INP - ENSEEIHT

End of Studies Internship Report

Using Transformers for Short Term Action Anticipation on Egocentric Videos

---

Gouneau Joceran
Sciences du Numérique
ENSEEIHT, Toulouse INP
Toulouse, France
joceran.gouneau@etu.toulouse-inp.fr

Ng Lai Xing (supervisor)
Institute for Infocomm Research
*A*STAR*
Singapore, Singapore
ng_lai_xing@i2r.a-star.edu.sg

21 March 2023 — 11 September 2023

Contents

1 Acknowledgements

I would like to thank every person that has contributed to the completion of this internship.

I would especially like to thank my tutor Ng Lai Xing, for his expertise, his benevolence, and his good spirit, as well as my teacher Axel Carlier and Christophe Jouffrais, to put me in contact with IPAL and A*STAR, without whom this internship would not have been possible.

Finally I am grateful to all the staff members for their kindness, their constant good mood, as well as their good tips and advice to live in Singapore.

2 Introduction

2.1 The Company

The Agency for Science, Technology and Research (A*STAR) is a national laboratory under the Ministry of Trade and Industry of Singapore ; as such it supports research and development that enhances the economic growth and technological advantage of Singapore, as well as advancing scientific discovery and technological innovation. This laboratory is divided into two Research Councils : the Biomedical Research Council (BMRC) and the Science and Engineering Research Council (SERC) ; my internship took place at the I^2R institute, one of the 8 Research Institutes SERC is composed of :

- Advanced Remanufacturing and Technology Centre (ARTC);
- Institute of Sustainability for Chemicals, Energy and Environment ($ISCE^2$);
- Institute of High Performance Computing (IHPC);
- Institute for Infocomm Research (I^2R);
- Institute of Materials Research & Engineering (IMRE);
- Institute of Microelectronics (IME);
- National Metrology Centre (NMC);
- Singapore Institute of Manufacturing Technology (SIMTech);

I^2R aims at fueling innovation in information technologies and its applications ; one of its search directions is toward human-in-the-loop systems, in which information systems directly assist humans in tasks.

2.2 The Problem

A concrete application of this work is embodied by the field of Augmented Reality, in which the user is provided more information on its environment, as well as virtual instances superposed to real objects with which he can interact.
One of the goals at the HFE Lab, within I^2R, is toward building a task assistant agent that can help the user know what are the next steps and actions in his task, where in the environment are the objects and tools he needs to complete an operation, or establish a detailed state of the task that can be sent in real time to a remote supervisor. This system has only access to the view of a head mounted camera that captures a 1st person view video stream, to the manual or the description of the task at hand, and to the device through which it can output information to the user – in our case Microsoft HOLO Lens.
This system would be on the border of multiple and various fields, such as a Natural Language Processing to understand the task description, or human-machine interfaces to interact with the user, but my internship focused on the computer vision part of extracting the information on the user's environment through the 1st person view video stream.
Computer Vision have been transformed in the recent years by Artificial Intelligence (AI) and especially Deep Learning (DL) ; these systems learn to excel at specified tasks by training on

large amounts of example data. To the specific task of understanding a user's environment through a 1st person view video is designed a dataset and set of benchmarks called Ego4D that we present in more details in the related work for this project (3.1).
My internship consisted of first choosing a task within the benchmarks proposed by Ego4D I would focus on, understand it as well as the state-of-the-art on it, find an improvement to the existing solutions, implement it, and test it.

3 Related Work

The first part of the internship (21 March - 24 April) was planned to be done in a remote setup before going on site and working at the lab, this was not a problem as I had to focus during that time on having an overview of the literature on the field of egocentric video processing and problems, which does not require a lot of computation resources or a teamwork setup more advanced than regulars video meetings. Two papers stood out as starting points:

- the original Ego4D [7] paper, in which the dataset and all the challenges related to it are defined, as well as baseline models for each challenge;
- the take of InternVideo [3] on 5 challenges of the Ego4D dataset;

Workflow: these two papers pointed to and referenced ideas and model architectures used in state-of-the-art (SOTA) applications of deep learning to video and image processing ; in order to keep track of all these concepts (most of which were new to me) I took notes in *Obsidian*, a markdown text editor that allows, within a note, to reference another note (see Fig. 2 where the *Transformer* note is referenced in the note on *Positional Encoding*) building at the same time a graph (Fig. 1) of all the notes and their links, easing the process of locating new concepts within the already explored literature and know what is yet to read.

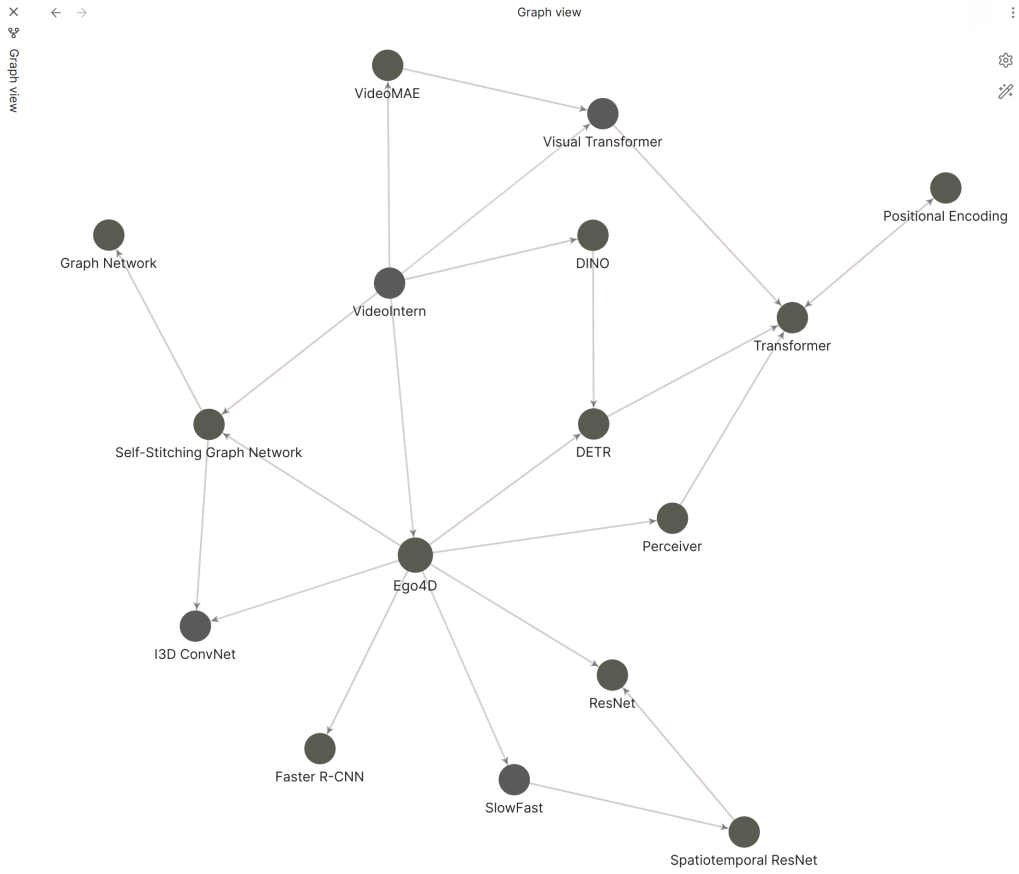

Figure 1: Obsidian's Graph View

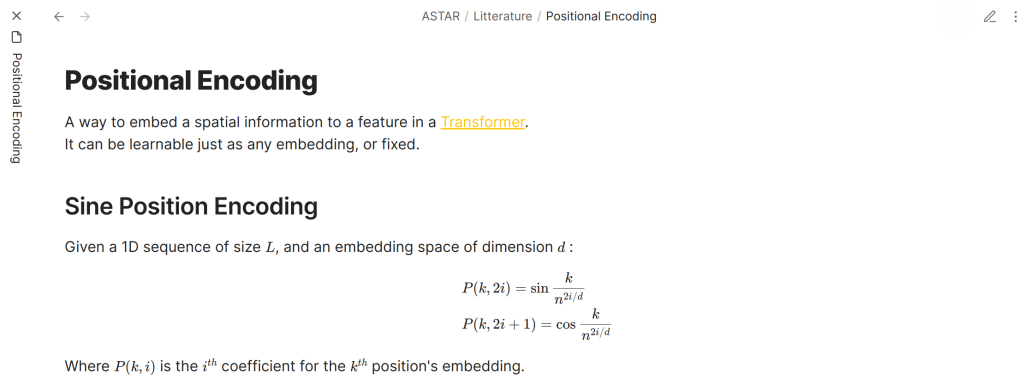

Figure 2: An Obsidian Note

3.1 Ego4D

3.1.1 Overview

Ego4D [7] is a 7.1 TB dataset containing over than 3,670 hours of daily life activity egocentric videos, covering a wide range of activities such as walking on the street, cooking or bike mechanic ; it has been collected by 923 unique participants from 74 worldwide locations in 9 different countries, using seven different cameras such as GoPro, Vuzix Blade or Pupil Labs. As such, it constitutes a high quality basis for many egocentric video processing applications. Two versions of this dataset exist : V1, the original one, and V2, with added data and cleaner annotations. On top of this data are defined a set of 14 challenges grouped within 5 more general problems areas :

- **Episodic Memory**: retrieving instances of objects the user interacted with in the past, the localization of this interaction in time or of the object in space;
- **Hands and Objects**: when, where and with what the user's hands interact, how the objects interacted with are changed;
- **Audio-Visual Diarization**: transcribing conversations, tracking the speakers on the video and knowing who said what;
- **Social Understanding**: detecting who is looking at or talking to the user, capturing non-verbal cues and detecting who is attending to whom;
- **Forecasting**: predicting future interactions of the users with its environment on a short term, or the list of its future actions on the long term;

I had to choose, within the Episodic Memory, Hands and Objects, and Forecasting challenges, one problem on which I would dedicate the rest of my internship ; after understanding the definition of each problem and of the baseline solution the original publisher had came up with for each of these, I chose to work on the Short-Term Action Anticipation challenge, within the Forecasting group.

3.1.2 Short-Term Action Anticipation

Definition

The Short-Term Action Anticipation (STA) problem is defined as follows: given an input video $V_{:t}$ up to a timestamp t that we will refer as stopping time, the goal is to detect on the last frame V_t a set of N objects the user will interact with in a short time (0s to 7s.) after t ; for each of these objects the model must also predict in which action the object will take part as well as an estimate of how many seconds after t the interaction with the object will begin. So, for a given input video $V_{:t}$, the model must predict N tuples (where N is arbitrary):

$$\left\{\left(\hat{b}_i, \hat{n}_i, \hat{v}_i, \hat{\delta}_i, \hat{s}_i\right)\right\}_{i=1}^{N} \tag{1}$$

where :

- $\hat{b}_i \in \mathbb{R}^4$ is a bounding box localizing the object on the last frame $V_{:t}$;
- $\hat{n}_i \in \mathcal{N}$ is a *noun* depicting the object, where $\mathcal{N}$ is the set of all nouns objects can be classified into;
- $\hat{v}_i \in \mathcal{V}$ is a *verb* depicting the action, where $\mathcal{V}$ is the set of all verbs actions can be classified into;
- $\hat{\delta}_i \in \mathbb{R}^+$ is the time at which the interaction with this object will begin, taking the stopping time t as origin;
- $\hat{s}_i \in [0, 1]$ is a confidence score used for evaluation;

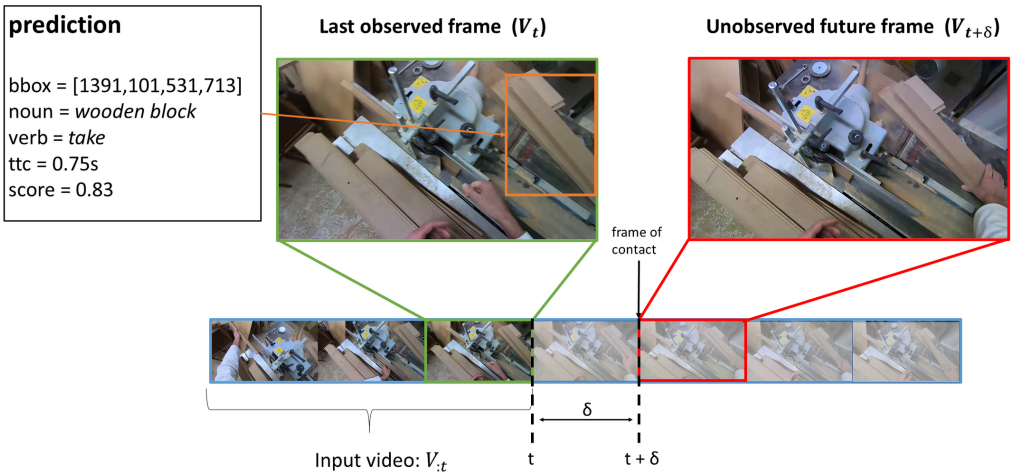

Figure 3: Example of short-term object interaction anticipation. F. Ragusa et al. [11].

Training

During training the object labels and bounding boxes predictions are trained using existing methods for networks for detection ; for the rest, these detections are used to attribute to each prediction its closest ground truth, against which it is evaluated with cross-entropy on the verb classes and smooth l1 distance on the ttc. The distance used to attribute a ground truth is the

intersection over union (IOU) of bounding boxes (see Fig. 4). All the attributed ground truths of every example in a training batch are then concatenated to form a single list like :

$$\{\mathrm{GT}_{ij_i}\}_{i\in[1,B],j_i\in[1,n_{\mathrm{obj},i}]}$$

where GT_{ij_i} corresponds to the j_ith object detected in the ith example of the batch, for which we have $n_{\mathrm{obj},i}$ detections.
The subset used for training is the *training* subset.

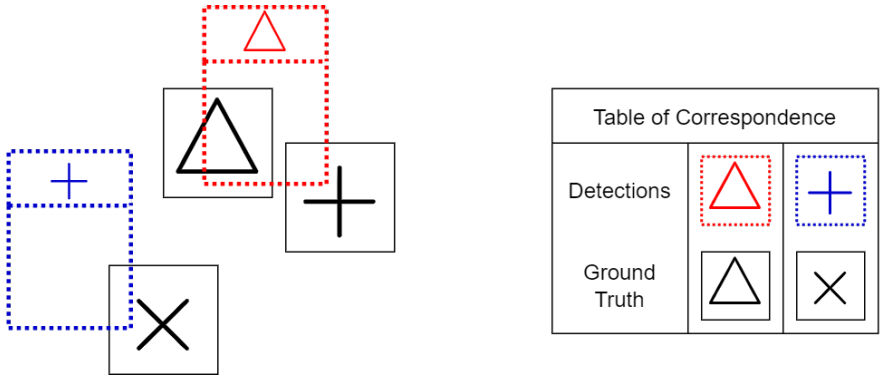

Figure 4: Attribution of a ground truth to each detection.

Evaluation
The metric used for evaluating and comparing models on this challenge is the Top-K mAP : to account for the fact that more than one next active object can be likely, the $(K-1)*T$ false positives with the highest confidence scores (K being a parameter of evaluation and T the number of ground truth example on this example) are ignored to compute the mAP (see E). Each ground truth is associated a unique prediction, this association is based on IOU and is done in decreasing confidence order.
We define different mAP criterions, for defining what a correct prediction is, in order to have a better grasp of the model performances, for each example, given a prediction i and an annotation j, we consider to have a correct prediction when the following conditions are satisfied :

- for **Box + Noun mAP**:
$$\begin{cases} IOU(\hat{b}_i, b_j) > 0.5 \\ \hat{n}_i = n_j \end{cases}$$
- for **Box + Noun + Verb mAP**:
$$\begin{cases} IOU(\hat{b}_i, b_j) > 0.5 \\ \hat{n}_i = n_j \\ \hat{v}_i = v_j \end{cases}$$
- for **Box + Noun + TTC mAP**:
$$\begin{cases} IOU(\hat{b}_i, b_j) > 0.5 \\ \hat{n}_i = n_j \\ |\hat{\delta}_i - \delta_j| < T_\delta \end{cases}$$

- for **Box + Noun + Verb + TTC mAP**, also refered to as **Overall mAP**:

$$\begin{cases} IOU(\hat{b}_i, b_j) > 0.5 \\ \hat{n}_i = n_j \\ \hat{v}_i = v_j \\ |\hat{\delta}_i - \delta_j| < T_\delta \end{cases}$$

where T_δ is a parameter of evaluation. The values used for evaluation are $K = 5$ and $T_\delta = 0.25$. We can deduce the following dependencies between these different metrics :

$$\begin{cases} \mathbf{mAP}_{\text{Box, Noun}} \geq \mathbf{mAP}_{\text{Box, Noun, Verb}} \geq \mathbf{mAP}_{\text{Overall}} \\ \mathbf{mAP}_{\text{Box, Noun}} \geq \mathbf{mAP}_{\text{Box, Noun, TTC}} \geq \mathbf{mAP}_{\text{Overall}} \\ \mathbf{mAP}_{\text{Overall}} \geq \max\left(0, \mathbf{mAP}_{\text{Box, Noun, Verb}} + \mathbf{mAP}_{\text{Box, Noun, TTC}} - \mathbf{mAP}_{\text{Box, Noun}}\right) \end{cases}$$

The subsets used for evaluation are the *validation* and *test* subsets. Annotations are provided for the validation subset, which allows to verify the performance of a model on data it has never seen ; the test set, on the other hand, has only the input data to generate predictions on, the annotations these are compared to in order to obtain the performance of the model on this set are kept secret by Ego4D : the only way to evaluate a model on the test set is to upload its predictions on the dedicated server, where the evaluation process is performed. $\mathbf{mAP}_{\text{Overall}}$ is the metric used to score submissions for the challenge.

3.2 Existing Approaches

To answer this challenge were designed various solutions, at the time of the literature review two stood out, which are summarized in Table. 1 :

- the baseline model developed by Ego4D, which was the first model developed (*Baseline V1*) but also the best one on the challenge once trained on the 2nd version of dataset (*Baseline V2*);
- the model developed by InternVideo (*InternVideo*), which had the best results before the 2nd version of the dataset came out, using the same video backbone they used to solve other Ego4D problems;

Model Name	Object Detector	Video Backbone	Dataset Version	$\mathbf{mAP}_{\text{Overall}}$
Baseline V1	Faster RCNN	SlowFast	V1	2.45
InternVideo	DINO DETR	VideoMAE	V1	3.40
Baseline V2	Faster RCNN	SlowFast	V2	3.61

Table 1: Existing approaches summary and their score on the benchmark. Video Backbone refers to what we call the video features extractor later.

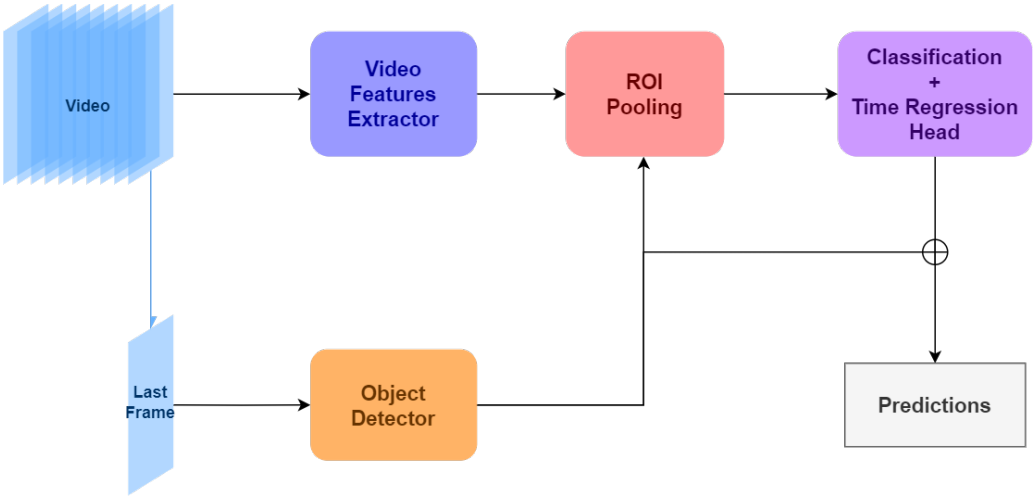

Figure 5: General architecture of the existing solutions.

Before going into details in the specific features of these models, we can highlight a similarity they share in their general architecture (see Fig. 5). Both of these models start off using an object detector finetuned on the forecasting data (on the objects to detect on last frames V_t of each clip) to produce active objects detections. These detections are then used as Regions Of Interest (ROI) in a ROI Pooling (see A) on the features extracted from the input video $V_{:t}$ by a video features extractor also finetuned on the forecasting data (on the video clips). The features pooled for each object are then processed by two Feed Forward Networks (FFN) to predict a probability distribution over verbs using softmax activation (Eq. 2) for the first one and a positive quantity for time to contact (TTC) prediction using a softplus activation (Eq. 3) for the second, these verb and TTC are then concatenated to their respective object detection to obtain the desired list of tuples from Eq. 1. During training the object detector is frozen.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}, \qquad \text{hence: } \sum_i \text{Softmax}(x_i) = 1 \tag{2}$$

$$\text{Softplus}(x) = \frac{1}{\beta} * \log(1 + \exp(\beta * x)), \qquad \text{hence: Softplus}(x) > 0 \tag{3}$$

Note: for the ROI Pooling to work it is important that the video features have a spatial dependency, otherwise the notion of *regions* on these features makes no sense. The choice of video features extractor is as such restricted to those that preserve this dependency.

3.2.1 Ego4D's Baseline

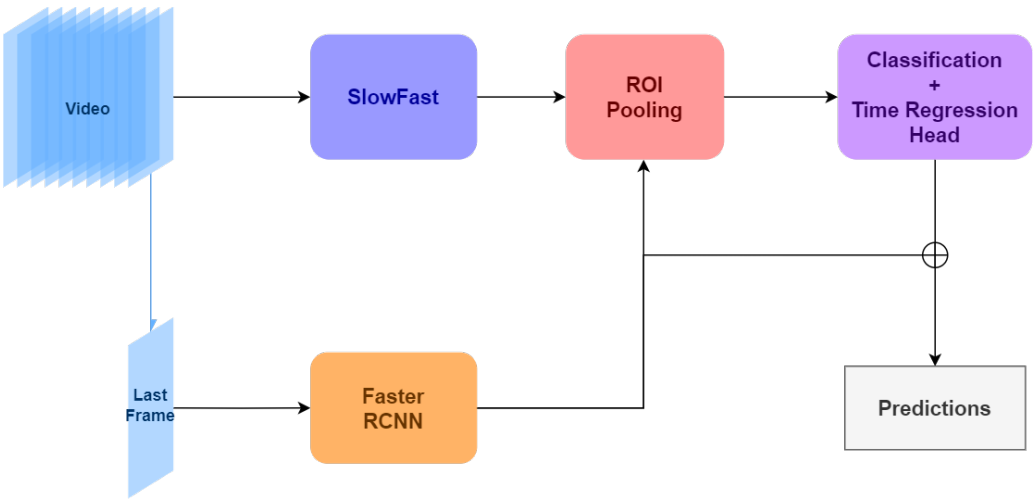

Figure 6: Architecture of the Baseline model.

The Baseline model uses Faster RCNN [12] (see B) to generate object detections and SlowFast [5] to extract video features ; no modifications are made to the general architecture shown in Fig. 5. In Table 1, the only difference between *Baseline V1* and *Baseline V2* is the data used for training, hence showing the improvement in quality of the Ego4D dataset between V1 and V2.

SlowFast

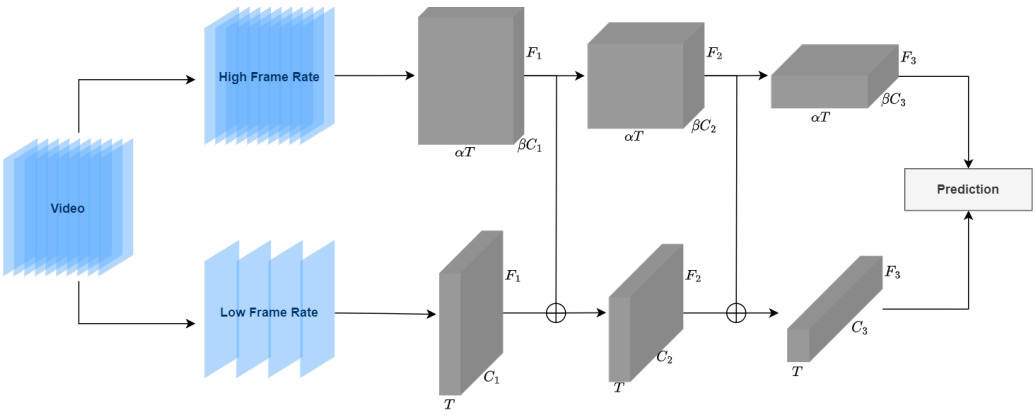

Figure 7: Architecture of a SlowFast network. F represents the spacial dimension ($F_1 = HW$ with H and W being the resolution of the input frames).

SlowFast [5] is an architecture for video processing in which the network is divided in two pathways : the slow pathway and fast pathway, both of which can be any type of 3D convolutional network:

- The slow pathway operates at low frame rate T but with a high channel dimension C, its role is to capture the structure of the scene in the video;

- The fast pathway operates at high frame rate αT (with $\alpha = 4$ in our case) and a small number of channels βC (with $\beta = 1/8$ in our case) in order to capture the motion information;

Lateral connections are also added, at different layers of the network, from the fast pathway to the slow pathway, for the slow pathway to also take into account the motion information extracted by the fast pathway ; these connections consist of a reshaping operation for the shapes to match, followed by addition or concatenation. The outputs of both pathways, which are used for prediction in the original case, are then used for ROI Pooling in the Baseline model.

3.2.2 InternVideo's Approach

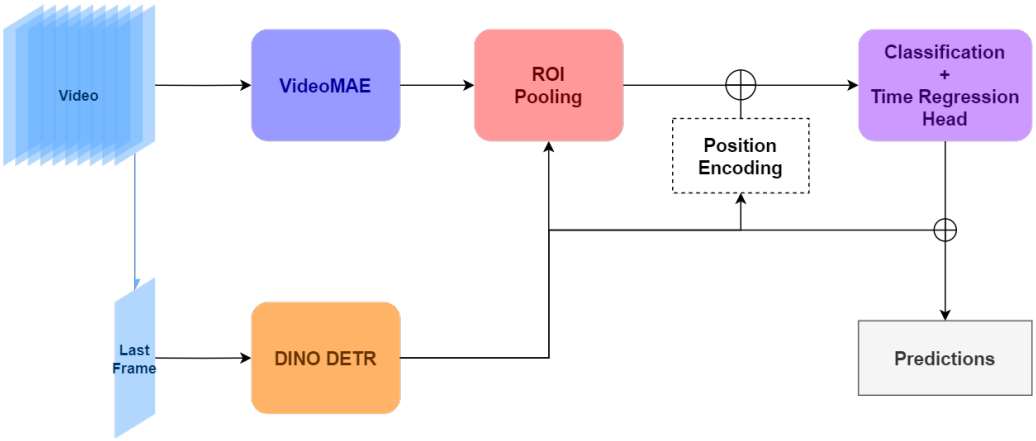

Figure 8: Architecture of the InternVideo model.

The InternVideo model uses DINO [16], an improved version of DETR (see 3.3), to generate object detections and the VideoMAE [14] Video Transformer to extract video features ; a position encoding (D) of each box is also added to its respective pooled feature with the aim to have more precision on the TTC prediction.

VideoMAE

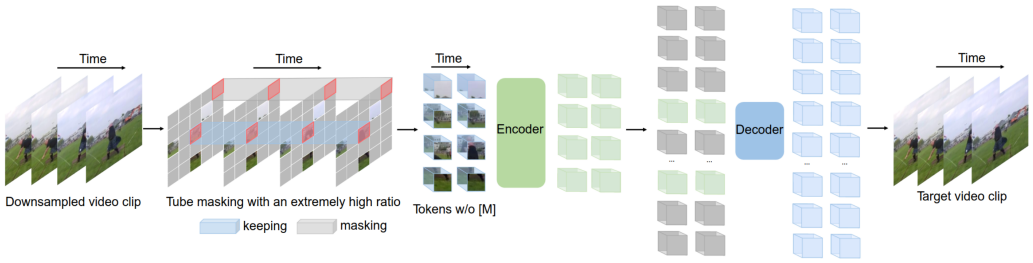

Figure 9: The VideoMAE model during training. Z. Tong et al. [14]

.

VideoMAE [14] is an implementation of a Video Transformer [1, 10]. This type of model tokenizes the video by first dividing it into 3D patches of size $(T_{\text{patch}}, H_{\text{patch}}, W_{\text{patch}}) = (2, 16, 16)$ (thus giving a total of $\frac{T}{2} \times \frac{H}{16} \times \frac{W}{16}$ patches, (T, H, W) being the size of the input video) and

applying to each a linear projection in order to project them in an embedding space of dimension d. These tokens are then fed to a Transformer [15] (see C) along with their space-time position in the video, added as a position encoding (D). The term *MAE* stands for *Masked Auto Encoders* and refers to the training process described in Fig. 9 where tubelets of tokens (tokens sharing the same spatial position but not the same temporal position) are masked, only the remaining ones being fed to the Transformer encoder ; the role of the Transformer decoder is to then reconstruct the video. In the InternVideo model, the video feature extractor we refer to is only the encoder part of a pre-trained VideoMAE model, of which the decoder was discarded ; the features returned by this encoder consist of a video feature token for each patch position $\frac{T}{2} \times \frac{H}{16} \times \frac{W}{16}$ in the input video.

3.3 Transformers for Detection

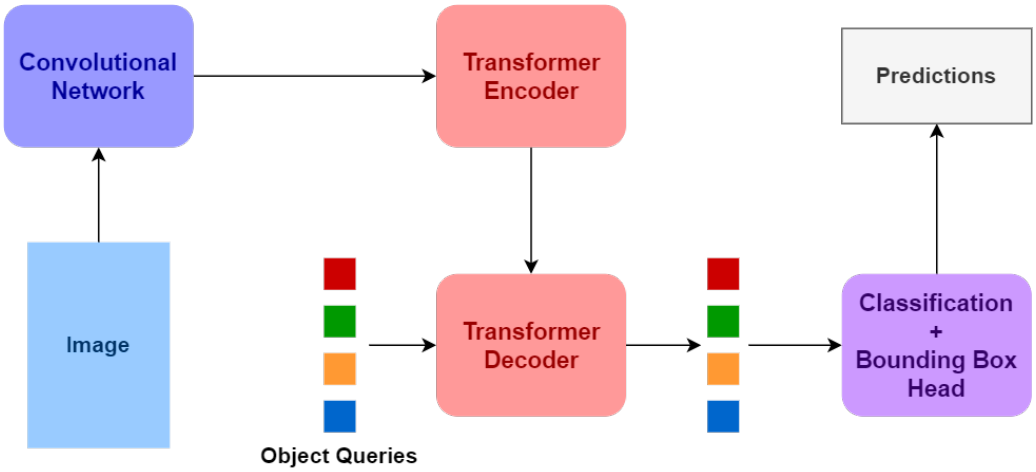

Figure 10: The DETR model.

DEtection TRansformer (DETR) [2] is a Transformer [15] (see C) based object detection model. Where a more classic approach like Faster RCNN [12] would use a ROI Pooling fed by Region Proposals to extract a feature for each object (see B), DETR uses a Transformer to decode N *Object Queries* (N being a fixed number, corresponding to the maximum number of objects detectable by the model), which are simply learnable embeddings, using the information encoded from the image features. The goal is, after decoding, that each object query refers either to a *unique* object on the image or to nothing ; these queries are finally fed to a FFN to predict a bounding box and a class (with a specific class $\varnothing$ for when the query refers to nothing). This approach achieves comparable results to Faster RCNN on the COCO dataset [9].

4 Method

By the end of my literature review (24 April), I had to come up with an idea to improve the existing solutions to this problem. My idea was to perform the same kind of change in perspective that was done with DETR on the problem of object detection on images, by using a Transformer (see C) in place of a ROI Pooling.

4.1 ROI Pooling Transformer

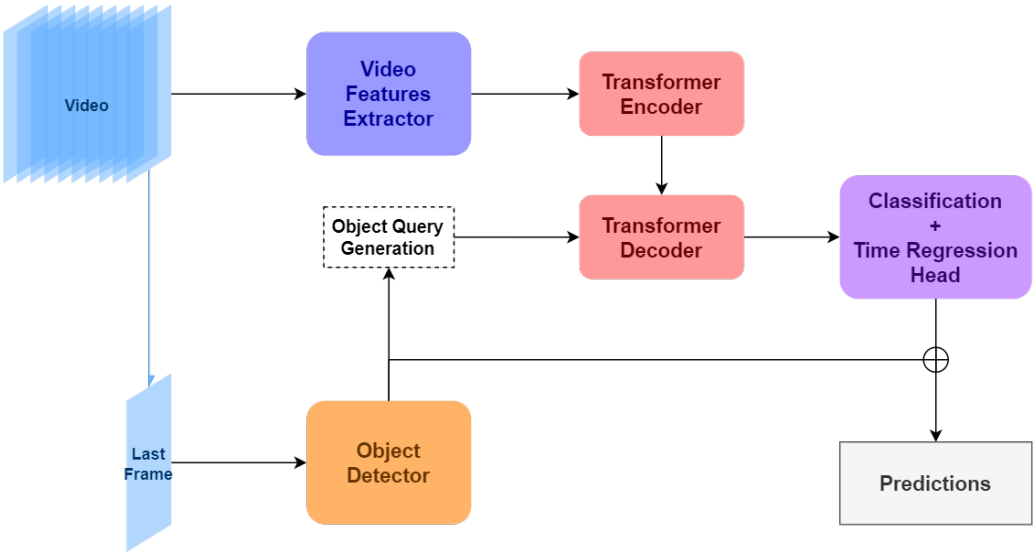

Figure 11: Model overview.

I decided to keep the same 2-phases process – extracting video features with a video backbone on one hand and producing object detections on the last frame with an object detector on the other hand – as is done in the existing solutions in order to have the most precise bounding boxes, as these must match the positions of the future active objects *on the last frame*. Where my solution differs is in the way these object detections and video features are used together to predict the actions and TTC of these objects ; following the example of DETR, I chose, instead of a ROI Pooling operation, to use a Transformer (see C), whose role is to decode object queries using the information from the encoded video features (see Fig. 11), these decoded object queries are then, like in the original solutions, fed to a FFN to predict a verb class and a time to contact. Rather than using learnable embeddings as object queries like is done in the original DETR model, I use the information from the object detections to generate them (see 4.2) ; doing so each object query corresponds to an object detected by the detector, which eases the operation of merging the predicted verbs and TTC to their corresponding object detection. Because our solution only affects the ROI Pooling operation, we can use any object detector and video backbone. To be more specific we will, through our experiments, use :

- for the **Object Detector:** the Faster-RCNN from the *Baseline V2* model (3.2.1) for simplicity, as Ego4D furnishes its detections on the STA subset within the data;
- for the **Video Backbone:**

- Omnivore (Swin L) [6] in 5.1;
- SlowFast [5] in 5.4;
- VideoMAE [14] in 5.5;

Note: by object queries we refer to the tokens which constitute the input sequence of the Transformer decoder, not the queries used in the attention operations (the Q in Attention(Q, K, V)) performed within the so-told Transformer.

4.2 Generating Object Queries

The rest of the architecture being unchanged and Transformers being already well studied and widely used on various types of data, the only part that needed more specification when I came up with this solution was the generation of object queries from the detections made by the detector.

4.2.1 Information Contained Within the Queries

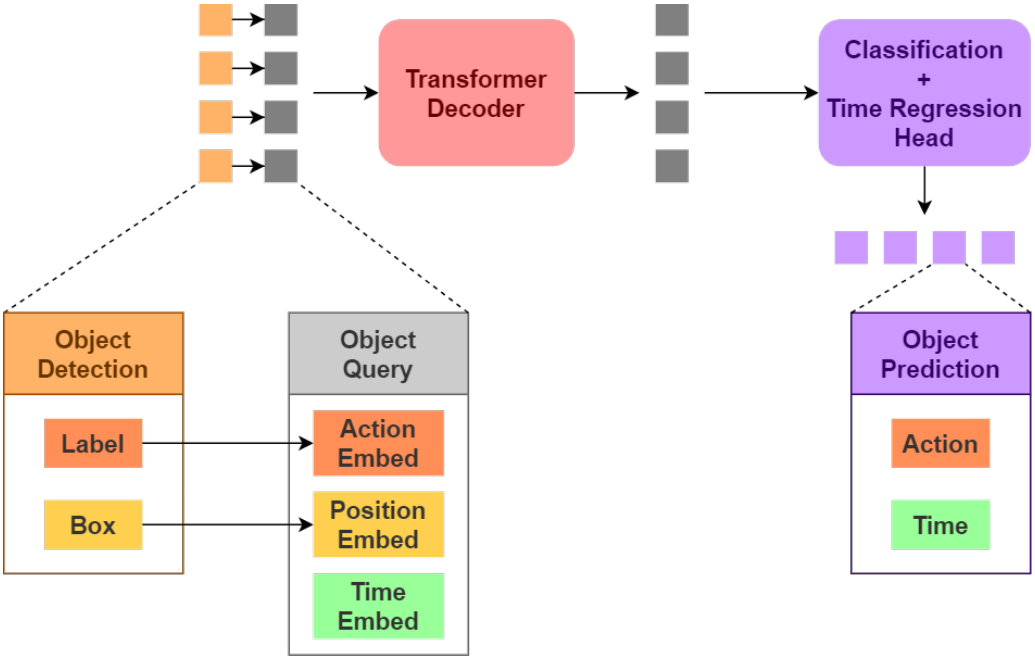

Figure 12: The Information within Object Queries.

Starting with the purpose of these object queries, we know that each one of them must contain, *after* decoding, the information of the action its associated object will take part into, as well as of the time this action will happen in the future ; so we would like to have this information initialized before decoding, as an action embedding and time embedding. Also, looking at what these object queries are associated to – object detections – we know that each object query *before* decoding must encapsulate the identity of the object it refers to, in order to be distinguishable from one another so that the decoder does not mix them up. To do so we use the bounding box information to create a spacial position embedding as a fixed position encoding (D), and the object label as a noun embedding with an embedding learned for each

noun ; intuitively an object should be used in a certain range of actions it is associated to, so we actually use this noun embedding as the initialization of the action embedding we referred earlier. We have no prior information on the time to contact from detections alone to initialize the time embedding, so it is left out as a fixed position encoding (D) or learnable embedding.

Note: one might notice that the idea from the InternVideo model of adding the box position information of each object detection to its corresponding feature before the Classification + Time Regression Head does not need to be re-implemented here, as this information is already present (and transformed) as a position embedding in each object query.

4.2.2 Merging the Information

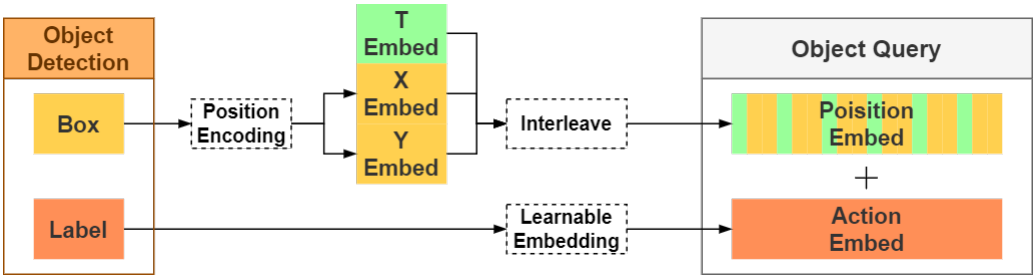

Figure 13: Object Queries Generation.

Finally it must be decided on the way these 3 embeddings (*Action*, *Position*, and *Time*) must be merged to create one object query ; it has to be known that these embeddings are 1D vectors of free and independent dimensions and that an object query must be a 1D vector whose dimension is the same as the token dimension of the Transformer. The first idea was to concatenate everything, but it does not follow the way it is done in the literature and generates queries of higher dimensionality, which requires the token size d_{token} of the Transformer to be just as big ; this is bad for computation because the attention operations that happen in a Transformer are of (simplified) complexity $\mathcal{O}(d_{\text{token}}^2)$. The second idea was to follow the way it is done in the literature which is to have a single embedding that represents a query and add to it a position embedding ; we notice how the *Position* and *Time* embeddings both capture information on the position of the object (spatial for the first and temporal for the second) while the *Action* embedding actually represents the nature of the object. As such we divide the *Position* embedding into an X and Y embeddings from the bounding box position and merge them to the T (*Time*) embedding to obtain a spatio-temporal position embedding P, these three embeddings are merged in an interleaved manner :

$$P \in \mathbb{R}^{d_{\text{tok}}}, \quad T \in \mathbb{R}^{d_{\text{token}}/3}, \quad X \in \mathbb{R}^{d_{\text{token}}/3}, \quad Y \in \mathbb{R}^{d_{\text{token}}/3}$$
$$\begin{cases} P_{3i-2} = T_i & \forall i \in [1, d_{\text{token}}/3] \\ P_{3i-1} = X_i & \forall i \in [1, d_{\text{token}}/3] \\ P_{3i} = Y_i & \forall i \in [1, d_{\text{token}}/3] \end{cases} \tag{4}$$

We require, as such, that $d_{\text{token}}|3$ and that each of X, Y and T are of the same size $d_{\text{token}}/3$. The final object query is simply the sum of this position embedding P to the *Action* embedding, which must then be of size d_{token}.

4.3 Choice of the Video Features Extractor

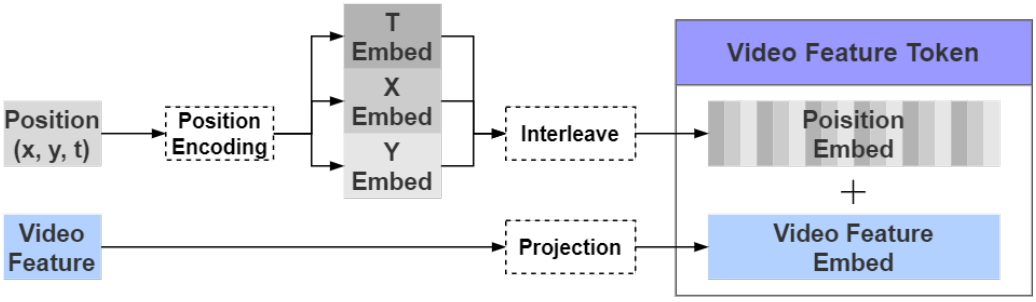

Figure 14: Video Feature Token Generation.

Any video backbone model could be used for video features extraction, as long as these features can be divided into different tokens in a way that makes sense for feeding it to the Transformer encoder, but we preferred the choice of one that keeps the spatial dependency of the input in its output – with the exception on experiment 5.1 in which the video features were provided. This way each feature has a corresponding spatio-temporal region (x_V, y_V, t_V) in the video matching that of the object queries $(x_{\text{box}}, y_{\text{box}}, t_{\text{init}})$; this makes it a more direct way for the Transformer to know where to focus its attention in the video features for each object query. We follow the same process of position encoding (D) for each position component followed by interleaving them to generate the position embedding associated to a video feature (see Fig. 14). We also apply a learnable projection on the video feature in the same way it is done on the input of Visual Transformers [1, 10] to generate a corresponding video feature embedding ; this also allows to project a video feature of any dimension d_{feat} to the token dimension d_{token} of the Transformer.

5 Experiments

5.1 A Simplified Notebook Example

I was supposed, on the 24th of April, to go to Singapore to start the on-site part of the internship. Unfortunately, due to a delay in the approval of my work pass, I could not enter the country for work purpose ; due to the lack of information on the nature of this delay the remote setup was extended of an indefinite amount of time. Despite this delicate situation – no access to the dataset on the lab machines, no access to computation resources – I could start experimenting at a small scale thanks to a Jupyter Notebook example on the STA task furnished by Ego4D.

Workflow: I executed and edited this notebook on *Google Colaboratory* (Colab), a hosted Jupyter Notebook service that provides access to computing resources, which I was lacking at the time. It also allows to share notebooks with other users, but to this sole purpose my tutor also created a repository on *Bitbucket*, a Git-based source code repository hosting service, to keep the progress I made on my scripts updated through Git.

5.1.1 Notebook Description

This notebook serves as a quickstart to the STA task, it covers a simplified training loop for this problem by utilizing the pre-generated bounding boxes of the Faster RCNN object detector from the Baseline model (3.2.1) as well as video features – much lighter to download than the original dataset – pre-extracted by an Omnivore (Swin L) [6] model ; it is not an end-to-end model as such, as the object detector and video backbone are frozen, only the ROI Pooling head is trained in the loop. This ROI Pooling head is also simplified for the training time to be reasonable on a Colab notebook ; it is not a real ROI Pooling operation, as the raw box position information is simply directly concatenated to the whole video features for each object detection before the FFN layers for verb and TTC predictions. We will refer to this original model proposed by Ego4D for this quickstart as the *Notebook Baseline*.
The training and validation annotations, as well as the evaluation process, are the same as those that are used in the original problem.

5.1.2 First Implementation

This notebook served as a sandbox environment for my first experiments ; I also had, before even experimenting anything, to first get familiar with Pytorch, as I had only been using Tensorflow at school or in previous projects.
Once I was comfortable enough I could start building the implementation of my ROI Pooling Transformer idea, which simply came in place of the ROI Pooling Head defined in the notebook. Most of the code that would then be used in the following experiments was written at this part of the project, with the exception made to the video feature token generator (4.3), because the video features produced by the Omnivore model are independent of spatial position in the input video ; they only depend on time, a single feature being produced for each 16 frames of the input. The encoder received the features as such, without any projection made and with only a time encoding added to localize each token in time.

5.1.3 Packing and Unpacking Data

Data in Different Shapes
During training, the model is fed batches of data, each batch is basically a certain fixed number B of training examples ; the major difficulty with this implementation was the handling of data in a *batched* manner, as inputs and outputs are packed differently :

- for **video features** each example in the batch being of the same shape $\text{Shape}_{V,\text{feats}}$, a video features patch is packed in a single tensor of shape $B \times \text{Shape}_{V,\text{feats}}$;
- for **object detections** each example in the batch is not of the same length, as each example does not necessarily have the same number of object detected, and as such :
  - a batch of object nouns is packed in an ensemble $\{N_i\}_{i=1}^{B}$, where $N_i \in \mathbb{R}^{n_{\text{obj},i}}, \forall i \in [1, B]$;
  - a batch of object boxes coordinates is packed in an ensemble $\{\text{Box}_i\}_{i=1}^{B}$, where $\text{Box}_i \in \mathbb{R}^{n_{\text{obj},i}} \times \mathbb{R}^4, \forall i \in [1, B]$;

  where $n_{\text{obj},i}$ denotes the number of objects detected for example i, and can be equal to 0 in the case where no objects were detected;
- for **verb labels and TTC targets**, against which are evaluated the predictions made by the network during training, we have a single long list of ground truths per batch (see 3.1.2), which gives :
  - a batch of verb labels is a tensor of shape $(n_{\text{obj,tot}}, N_{\text{verbs}})$;
  - a batch of time to contact targets is a 1D tensor of dimension $n_{\text{obj,tot}}$;

  where $n_{\text{obj,tot}} = \sum_{i=1}^{B} n_{\text{obj},i}$ is the total number of detections in the batch;

Unfortunately the Transformer implementation of Pytorch only accepts sequences of same length for each example in a batch ; for the video features this was not a problem as they are all of the same size and generate as such the same number of video feature tokens for each example, but for the object detections this was a problem, as different numbers of objects give different numbers of object queries for each example in a batch.

Packing Data
To address this problem, we want to generate *dummy* object detections for the examples that lack objects, so that every example in a given batch has the same number $n_{\text{obj,max}} = \max_{i \in [1,B]} n_{\text{obj},i}$ of objects detections ; a *dummy* bounding box will have all its coordinates set to 0, while a *dummy* noun label will be the first noun in $\mathcal{N}$. In order to prevent the *dummy* object queries that are generated from these *dummy* object detections to have an impact on the *real* object queries in the decoding process, we also generate an attention mask, that tells the decoder which queries not to look at. This process is described in the step 3 of Fig. 15.
It was found later on that *empty* examples (for which there are no object detections) created instabilities in the learning process ; to address this we also remove every *empty* example from the batch (step 2 on Fig. 15), giving a new batch with B' examples.

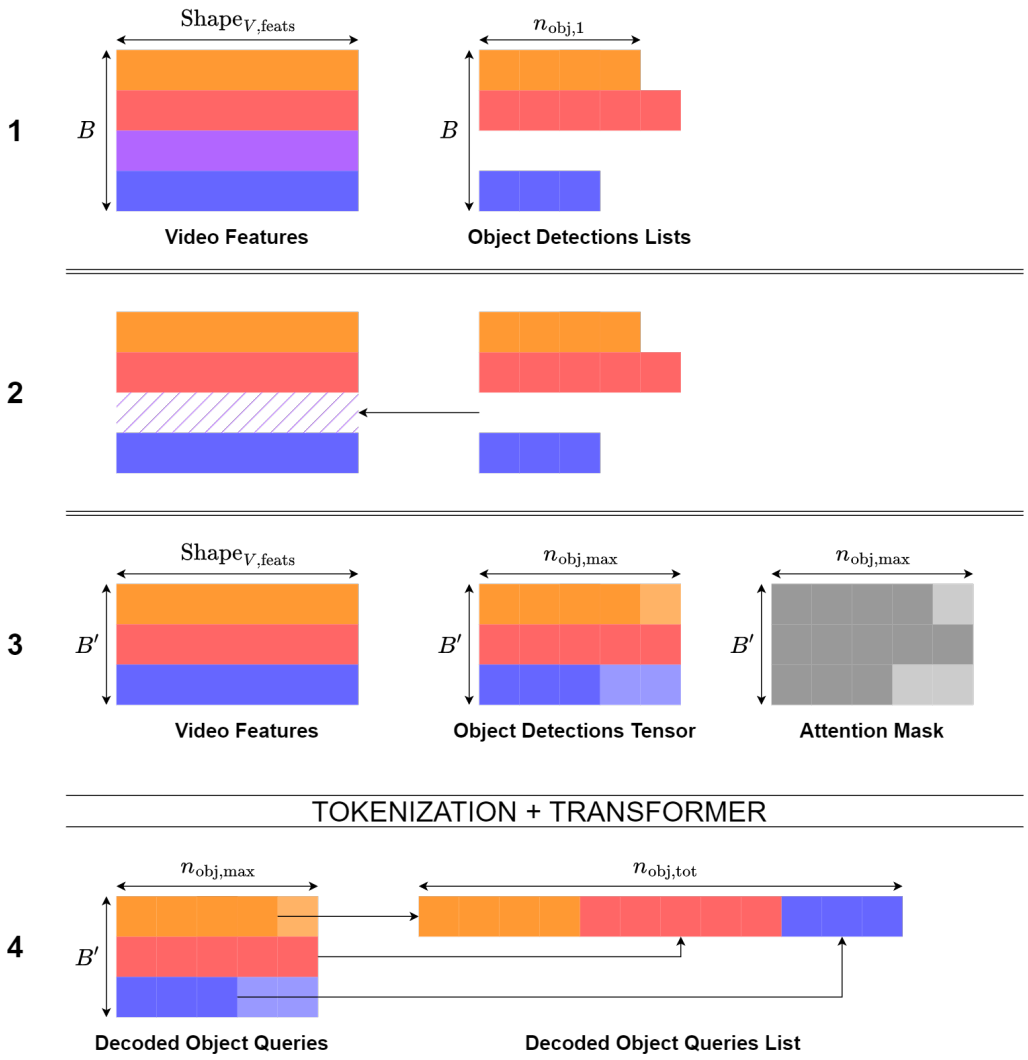

Figure 15: The data packing and un-packing processes.

Un-packing Data

After packing the input data like so we can tokenize it, following the object queries generation process (4.2) and video features tokenization process (4.3) and feed them to the transformer. What we obtain is a decoded object queries (DOQ) tensor of shape $(B', n_{\text{obj,max}}, d_{\text{token}})$; we want to reshape it into a long list of all the object queries of this batch, in order to match the shape of the labels it will be evaluated against after verbs and TTC extraction by the FFN layers at the end of the network. To do so we first discard the decoded *dummy* queries, and then put all the queries of each example one example after another to obtain a list :

$$\left\{\text{DOQ}_{ij_i}\right\}_{i\in[1,B'],j_i\in[1,n_{\text{obj},i}]}$$

where DOQ_{ij_i} corresponds to the j_ith object detected in the ith non-*empty* example of the batch. This process corresponds to the 4th step on Fig. 15.

Results

Because we used the raw video features without reshaping them, the token dimension of the

Transformer had to be the same as the video features dimension, which is 1536 ; the other parameters were chosen not too large so that the training could be performed in a reasonable time on a Colab notebook :

Parameter	Value
d_{token}	1536
n_{head}	6
$n_{\text{encoder layers}}$	4
$n_{\text{decoder layers}}$	4
$d_{\text{feedforward}}$	1024

Table 2: Parameters of the notebook ROI Pooling Transformer.

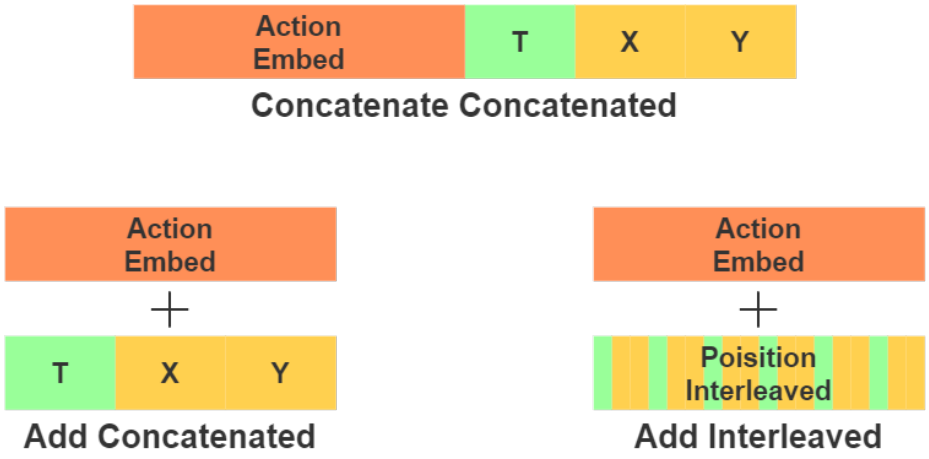

Figure 16: The different forms of object queries tested.

At that time the way the object queries were generated was not fixed yet, so this series of experiments also served to check if one way was better than another. The three ways of doing were :

1. **Concat Concatenated:** concatenate the 3 position encodings T, X and Y and concatenate this new vector to the action embedding;
2. **Add Concatenated:** concatenate the 3 position encodings T, X and Y and add this new vector to the action embedding;
3. **Add Interleaved:** interleave the 3 position encodings T, X and Y and add the resulting vector to the action embedding. This corresponds to the final solution presented earlier (4.2);

We trained each model for 10 epochs and collected for each their results on the epoch at which they perform best in the validation $\mathbf{mAP}_{\text{Overall}}$ in Table 3. We obtained similar results, with a slight hint of improvement for the *Add Interleaved* architecture ; we can hope to obtain results at least as good as by using a classic ROI Pooling.

Model Name	$\mathbf{mAP}_{\text{Box, Noun}}$	$\mathbf{mAP}_{\text{Box, Noun, Verb}}$	$\mathbf{mAP}_{\text{Box, Noun, TTC}}$	$\mathbf{mAP}_{\text{Overall}}$
Notebook Baseline	29.11	10.08	**7.16**	2.61
Concat Concatenate		10.00	6.57	2.38
Add Concatenate		9.81	5.92	2.21
Add Interleaved		**10.38**	6.68	**2.69**

Table 3: Results of the different versions of tokens and the Baseline.

5.2 Scaling Up to the Original Data

My work pass was finally approved the 5th of June, and I arrived the 14th of June in Singapore, a week later I could start working at the premises of A*STAR. I had a dedicated computer – we will refer to as the local machine – to configure the way I wanted it to be configured, so my first task was to set up my work environment.

5.2.1 Setting Up the Environment

I had access to a graphic processing unit (GPU) enhanced computer with the following specs :

GPU	model	NVIDIA GeForce RTX 3090
	cores	1
CPU	model	Intel Core i7-8700K
	cores	6
Memory	RAM	64 GB
	VRAM	24 GB
Storage		1 TB

Table 4: Local Machine Hardware Specifications. VRAM refers to the GPU memory.

The computer was setup with Windows 11 so I first tried the installation procedure in a Linux environment hosted by Windows Subsystem for Linux on Windows, but got issues with using the GPU later on so I started all over again on a Linux distribution installed in dual-boot alongside the native Windows. The goal of the installation was not only to have a functional workspace with Pytorch installed, but to be able to use the main computation resource – the GPU – in our experiments. To do so I had to install Pytorch on top of CUDA, a parallel computing platform and programming model for general computing on GPUs, with the right versions to avoid any issue. The installation procedure went as :

1. installing the latest version of Linux **Ubuntu** (22.04) compatible with the CUDA installed in next step, in dual boot alongside Windows on a 500 GB partition of the 1 TB disk;
2. installing the latest version of **CUDA** (12.1) compatible with the version of Pytorch installed at the next step. Requires Ubuntu 18.04, 20.04, or 22.04;
3. installing the latest version of CUDA enabled **Pytorch** (2.0.1) with Conda, an open source package management system for Python. Requires CUDA 11.8 or 12.1;
4. installing **other packages** required by original scripts I use in my experiments (5.3);

I then installed the code editor *Visual Studio Code* to manage my scripts, and the following experiments and results. The *Bitbucket* repository initialized at 5.1 was used to keep track of the modifications in the code, but not really for shared code management, as I was the only person working on this project.

5.2.2 Obtaining source codes

I used *Git* to clone the repositories of the Ego4D Forecasting Benchmark (`https://github.com/EGO4D/forecasting`) and of InternVideo (`https://github.com/OpenGVLab/ego4d-eccv2022-solutions`), these two repos contain scripts to perform, among others, the following actions:

- **Ego4D Forecasting**:
  - extract videos frames in a format accepted by the dataloader;
  - define the models used in the scripts – SlowFast, the ROI Pooling module and the Baseline (SlowFast + ROI Pooling Head);
  - train or evaluate a model on a given subset of the dataset, with given object detections and model checkpoint;
- **InternVideo**:
  - define the models used in the scripts – VideoMAE, the ROI Pooling with boxes position encoding module and the Baseline (VideoMAE + ROI Pooling Head);
  - train or evaluate a model on a given subset of the dataset, with given object detections and model checkpoint, using the original script from Ego4D Forecasting;
  - train a model more quickly using *DeepSpeed* on a given subset of the dataset, with given object detections and model checkpoint;

5.2.3 Using the Original Data

Access to the Data
The full Ego4D dataset (not only the Omnivore features used in the Notebook 5.1 but also the full scale videos) represents approximately 7 TB of data, which is far too much to be directly stored on the local machine. It is nevertheless stored on a machine at I^2R, and I can access the shared folder containing the dataset on my computer.

Extracting Video Frames
The dataloader provided by Ego4D, that makes the interface between the dataset and the models, works on data in an images list format rather than video format, so it is first necessary to extract the frames of the videos ; for every stopping time of every clip in the dataset, the 32 preceding frames are extracted : this is the input window the model can attend to to make its prediction, it is a parameter that can be freely changed but 32 is the default value used by every model on this problem. Extracting 32 frames before every time to contact in the dataset already generates 71 GB of data, which is stored on the local machine. For speedup reasons the annotations and pre-generated Faster RCNN detections, which represent only 215 MB of data, are also copied on the local machine, this way all the data used for training or evaluation is in local, reducing loading times compared to if it had been on a distant server (see Fig. 17). I am using the V2 version of the Ego4D dataset.

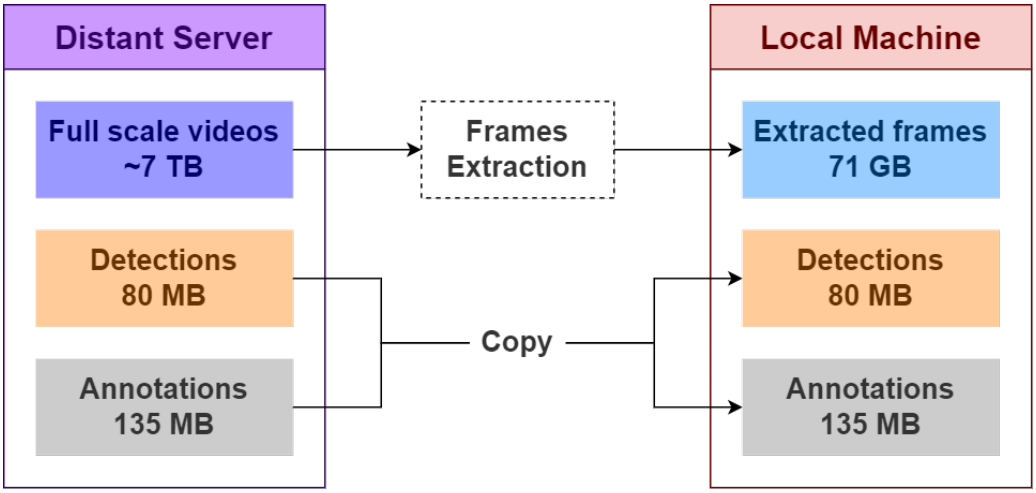

Figure 17: Organization of the data on the distant server and local machine.

5.2.4 Running the Original Scripts

Once the data was ready, I could check if the different pipelines provided in *Ego4D Forecasting* and *InternVideo* were functional.

Ego4D Forecasting
Ego4D provides the weight of the whole *Baseline V2* model, as well as the weights of the video backbone (SlowFast) pre-trained on KINETICS-400 [8]. So we made these tests:

1. train a model from scratch for 100 steps, in order to see the typical values for the loss for an untrained model as well as the expected time of training (ETA):
   - Loss: $\sim$ 9;
   - ETA: $\sim$ 2h40;
2. train the *Baseline V2* model – using the provided weights – for 100 steps in order to see if the loss is indeed smaller:
   - Loss: $\sim$ 3.5;
3. test the *Baseline V2* model – using the provided weights – to obtain the results of the *Baseline V2* on the validation sets of Ego4D V2 (along with the provided results on the test set):

Subset	$\mathbf{mAP}_{\text{Box, Noun}}$	$\mathbf{mAP}_{\text{Box, Noun, Verb}}$	$\mathbf{mAP}_{\text{Box, Noun, TTC}}$	$\mathbf{mAP}_{\text{Overall}}$
validation	24.79	8.59	8.13	3.65
test	26.15	9.45	8.69	3.61

Table 5: Results of the *Baseline V2* model.

With the test of the trained model we could verify that the evaluation script was functional and that the weights provided by Ego4D are the right ones. With the significant drop in the

loss between the untrained and trained model we could verify that the loss indicated during the training process is consistent.

InternVideo
The only weights provided by InternVideo are those of the video backbone (VideoMAE) pre-trained on Ego4D V1 – we do not have access to the weights of the whole model with the ROI pooling head – so even when loading these we cannot expect the loss to be significantly lower right at the beginning of the training ; for the same reasons we cannot evaluate this model directly, so we only did one test:

1. train a model from scratch for 100 steps, in order to see the typical values for the loss for an untrained model as well as the expected time of training:
    - Loss: ~ 9;
    - ETA: $\sim$ 2h;

The starting loss obtained with this training script and this untrained model is close to the one obtained with the Ego4D benchmark script which keeps the consistency between the two scripts ; furthermore the training time is significantly shorter, especially in this case where we have a bigger model to train.

5.3 Building the Code Base

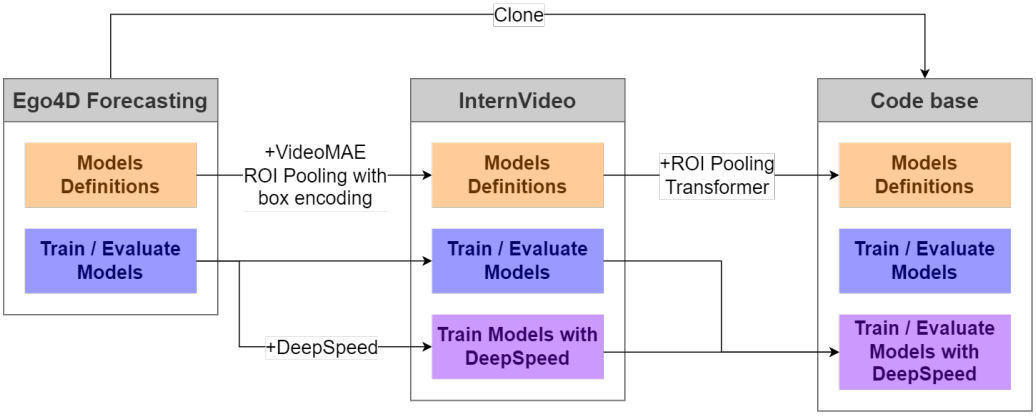

Figure 18: Dependencies between the sources and the code base.

What we refer to as the code base are the scripts on top of which will be specified and run the experiments which will follow. (see Fig. 18) I first cloned the Ego4D Forecasting repository in order to have the correct base (especially the evaluation script which must be unchanged to have the right evaluation process), I then took code snippets from the InternVideo repository – which was also built upon Ego4D Forecasting – especially the definition of the VideoMAE model and the use of DeepSpeed in the training script, which was responsible for the reduction of training time observed in 5.2.4, that I extended to the evaluation script too.

5.3.1 DeepSpeed

DeepSpeed is a deep learning optimization library that allows to easily speedup the training and inference processes of deep learning models. It does so by optimizing the use of the GPU as well as by leveraging mixed precision training and inference. Mixed precision is the combined use of different numerical precisions to represent floating point numbers in a computational method:

- double precision, which uses 64 bits;
- single precision, which uses 32 bits;
- half precision, which uses 16 bits, and reduces the memory usage of the neural network compared to the two previous – more commonly used – data formats;

Being able to use half precision, especially, allows to decrease the required amount of memory and shorten the training and inference times.
The only thing that needs to be done in order to use this library is, after initializing the model and the optimizer used during training, to give those as parameters to a `deepspeed.initialize()` function, which returns the same model and optimizer but running on the DeepSpeed engine. Doing so I could observe an improvement on the *Baseline V1* model with a training time cut in half :

DeepSpeed	Memory Use	t_{epoch}
without	20 GB	2h40
with	12 GB	1h20

Table 6: DeepSpeed improvement on the *Baseline V1* training. t_{epoch} refers to the time it takes for the model to complete one epoch of training.

Having less memory used by the model also allows to increase the batch size, which usually improves the quality of training and decreases training time. For this model we could increase the maximum batch size before saturating the GPU memory from 12 to 16.

5.3.2 ROI Pooling Transformer implementation

Parameter	Value
d_{token}	504
n_{head}	8
$n_{\text{encoder layers}}$	6
$n_{\text{decoder layers}}$	6
$d_{\text{feedforward}}$	2048

Table 7: Parameters of the ROI Pooling Transformer to-scale.

Once the useful assets of both source codes were merged I could define my implementation of the ROI Pooling Transformer Head (4.1) in the modules that can be used within the models. Most of the work had already been done in the Notebook experiment (5.1), the only part left

was the implementation of the video features tokenizer (14) between the video backbone and the encoder of the Transformer, along with the position encoding generator for these features. Both of these modules were largely inspired by the video tokenizer of VideoMAE, and position encoder of DETR ; both of these Transformer-based implementations were also used to choose the parameters values of this new implementation *to-scale* (Table 7).

5.3.3 Establishing the Baseline

With the code base complete, the last task was to test it by establishing our instance of the *Baseline V2* model, by training the baseline model on the Ego4D dataset V2, with the weights only of SlowFast pre-trained on KINETICS-400 [8] ; this would serve as a baseline on this pipeline. We will refer to this model – our instance of the Baseline V2 model – as *Our Baseline* ; we used the following training parameters:

batch size	16
learning rate	10^{-3}
learning rate policy	cosine schedule
momentum	0.9
weight decay	10^{-4}
optimizing method	adam

which are the same as those used in the original script, except for the batch size (due to GPU memory limitations, even with DeepSpeed) which was 32, and the optimizing method which was stochastic gradient descent. We trained the model for 20 epochs – the evolution of the loss on these 20 epochs can be seen in Fig 19 – and obtained the following results on validation and test:

Subset	$\mathbf{mAP}_{\text{Box, Noun}}$	$\mathbf{mAP}_{\text{Box, Noun, Verb}}$	$\mathbf{mAP}_{\text{Box, Noun, TTC}}$	$\mathbf{mAP}_{\text{Overall}}$
validation	24.79	8.86	7.58	2.66
test	26.15	9.48	8.11	3.36

Table 8: Results of Our Baseline.

5.4 Using SlowFast

The goal of this experiment was to see if, just by changing the head of the network – going from a ROI Pooling Head to a ROI Pooling Transformer Head – we could improve the results of the previously established baseline, and maybe even of the original *Baseline V2*. We will refer to this model – ROI Pooling Transformer Head on top of the SlowFast backbone – as the *Using Slowfast* model ; we used the same training parameters as in our baseline (5.3.3):

batch size	16
learning rate	10^{-3}
learning rate policy	cosine schedule
momentum	0.9
weight decay	10^{-4}
optimizing method	adam

We trained the model for 20 epochs – the evolution of the loss on these 20 epochs can be seen in Fig 19 – and obtained the following results on validation and test:

Subset	$\mathbf{mAP}_{\text{Box, Noun}}$	$\mathbf{mAP}_{\text{Box, Noun, Verb}}$	$\mathbf{mAP}_{\text{Box, Noun, TTC}}$	$\mathbf{mAP}_{\text{Overall}}$
validation	24.79	8.83	7.21	3.24
test	26.15	9.90	7.62	3.28

Table 9: Results of the Using SlowFast model.

5.5 Using VideoMAE

The goal of this experiment was to see if we could get even better results with the more advanced video backbone VideoMAE ; but this model is so much more complex and heavy that it almost fills up the GPU memory – 20 GB for a batch size of only 4 – and takes more than 3h for one epoch. This seemed too much as the VideoMAE backbone weights are only 3 times bigger than those of SlowFast, but the model seem to be more than 4 times larger in action ; my first task before actually training the model was to investigate on how the memory is used during training.

5.5.1 Improving the Memory Usage

With the help of the `torch.cuda.memory_allocated()` function provided by Pytorch, we can check how much GPU memory is allocated for Pytorch anywhere in the script ; at the beginning of the training memory is only allocated to store the weights of the model, but, during inference, memory is used to compute the operations happening within the network. We can track which operations are the most costly during the first inference of the model on the training set ; this is during this first inference that the memory usage goes from only used for storing the weights to storing the whole computation graph of the model.
One operation was much more costly than the others : the attention operation. VideoMAE is a Video Transformer and re-implements to this purpose a Transformer, so the attention operation is not from the original Pytorch implementation. Looking at the original implementation, it was noticeable that the attention performed in VideoMAE was largely inspired from an original code snippet ; in List. 1, this snippet corresponds to the "Process to compute the attention operation and save the attention weights" operation. Pytorch indeed allows to return the attention weights along with the attention result of the attention operation, but it is optional, and when these are not needed and unused, it allocates GPU memory for nothing. In the case of VideoMAE these attention weights were indeed not used, and as such, useless to compute.

I adapted the "Process to only compute the attention operation" code snippet to integrate it in the code of VideoMAE in place of the previous attention operation, verifying that the two different versions give the same results on random data in a unit test manner. I could then check how the performances were impacted and saw a memory usage almost cut in half and a training time slightly reduced (Table 10).

```python
def multi_head_attention_forward(
    query: Tensor,
    key: Tensor,
    value: Tensor,
    need_weights: bool,
    ...
) -> Tuple[Tensor, Optional[Tensor]]:

    ...

    if need_weights:
        """
        Process to compute the attention operation and
        save the attention weights
        """
        ...
        return attn_output, attn_output_weights
    else:
        """
        Process to only compute the attention operation
        """
        return attn_output, None
```

Listing 1: Simplified version of the attention operation from the original Pytorch source code. Facebook Inc. [4]

Attention Weights	Memory Use	t_{epoch}
saved	20 GB	3h40
not saved	12 GB	3h00

Table 10: Difference of performance between saving and not saving attention weights.

Furthermore we had a better loss evolution when freezing the video backbone – meaning that the forward operations are still performed, but the backward operations are not, and every information relevant for these backward operations are not stored – during training ; when unfrozen the loss would quickly reach a point of stagnation, and even go slightly up. So, by freezing the video backbone, we could improve the GPU usage and inference time even more, setting the batch size to 16 and performing one training epoch in 1h10.

5.5.2 Results

We tested on this model a detail architecture of the Transformer used in DETR, which is that position embeddings are added, at each attention operation, in the K and Q of the attention operation ; this showed no improvement in the loss evolution or validation mAE so we kept the same transformer architecture as for 5.4. We will refer to this model – ROI Pooling Transformer Head on top of the VideoMAE backbone – as the *Using VideoMAE* model ; we used the following learning parameters for training:

batch size	16
learning rate	$2.5 \cdot 10^{-4}$
learning rate policy	cosine schedule
momentum	0.9
weight decay	$5 \cdot 10^{-2}$
optimizing method	adam

which are the same as those used in the original InternVideo training script, except for the batch size which was 4, and the learning rate which was $3.125 \cdot 10^{-5}$. We trained the model for 20 epochs – the evolution of the loss on these 20 epochs can be seen in Fig 19 – and obtained the following results on validation and test:

Subset	$\mathbf{mAP}_{\text{Box, Noun}}$	$\mathbf{mAP}_{\text{Box, Noun, Verb}}$	$\mathbf{mAP}_{\text{Box, Noun, TTC}}$	$\mathbf{mAP}_{\text{Overall}}$
validation	24.79	10.48	8.70	3.92
test	26.15	11.25	9.22	4.75

Table 11: Results of the Using VideoMAE model.

6 Discussion

6.1 Comparison

We can first start by summing up our results on the test set, and comparing it to the existing solutions (Table 12).

Model	**mAP**$_{\text{B, N}}$	**mAP**$_{\text{B, N, V}}$	**mAP**$_{\text{B, N, TTC}}$	**mAP**$_{\text{Overall}}$
Baseline V1	20.45	6.78	6.17	2.45
InternVideo	24.60	9.19	7.64	3.40
Using SlowFast	**26.15**	9.90	7.62	3.28
Our Baseline		9.48	8.11	3.36
Baseline V2		9.45	8.69	3.61
Using VideoMAE		11.25	**9.22**	4.75
StillFast	25.06	13.29	9.14	5.12
GANO	25.67	**13.60**	9.02	**5.16**

Table 12: Results Comparison.

6.1.1 Using SlowFast VS Our Baseline

The first comparison that can be made is between *Our Baseline* and the *Using SlowFast* model, as the only difference between these networks architectures and training processes is the use of our ROI Pooling Transformer in place of ROI Pooling. We can see that, despite a better performance of the *Using Slowfast* model on validation set – 3.24 VS 2.66 –, it was beaten by *Our Baseline* on the test set.

6.1.2 Using VideoMAE VS Baseline V2

This experiment showed better results, with the *Using VideoMAE* model beating the *Baseline V2* model both on validation and test data by far, making it the best model within the models known **at the time of the literature study**.

6.1.3 Using VideoMAE VS Current SOTA

The problem is that since the literature study, other architectures have been developed : the StillFast model [11] and GANO [13], another model built on top of StillFast. Both of these models achieve more the 5 mAP on test set, beating the Using VideoMAE model.

6.2 Analysis

6.2.1 Overfitting, Misalignement ?

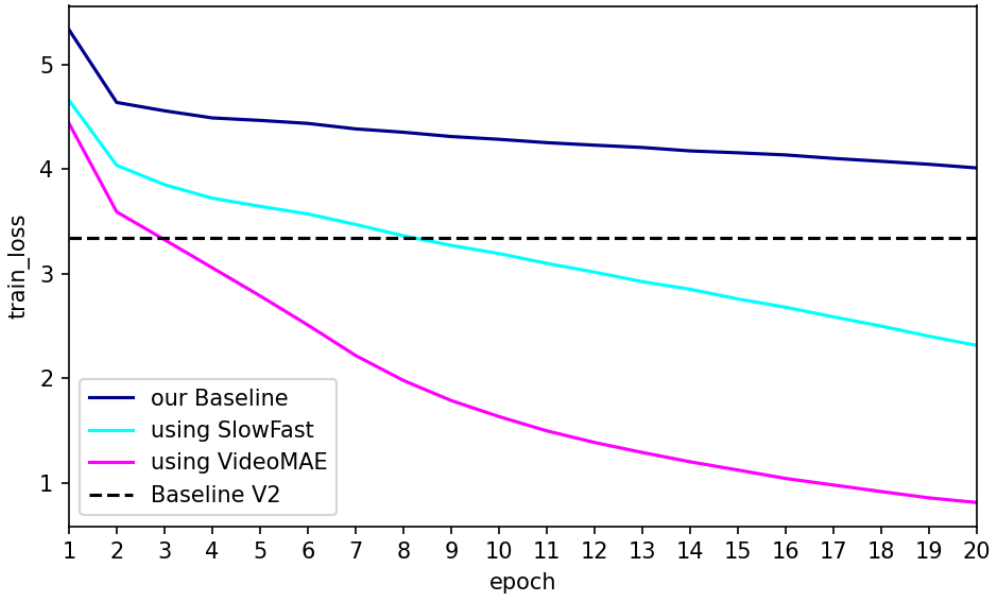

Figure 19: Evolution of the train losses of the three models trained, compared to the train loss achieved by the provided Baseline V2 model.

Evaluating the model takes time, so most of the decisions in the training hyperparameters and architecture details choices were based on the evolution of the loss during training ; this is for example looking at the loss that we chose to freeze the VideoMAE backbone in the *Using VideoMAE* model. These losses showed rather promising models in comparison to the *Baseline V2* model (Fig. 19). We can first see that, compared to the final training loss attained by the *Baseline V2*, *Our Baseline* very slowly converges towards it, maybe is it the use of DeepSpeed and mixed-precision training or the smaller batch size that slows the convergence. Nevertheless, in comparison to these two models, our two implementations using the ROI Pooling Transformer overtake the *Baseline V2*, in terms of training loss, in less than 10 epochs for both of them and go way below. Looking at their performance on validation and test set we may presume that these models are overfitting or misaligned with the evaluation metric. Even though the time lacked to compute the evolution of the loss on the validation set, some quick tests showed that the *Using VideoMAE* model still had a validation loss above 4 on a subset of the validation data, when its training loss was below 1, which would strongly point towards overfitting. Time lacked to retrain this model with a higher weight decay, or FFN layers with less units, to counter this problem.

6.2.2 Quality of the Detections

Because we use the ground truth (GT) nouns and bounding boxes as detections for training, the loss in detection quality when predicting on the validation or test sets could be a source of loss of precision. We tried to bring an insight on this question on our best performing model,

Using VideoMAE, by checking how its mAP changes on the training and validation datasets when using the predicted object detection or the GT ; the results are compiled in Tab. 13.

Subset	Training		Validation	
Detections	GT	Pred	GT	Pred
$\mathbf{mAP}_{\text{Box, Noun}}$	100	46.01	100	24.79
$\mathbf{mAP}_{\text{Box, Noun, Verb}}/\mathbf{mAP}_{\text{Box, Noun}}$	84.15	77.14	42.41	42.28
$\mathbf{mAP}_{\text{Box, Noun, TTC}}/\mathbf{mAP}_{\text{Box, Noun}}$	45.31	41.80	38.30	35.09
$\mathbf{mAP}_{\text{Overall}}/\mathbf{mAP}_{\text{Box, Noun}}$	39.02	33.71	18.52	15.81

Table 13: Object detection quality impact on performance. Values are in percent.

Here we are interested in the **proprtion** of $\mathbf{mAP}_{\text{Box, Noun}}$ *converted* in the other metrics, which each directly depends on the number of correct object detections. We can see that we have a slight loss on mAP when using predicted oject detection rather than GT, but going from training to validation has a much greater impact, reinforcing the hypothesis of a strong overfitting of the model.

6.2.3 Balance of the Data

The last point of concern was on the quality of the data ; looking at the verbs predictions of the model we could notice a strong presence of the label 62, corresponding to the "take" verb. Indeed, as shown in Fig. 20, the dataset is strongly imbalanced in favor to this label. It is more balanced, on the other hand, on the nouns labels.

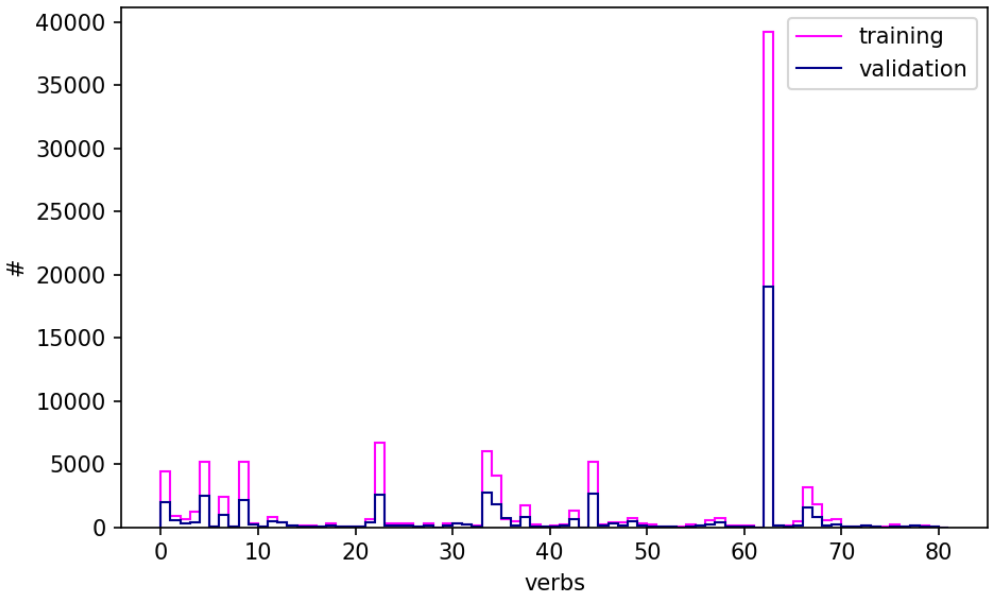

Figure 20: Distribution of the verbs in the training and validation sets.

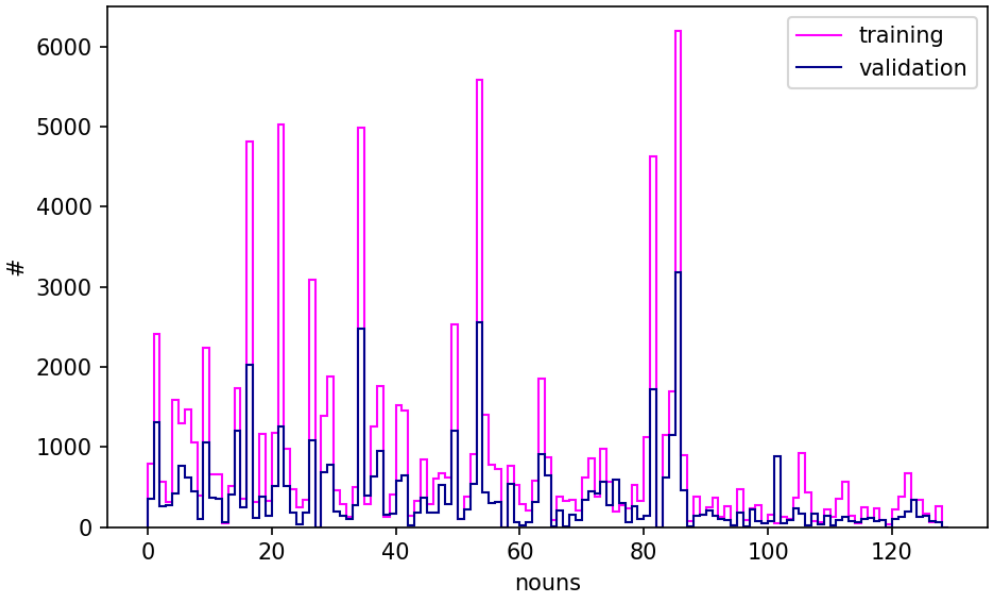

Figure 21: Distribution of the nouns in the training and validation sets.

6.3 Areas for Improvement

In this project we managed to implement and test a few models with a few variations, we lacked the time and computation power to then find the best performing training hyperparameters and architecture details – d_{token}, $n_{\text{encoder layers}}$, $n_{\text{decoder layers}}$, $d_{\text{feedforward}}$, etc – of the ROI Pooling Transformer Head, which, seeing how our solution might overfit the training data, may improve even more our results.
Our model, like any other solution on this problem, seems to be strongly dependent on the video feature extractor ; only models using StillFast are better than our solution for now, we could adapt our solution to use this model as a video backbone, as it can be seen as one, and see if it can be enough to beat the current top-1 solution.
Finally, looking at the strong imbalance of the verbs labels, techniques and methods already commonly used to deal with imbalanced datasets could be implemented to improve the training process.

7 Conclusion

Short Term Action Anticipation – and more generally tasks on egocentric videos – is a very recent problem and not yet as competitive as older and well known ones, hence making the room for a lot of experimentation on the solutions to find ; as such I was from the start considering the possibility, with this experimental solution, that I could get decent results even in comparison to the SOTA. I was, when starting the experiments, only considering to use VideoMAE as a video backbone, as it seemed more probable – and it turned out to be the case – to obtain better results with it than with SlowFast, and was pretty satisfied with the 3rd place on the public leaderboard I managed to obtain with it. But I was then more interested in the very validity of the core idea of replacing a ROI Pooling operation with a Transformer, and focused my efforts on SlowFast, on which we had a Baseline trained on the same data to beat. As discussed before, time and computation power may have lacked to test a mature implementation of this idea, and the question of using Transformers as ROI Pooling in other problems could still arise ; it could even generalize the ROI Pooling operation to work on features that are not necessarily dependant on the space the regions are defined in, for which the model would learn, for a given region, where in the features to focus its *attention*.

This lack of time reflects the difficulties and challenges I was faced with during this experience ; from difficulties to implement the solution and find what are the sources of instabilities, to delays in my arrival in Singapore, extending a situation of remote work and generating lots of uncertainties. For worse *and* for better this internship was a full experience on a personal level, especially as a first overseas travel – performed alone moreover – it forced myself into meeting new people and discovering new cultures ; but also – of course – on a professional level, where I learned to be autonomous on every level : on the choice of the benchmark to work on, the solution to implement, the configuration of my workspace at the lab, and the choices of implementation, with the responsibility that comes with it to assure that I will be capable to build, test, and obtain results in time. The teamwork might have lacked for a project I was the only one working on, but I still had to show good communication skills to discuss complex ideas with my tutor, as well as the progress of the experiments and issues encountered. In any case I was able to put into practice all the skills and knowledge I gained throughout my studies at ENSEEIHT, past internships, and personal experience ; the world of research is truly fascinating and stimulating, with my studies on their end a PhD could be in my future prospects.

References

[1] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, *Vivit: A video vision transformer*, 2021.

[2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, *End-to-end object detection with transformers*, 2020.

[3] G. Chen, S. Xing, Z. Chen, Y. Wang, K. Li, Y. Li, Y. Liu, J. Wang, Y.-D. Zheng, B. Huang, Z. Zhao, J. Pan, Y. Huang, Z. Wang, J. Yu, Y. He, H. Zhang, T. Lu, Y. Wang, L. Wang, and Y. Qiao, *Internvideo-ego4d: A pack of champion solutions to ego4d challenges*, 2022.

[4] I. Facebook, *Multihead attention operation implementation from pytorch*, 2023. `https://github.com/pytorch/pytorch/blob/main/torch/nn/functional.py#L5094` [Accessed: 27 August 2023].

[5] C. Feichtenhofer, H. Fan, J. Malik, and K. He, *Slowfast networks for video recognition*, 2019.

[6] R. Girdhar, M. Singh, N. Ravi, L. van der Maaten, A. Joulin, and I. Misra, *Omnivore: A single model for many visual modalities*, 2022.

[7] K. Grauman, A. Westbury, E. Byrne, Z. Chavis, A. Furnari, R. Girdhar, J. Hamburger, H. Jiang, M. Liu, X. Liu, M. Martin, T. Nagarajan, I. Radosavovic, S. K. Ramakrishnan, F. Ryan, J. Sharma, M. Wray, M. Xu, E. Z. Xu, C. Zhao, S. Bansal, D. Batra, V. Cartillier, S. Crane, T. Do, M. Doulaty, A. Erapalli, C. Feichtenhofer, A. Fragomeni, Q. Fu, A. Gebreselasie, C. Gonzalez, J. Hillis, X. Huang, Y. Huang, W. Jia, W. Khoo, J. Kolar, S. Kottur, A. Kumar, F. Landini, C. Li, Y. Li, Z. Li, K. Mangalam, R. Modhugu, J. Munro, T. Murrell, T. Nishiyasu, W. Price, P. R. Puentes, M. Ramazanova, L. Sari, K. Somasundaram, A. Southerland, Y. Sugano, R. Tao, M. Vo, Y. Wang, X. Wu, T. Yagi, Z. Zhao, Y. Zhu, P. Arbelaez, D. Crandall, D. Damen, G. M. Farinella, C. Fuegen, B. Ghanem, V. K. Ithapu, C. V. Jawahar, H. Joo, K. Kitani, H. Li, R. Newcombe, A. Oliva, H. S. Park, J. M. Rehg, Y. Sato, J. Shi, M. Z. Shou, A. Torralba, L. Torresani, M. Yan, and J. Malik, *Ego4d: Around the world in 3,000 hours of egocentric video*, 2022.

[8] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman, and A. Zisserman, *The kinetics human action video dataset*, 2017.

[9] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, *Microsoft coco: Common objects in context*, 2015.

[10] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, *Video swin transformer*, 2021.

[11] F. Ragusa, G. M. Farinella, and A. Furnari, *Stillfast: An end-to-end approach for short-term object interaction anticipation*, 2023.

[12] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016.

[13] S. Thakur, C. Beyan, P. Morerio, V. Murino, and A. D. Bue, *Guided attention for next active object @ ego4d sta challenge*, 2023.

[14] Z. Tong, Y. Song, J. Wang, and L. Wang, *Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training*, 2022.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, 2017.

[16] H. Zhang, F. Li, S. Liu, L. Zhang, H. Su, J. Zhu, L. M. Ni, and H.-Y. Shum, *Dino: Detr with improved denoising anchor boxes for end-to-end object detection*, 2022.

A ROI Pooling

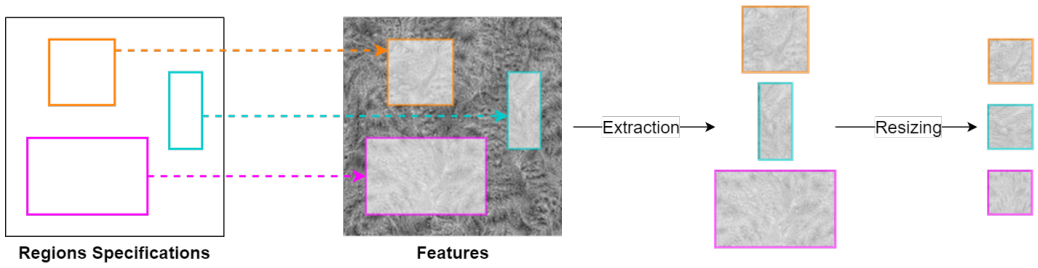

Figure 22: The ROI Pooling Process.

Region of Interest (ROI) Pooling is an operation to isolate features related to specific regions. It takes as input the specifications of regions of interest on a given space (eg. bounding boxes on a 2D image) as well as features defined over the same space (eg. images features extracted by a convolutional network, each of which is related to a specific zone in the image dependent on its spatial position), and extracts the corresponding region features for each region, to then resize them to a fixed size, so that another module can use them regardless of the shape or size of the corresponding region.

B Faster RCNN

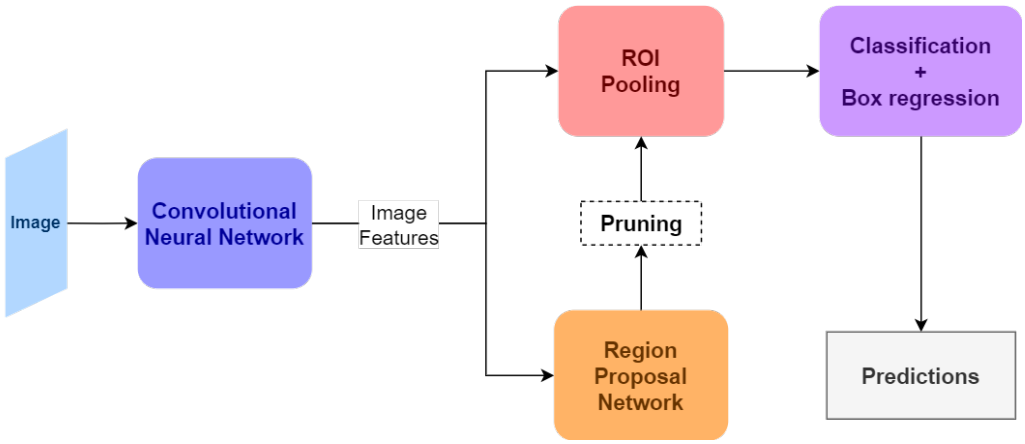

Figure 23: The Faster RCNN Architecture.

Faster RCNN is a Region-based Convolutional Neural Network (RCNN) used for detection of objects in images. It uses a convolution neural network to extract the image features, which are fed to a Region Proposal Network (RPN) which outputs a fixed number of coordinates and confidence values associated with a fixed number of pre-defined bounding boxes. These confidence values are used to prune these bounding boxes to only keep the bounding boxes of interest, which are used in a ROI Pooling operation on the same extracted image features that were used for the RPN. The features pooled are fed to a FFN that outputs the final coordinates of the bounding box as well as a class for each one of them.

C Transformer

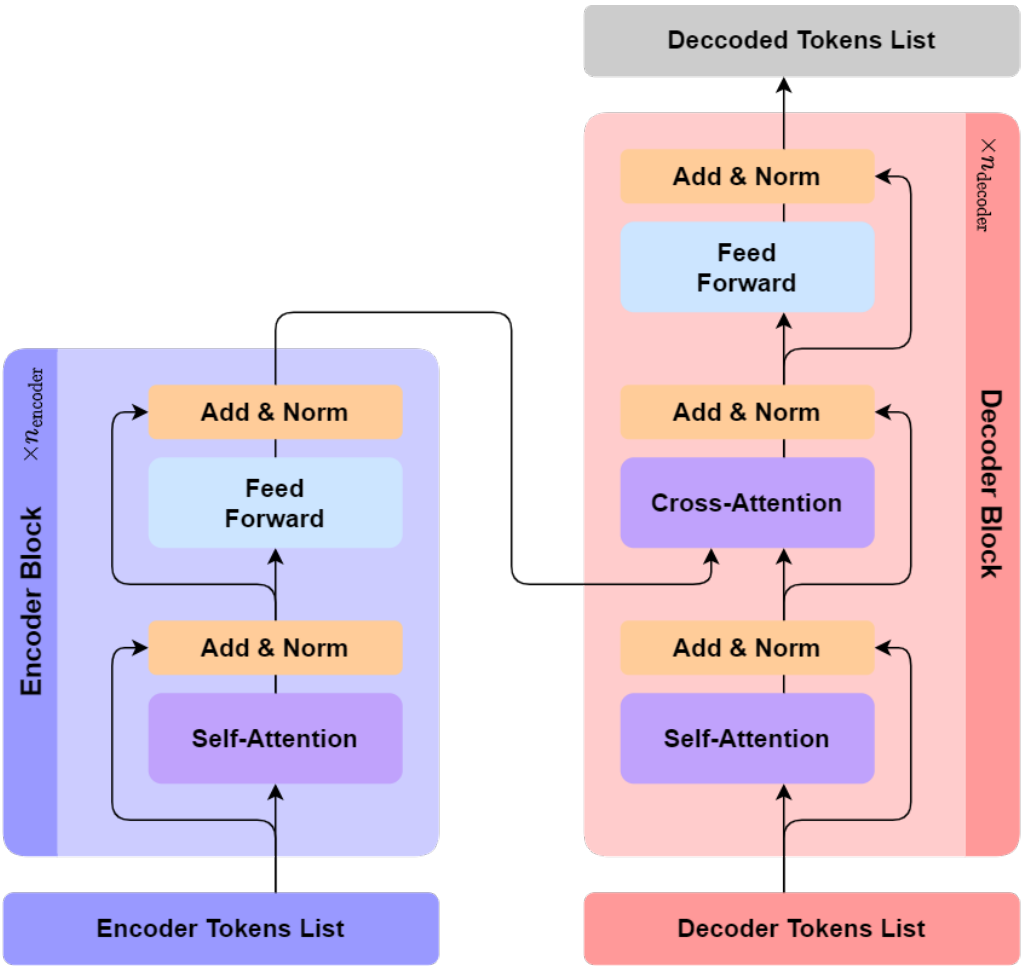

Figure 24: The Transformer Architecture.

A Transformer [15] is composed of an encoder and a decoder.
The encoder is composed of n_{encoder} consecutive encoder blocks, in each block an operation of self attention over the encoder tokens is performed, followed by a Feed Forward Network (FFN) on each of these independently ; residual connections are made before normalization for each of these operations.
The decoder is composed of n_{decoder} consecutive decoder blocks, in each block an operation of self attention over the decoder tokens is performed, followed by a cross-attention operation on the decoder tokens as keys and encoded tokens as queries and values in order to use the information extracted by the encoder, and a Feed Forward Network (FFN) on each of the decoder tokens independently ; residual connections are made before normalization for each of these operations. None of the Feed Forward, Self-Attention, and Cross-Attention operations are variant to the tokens positions in the token sequence, so position encoding (D) is usually added to those tokens if their position has to be taken into account by the transformer.

D Position Encoding

Position encoding designs the process of translating a notion of position – which can be represented, for example, by an integer for an index, a real number for a 1D spatial dimension, or an n-tuple of real numbers for a position in $\mathbb{R}^n$ – in a vector of given dimension d that we refer to as the position embedding.
The associations between vectors and positions can either be learnable, in which the network builds itself its own position embeddings, or fixed, by a specific process. The most used position encoding in state-of-the-art Transformers is the Sine Position Encoding, which is a fixed position encoding over a 1D space which, for a given position $k \in [0, L]$, associates the position embedding $P^k \in \mathbb{R}^d$ such that $\forall i \in [1, \lfloor \frac{d}{2} \rfloor]$ and $\forall j \in [0, \lceil \frac{d}{2} \rceil - 1]$:

$$\begin{cases} P^k_{2i} & = \sin \frac{k}{T^{2i/d}} \\ P^k_{2j+1} & = \cos \frac{k}{T^{2j/d}} \end{cases} \tag{5}$$

where T is a parameter called *temperature*.

E Mean Average Precision

The Mean Average Precision (mAP) is an evaluation metric for problems in which we assume to have ground truths and a set of candidate predictions with a certain confidence for each.
First, for each example, each of the detected objects are matched to a ground truth annotation in a given order until no more ground truth is available ; the matching criterion and order of attribution depend on the problem (these can be respectively IOU and in a decreasing confidence order for object detection). The predictions are then compared to their respective ground truth to establish which ones are correct or wrong, based on another given criterion (for example a correct prediction is when the predicted class matches that of the ground truth and the IOU is over 0.5 for object detection).
Then, for a specific threshold T on the confidence s, the number of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) on the whole set are computed, which correspond to:

- **TP:** when the prediction is correct and $s \geq T$;
- **TN:** when the prediction is wrong (or the prediction was discarded) and $s < T$;
- **FP:** when the prediction is wrong (or the prediction was discarded) and $s \geq T$;
- **FN:** when the prediction is correct and $s < T$;

The recall and precision for this specific threshold can then be computed as such :

$$\text{Precision}(T) = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}}$$
$$\text{Recall}(T) = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}}$$

The Average Precision (AP) is defined for a set of thresholds in increasing order $\{T_i\}_{i=1}^{N}$ as the approximation of the number $\int_0^1 p(r)dr$, where p is defined as:

$$\begin{gathered} p : [0,1] \to [0,1] \\ p(r_i) = \text{Precision}(T_i), \forall i \in [1,N] \\ \text{with } r_i = \text{Recall}(T_i) \end{gathered}$$

For mAP the AP is computed independently for each class $c \in \mathcal{C}$ and we compute the mean over all classes:

$$\text{mAP} = \frac{1}{n} \sum_{c \in \mathcal{C}} AP_c$$

We have as a main property that mAP $\in [0,1]$.