OLORAS: Online LOng Range Action Segmentation for edge devices

1st Filippo Ziche
University of Verona
Verona, Italy
filippo.ziche@univr.it

2nd Nicola Bombieri
University of Verona
Verona, Italy
nicola.bombieri@univr.it

***Abstract*—Temporal action segmentation (TAS) is essential for identifying when actions are performed by a subject, with applications ranging from healthcare to Industry 5.0. In such contexts, the need for real-time, low-latency responses and privacy-aware data handling often requires the use of edge devices, despite their limited memory, power, and computational resources. This paper presents OLORAS, a novel TAS model designed for real-time performance on edge devices. By leveraging human pose data instead of video frames and employing linear recurrent units (LRUs), OLORAS efficiently processes long sequences while minimizing memory usage. Tested on the standard Assembly101 dataset, the model outperforms state-of-the-art TAS methods in accuracy with 10x memory footprint reduction, making it well-suited for deployment on resource-constrained devices.**

***Index Terms*—temporal action segmentation, edge devices, linear recurrent units**

I. Introduction

Temporal action segmentation (TAS) is a crucial task that helps to understand the actions performed by a subject, including when they start, how they progress, when they finish, and how they transform the environment [1]. It is the one-dimensional equivalent to the more established two-dimensional semantic segmentation on images, where each temporal segment of the input data stream is labeled with a set of predefined action labels.

Online TAS aims to identify actions in a *live* video stream, even when only partial observations of the actions are available. This evolution of TAS represents a significant advancement, as it enables the development of practical, real-world applications. The ability to process data online is crucial for various industrial and telemedicine applications, such as human-robot interaction, safety, and workers' health reporting [2]. Also, we target edge AI systems, as they provide several advantages. By processing data on-site, they provide near-instantaneous feedback without network delays. They are more robust to network outages, which is crucial for safety. Keeping data and inference within the device ensures that patient or subject information remains secure and less vulnerable to attacks and data breaches.

The action segmentation solutions in the literature primarily rely on RGB/D video analysis. However, the high computational and memory requirements of this approach make it unsuitable for online classification on edge devices. A potential alternative is action segmentation based on the analysis of human-pose keypoints. Using such a human representation of skeletal keypoints, which can be extracted at runtime by a *human-pose estimator* (HPE) [3], offers several advantages for human action segmentation: It reduces memory and computational demands, removes irrelevant background information, and clarifies the spatio-temporal evolution of body movements, making the data easier to process.

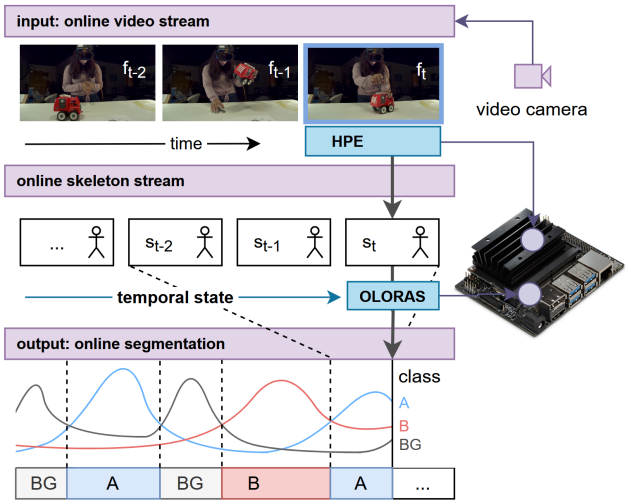

Fig. 1. Overview of our methodology

Current solutions are based on standard recurrent neural network (RNN) architectures, such as gated recurrent units (GRU) [4] and long short-term memory (LSTM) [5]. Their memory requirements typically prevent the model from processing large temporal windows, limiting its ability to capture the sequential relationships between different actions—an ability recognized as crucial for effective temporal action segmentation (TAS) [6]. They are also limited to sequentially processing the input data, limiting their performance during training. In alternative, other works use temporal convolution networks [7]. They are not suited for online inference as they require the entire input sequence before prediction. Recently, there have also been models based on *transformers* that can be used both online and offline [8], [9]. If they are online they still require a memory buffer of historical inputs (called *cache*) for their attention mechanism. Despite their successes, they are difficult to train with the limited amount of data available in this task and have a quadratic memory complexity, making them unsuitable for edge devices with limited memory availability.

978-3-9826741-0-0/DATE25/© 2025 EDAA

In this paper we propose OLORAS, a model that takes advantage of linear recurrent units (LRU) [10], which have shown great potential in the edge AI scenario. Our approach is pioneering in using these new models for online action segmentation while directly employing human pose data. Figure 1 shows the high-level view of our approach. We start from the skeletons of keypoints (s_t) extracted at runtime from each frame (f_t) of the video stream by a human pose estimator (HPE). OLORAS analyzes the keypoints of the current frame together with an inner state generated by previous inputs to classify the frame into an action class at runtime (e.g., classes A, B or background BG in the figure). OLORAS achieves higher classification accuracy and requires less total memory than other state-of-the-art with comparable or higher inference speed. We present the evaluation of the proposed model on both a standard NIVIDA GPU and a Jetson edge device by using a widespread and standard large dataset.

The paper is organized as follows. Section II presents the background and the related work analysis. Section III presents the OLORAS model, while Section IV presents the experimental results. Section V concludes the paper with final remarks.

II. Background and related works

A. Temporal segmentation

Initial work by Colin et al. [11] uses a sequence of 1D convolutions, pooling, and upsampling operations to compress the temporal sequence and then expand into the final segmentation. This method is not suited for online TAS as it requires the entire input sequence in advance. Online TAS has been implemented in MiniROAD [12], which uses a simple GRU [4] together with a non-uniform weight applied to the loss function to learn in a way that is more optimized for the actual inference phase. De et al. [13] propose a multistream architecture with two LSTMs, one that directly processes the input sequence and produces an initial segmentation, and another LSTM that processes the segmentation to refine the final prediction. Mingze et al. proposed TRN [14], a modified RNN architecture that, other than online detection, anticipates possible actions in the immediate future and uses them to increase the accuracy of their model in the present. Mingze et al. propose LSTR [15], a *transformer* architecture that combines a long-range encoder that analyzes and compresses long-range historical data and a short-term decoder that uses recent frames together with the compressed state to make the final predictions. ASFormer [8] uses a single decoder and multiple encoders in sequence to refine the model output at each step. Xiang et al. proposed OadTR [16], which uses a decoder to compress the temporal sequence and a decoder to anticipate future predictions. Then, similar to TRN, they are combined to refine the current frame classification. All transformer models require one or multiple historical buffers of previous inputs. They are not suited for low-memory and low-latency devices.

B. Human Pose Estimator (HPE)

A HPE is a SW platform based on a neural network that estimates the positions of human body joints from images or video frames. These networks predict the locations of keypoints (joints) on the human body, such as the head, shoulders, elbows, wrists, hips, knees, and ankles. The result is a skeletal representation of the person's pose, which can be used for various applications including action recognition, human-computer interaction, sports analysis, and more. Example of real-time HPEs are [3], [17] and TRT-Pose [18] for edge devices.

C. State Space Models (SSMs)

Recently, there has been a renewed interest in RNN, driven by the introduction of new RNN-like models based on state space models (SSMs). Examples are S4, S4D, S5, LRU, and Mamba [10], [19]–[22]. They efficiently compress the entire input sequence into a fixed-size hidden state. Formally, SSMs model the input data based on the following ordinary differential equation (ODE): $h'(t) = Ah(t) + Bx(t)$, $y(t) = Ch(t) + Dx(t)$. $x(t)$ and $y(t)$ are continuous input and output signals in the time domain, respectively. SSMs discretize the A, B, C, D matrices in this ODE to obtain the following update rules: $h_t = \tilde{A}h_{t-1} + \tilde{B}x_t$ and $y_t = \tilde{C}h_t + \tilde{D}u_t$. Instead of applying a non-linearity $\sigma(\cdot)$ after each update, they first process the entire sequence in parallel and then apply it to the final hidden states $z_t = \sigma(y_t)$, which provides notable speedup compared to standard RNNs that are sequential in nature. Using linearity, the hidden state h_t can be generated efficiently on modern hardware such as GPU with a modified Blelloch's parallel prefix scan algorithm [23]. For an input sequence of length n, SSMs require to materialize the $\tilde{A}^n$ matrix. Recent works diagonalize with great effect this matrix to limit the update complexity from $O(n^3)$ to $O(n^2)$ while still obtaining comparable modeling capabilities. Examples of such works are S4D and LRU, which are simpler to implement and aggregate several advancements in the category of temporal models. In this work we use LRU [10], which are based on the following update rule:

$$
\begin{aligned}
h_k &= diag(\Lambda)h_{k-1} + \gamma \odot \tilde{B}x_k \\
y_k &= \tilde{C}h_k + \tilde{D}x_t \\
z_k &= \sigma(y_k)
\end{aligned}
$$

where $diag(\Lambda) \in \mathbb{C}^{d\times d}$ and γ is a normalization factor to aid the training for long-range sequences.

III. The OLORAS model

A. Problem description

The input of our model is a stream of human skeletons, where each skeleton has been extrapolated by a video frame through a HPE. The objective is to classify each frame at runtime on an edge device. We refer to the input sequence of the skeleton of keypoints as the tensor $\mathcal{S}$ of size (T, J, F), where T is the sequence length, J is the number of skeleton keypoints (*joints* in the following), and F is the dimensionality of the features of each joint. We refer to each skeleton at time t as $\mathcal{S}_t$, and at its joint j as $\mathcal{S}_{t,j} = [\mathcal{S}^x_{t,j}, \mathcal{S}^y_{t,j}, \mathcal{S}^z_{t,j}]$, where $\mathcal{S}^x_{t,j}$, $\mathcal{S}^y_{t,j}$, and $\mathcal{S}^z_{t,j}$ are the 3D keypoint coordinates of joint j at time t[1]. Our

[1]In this work, we consider 3D HPE information. However, OLORAS also supports 2D HPE data.

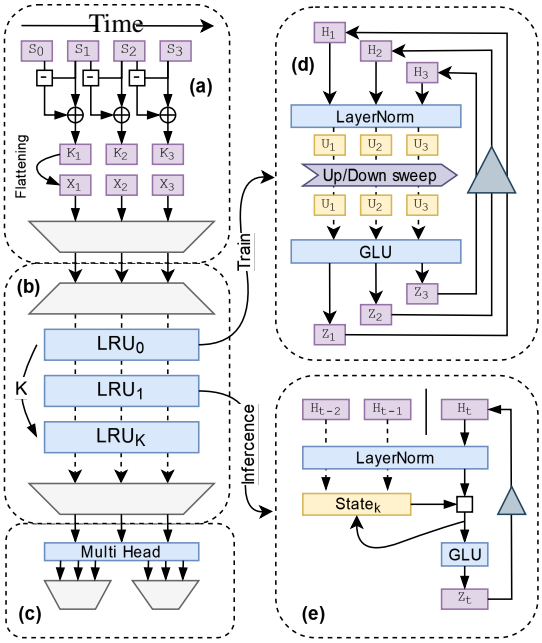

Fig. 2. Architecture of the model.

goal is to predict c_t given only the historical inputs $\mathcal{S}_{0:t}$, $c_t \in \{0, 1, \ldots, K-1\}$ where K is the number of classes of interest.

B. Architecture

Figure 2 shows the OLORAS architecture. It is composed of a preprocessing module *a*, a temporal module *b*, and a final frame classifier *c*. The figure also shows the inner definition of the LRU blocks for the train phase *d* and inference-phase *e*.

1) Preprocessing module (a): Figure 2 section *a* shows this initial module. It inserts velocity information in the skeleton. Given the input tensor $\mathcal{S}$, it creates a new tensor $\mathcal{K}$ of size $(T-1, J, F')$, where $F' = 2F$. Each joint features F is augmented to F' as follows:

$$\mathcal{S}^d_{t,j} = [\mathcal{S}^x_{t,j} - \mathcal{S}^x_{t-1,j}, \quad \mathcal{S}^y_{t,j} - \mathcal{S}^y_{t-1,j}, \quad \mathcal{S}^z_{t,j} - \mathcal{S}^z_{t-1,j}]$$
$$\mathcal{K}_{t,j} = \mathcal{S}_{t,j} \oplus \mathcal{S}^d_{t,j}$$

where $\oplus$ is the concatenation operator, $\mathcal{S}^d_{t,j}$ is the first time difference between the coordinates of joint j at frames t and $t-1$. Then, it flattens the tensor $\mathcal{K}$ from $(T-1, J, F')$ to $(T-1, C)$, where $C = J \times F'$. This aggregates in the same dimension all the joint features. At the end, a linear layer augmented with layer normalization and dropout decreases the dimensionality of the last dimension C of the augmented tensor into the final tensor $\mathcal{X}$ of size $(T-1, d_m)$, where d_m is the inner dimensionality of the model.

2) Temporal model (b): Figure 2 section *b* shows the block model, which processes the temporal dimension of the $\mathcal{X}$ tensor using an LRU-block. The block is composed of an initial linear projection from the model dimension d_m to the temporal dimension d_t, a stack of LRU layers, and a final linear projection back to the model dimension d_m. It stacks K LRU layers to increase the modelling capability of the architecture to handle more complex temporal relations. Each layer receives as input the output of the previous one. The block implements the following operations:

$$\mathcal{H}^0 = \mathcal{X}W_0 + b_0$$
$$\mathcal{H}^{k+1} = LRU_k(\mathcal{H}^k)$$
$$\mathcal{R} = \mathcal{H}^K W_1 + b_1$$

where $\mathcal{H}^0$ is the result of the initial projection, $\mathcal{H}^{k+1}$ is the output of the LRU layer from the previously processed tensor $\mathcal{H}^k$, $\mathcal{R}$ is the result of the entire block after the final projection, and W_0, b_0, W_1, b_1 are the weights and biases of the network. Each LRU layer LRU_k contains a layer normalization, the LRU update rule as presented in the background section, and a final gated linear unit (GLU) as the activation function [24]. The layer has two modalities, one for training and one for inference. The first is shown in Figure 2 section *d*. The sequence elements are processed in parallel with an up-down sweep and sent to the next LRU layer in the block together. Figure 2 section *e* shows the inference mode, where only the last hidden state is kept inside each layer before the activation function. Each new input updates the hidden state $state_k$ of all K LRU layers in sequence.

3) Classifier (c): The model finally implements a multi-head classifier, as shown in Figure 2 section *c*. Each head classifies all the temporal elements in the tensor $\mathcal{R}$ independently to different label sets. The block implements a linear classifier in conjunction with a softmax activation function. It generates the probabilities of each possible label for each frame. By selecting the label with the highest probability, the block generates the final temporal segmentation.

C. Loss

The model is trained using the custom loss function proposed in [25]. It combines the cross-entropy (CE) and mean-square-error (MSE) losses. Cross-entropy is used for the multiclass classification of the single frames, while mean-square error is used as a secondary loss to help reduce the problem of over-segmentation., as follows:

$$\mathcal{L}_{\text{CE}} = -\sum_{t}^{T} \log(y_{t,c}) \quad \mathcal{L}_{\text{MSE}} = \sum_{t}^{T} (\Delta_{t,\epsilon})^2$$

where $y_{t,c}$ is the predicted probability for the ground truth label at time t and class c, and where $\Delta_{t,\epsilon}$ is defined as the minimum between $|\log(y_{t,c}) - \log(y_{t-1,c})|$. and ϵ, that is the difference between predicted values at consequent timesteps. It suggests the network to maintain a similar classification between near frames. The two losses are combined linearly using a coefficient λ as follows:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda * \mathcal{L}_{\text{MSE}}$$

Since the λ coefficient has to be kept small, we use a value of 0.22, and set ϵ as 4, following the original paper. Values too large make the model output a single classification class for all frames to reduce the penalty caused by a change in the predicted class. We also use a standard L2 regularization to limit the absolute value of the model's weights. L2 adds the square of the sum of the coefficients (L2 norm) as a penalty

term. This shrinks the coefficients towards zero but does not necessarily set them to zero, resulting in a smoother reduction in complexity. The final loss function is:

$$\mathcal{L}_{final} = \mathcal{L} + \delta \sum_{i}^{p} \beta_i^2$$

where p is the number of network parameters, β_i is a single parameter and δ is the regularization coefficient.

IV. Experimental results

A. Datasets

We evaluated OLORAS on Assembly101, a publicly available and reference standard dataset for action segmentation. It is a large dataset in which 53 participants are tasked with assembling and disassembling take-apart toys without instructions. It contains 513 hours of footage divided into 4,321 videos from multiple viewpoints (3rd person and egocentric). It provides HPE data extracted from each video frame. Each video can span over 20 minutes (over 36,000 frames). The dataset provides coarse and fine labels of the actions. For the sake of space we report the results obtained with fine labels as they require a more difficult segmentation task. We observed similar or better results with coarse labels. Fine labels are composed of a combination of 25 actions applied to 90 different toys' parts, together with a background class when no known action is being performed. The dataset provides both the temporal labeling for the action and the toy components used in the action. We classified both labels using the multi-head classifier. We used only one of the multiple 3rd person views provided for training and evaluation.

B. Settings

OLORAS was trained on an NVIDIA RTX4090 with 24GB of VRAM and an MD Ryzen 9 7950X 16-Core Processor with 64GB of RAM. During the first training epochs, the gradients of the network exploded. This can be explained by the particularly long input sequences and initial high loss of the frame classification. For this reason, we also implemented gradient clipping [26] only in the first epochs of training. This technique clips the gradients to a maximum value of 100. As the optimizer, we used SGD [27] with a learning rate of 0.01. The learning rate decreases every 100 epochs by a factor of 0.5, and we trained the model for 400 epochs in total. During training, we sectioned the dataset sequences into clips of 10,000 frames each to allow the network to learn long-range relationships between actions. Dropout was set to a standard 0.20.

C. Evaluation metrics

We adopt three common metrics to evaluate the temporal action segmentation in a supervised training scheme. They are *mean over frames* (MoF), *edit score*, and *F1 score*. The MoF metric is a frame-wise accuracy and is defined as the fraction of the model's correct frame predictions:

$$MoF = \frac{\text{nun. correct frames}}{\text{num. total frames}}$$

Since it only evaluates predictions at the individual frame level and does not reflect the quality of the predicted segments, we also include the edit score and F1. The edit score calculates the distance between two sequences. It is based on the Levenshtein distance (i.e., edit distance). It calculates the number of operations (insertion, deletion, replacement) required to make two sequences equal. A high edit score means that the required operations are low in number and that there is less over-segmentation in the output. The F1 score or F1@γ compares the intersection over union (IoU) of each segment with respect to the corrisponding ground truth based on some threshold $\gamma/100$. Commonly used γ values are $\{10, 25, 50\}$. A segment is considered a true positive if its IoU with respect to the ground truth exceeds the threshold. If there is more than one correct segment within the span of a single ground truth action, then only one segment is considered a true positive (TP) and the others are marked as false positives (FP). Based on the true and false positives as well as false negatives (missed segments, FN), one can compute the precision and recall. F1 score is defined as:

$$F1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

where precision is $\frac{TP}{TP+FP}$ and recall is $\frac{TP}{TP+FN}$.

D. Comparison with the state of the art

We compared OLORAS with MiniROAD [12] and TRN [14], which are efficient solutions at the state of the art that can work in resource-constrained settings, where the model has to generate frame labels at runtime with minimal latency. Both models are based on RNNs and are presented in details in the related work section.

We do not consider, in the comparison, *non-causal* (i.e., not online) action segmentation models at the state-of-the-art, which require very large historical caches or the entire sequence of frames before execution.

We evaluated the efficiency of all models to segment the human actions *and* to segment the components of the toys manipulated during each action. For both segmentation tasks, we compared the efficiency of all solutions in two ways. The first starting from human keypoint sequences while the second starting from the video frames. The second has been done to compare the different solutions with the native configuration of the state of the art counterparts. Both data representations (i.e., keypoint and frame sequences) are provided by the Assembly101 dataset.

To run MiniROAD and TRN on keypoint sequences, we implemented a *skeleton-to-embedding* connector. It converts the inputs represented by skeletons of keypoints (i.e., tensor $\mathcal{S}$ of size (T, J, F)) to the embedding format $\mathcal{S}_E$ of size (T, C)) supported by the two state-of-the-art models, where $C = J \times F$.

We compared all models in terms of accuracy, memory usage, inference and training speed, both on an NVIDIA RTX4090 and on a Jetson Nano as edge device. We run the experimental analysis with different model sizes. For the sake of clarity, we present the results with two representative

TABLE I
METRIC SCORES OF THE ACTION SEGMENTATION TASK USING HUMAN KEYPOINTS AS INPUT. BOLD AND ITALIC VALUES SHOW THE BETTER SCORES FOR HIGHER AND LOWER PARAMETER COUNT CONFIGURATIONS, RESPECTIVELY.

Model	Params	Mof	Edit	F1@10	F1@25	F1@50
MiniROAD	˜16M	0.03	0.08	0.026	0.014	0.008
	˜1M	0.05	0.12	0.037	0.021	0.007
TRN	˜16M	0.04	0.03	0.008	0.004	0.001
	˜1M	0.05	0.02	0.009	0.005	0.001
OLORAS	˜15M	**0.10**	**0.21**	**0.080**	**0.040**	**0.012**
	˜1M	***0.06***	***0.14***	***0.041***	***0.023***	***0.008***

TABLE II
METRIC SCORES FOR THE SEGMENTATION TASK OF THE TOY COMPONENTS MANIPULATED DURING ACTIONS USING HUMAN KEYPOINTS AS INPUT. BOLD AND ITALIC VALUES SHOW THE BETTER SCORES FOR HIGH AND LOW PARAMETER COUNT CONFIGURATIONS, RESPECTIVELY.

Model	Params	Mof	Edit	F1@10	F1@25	F1@50
MiniROAD	˜16M	0.03	0.02	0.017	0.008	0.002
	˜1M	0.01	0.03	0.008	0.004	0.001
TRN	˜16M	0.01	0.00	0.001	0.000	0.000
	˜1M	0.01	0.01	0.001	0.001	0.000
OLORAS	˜15M	**0.04**	**0.10**	**0.025**	**0.014**	**0.004**
	˜1M	***0.03***	***0.05***	***0.016***	***0.010***	***0.003***

TABLE III
METRIC SCORES OF THE ACTION SEGMENTATION TASK USING FRAME EMBEDDINGS AS INPUT. BOLD VALUES SHOW THE BETTER SCORES FOR ALL PARAMETER COUNT CONFIGURATIONS.

Model	Params	Mof	Edit	F1@10	F1@25	F1@50
MiniROAD	˜14M	0.03	**0.14**	0.028	0.014	**0.010**
TRN	˜38M	0.05	0.07	0.021	0.011	0.004
OLORAS	˜15M	**0.11**	**0.14**	**0.056**	**0.026**	0.008

TABLE IV
METRIC SCORES FOR THE SEGMENTATION TASK OF THE TOY COMPONENTS MANIPULATED DURING ACTIONS USING FRAME EMBEDDINGS AS INPUT. BOLD VALUES SHOW THE BETTER SCORES FOR ALL PARAMETER COUNT CONFIGURATIONS.

Model	Params	Mof	Edit	F1@10	F1@25	F1@50
MiniROAD	˜14M	0.06	0.08	**0.033**	**0.018**	**0.008**
TRN	˜38M	0.03	0.02	0.005	0.002	0.001
OLORAS	˜15M	**0.08**	**0.10**	0.030	0.015	0.006

configurations: A very low-parameter version with ˜1 million parameters and a higher-parameter version with ˜16 million parameters for each model.

For OLORAS, the low-parameter version uses 3 LRU layers, has a model dimension d_m of 64, and a temporal state dimension d_t of 256. The higher-parameter version has 6 LRU layers with a d_m and d_t of 256 and 768, respectively. We changed the MiniROAD and TRN hidden state size to obtain the total parameter count comparable to OLORAS.

1) Evaluation: Tables I and II show the results for the segmentation of actions and toy components, respectively. We found that MiniROAD achieves its best results in segmenting human actions with the lower-parameter configuration. In contrast, its accuracy scales with the model size in segmenting the toy components. This is due to the fact that by using a more representative input (i.e., the skeleton poses) the network has difficulty in optimizing all its weights compared to the 1M parameters configuration for the action task. For the same reason, the TRN accuracy does not scale with the model size in both the segmentation tasks.

In general, OLORAS achieves better results in all metrics for both higher- and lower-parameter configurations. In particular, the OLORAS accuracy with the lower-parameter configuration is higher than the accuracy of the counterparts with the higher-parameter configuration. This is due to the more optimal parameter utilization of the adopted LRU architecture.

As expected, all models are less accurate in segmenting toy components than human actions. This is due to the fact that, starting from HPE rather than RGB/D data, the models cannot take advantage of the background video information. However, differently from the state-of-the-art models, the accuracy of OLORAS in segmenting components without video background is more than acceptable with both parameter configurations.

For the second set of experiments, we compared the segmentation accuracy of the different solutions with the native configuration of the state of the art counterparts (i.e., starting from video frames rather than keypoints and with the best model size suggested by the corresponding authors).

Tables III and IV show the results. We found that the MiniROAD and TRN accuracy in segmenting human actions starting from video frames is slightly higher than the accuracy achieved by the same model in the lower-parameter configuration starting from keypoints (see Tables III and I). In contrast, they better exploits the video information for segmenting toy comonents (Tables IV and II).

In general, OLORAS is more accurate or comparable to its counterparts in both action an component segmentation tasks. It is less accurate than in the first set of experiments in all metrics except MoF. The lower scores are due to the fact that the SSMs adopted in the model are more accurate with input data similar to a continuous function [22]. In contrast, embeddings contain information in a more condensed way, which is less ideal for our architecture. This is reflected in the other metrics that consider segments. For the component segmentation, OLORAS achieves better results in terms of Mof and Edit, while it is comparable to MiniROAD in the other metrics. As expected, the accuracy for component segmentation is higer as the models have access to visual information.

It is important to note that, even considering the state-of-the-art models with their best configuration and starting from videos, OLORAS achieves a comparable or better accuracy with the lower-parameter setting and starting from keypoints.

E. Memory consumption and inference time

We evaluated the memory consumption and inference time (in frames per second - FPS) of each model during the

TABLE V
STATE MEMORY, TOTAL MEMORY CONSUMPTION, AND INFERENCE TIME (IN FRAMES PER SECOND - FPS) ON AN NVIDIA RTX4090 AND ON AN NVIDIA JETSON NANO

Model	**Params** (#)	**State mem.**	**Total mem.**	**FPS**	
				RTX4090	Nano
MiniROAD	~16M	8 KB	125 MB	4000	84
	~14M	4 KB	85 MB	4000	160
	~1M	1 KB	7 MB	6000	170
TRN	~38M	16 KB	151 MB	600	17
	~16M	8 KB	61 MB	600	17
	~1M	2.5 KB	5 MB	900	16
OLORAS	~15M	37 KB	86 MB	1100	48
	~1M	8 KB	8 MB	1600	70

segmentation tasks. Table V presents an overview of the results (i.e., starting from human keypoints for MiniROAD 1M/16M, TRN 1M/16M, and OLORAS 1M/15M and from videos for MiniROAD 14M and TRN 38M). The total memory values include the state memory used for the inner state. OLORAS requires more memory to maintain the inner state as it keeps a hidden state for each LRU layer, while the others use a single hidden state. In general, the total memory consumption of the three models with the lower-parameter configuration is comparable. TRN and OLORAS scale less than linearly with the number of parameters (5/8MB to 86/61MB from ∼1M to ∼16M configurations) while Miniroad scales more than linearly. This happens with the models configured for human keypoints. MiniROAD and TRN require twice the memory for the original configurations.

It is important to remark that, considering the models at the state of the art with their best configuration and starting from videos, OLORAS achieves a comparable or better accuracy with the lower-parameter setting and starting from keypoints, which translates in a ∼16x total memory reduction.

Finally, we evaluated the inference time of the models to assess their ability to perform TAS tasks online. The input frame rate of the dataset is set to 60 FPS (consistently with standard RGBD camera sensors at the state of the art). We measured the latency of TRTPose [18], an HPE backbone that can be used online to pre-process the video frames for the keypoint-based models on a Jetson Nano device. Similarly, we measured the latency of the embedding generator used in the dataset (TSM [28]), which pre-processes the videos for the frame-based models. Both pre-processing tools, essential for running the segmentation models, exhibited comparable computation times. This was expected, as both involve inference on CNNs of similar size. The pre-processing step reduces the maximum online frequency to 42 FPS. In this context, MiniROAD and OLORAS fully support the online processing on the Jetson Nano, while the TRN execution results in frame dropping.

In conclusion, the best configuration of OLORAS (∼1M) largely supports the system frame rate required for the online application. MiniROAD, in any configuration, is faster than OLORAS. Nevertheless, the accuracy of OLORAS in segmenting toy components is significantly higher than that of

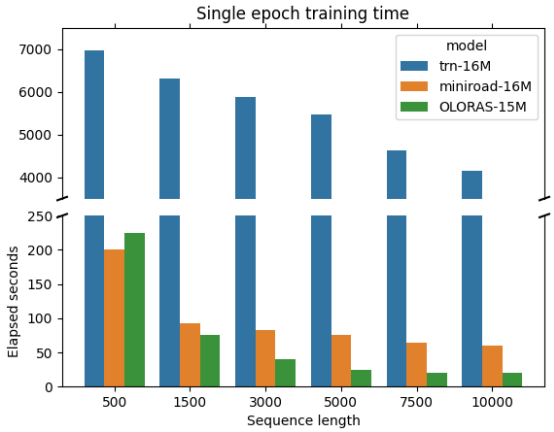

Fig. 3. Training time of the three models. They process the same amount of frames but with varying sample sequence lengths.

MiniROAD, which demonstrates strong limitations in this task, while also being slightly more accurate in segmenting actions. Additionally, ∼1M OLORAS achieves higher accuracy than ∼16M MiniROAD, with a 16x reduction in memory usage. Compared to the native ∼14M MiniROAD, ∼1M OLORAS achieves higher accuracy with 10x lower memory consumption.

1) Training time: We analyzed the training time of all models by varying the length of the training clip sequence. Figure 3 shows the results. OLORAS, by leveraging the parallel scan implementation of LRUs, has much more efficient training times. TRN requires hours of training time, which is out of scale compared to OLORAS and MiniROAD. For very short input sequences with up to 500 frames (i.e., less than 9 seconds), MiniROAD has a faster training phase. With sequences longer (i.e., in the majority of real cases) OLORAS achieves up to 3x training speedup. We also note that MiniROAD uses an optimized pytorch implementation, while OLORAS does not use fused kernels and can be further optimized. This is part of our future work.

V. CONCLUSION

In this paper, we presented OLORAS, an online action segmentation model optimized for resource-constrained edge devices. We target online TAS using skeleton data as an intermediary step in the segmentation pipeline. We demonstrated that recent innovations in RNN architectures, such as LRUs, can be very efficient for segmentation tasks computed in devices with such temporal and computational constraints. We presented experimental analysis with a standard dataset that also include very long sequences (up to 10,000 frames) on Jetons Nano as edge device. OLORAS achieves better accuracy than the competitors in the temporal action segmentation task for both the human actions and the manipulated toy components tasks, with up to 10x lower memory consumption.

VI. ACKNOWLEDGMENT

This work has been supported by the ”PREPARE” project (n. F/310130/05/X56 - CUP: B39J23001730005) - D.M. MiSE 31/12/2021.

REFERENCES

[1] G. Ding, F. Sener, and A. Yao, "Temporal action segmentation: An analysis of modern techniques," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[2] T. Benmessabih, R. Slama, V. Havard, and D. Baudry, "Online human motion analysis in industrial context: A review," *Engineering Applications of Artificial Intelligence*, vol. 131, p. 107850, 2024.

[3] Z. Yang, A. Zeng, C. Yuan, and Y. Li, "Effective whole-body pose estimation with two-stages distillation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4210–4220.

[4] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[6] E. Bahrami, G. Francesca, and J. Gall, "How much temporal long-term context is needed for action segmentation?" in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 10 351–10 361.

[7] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 156–165.

[8] F. Yi, H. Wen, and T. Jiang, "Asformer: Transformer for action segmentation," *arXiv preprint arXiv:2110.08568*, 2021.

[9] F. Sener, D. Singhania, and A. Yao, "Temporal aggregate representations for long-range video understanding," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*. Springer, 2020, pp. 154–171.

[10] A. Orvieto, S. L. Smith, A. Gu, A. Fernando, C. Gulcehre, R. Pascanu, and S. De, "Resurrecting recurrent neural networks for long sequences," in *International Conference on Machine Learning*. PMLR, 2023, pp. 26 670–26 698.

[11] C. Lea, M. Flynn, R. Vidal, A. Reiter, and G. Hager, "Temporal convolutional networks for action segmentation and detection," 07 2017, pp. 1003–1012.

[12] J. An, H. Kang, S. H. Han, M.-H. Yang, and S. J. Kim, "Miniroad: Minimal rnn framework for online action detection," in *International Conference on Computer Vision (ICCV)*, 2023.

[13] R. De Geest and T. Tuytelaars, "Modeling temporal structure with lstm for online action detection," in *2018 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2018, pp. 1549–1557.

[14] M. Xu, M. Gao, Y.-T. Chen, L. S. Davis, and D. J. Crandall, "Temporal recurrent networks for online action detection," in *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[15] Y. Huang, Y. Sugano, and Y. Sato, "Improving action segmentation via graph-based temporal reasoning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 14 024–14 034.

[16] X. Wang, S. Zhang, Z. Qing, Y. Shao, Z. Zuo, C. Gao, and N. Sang, "Oadtr: Online action detection with transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 7565–7575.

[17] Y. Wang, M. Li, H. Cai, W.-M. Chen, and S. Han, "Lite pose: Efficient architecture design for 2d human pose estimation," *arXiv preprint arXiv:2205.01271*, 2022.

[18] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 466–481.

[19] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.

[20] A. Gu, K. Goel, A. Gupta, and C. Ré, "On the parameterization and initialization of diagonal state space models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 35 971–35 983, 2022.

[21] J. T. Smith, A. Warrington, and S. W. Linderman, "Simplified state space layers for sequence modeling," *arXiv preprint arXiv:2208.04933*, 2022.

[22] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[23] G. E. Blelloch, "Prefix sums and their applications," 1990.

[24] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *International conference on machine learning*. PMLR, 2017, pp. 933–941.

[25] Y. A. Farha and J. Gall, "Ms-tcn: Multi-stage temporal convolutional network for action segmentation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 3575–3584.

[26] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*. Pmlr, 2013, pp. 1310–1318.

[27] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, pp. 462–466, 1952.

[28] J. Lin, C. Gan, and S. Han, "Tsm: Temporal shift module for efficient video understanding," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 7083–7093.