train a linear classifier on frozen features. For FT, we fine-tune the extractor and head, initializing the head from the LP checkpoint. LP reflects performance in low-compute settings, while FT captures performance when more accuracy is desired. Unless otherwise noted, we evaluate at 90% sparsity for these main-paper and Appendix (§A.7.4) ablations.

4.1 *When* and *where* to select

Attention vs. token uniqueness (Tab. 2a). We first explore selection training paradigms: attention-based or representation-based. Directly comparing our method (top1-dist using DINOv3) to a naive extension of Look*Where* [6] to video (attention using IV2), our method wins easily: +3.9% LP and +7.2% FT. Training to select IV2's attention is no better than random selection, showing that IV2's noisy space-time attention cannot effectively guide selection. Plugging our top1-dist into IV2's feature map also fails—it is much worse than random selection. This aligns with our intuition that positionally-grounded representations or attention is needed for effective selection. We also tried to exploit time redundancy via "Δattn" (row #3), which measures how rapidly each patch's *DINOv3* attention changes across neighboring frames. This method works well-enough but underperforms our top1-dist overall by 0.4% LP and 1.4% FT. In particular, Δattn is least effective on Diving48, which follows divers while panning the camera—our top1-distance is robust to this challenging motion.

Measuring token uniqueness (Tab. 2b). We explore different variations of our top1-dist method, both in DINOv3's feature space (rows #1-4) and the pixel space (row #5). Computing the average distance to more neighbors still works well (rows #2,3), but 1 neighbor is best overall (row #1).

Table 2: **Ablating *when* and *where* to select.** We train with different selection targets to study its effect on downstream tasks. LP is linear probing the frozen extractor, FT is fine-tuning the extractor. All runs use InternVideo2 video-token target, frame and patch loss (no norm), and K400+SSv2 data.

(a) **Attention vs. token uniqueness.** Training to select unique tokens (top1-dist) beats highly-attended tokens, if we have suitable teacher features (e.g. DINOv3). The *change* in DINOv3's attention between successive frames (Δ) beats InternVideo2's space-time attention (which contains artifacts) and DINOv3's space-only attention.

		K400-20K		SSv2-20K		Diving48		EK100-20K		Jester-20K		Charades	
Teacher	Method	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT
DINOv3	top1-dist	72.9	73.6	31.5	45.2	**29.7**	**83.2**	**11.6**	**22.2**	**63.4**	**94.7**	**29.3**	**36.7**
DINOv3	attn	72.1	72.8	30.1	41.1	28.0	77.7	9.1	17.6	53.8	90.1	26.6	33.2
DINOv3	Δattn	**73.5**	**74.2**	**31.7**	**45.6**	27.6	76.0	11.0	21.3	**63.4**	94.4	28.6	35.7
InternVideo2	top1-dist	64.7	64.7	27.0	31.2	15.8	44.6	7.3	13.4	56.8	89.2	21.9	26.0
InternVideo2	attn	68.6	68.9	28.6	35.4	26.2	72.9	9.3	15.2	58.2	89.9	24.4	30.4
none	random	72.5	72.7	30.4	42.8	18.3	57.2	10.2	20.3	57.2	91.8	25.9	30.8

(b) **Computing token uniqueness.** We vary K in our "topK-distance" method, which is each patch's mean distance to its K-nearest neighbors in DINOv3's feature space. K=1 performs the best on average. We also try "K-center" on both features and pixels [9], which runs farthest-point sampling [35] for maximum diversity.

	K400-20K		SSv2-20K		Diving48		EK100-20K		Jester-20K		Charades	
Method	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT
top1-distance	**72.9**	**73.6**	31.5	**45.2**	29.7	83.2	**11.6**	**22.2**	63.4	94.7	**29.3**	**36.7**
top10-distance	72.7	73.2	31.6	44.7	30.5	**84.0**	11.3	21.6	63.3	**94.8**	28.9	36.4
top100-distance	71.4	72.3	31.4	43.5	**31.0**	83.5	11.0	20.5	61.8	94.4	27.8	35.4
feature K-center	**72.9**	**73.6**	**31.8**	44.9	27.1	79.5	10.7	20.8	63.1	**94.8**	28.2	35.4
pixel K-center	72.1	72.8	31.4	42.9	25.9	72.7	9.8	16.7	**66.0**	93.8	28.6	35.3

Diving48 is an exception: larger neighborhoods yield more contiguous selections that appear to capture more of the diver. At 90% sparsity, we find the largest connected region in each video's space-time selection, then average its size across the Diving48 training set. This average increases by 69% from 1 to 10 neighbors and a further 22% from 10 to 100. Capturing more of the diver may help distinguish fine-grained motions, offering a plausible explanation for the gains from more neighbors on Diving48. We also try further increasing the diversity of token-selection targets via farthest-point sampling [35] in DINOv3's feature space (row #4) and the pixel space (row #5); the latter is used in prior work [9].

4.2 *What* to extract

Video-token targets (Tab. 3a). We first explore video-token supervision with frame-level and patch-level losses disabled. Video-token distillation with IV2 (row #3) significantly outperforms distillation with DINOv3 (row #4) by +12.8% in LP and +2.5% in FT on average. Yet DINOv3 alone beats IV2 alone in some cases (+8.8% on Jester LP). Including *both* targets (row #1) boosts gains over IV2 alone (+3% in LP and 0.7% in FT on average), confirming that the two forms of supervision are complementary. *Removing* within-video target-feature normalization of DINOv3 (row #2) drops performance (-2.1% LP and -0.4% FT on average) and sometimes by large margins (-6.4% on SSv2 LP and -5.1% on Jester LP), showing that learning *what changes* is an effective strategy.

Frame- and patch-token targets (Tab. 3b). We explore frame-level and patch-level supervision with both video-token losses enabled. Supervising both frame and patch tokens (row #1) outperforms supervising neither (row #4) by +3.1% LP and +1.3% FT on average; these gains are sometimes massive, e.g. +7.7% on Diving48 LP and +10% on Jester LP. Supervising either frame (row #2) or patch tokens (row #3) narrows the gap. We note that supervising all tokens is not uniformly optimal, e.g. K400 is best without these denser supervision signals. See Appendix §A.7.4 for more ablations.

Sparse versus dense extraction (Tab. 4). Even at 90% sparsity, LookWhen outperforms the 224px dense extractor on SSv2 in both LP and FT, and on Diving48 and Jester in LP, despite the substantially higher computational cost of dense processing, showing that selection-extraction can boost accuracy.

4.3 Long-video recognition

We further evaluate LookWhen on longer videos, where dense processing is especially costly. We train a long-video model for 20 epochs with the same objectives, varying the frame count, spatial

Table 3: **Ablating *what* to extract.** We train with different extractor targets to study its effect on downstream tasks. LP is linear probing the frozen extractor, FT is fine-tuning the extractor. All runs use top1-distance using DINOv3's feature space to train the selector map and K400+SSv2 datasets.

(a) **Video-token target.** Predicting *both* InternVideo2's video token and a video token we make from DINOv3's frame-wise class tokens improves representations. We time-normalize each dimension before concatenating DINOv3 tokens to learn *what* changes. Frame and patch losses are disabled to isolate video-token supervision.

	DINOv3				K400-20K		SSv2-20K		Diving48		EK100-20K		Jester-20K		Charades	
IntVid2	Vid	Frame	Patch	Norm	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT
✔	✔	✘	✘	✔	**73.3**	**74.0**	**37.9**	48.0	29.9	81.9	**12.8**	**23.3**	69.0	94.6	**30.3**	**37.4**
✔	✔	✘	✘	✘	72.1	73.3	31.5	45.1	**32.1**	84.2	11.3	22.4	63.9	94.9	29.5	36.9
✔	✘	✘	✘	✘	**73.3**	73.9	32.1	45.6	27.9	83.1	11.0	22.0	61.6	94.4	29.0	36.3
✘	✔	✘	✘	✔	17.9	61.1	26.7	**48.8**	26.7	**85.7**	2.6	18.7	**70.4**	**95.2**	14.0	30.5

(b) **Frame and patch-token targets.** We turn off frame and patch distillation losses to study their effects within the full IV2+DINOv3 configuration. Using both losses helps on average; removing both hurts Jester and Diving48 the most. All rows include normalization over time for constructing DINOv3 Vid representation.

	DINOv3			K400-20K		SSv2-20K		Diving48		EK100-20K		Jester-20K		Charades	
IntVid2	Vid	Frame	Patch	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT
✔	✔	✔	✔	72.5	73.8	38.1	**50.1**	**37.6**	**84.8**	13.5	23.8	**79.0**	95.2	**31.3**	**39.3**
✔	✔	✔	✘	72.5	73.8	38.6	**50.1**	35.5	84.1	**13.8**	**24.2**	77.0	95.2	31.0	39.1
✔	✔	✘	✔	72.7	73.8	**39.1**	49.6	31.9	84.6	**13.8**	23.8	77.1	**95.3**	30.7	38.7
✔	✔	✘	✘	**73.3**	**74.0**	37.9	48.0	29.9	81.9	12.8	23.3	69.0	94.6	30.3	37.4

Table 4: **Ablating the selector.** We compare dense and sparse selection to isolate the effect of our selector. Dense extractors use the same pre-training data and dual-teacher distillation as LookWhen, but retain all patches in every frame (0% sparsity) at the stated spatial resolution. All inputs contain 16 frames. FLOPs ratios are normalized to LookWhen at 90% sparsity.

			K400-20K		SSv2-20K		Diving48		EK100-20K		Jester-20K		Charades	
Model	Sparsity	FLOPs ratio	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT
LookWhen (224px)	90%	1.0×	72.5	73.8	**38.1**	**50.1**	**37.6**	84.8	13.5	23.8	**79.0**	95.2	31.3	39.3
Dense extractor (224px)	0%	11.3×	**73.7**	**75.8**	34.5	47.7	37.2	**87.4**	**13.9**	**27.8**	78.1	**95.5**	**34.5**	**44.7**
Dense extractor (80px)	0%	0.9×	68.4	70.8	31.9	45.2	29.4	82.5	8.9	18.7	77.6	95.2	26.3	34.4

resolution, and retained patch budget. Table 5 reports linear-probe results on frozen representations for Charades, ActivityNet, and UCF-Crime using $128 \times 224 \times 224$ inputs at 90% and 95% sparsity. We further explore increasing the number of frames while fixing the retained patch budget in Appendix Table 13, finding that more frames do not consistently improve recognition under a fixed token budget.

5 Related work: Looking around

Token reduction methods drop or merge tokens across layers to reduce computation. For example, ToMe [10] merges tokens via soft bipartite matching. Other methods have not been extended from images to video, so we do not include them as baselines [36, 37, 38]. STA [39] progressively prunes video tokens by accumulating inter-frame similarity over time and reweighting via activation-based semantic scores. vid-TLDR [11] defines a saliency score based on attention sharpness and merges accordingly, enabling earlier reduction; it beats ToMe, so we use it as a baseline in our controlled experiments (§3). Merging methods typically reduce FLOPs more than runtime, since their algorithms may not be GPU-friendly or Flash-Attention-compatible; hence LookWhen's gains over IV2+vid-TLDR grow in reality. These methods still process all tokens in the first layer and only reduce them gradually, so their peak memory use can be very high (see memory-use statistics in §A.1). In contrast, LookWhen never processes all tokens, is GPU-friendly, and is FLOP, runtime, *and* memory efficient.

Token selection methods need to learn to efficiently select informative parts of the input. RLT [8] uses pixel similarity to remove static patches. K-centered [9] samples patches to maximize pixel-level diversity. EVEREST [40] selects high-motion tokens via adjacent-frame embedding differences. In

Table 5: **Long-video recognition.** We compare sparse and dense frozen video representations on three long-video recognition benchmarks. We train probes on frozen features; DINOv3 attention pooling is learned jointly with its classifier. FLOPs estimate transformer forward passes and are normalized to LookWhen at 90% sparsity.

Model	Sparsity	Input ($T \times H \times W$)	FLOPs	Charades mAP	ActivityNet mAP	UCF-Crime Top-1 Acc.
LookWhen-B	90%	128×224×224	1.0×	35.0	87.0	49.7
LookWhen-B	95%	128×224×224	0.6×	33.2	85.4	48.5
DINOv3-B (mean-pooled)	0%	128×224×224	4.9×	22.8	74.3	39.0
DINOv3-B (attn-pooled)	0%	128×224×224	4.9×	31.3	84.2	43.9
InternVideo2-B	0%	8×224×224	0.5×	30.4	88.1	45.7

contrast, LookWhen selects tokens based on *feature-space* uniqueness, enabling more aggressive sparsity. AutoGaze [41], a concurrent work, uses a small encoder with an autoregressive transformer decoder, trained by next-token prediction and reinforcement learning (RL) to predict the smallest set of multi-scale patches that reconstructs the video within a user-specified error threshold. Our selector is instead trained in a single distillation stage, without RL, to predict which tokens are most isolated in a teacher's feature space, and predicts its full map in one forward pass rather than one patch at a time. STTS [31] trains a scorer network end-to-end to rank token importance using a differentiable top-K operator. LITE [26] trains a selector using gradients of class scores with respect to feature activations. Although LITE shares the spirit of LookWhen, the methods differ in important ways: LookWhen trains a joint selector-extractor rather than only a selector; its selector exploits general video redundancy rather than task-specific redundancy; and its extractor predicts general representations from two teachers, making it directly usable as an effective video feature extractor for linear probing as well as an effective model for fine-tuning. LookThere [42] generalizes LookWhere's [6] selector-extractor framework by learning to select with RL.

Architectural changes. Other work modifies token interactions, e.g. attention [43, 44, 45] or the overall architecture [46, 47, 29, 48]. They reduce the cost of token processing, but still operate on the full set of tokens: they process all tokens, less. LookWhen processes fewer tokens, more.

Multimodal models. Recent work makes vision-language models more efficient [49, 50, 51, 52, 53, 54, 55, 56, 57]. They reduce tokens *after* vision encoder processing, sparing language model computation, which dominates the cost. They are not designed for unimodal video recognition, while LookWhen spares vision computation.

6 Closing: Looking to the end

Limitations and future work. Due to computational constraints, we are restricted to training ViT-Base models and could not include larger ViTs. One potential limitation of LookWhen's design is that it requires a teacher with positionally-grounded patch-level attention or representation to effectively supervise the selector. However, concurrent foundation model development now includes ongoing efforts to achieve this property for images [58, 59, 60] and video [61], due to the utility and transferability of positionally-grounded outputs. As a result more compatible teachers may be available and soon. Future efforts can further explore long video (Look*Then*), multi-view (Look*How*), and hyperspectral (Look*Which*) processing, all of which introduce significant computational costs and their own forms of redundancy and opportunities for selection.

Conclusion. We extend the selector-extractor framework from images to video by (1) adapting the network architecture, (2) pre-training to select tokens that have unique features, and (3) pre-training to extract features from a video teacher *and* from an image teacher to learn both global video-level and fine-grained patch-level features. Through ablations we show that these strategies explain why our LookWhen achieves better accuracy-compute trade-offs than existing models and other baselines on six video benchmarks. We hope our pre-trained LookWhen models make video recognition computationally feasible for you.

Acknowledgements

AF was funded by an NSERC PGS-D. ES is supported by a Canada CIFAR AI Chair and the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant (RGPIN-2025-06878). Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *International Conference on Computer Vision (ICCV)*, 2021.

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

[4] Rohit Girdhar and Kristen Grauman. Anticipative video transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 13505–13515, 2021.

[5] Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang, and Yu Qiao. Videomae v2: Scaling video masked autoencoders with dual masking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14549–14560, 2023.

[6] Anthony Fuller, Yousef Yassin, Junfeng Wen, Tarek Ibrahim, Daniel Kyrollos, James R Green, and Evan Shelhamer. Lookwhere? efficient visual recognition by learning where to look and what to see from self-supervision. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

[7] Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. In *The Twelfth International Conference on Learning Representations*, 2024.

[8] Rohan Choudhury, Guanglei Zhu, Sihan Liu, Koichiro Niinuma, Kris M Kitani, and László A Jeni. Don't look twice: Faster video transformers with run-length tokenization. *Advances in Neural Information Processing Systems*, 37:28127–28149, 2024.

[9] Seong Hyeon Park, Jihoon Tack, Byeongho Heo, Jung-Woo Ha, and Jinwoo Shin. K-centered patch sampling for efficient video recognition. In *European Conference on Computer Vision*, pages 160–176. Springer, 2022.

[10] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. *arXiv preprint arXiv:2210.09461*, 2022.

[11] Joonmyung Choi, Sanghyeok Lee, Jaewon Chu, Minhyuk Choi, and Hyunwoo J Kim. vid-tldr: Training free token merging for light-weight video transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18771–18781, 2024.

[12] 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. The kinetics human action video dataset, 2017.

[13] 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 *Proceedings of the IEEE international conference on computer vision*, pages 5842–5850, 2017.

[14] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Scaling egocentric vision: The epic-kitchens dataset. In *European Conference on Computer Vision (ECCV)*, 2018.

[15] Yingwei Li, Yi Li, and Nuno Vasconcelos. Resound: Towards action recognition without representation bias. In *Proceedings of the European conference on computer vision (ECCV)*, pages 513–528, 2018.

[16] Joanna Materzynska, Guillaume Berger, Ingo Bax, and Roland Memisevic. The jester dataset: A large-scale video dataset of human gestures. In *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pages 0–0, 2019.

[17] Gunnar A. Sigurdsson, Gül Varol, Xiaolong Wang, Ivan Laptev, Ali Farhadi, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. *ArXiv e-prints*, 2016.

[18] Yi Wang, Kunchang Li, Xinhao Li, Jiashuo Yu, Yinan He, Guo Chen, Baoqi Pei, Rongkun Zheng, Zun Wang, Yansong Shi, et al. Internvideo2: Scaling foundation models for multimodal video understanding. In *European conference on computer vision*, pages 396–416. Springer, 2024.

[19] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[20] Spiros Baxevanakis, Platon Karageorgis, Ioannis Dravilas, and Konrad Szewczyk. Do all vision transformers need registers? a cross-architectural reassessment. *arXiv preprint arXiv:2603.25803*, 2026.

[21] Jiawei Yang, Katie Z Luo, Jiefeng Li, Congyue Deng, Leonidas J. Guibas, Dilip Krishnan, Kilian Q Weinberger, Yonglong Tian, and Yue Wang. Dvt: Denoising vision transformers. *arXiv preprint arXiv:2401.02957*, 2024.

[22] Cheng Shi, Yizhou Yu, and Sibei Yang. Vision transformers need more than registers. *arXiv preprint arXiv:2602.22394*, 2026.

[23] Zipeng Yan, Yinjie Chen, Chong Zhou, Bo Dai, and Andrew Luo. Vision transformers with self-distilled registers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026.

[24] Anthony Fuller, Yousef Yassin, Daniel Kyrollos, Evan Shelhamer, and James R Green. Thicker and quicker: The jumbo token for fast plain vision transformers. In *The Fourteenth International Conference on Learning Representations*, 2026.

[25] 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, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-jepa 2: Self-supervised video models enable understanding, prediction and planning, 2025.

[26] Xinyue Hao, Gen Li, Shreyank N Gowda, Robert B Fisher, Jonathan Huang, Anurag Arnab, and Laura Sevilla-Lara. Principles of visual tokens for efficient video understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 21254–21264, 2025.

[27] Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, Francisco Massa, Daniel Haziza, Luca Wehrstedt, Jianyuan Wang, Timothée Darcet, Théo Moutakanni, Leonel Sentana, Claire Roberts, Andrea Vedaldi, Jamie Tolan, John Brandt, Camille Couprie, Julien Mairal, Hervé Jégou, Patrick Labatut, and Piotr Bojanowski. Dinov3, 2025.

[28] Kunchang Li, Yali Wang, Yizhuo Li, Yi Wang, Yinan He, Limin Wang, and Yu Qiao. Unmasked teacher: Towards training-efficient video foundation models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 19948–19960, 2023.

[29] Kunchang Li, Xinhao Li, Yi Wang, Yinan He, Yali Wang, Limin Wang, and Yu Qiao. Videomamba: State space model for efficient video understanding. In *European conference on computer vision*, pages 237–255. Springer, 2024.

[30] Hui Lu, Albert A Salah, and Ronald Poppe. Snakes and ladders: Two steps up for videomamba. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 24234–24244, 2025.

[31] Junke Wang, Xitong Yang, Hengduo Li, Li Liu, Zuxuan Wu, and Yu-Gang Jiang. Efficient video transformers with spatial-temporal token selection. In *European Conference on Computer Vision*, pages 69–86. Springer, 2022.

[32] Christoph Feichtenhofer, Haoqi Fan, Yanghao Li, and Kaiming He. Masked autoencoders as spatiotemporal learners. *arXiv:2205.09113*, 2022.

[33] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[35] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30.

[36] Joakim Bruslund Haurum, Sergio Escalera, Graham W Taylor, and Thomas B Moeslund. Agglomerative token clustering. In *European Conference on Computer Vision*, pages 200–218. Springer, 2024.

[37] Dong Hoon Lee and Seunghoon Hong. Learning to merge tokens via decoupled embedding for efficient vision transformers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[38] Hoai-Chau Tran, Duy M Nguyen, TrungTin Nguyen, Ngan Le, Pengtao Xie, Daniel Sonntag, James Zou, Binh T Nguyen, and Mathias Niepert. Accelerating transformers with spectrum-preserving token merging. *Advances in Neural Information Processing Systems*, 37:30772–30810, 2024.

[39] Shuangrui Ding, Peisen Zhao, Xiaopeng Zhang, Rui Qian, Hongkai Xiong, and Qi Tian. Prune spatio-temporal tokens by semantic-aware temporal accumulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16945–16956, 2023.

[40] Sunil Hwang, Jaehong Yoon, Youngwan Lee, and Sung Ju Hwang. Everest: Efficient masked video autoencoder by removing redundant spatiotemporal tokens. In *International Conference on Machine Learning*, 2024.

[41] Baifeng Shi, Stephanie Fu, Long Lian, Hanrong Ye, David Eigen, Aaron Reite, Boyi Li, Jan Kautz, Song Han, David M Chan, et al. Attend before attention: Efficient and scalable video understanding via autoregressive gazing. *arXiv preprint arXiv:2603.12254*, 2026.

[42] Sreehari Rammohan, Yousef Yassin, Anthony Fuller, Junfeng Wen, Carl Vondrick, and Evan Shelhamer. LookThere! Sparse Vision by Reinforced Selection. In *Advances in Neural Information Processing Systems*, 2026.

[43] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? *arXiv preprint arXiv:2102.05095*, 2021.

[44] Adrian Bulat, Juan-Manuel Perez-Rua, Swathikiran Sudhakaran, Brais Martinez, and Georgios Tzimiropoulos. Space-time mixing attention for video transformer. In *Advances in Neural Information Processing Systems*.

[45] Syed Talal Wasim, Muhammad Uzair Khattak, Muzammal Naseer, Salman Khan, Mubarak Shah, and Fahad Shahbaz Khan. Video-focalnets: Spatio-temporal focal modulation for video action recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13778–13789, 2023.

[46] Christoph Feichtenhofer. X3d: Expanding architectures for efficient video recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 203–213, 2020.

[47] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6202–6211, 2019.

[48] Chao-Yuan Wu, Yanghao Li, Karttikeya Mangalam, Haoqi Fan, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. Memvit: Memory-augmented multiscale vision transformer for efficient long-term video recognition. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pages 13587–13597, 2022.

[49] Zhijian Liu, Ligeng Zhu, Baifeng Shi, Zhuoyang Zhang, Yuming Lou, Shang Yang, Haocheng Xi, Shiyi Cao, Yuxian Gu, Dacheng Li, et al. Nvila: Efficient frontier visual language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4122–4134, 2025.

[50] Qingqing Cao, Bhargavi Paranjape, and Hannaneh Hajishirzi. Pumer: Pruning and merging tokens for efficient vision language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12890–12903, 2023.

[51] Jindong Jiang, Xiuyu Li, Zhijian Liu, Muyang Li, Guo Chen, Zhiqi Li, De-An Huang, Guilin Liu, Zhiding Yu, Kurt Keutzer, et al. Storm: Token-efficient long video understanding for multimodal llms. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5830–5841, 2025.

[52] Xiaoqian Shen, Yunyang Xiong, Changsheng Zhao, Lemeng Wu, Jun Chen, Chenchen Zhu, Zechun Liu, Fanyi Xiao, Balakrishnan Varadarajan, Florian Bordes, et al. Longvu: Spatiotemporal adaptive compression for long video-language understanding. *arXiv preprint arXiv:2410.17434*, 2024.

[53] Peng Jin, Ryuichi Takanobu, Wancai Zhang, Xiaochun Cao, and Li Yuan. Chat-univi: Unified visual representation empowers large language models with image and video understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13700–13710, 2024.

[54] Shuhuai Ren, Sishuo Chen, Shicheng Li, Xu Sun, and Lu Hou. Testa: Temporal-spatial token aggregation for long-form video-language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 932–947, 2023.

[55] Mingze Xu, Mingfei Gao, Shiyu Li, Jiasen Lu, Zhe Gan, Zhengfeng Lai, Meng Cao, Kai Kang, Yinfei Yang, and Afshin Dehghan. Slowfast-llava-1.5: A family of token-efficient video large language models for long-form video understanding. *arXiv preprint arXiv:2503.18943*, 2025.

[56] Shaolei Zhang, Qingkai Fang, Zhe Yang, and Yang Feng. Llava-mini: Efficient image and video large multimodal models with one vision token. *arXiv preprint arXiv:2501.03895*, 2025.

[57] Keda Tao, Can Qin, Haoxuan You, Yang Sui, and Huan Wang. Dycoke: Dynamic compression of tokens for fast video large language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 18992–19001, 2025.

[58] Chenchen Zhu, Saksham Suri, Cijo Jose, Maxime Oquab, Marc Szafraniec, Wei Wen, Yunyang Xiong, Patrick Labatut, Piotr Bojanowski, Raghuraman Krishnamoorthi, et al. Efficient universal perception encoder. *arXiv preprint arXiv:2603.22387*, 2026.

[59] Savya Khosla, Sethuraman TV, Aryan Chadha, Alex Schwing, and Derek Hoiem. T-ren: Learning text-aligned region tokens improves dense vision-language alignment and scalability. *arXiv preprint arXiv:2604.18573*, 2026.

[60] Bingyi Cao, Koert Chen, Kevis-Kokitsi Maninis, Kaifeng Chen, Arjun Karpur, Ye Xia, Sahil Dua, Tanmaya Dabral, Guangxing Han, Bohyung Han, et al. Tipsv2: Advancing vision-language pretraining with enhanced patch-text alignment. *arXiv preprint arXiv:2604.12012*, 2026.

[61] Lorenzo Mur-Labadia, Matthew Muckley, Amir Bar, Mido Assran, Koustuv Sinha, Mike Rabbat, Yann LeCun, Nicolas Ballas, and Adrien Bardes. V-jepa 2.1: Unlocking dense features in video self-supervised learning. *arXiv preprint arXiv:2603.14482*, 2026.

[62] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alex Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv preprint arXiv:1704.00675*, 2017.

[63] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[64] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[65] Christoph Feichtenhofer, Haoqi Fan, Bo Xiong, Ross Girshick, and Kaiming He. A large-scale study on unsupervised spatiotemporal representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3299–3309, 2021.

[66] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks for action recognition in videos. *IEEE transactions on pattern analysis and machine intelligence*, 41(11):2740–2755, 2018.

A Technical Appendices and Supplementary Material

A.1 Efficiency in Practice: Throughput and Memory Measurements

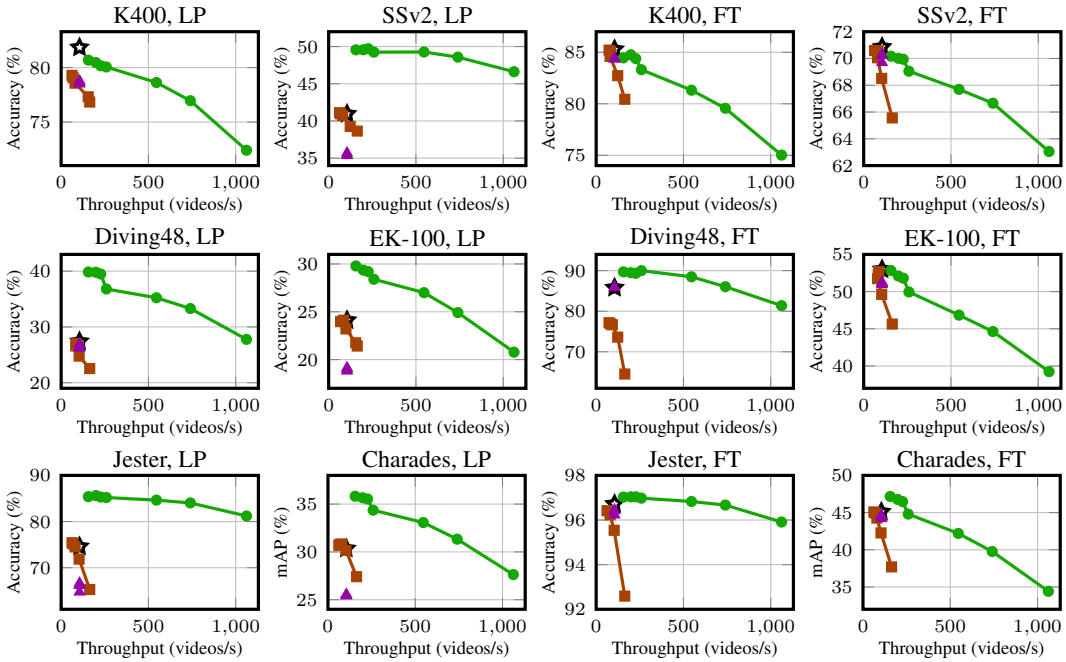


Figure 6: **Throughput (videos/s) at inference time.** All measurements are taken on an NVIDIA L40S GPU with batch size 32 and `bfloat16` automatic mixed precision. RLT models with different sparsity have the same throughput because RLT requires token masking through attention masking (not token dropping!) for batch sizes greater than 1. Markers: LookWhen (●), IV2 (⋆), IV2+vid-TLDR (■), and IV2+RLT (▲).

Table 6: **Peak memory use (GB) at inference time.** For RLT, all sparsity levels reach equal peak memory because RLT requires masking for batch processing; masking still computes all tokens, it just ignores their contribution. For vid-TLDR, all sparsity levels are equal because the first layer processes *all tokens* regardless of the token-merging schedule; it consumes much more memory because the token-merging layers are incompatible with Flash Attention [33]. All measurements are taken on an NVIDIA L40S GPU with batch size 32 and `bfloat16` automatic mixed precision.

LookWhen (sparsity)									
95%	90%	85%	70%	65%	60%	50%	IV2 [18]	+RLT [8]	+vid-TLDR [11]
1.51	1.51	1.51	1.68	1.80	1.91	2.15	2.76	2.76	16.59

A.2 Learned Selection Examples

Figure 7: Example from Kinetics-400.

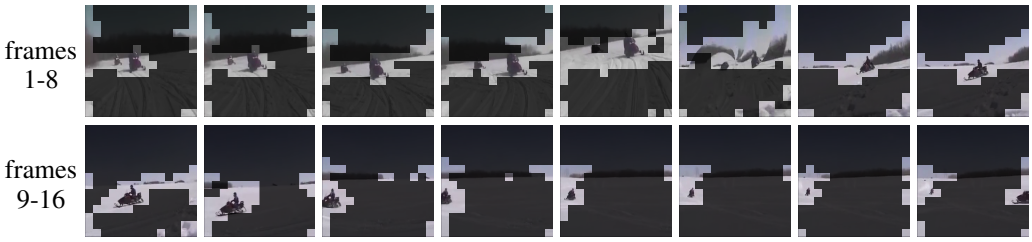


Figure 8: Example from Something-Something-v2.

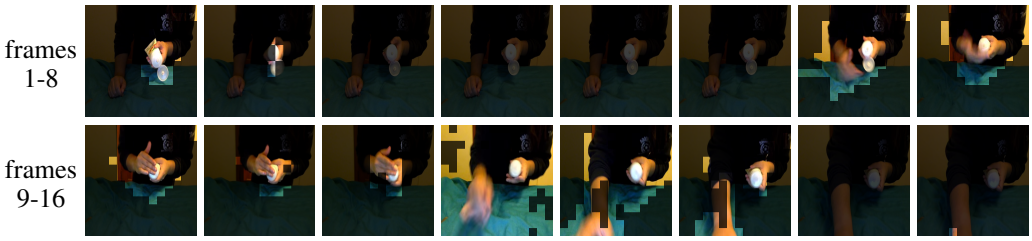


Figure 9: Example from Epic-Kitchens-100.

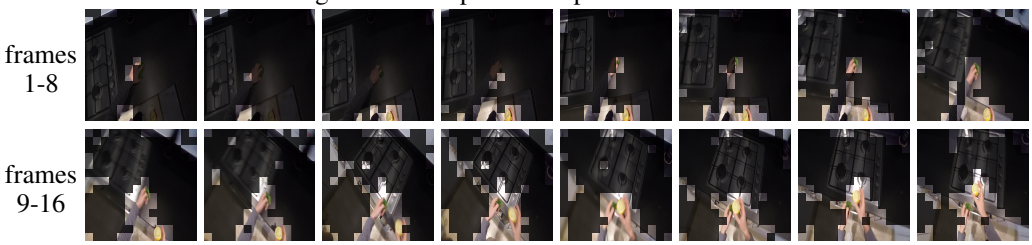


Figure 10: Example from Diving48.

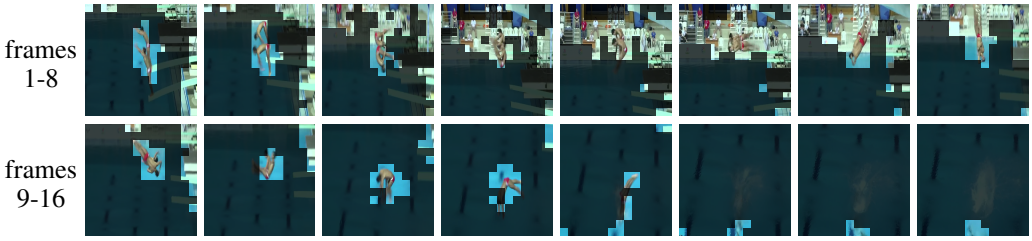


Figure 11: Example from Jester.

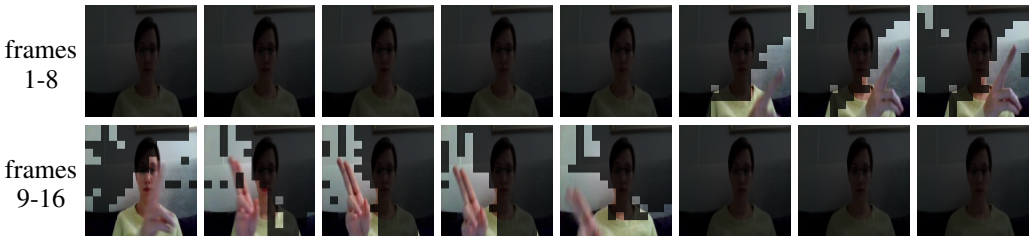
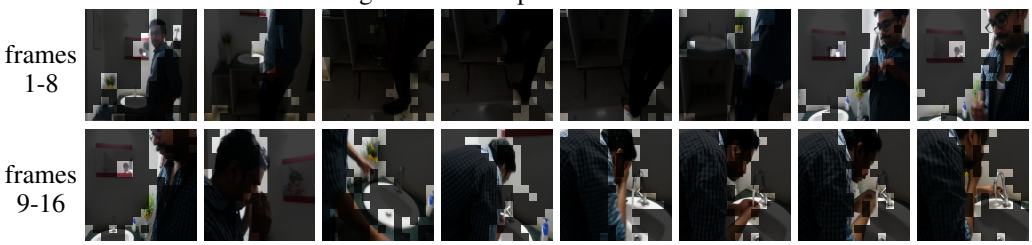


Figure 12: Example from Charades.



A.3 Video object segmentation

On DAVIS-2017 [62], LookWhen with learned selection outperforms random selection by 7.57 percentage points in $\mathcal{J}\&\mathcal{F}$, the mean of Jaccard region similarity and boundary F-score (Table 7). Both selection policies use the same extractor and retain 30% of the patches (70% sparsity). This comparison assesses the contribution of our selector and shows that learned selection improves transfer to a dense prediction task.

Table 7: **Video object segmentation.** We evaluate how well our selector transfers to a dense prediction task. Scores are percentages; random selection reports the mean $\pm$ sample standard deviation across three seeds. Propagation hyperparameters are tuned on the DAVIS training split separately for learned and random selection.

DAVIS-2017	$\mathcal{J}$	$\mathcal{F}$	$\mathcal{J}\&\mathcal{F}$
LookWhen ($k = 0.3$)	**49.4**	**47.9**	**48.6**
Random ($k = 0.3$)	44.7 ± 0.5	37.4 ± 0.4	41.1 ± 0.4

A.4 Fidelity to the selection target

We train a linear probe on features extracted when selection uses our learned selector and compare against features extracted when selection uses DINOv3's top1-distance map (our training target). At 70% sparsity, using the target map improves accuracy by +2.0% on Diving48 and by +0.8% on Epic-Kitchens-100. We also compare our learned selection maps directly to DINOv3's top1-distance map. Measured on K400, these two maps have a mean Spearman correlation coefficient of 0.61, and their selections overlap by 40% at 90% sparsity and by 61% at 70% sparsity. We find similar correlations and overlaps for the other datasets. The modest accuracy gap despite moderate selection overlap suggests that downstream accuracy is insensitive to exact agreement with the target.

A.5 Scaling across ViT sizes

To study scalability across ViT sizes, we also train a ViT-Small (ViT-S) version of LookWhen. It achieves 62% on K400-20K, 44% on SSv2-20K, 75% on Diving48, 17% on EK-100-20K, 93% on Jester-20K, and 29% on Charades. Although these results are lower than those of our ViT-Base model, varying model size provides another axis for trading accuracy for computational cost.

A.6 V-JEPA 2.1 supervision

We also evaluate V-JEPA 2.1-B [61] as a teacher for selection and extraction. Under this recipe, top1-distance outperforms attention-based selection on all six datasets for both LP and FT, although absolute performance is lower than our main dual-teacher configuration (Table 8).

Table 8: **V-JEPA 2.1 supervision.** We average V-JEPA 2.1-B patch features over space to form frame targets, and over space and time to form the video target. We compare attention-based and top1-distance selection at 90% sparsity and include our full dual-teacher configuration (Table 3b, first row) for reference.

		K400-20K		SSv2-20K		Diving48		EK100-20K		Jester-20K		Charades	
Teacher	Selection	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT
V-JEPA 2.1-B	Attention	42.7	50.4	26.4	41.3	14.3	38.0	4.2	11.1	59.4	90.5	15.0	20.2
V-JEPA 2.1-B	top1-distance	43.8	53.2	28.3	43.2	17.9	63.7	5.1	15.0	70.2	94.3	15.8	23.1
Dual-teacher	top1-distance	**72.5**	**73.8**	**38.1**	**50.1**	**37.6**	**84.8**	**13.5**	**23.8**	**79.0**	**95.2**	**31.3**	**39.3**

A.7 Implementation Details

A.7.1 Pre-training Settings

We follow the data augmentation and input size used in fine-tuning (Tab. 10). We use $1\times16\times16$ ($T\times H\times W$) patch tokens for simplicity and to match DINOv3's image tokenization. We pre-train for 20 epochs but each video is actually seen 20×8 times because of batch repetition, which reduces the fraction of time spent data loading and teacher processing. Specifically, we load a batch, compute DINOv3 representations, then horizontally flip and stack both videos and DINOv3 representations

(to align them); this is the first repeat. We repeat another $4\times$ by mixing up [63] videos and their corresponding DINOv3 representations. We thus compute the teacher once for every 8 times the selector-extractor is computed (which is more efficient because it is sparse). For ablations, we pre-train for 10 epochs. We did not tune the pre-training learning rate, we used a max value of 0.0001 and a batch size of 16 (repeated $8\times$ per above).

A.7.2 Variable-size Pre-training for Long Videos

For the long-video experiments in Tables 5 and 13, we follow the main model's pre-training setup in Appendix A.7.1, with the following changes. We vary the frame count $T \in \{16, 32, 64, 96, 128\}$, the square frame size $F \in \{128, 160, 192, 224, 256\}$ pixels, and the retained patch budget $K \in [128, 768]$, allowing repeated frames for short videos. We reduce the batch size to 4 videos before augmentation and the learning rate to 4×10^{-5}.

A.7.3 Fine-tuning Settings

For all datasets, we first train a linear probe for 100 epochs using the settings in Table 9, and initialize the fine-tuning classification head from this probe. For ablations, we subsample up to 20K training examples per dataset, uniformly across classes, and evaluate on the full validation set. For all other experiments, we use the full training data. For the efficiency results in Fig. 4, we follow the settings in Tab. 10. For Tab. 1, we use the same settings but train for 30 epochs on K400 and SSv2. All ablations use the same configuration with 20 training epochs. Following VideoMAE [3], we use dense sampling [64, 65] on K400, and TSN-style sampling [66] on SSv2 and the remaining datasets.

Importantly, we run the exact same learning-rate sweep separately for IV2, IV2+RLT, and our LookWhen. For IV2+vid-TLDR, we follow the original paper [11] by tuning the merging schedule. Specifically, we first make 5 sparsity levels that merge 500, 700, 1000, 1400, or 1800 tokens. For each sparsity level, we make 4 schedules (following templates in vid-TLDR's Table A) and choose the schedule that performs best.

Table 9: **Linear probing configuration.**

Config	Value
Optimizer	AdamW
Base learning rate	1e-4, 5e-4, 1e-3, 5e-3
Batch size	256
Learning rate schedule	Cosine annealing

Table 10: **Fine-tuning configuration.**

Config	K400	SSv2	Diving48	EK100	Jester	Charades
Optimizer	AdamW					
Weight decay	0.01					
Min. LR	5×10^{-6}					
Batch size	128					
LR schedule	Cosine annealing w/ linear warmup					
Base LR (swept)	1e-5, 3e-5 1e-4, 3e-4	1e-4, 3e-4 5e-4, 1e-3	5e-4	1e-4, 3e-4 5e-4, 1e-3	1e-5, 3e-5 1e-4, 3e-4	1e-5, 3e-5 1e-4, 3e-4
Training epochs	10	10	50	10	10	50
Warmup epochs	1	1	5	1	1	5
Input resolution	224×224					
# input frames	16					
Drop path rate	0.1					
Random crop scale	$[0.5,\ 1.0]$					
Horizontal flip	✓	✗	✓	✗	✗	✗
Color jitter	(p=0.8, $b/c/s$=0.4, h=0.1)					
Gaussian noise	(p=0.5, σ=0.1)					
Batch renorm. aug.	(p=0.5)					
Mixup	(α=0.8)					
CutMix	(α=1.0)					
Label smoothing	0.1	0.1	0.1	0.1	0.1	0.0

A.7.4 More results and Ablations

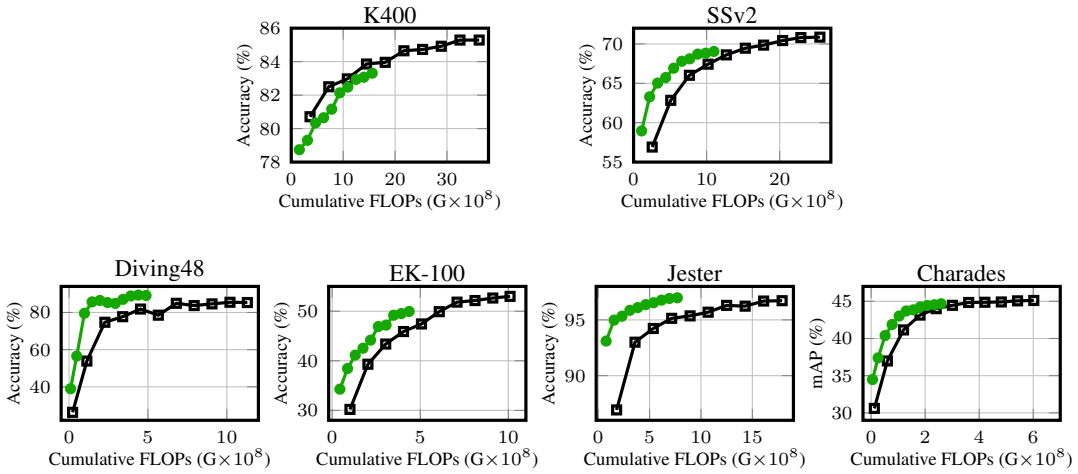


Figure 13: **Fine-tuning efficiency.** We plot *cumulative* fine-tuning cost vs. accuracy. At 70% sparsity, LookWhen (●) reaches a given accuracy faster than the dense InternVideo2 (■) during fine-tuning. Each marker represents 1 epoch for EK-100 and Jester, and 5 epochs for Diving48 and Charades.

Table 11: **Full ablation over what to extract.** Settings are the same as the main-paper ablations.

	DINOv3				K400-20K		SSv2-20K		Diving48		EK100-20K		Jester-20K		Charades	
IntVid2	Vid	Frame	Patch	norm	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT
✔	✔	✔	✔	✔	72.5	73.8	38.1	50.1	37.6	84.8	13.5	23.8	**79.0**	95.2	**31.3**	**39.3**
✔	✔	✔	✘	✔	72.5	73.8	38.6	50.1	35.5	84.1	**13.8**	**24.2**	77.0	95.2	31.0	39.1
✔	✔	✘	✔	✔	72.7	73.8	**39.1**	49.6	31.9	84.6	**13.8**	23.8	77.1	**95.3**	30.7	38.7
✔	✔	✘	✘	✔	**73.3**	**74.0**	37.9	48.0	29.9	81.9	12.8	23.3	69.0	94.6	30.3	37.4
✔	✔	✘	✘	✘	72.1	73.3	31.5	45.1	32.1	84.2	11.3	22.4	63.9	94.9	29.5	36.9
✔	✘	✔	✔	✘	72.9	73.6	31.5	45.2	29.7	83.2	11.6	22.2	63.4	94.7	29.3	36.7
✔	✘	✔	✘	✘	72.9	73.7	31.8	46.4	30.4	82.7	11.3	22.7	62.7	94.9	29.3	37.0
✔	✘	✘	✔	✘	73.2	73.9	32.0	45.0	29.4	83.2	11.3	21.6	60.8	94.4	28.9	36.2
✔	✘	✘	✘	✘	**73.3**	73.9	32.1	45.6	27.9	83.1	11.0	22.0	61.6	94.4	29.0	36.3
✘	✔	✔	✔	✔	24.3	65.3	25.5	**51.5**	29.8	84.9	2.8	20.9	70.8	95.2	14.9	35.5
✘	✔	✔	✔	✘	63.8	68.2	21.8	43.4	**37.8**	84.7	6.6	21.2	56.3	94.9	23.0	34.3
✘	✔	✘	✘	✔	17.9	61.1	26.7	48.8	26.7	**85.7**	2.6	18.7	70.4	95.2	14.0	30.5
✘	✘	✔	✔	✘	53.9	68.0	25.1	45.6	31.6	85.6	6.5	21.0	77.0	94.9	21.6	36.0

Table 12: **LookWhen-TIPSv2.** The selector-extractor framework also works with a strong frame-wise teacher other than DINOv3. We replace DINOv3's selection and image-based extraction targets with TIPSv2 [60], keeping the selector-extractor architecture and initialization unchanged.

	TIPSv2				K400-20K		SSv2-20K		Diving48		EK100-20K		Jester-20K		Charades	
IntVid2	Vid	Frame	Patch	Norm	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT
✔	✔	✔	✔	✔	67.5	68.5	37.6	48.2	**32.6**	83.2	10.9	21.2	74.0	94.7	26.3	33.6
✔	✔	✔	✘	✔	**72.7**	**73.3**	**39.1**	48.8	30.2	84.1	13.3	**23.9**	**75.8**	**94.8**	**29.5**	**36.9**
✔	✔	✘	✔	✔	68.3	68.9	38.1	46.7	32.2	82.5	11.3	21.1	72.7	94.8	25.6	33.1
✔	✔	✘	✘	✔	72.5	73.0	39.0	**49.3**	32.2	**84.5**	**13.9**	23.6	74.4	94.8	29.0	36.4
✔	✔	✔	✔	✘	71.8	72.4	32.3	42.9	26.4	82.1	12.0	21.2	66.4	94.1	27.8	35.0
✔	✘	✔	✔	✘	72.0	72.7	32.0	43.6	29.1	82.2	11.3	21.7	65.9	94.3	28.0	35.0
✔	✘	✔	✔	✔	69.7	70.3	31.6	45.8	30.0	82.3	10.4	20.7	65.5	94.7	26.6	33.8
✘	✔	✔	✔	✔	23.6	56.8	26.1	46.3	25.2	83.9	3.3	16.8	67.1	94.6	13.6	26.1
✘	✔	✔	✔	✘	57.4	61.5	23.5	40.6	25.7	82.6	6.7	18.3	57.7	94.2	20.3	29.2
✘	✘	✔	✔	✔	49.2	58.6	21.8	44.8	25.5	82.9	5.0	16.7	62.1	94.6	17.2	29.5

Table 13: **Long-video recognition with a fixed patch budget.** We examine temporal and spatial resolution while fixing the number of retained patch tokens K. T is the number of sampled frames and F is the spatial side length in pixels; each input has size $T \times F \times F$. We report linear-probe scores from the frozen epoch-20 variable-input checkpoint. Variable-size pre-training settings are given in Appendix A.7.2. Charades and ActivityNet use macro-mAP (%); UCF-Crime reports top-1 accuracy (%), as the four-fold mean ± population standard deviation.

(a) **Charades (mAP).**

	$T = 16$		$T = 64$		$T = 128$	
K	$F = 128$	$F = 256$	$F = 128$	$F = 256$	$F = 128$	$F = 256$
128	23.9	22.3	22.2	21.6	21.8	21.4
256	27.7	27.6	26.5	26.0	25.8	25.3
512	30.4	31.3	29.8	30.3	29.2	29.1
768	30.6	**33.3**	31.1	32.3	30.6	31.4

(b) **ActivityNet (mAP).**

	$T = 16$		$T = 64$		$T = 128$	
K	$F = 128$	$F = 256$	$F = 128$	$F = 256$	$F = 128$	$F = 256$
128	75.8	72.5	72.8	70.4	70.9	67.9
256	82.3	80.1	79.7	77.9	77.5	76.0
512	85.7	85.2	83.8	82.7	82.4	81.4
768	86.2	**86.6**	85.7	84.9	84.3	83.8

(c) **UCF-Crime (top-1 accuracy).**

	$T = 16$		$T = 64$		$T = 128$	
K	$F = 128$	$F = 256$	$F = 128$	$F = 256$	$F = 128$	$F = 256$
128	41.5 ± 0.9	41.2 ± 1.5	41.7 ± 0.9	42.1 ± 1.4	40.6 ± 1.5	39.7 ± 2.4
256	44.9 ± 1.7	46.6 ± 2.9	44.5 ± 1.6	47.0 ± 3.2	42.9 ± 2.6	43.9 ± 4.3
512	44.9 ± 3.2	47.2 ± 1.7	46.0 ± 3.9	47.6 ± 2.3	45.5 ± 2.9	46.4 ± 3.8
768	46.3 ± 2.9	46.9 ± 1.1	46.3 ± 3.8	$\mathbf{48.7 \pm 2.5}$	45.1 ± 2.6	45.7 ± 2.4

Table 14: **Ablating pre-training data.** Pre-training on unlabeled K400 mostly helps downstream accuracy on K400, Diving48, and Charades; pre-training on unlabeled SSv2 mostly improves SSv2.

		K400-20K		SSv2-20K		Diving48		EK100-20K		Jester-20K		Charades	
Dataset	Steps	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT	LP	FT
K400+SSv2	1×	72.9	**73.6**	31.5	**45.2**	29.7	**83.2**	**11.6**	**22.2**	63.4	**94.7**	**29.3**	**36.7**
K400	0.6×	**73.0**	73.3	24.5	42.8	**31.9**	83.1	9.8	20.8	**64.1**	94.6	28.2	36.0
SSv2	0.4×	55.2	63.3	**32.1**	43.9	21.4	71.8	10.2	20.2	63.4	94.3	23.3	32.1

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state our contributions and scope; Sections 3 and 4 evaluate the accuracy–computation trade-offs and the design choices underlying these claims.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 discusses the requirement for positionally-grounded teacher representations and the limited exploration of larger ViT models.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sections 2.3 and 2.4 describe the architecture and training objectives. Section 3 and Appendix A.7 specify the data, evaluation protocols, and training settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The project link on the first page provides public code, pre-trained weights, and installation, data-preparation, training, and evaluation instructions. Appendix A.7 gives the experimental settings, and the benchmark datasets are obtained from their original sources.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Section 3 and Appendix A.7 describe the data splits, sampling, augmentations, optimization, and hyperparameter searches; Tables 9 and 10 summarize the downstream settings.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Repeating all main experiments across random seeds was not computationally feasible. We report sample standard deviations across random-selection seeds in Table 7 and population standard deviations across four evaluation folds for UCF-Crime in Table 13.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 3 and Appendix A.1 report the GPU used for efficiency measurements, FLOPs, inference throughput, and peak inference memory; Figure 13 reports cumulative fine-tuning cost.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have read it.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: Our paper does not address societal impact because it is a general method for speeding-up video recognition. Potential negative societal impacts of our work could be to speed-up surveillance. Potential positive societal impacts could be to speed-up perception for autonomous cars or healthcare settings (e.g. patient monitoring).

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This work does not introduce high-risk generative models or a newly collected dataset requiring release safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite all datasets that we use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The public repository and pre-trained model are linked on the first page. The repository documents installation, data preparation, pre-training, linear probing, fine-tuning, and inference, with experimental details in Appendix A.7.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing or newly recruited human participants; experiments use existing benchmark datasets.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Our research did not involve human subjects.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: We only used LLMs a bit for writing and coding help.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.