VATE: Edge-Cloud System for Object Detection in Real-Time Video Streams

Maximilian Maresch
Distributed Systems Group
TU Wien
maxi.maresch@gmail.com

Stefan Nastic
Distributed Systems Group
TU Wien
snastic@dsg.tuwien.ac.at

***Abstract*—In the realm of edge intelligence, emerging video analytics applications are often based on resource constrained edge devices. These applications need systems which are able to provide both low-latency and high-accuracy video stream processing, such as for object detection in real-time video streams. State-of-the-art systems tackle this challenge by leveraging edge computing and cloud computing. Such edge-cloud approaches typically combine low-latency results from the edge and high accuracy results from the cloud when processing a frame of the video stream. However, the accuracy achieved so far leaves much room for improvement. Furthermore, using more accurate object detection often requires having more capable hardware. This limits the edge devices which can be used. Applications related to autonomous drones, with the drone being the edge device, give one example. A wide variety of objects needs to be detected reliably for drones to operate safely. Drones with more computing capabilities are often more expensive and suffer from short battery life, as they consume more energy. In this paper, we introduce VATE, a novel edge-cloud system for object detection in real-time video streams. An enhanced approach for edge-cloud fusion is presented, leading to improved object detection accuracy. A novel multi-object tracker is introduced, allowing VATE to run on less capable edge devices. The architecture of VATE enables it to be used when edge devices are capable of running on-device object detection frequently and when edge devices need to minimise on-device object detection to preserve battery life. Its performance is evaluated on a challenging, drone-based video dataset. The experimental results show that VATE improves accuracy by up to 27.5% compared to the state-of-the-art system, while running on less capable and cheaper hardware.**

***Index Terms*—Video Analytics, Edge Intelligence, Edge Computing, Edge-Cloud Systems, Object Detection, Object Tracking**

I. Introduction

Detecting objects fast and accurately in dynamic environments remains a challenge [1] [2]. Applications that utilize intelligent drones, intelligent cars, and augmented reality headsets, are all exposed to such environments, where the surroundings change rapidly within a few frames as the drone, car, or headset moves or as any object in the environment moves [3]. Reliable object detection is one of the foundational problems that needs to be solved to enable these applications [1]. Streaming video analytics systems can be built for operation at the edge to tackle this. These systems need to process frames with high accuracy and at a high frame rate [3].

The most accurate models can be leveraged with fast inference only by running them in the cloud [3]. Vision transformers, like [4], have for example made significant progress in recent years, but are computationally expensive models, making them infeasible to be used on edge devices for fast object detection. At the same time, communicating with models in the cloud incurs a latency overhead in itself [3]. In the context of systems that need to operate at a high frame rate and target dynamic environments, this becomes problematic as object detections coming from cloud models can be outdated when they finally arrive.

Edge computing offers a lower latency alternative to using the cloud by moving data processing closer to the data sources [5] [3]. Nowadays GPUs and tensor processing units (TPUs) provide computational acceleration at the edge, making it possible to leverage deep neural networks, in particular convolutional neural networks for object detection, on edge devices by running inference operations directly at the edge [5] [6]. However, the computational power of edge devices does not match that of the cloud, resulting in limitations on the accuracy of the object detection models that can be used at the edge [6].

State-of-the-art systems combine object tracking, edge object detection, and cloud object detection [6]. Typically, results from the edge and the cloud are combined, which enables the system to achieve higher object detection accuracy. REACT [6] showed that up-to-date, lower-accuracy edge and outdated, higher-accuracy cloud object detections can be fused to improve the accuracy in dynamic environments while being able to process frames at a high frame rate. Nevertheless, many challenges remain in developing these systems [3].

In this paper, we introduce VATE[1], a novel edge-cloud system for object detection in real-time video streams. VATE is inspired by REACT, which gives the state-the-of-art system. VATE extends the design of REACT in several directions and presents the following contributions:

- An enhanced algorithm for fusing edge and cloud object detections, leading to improved object detection accuracy by 27.5% compared to the state-of-the-art algorithm, while the system is running on less capable and cheaper edge hardware.
- A novel multi-object tracker that enables running VATE on the less capable edge devices. VATE runs on a Nvidia Jetson TX2, which is less capable than a Nvidia Jetson

[1]The source code is available at https://github.com/polaris-slo-cloud/vate

Xavier, as was used to evaluate the state of the art. The TX2 is also available on many commercial drones.
- Two novel coordination algorithms to support on-device object detection and offloading of object detection to a combination of edge and cloud servers. This allows the system to be used with the less capable edge devices and in scenarios where object detection on edge devices needs to be avoided to preserve battery life.

The system is evaluated on the VisDrone-VID2019 dataset [2] and the performance of our system under different deployment options is investigated.

The rest of this paper is organized as follows: In Section II, the system design, architecture, and deployment modes of VATE are described. Section III presents the main runtime mechanisms used by VATE. The implementation is depicted in Section IV. Section V describes how VATE is evaluated and the experimental results achieved by it. Sections VI and VII introduce related work and conclude this paper, respectively.

II. VATE System Design & Architecture Overview

The system consists of 3 components:

1) Edge Server: Runs inference using the edge object detection model
2) Cloud Server: Runs inference using the cloud object detection model
3) Edge Device: Runs object tracking and is supported by both: (i) Object detections from the Edge Server: Depending on the configured mode, these can be received synchronously or asynchronously. (ii) Object detections from the Cloud Server: These are always received asynchronously.

An overview of the architecture of VATE is shown in Figure 1. A frame from the VisDrone-VID2019 dataset being processed by VATE is shown in Figure 2. Annotations are displayed in green. The objects detected by the edge model are displayed in blue, if enhanced by the cloud model in red.

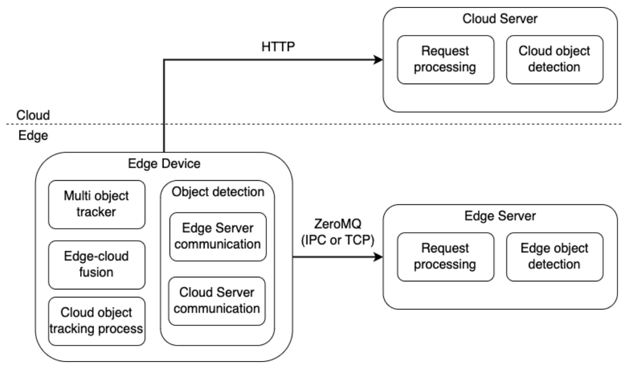

Fig. 1. VATE architecture overview

An Edge Server request consists of an encoded frame. A predictor runs the object detection for this frame. Multiple predictors are supported by VATE. In particular, a Jetson predictor is available, which uses an object detection model and leverages the hardware capabilities of the Nvidia Jetson platform. The response to these requests represents the detected objects, each consisting of a category, a bounding box and a score. A Cloud Server request consists of a frame as well. An object detection model is used to respond to the requests with detected objects.

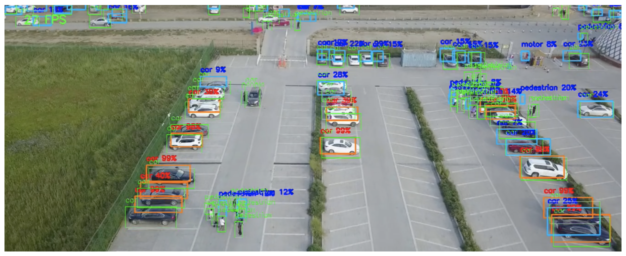

Fig. 2. Example frame with objects detected by VATE

The Edge Device represents the component that uses the detected objects, for example running on a drone, a car, or an augmented reality headset. It sends frames to the Edge Server to get up-to-date, lower-accuracy detected objects. Inter-process communication (IPC) is used for this communication if the Edge Server component is running on the same device as this component, otherwise, TCP is used. For communication with the Edge Server, the Edge Device can use sync or async mode. A detection rate d is used by these modes. It influences how often frames are sent to the Edge Server and how often object tracking is used. The modes are described in Section III by the coordination algorithms. Additionally, the Edge Device sends a frame to the cloud to get outdated, higher-accuracy detected objects as often as possible. It sends the current frame to the Cloud Server every time objects from the cloud are received and processed. This happens irregularly and is dependent on the response time of the Cloud Server and the network.

On start-up, the Edge Device creates a separate process that is used for cloud object tracking. This process takes the objects detected by the Cloud Server and uses a multi-object tracker and previous frames to provide up-to-date cloud-detected objects to the main process. To reduce CPU usage and the amount of data being exchanged, a cloud tracking stride $s \geq 1$ can be specified. Only every s^{th} frame of the previous frames is sent to the cloud tracking process. This results in certain frames not being used for cloud object tracking. $s = 2$ and $s = 3$ were found to work well. See [6] for more information on the concepts of a detection rate/frequency and a cloud tracking stride.

The up-to-date cloud-detected objects are fused with edge-detected objects by the Edge Device. This is called edge-cloud fusion. A custom edge-cloud fusion algorithm is used, which is presented in Section III. The result of this edge-cloud fusion is up-to-date, higher accuracy detected objects. The detected objects coming from the Edge Server are enhanced with detected objects from the Cloud Server.

The detected objects are used to prepare the objects to be tracked by the multi-object tracker for the next frames. This object tracking uses a custom CPU-based multi-object tracker.

The multi-object tracker takes a frame as input and outputs a tracking result based on the current objects to track.

Figure 3 illustrates the high-level model of VATE from the perspective of the Edge Device. The inputs and their impact on aspects of the Edge Device are illustrated. The result corresponds to the detected objects. The result processing is determined by the concrete application.

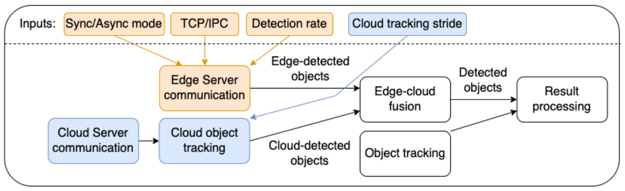

Fig. 3. The model of VATE from the perspective of the Edge Device.

A. Deployment options

1) Edge Device and Edge Server on the same physical device: This option is shown in Figure 4. It is seen as suitable for intelligent cars, which have onboard hardware acceleration. As both edge components are running on the same device, IPC can be used. There is no network between the edge components, thus object detections can be received synchronously without severely limiting the number of frames that the system can process.

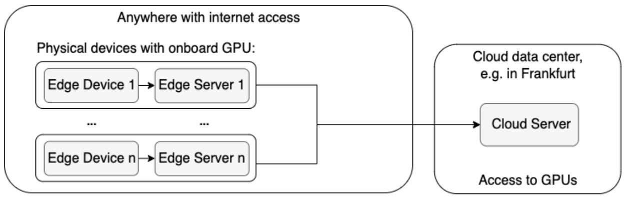
Anywhere with internet access
Physical devices with onboard GPU:
Edge Device 1
Edge Server 1
...
...
Edge Device n
Edge Server n
Cloud data center, e.g. in Frankfurt
Cloud Server
Access to GPUs

Fig. 4. The deployment option with Edge Device and Server on the same physical device.

2) Edge Device and Edge Server on different physical devices: This option is shown in Figure 5. It is seen as suitable for drones used at industrial facilities and AR headsets used in cities. Battery life is assumed to be a major concern. Offloading to a nearby Edge Server is possible. The Edge Server leverages hardware accelerators (like GPUs). It needs to be located close to the Edge Device, for example in the same city, to able to respond to requests of the edge devices sufficiently fast for the desired frame rate. As the edge components are not running on the same device, TCP must be used. Since there is a network between the edge components, edge object detections should be received asynchronously to avoid lag in the video.

III. VATE Main Runtime Mechanisms

A. Enhanced edge-cloud fusion algorithm

A custom edge-cloud fusion algorithm is used by the Edge Device. The algorithm is shown by Algorithm 1. The edge-cloud fusion algorithm takes current objects, new objects, and a source of the new objects (Edge or Cloud Server) as its

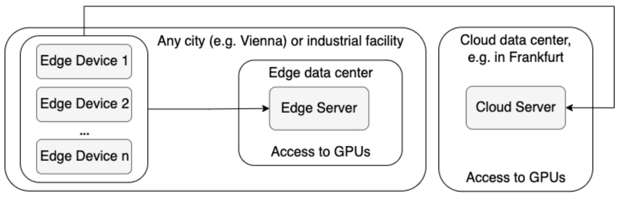

Fig. 5. The deployment option with Edge Device and Server on different physical devices.

input and returns a list of enhanced detected objects. The current objects can be edge detected and the new objects cloud detected or vice versa.

An Intersection over Union (IoU) matrix based on the bounding boxes of the current and new objects is calculated, giving the IoU of each current object with each new object (line 1). A linear sum assignment is performed (line 3). The result is used to build the collection of enhanced detected objects. For current objects that are matched by new objects (line 5), the resulting category and bounding box are determined by the source of the new objects. If the source is the cloud, then the category is taken from the new object, and the bounding box is taken from the current object (lines 6-9). This leverages the fact that the cloud model is more accurate, thus the category determined by it is used. As frames may have passed and there is a matching current object, its up-to-date bounding box is used. If the source is the edge, then the category of the current object is taken, and the bounding box from the new object is used (lines 10-13). This ensures that the more accurate categories are retained, and up-to-date bounding boxes are used. For current objects which are unmatched by new objects, the new object is used as-is (line 17). For details on these aspects of the algorithm, see [6].

Unlike existing approaches, the current objects that are unmatched by new objects are kept in the result (lines 18-20) if the current objects are coming from the Edge Server. This avoids discarding a detected object from the Edge Server when there is no corresponding detected object from the Cloud Server and was found to increase accuracy.

B. Coordination algorithms

VATE provides two coordination algorithms, which showcase the main loop used by the system for processing a frame. Both algorithms include the details on how the Cloud Server is incorporated asynchronously. The algorithms differ in how the communication between Edge Device and Edge Server works.

The coordination algorithm for sync mode is shown by Algorithm 2. With sync mode, the Edge Device sends every d^{th} frame to the Edge Server to get detected objects for that frame (lines 4-5). The Edge Device waits until it has received the detected objects. For the frames between every d^{th} frame, object tracking is used based on the last detected objects (line 10).

The coordination algorithm for async mode is shown by Algorithm 3. With async mode, at least every d^{th} frame is sent to the Edge Server. The Edge Device does not wait

```
Input: A list of current objects $objs_{curr}$, a list of new objects
    $objs_{new}$, a source $type_{new}$ ("EDGE" or "CLOUD")
Output: Collection of objects with their source
 1: $M \leftarrow ComputeIOUMatrix(objs_{curr}, objs_{new})$
 2: $M[M < 0.5] \leftarrow 0$
 3: $curr\_objs, new\_objs \leftarrow LinearSumAssignment(M,$
        $maximize = true)$
 4: for $curr\_obj$, $new\_obj$ in $curr\_objs$, $new\_objs$ do
 5:     if $M[curr\_obj][new\_obj] \neq 0$ then
 6:         if $type_{new} =$ "$CLOUD$" then
 7:             $category \leftarrow new\_obj.category$
 8:             $bbox \leftarrow curr\_obj.bbox$
 9:         end if
10:         if $type_{new} =$ "$EDGE$" then
11:             $category \leftarrow curr\_obj.category$
12:             $bbox \leftarrow new\_obj.bbox$
13:         end if
14:         $o \leftarrow \{category, bbox, new\_obj.score\}$
15:         $result.append((o, type_{new}))$
16:     else
17:         $result.append((new\_obj, type_{new}))$
18:         if $type_{new} =$ "$CLOUD$" then
19:             $result.append((curr\_obj,$ "$EDGE$"$))$
20:         end if
21:     end if
22: end for
23: return $result$
```

Alg. 1. Enhanced edge-cloud fusion algorithm

```
 1: while $true$ do
 2:     $frame \leftarrow next\_frame()$
 3:     $reset\_tracker \leftarrow false$
 4:     if $frame\_count \% detection\_rate = 0$ then
 5:         $edge_{new} \leftarrow request\_edge\_objects\_sync($
                $frame)$
 6:         $cloud_{curr} \leftarrow cloud\ objects\ of\ curr\_objs$
 7:         $curr\_objs \leftarrow fuse\_edge\_cloud\_objects($
                $cloud_{curr}, edge_{new},$ "$EDGE$"$)$
 8:         $reset\_tracker \leftarrow true$
 9:     else
10:         $curr\_objs \leftarrow track\_objects(frame)$
11:     end if
12:     $process\_objects\_received\_from\_cloud()$
13:     if $cloud\ tracking\ result\ is\ available$ then
14:         $cloud_{new} \leftarrow cloud\ tracking\ result$
15:         $edge_{curr} \leftarrow edge\ objects\ of\ curr\_objs$
16:         $curr\_objs \leftarrow fuse\_objects($
                $edge_{curr}, cloud_{new},$ "$CLOUD$"$)$
17:         $reset\_tracker \leftarrow true$
18:     end if
19:     if $reset\_tracker$ then
20:         $set\_tracker\_objects(frame, curr\_objs)$
21:     end if
22:     $frame\_count \leftarrow frame\_count + 1$
23:     $process\_objects(curr\_objs)$
24: end while
```

Alg. 2. Coordination algorithm - Sync mode

for the detected objects, except when d frames have passed without newly detected objects (line 9). Instead of waiting, after sending the frame to the Edge Server, it continues to use object tracking and checks once every frame whether newly detected objects are available from the Edge Server (line 12). If newly detected objects are available, object tracking is used on the frames that have passed since sending a frame to the Edge Server (lines 16-17). This makes the received detected objects up to date with regards to the current frame. The assumption is that since the Edge Server provides up-to-date, lower-accuracy detected objects, only a few frames will pass between sending a frame and receiving the detected objects. This object tracking fills the small gap introduced by these frames passing and by the async mode.

$process\_objects\_received\_from\_cloud()$ checks whether objects have been received from the Cloud Server. If this is the case, it sends them to the cloud tracking process and triggers the next asynchronous request for cloud objects.

C. Multi-object tracker

A custom CPU-based multi-object tracker, which manages a list of objects to track, is used by the Edge Device. The multi-object tracker uses a single object tracker for each object to track. As such, potentially many single-object trackers are used at the same time. It was found that using many CSRT or KCF trackers makes it infeasible to reach the desired frames per second (FPS) on limited edge devices, like the Nvidia Jetson TX2. For this reason, the MOSSE tracker is leveraged instead.

When adding an object to track to the multi-object tracker, the tracker is created and initialized with the bounding box of the detected object. This collection of MOSSE trackers is used when object tracking is leveraged on new frames. If the object to track has a score lower than a threshold, no tracker is created, meaning only objects with a score greater than or equal to this threshold are tracked. This is done to reduce CPU usage and to be able to process frames at the desired rate.

For objects to track with a score below this threshold, the behaviour is dependent on the configuration of the multi-object tracker. The tracker can assume that the objects do not move and decay their score. As there are frequent requests for detected objects from the Edge Server, there are only a few frames where tracking is used in between those detected objects. In these few frames, many objects will not move drastically due to the spatiotemporal correlation of video frames, which is why this option was introduced. The same behaviour is used when tracking of a single object fails, meaning it is assumed that the object does not move and its score is decayed, when the said option is used. Alternatively, when the score is too low or tracking of a single object fails, the tracker can be configured to drop the object. The main object tracker uses the option to assume no movement and the cloud object tracker uses the drop option. The objects tracked

```
1: while $true$ do
2:     $frame \leftarrow next\_frame()$
3:     $reset\_tracker \leftarrow false$
4:     if $no\ edge\ request\ in\ progress$ then
5:         $request\_edge\_objects\_async(frame)$
6:         $frames\_until\_current \leftarrow []$
7:     end if
8:     $frames\_until\_current.append(frame)$
9:     if $length(frames\_until\_current) \geq detection\_rate$ then
10:        $wait\ for\ objects\ from\ edge$
11:    end if
12:    if $objects\ received\ from\ edge$ then
13:        $edge_{new} \leftarrow objects\ from\ edge$
14:        $cloud_{curr} \leftarrow cloud\ objects\ of\ curr\_objs$
15:        $objs \leftarrow fuse\_objects($
               $cloud_{curr}, edge_{new}, \text{“}EDGE'')$
16:        $set\_tracker\_objects($
               $head(frames\_until\_current), objs)$
17:        $curr\_objs \leftarrow track\_objects\_until\_current($
               $tail(frames\_until\_current))$
18:    else
19:        $curr\_objs \leftarrow track\_objects(frame)$
20:    end if
21:    $process\_objects\_received\_from\_cloud()$
22:    if $cloud\ tracking\ result\ is\ available$ then
23:        $cloud_{new} \leftarrow cloud\ tracking\ result$
24:        $edge_{curr} \leftarrow edge\ objects\ of\ curr\_objs$
25:        $curr\_objs \leftarrow fuse\_objects($
               $edge_{curr}, cloud_{new}, \text{“}CLOUD'')$
26:        $reset\_tracker \leftarrow true$
27:    end if
28:    if $reset\_tracker$ then
29:        $set\_tracker\_objects(frame, curr\_objs)$
30:    end if
31:    $frame\_count \leftarrow frame\_count + 1$
32:    $process\_objects(curr\_objs)$
33: end while
```

Alg. 3. Coordination algorithm - Async mode

by the main object tracker are updated frequently, as such it was found that said assumption works well here. For the cloud object tracker, it was found that dropping is preferable as the detected objects are too outdated to assume no movement.

IV. Implementation

The Edge Device and Edge Server components are implemented using Python. jetson-inference is used for detecting objects on the edge, leveraging the edge object detection model. It uses TensorRT to optimize and run networks on GPUs. pycocotools and scipy are used to implement the edge-cloud fusion algorithm. A collection of the single object trackers provided by opencv-contrib-python is used for multi-object tracking. In particular, the MOSSE tracker is leveraged. The communication between the Edge Device and Edge Server is implemented using ZeroMQ sockets via pyzmq. When using IPC, the ZeroMQ local inter-process communication transport is used. Otherwise, the ZeroMQ unicast transport using TCP is leveraged. The Cloud Server corresponds to a model being served by TorchServe in the cloud, on a machine with access to data center GPUs.

The models used by the Edge and Cloud Server are trained using the VisDrone-DET2019 dataset [1]. It consists of images captured by drones. MobilenetV2 SSD [7] is used as the edge model. Either Faster R-CNN [8] or Swin transformer [4] is used as the cloud model. The MobileNetV2 SSD which was trained works with images of size 512x512 as its input, given to it by the Edge Server. Images of size 1333x800 are sent to the Cloud Server and used by the cloud models. The tools provided by [9] are used to train the MobileNetV2 SSD and to convert it to an ONNX graph. Faster R-CNN and Swin transformer are trained using the mmdetection toolbox [10].

V. Evaluation

VATE is evaluated using 4 experiments. In our experiments, VATE processes videos of the VisDrone-VID2019 test-dev dataset [2]. Accuracy is measured using mAP@50. This is repeated using different modes, models, and configurations. The Edge Device and Edge Server are located in Vienna. The Cloud Server is deployed on an AWS EC2 instance of type p3.2xlarge in the Frankfurt region. For the deployment option with Edge Device and Edge Server on the same physical device, the Edge Device and Edge Server components are running on an Nvidia Jetson TX2. For the deployment option with Edge Device and Edge Server on different physical devices, the Edge Server component is running on an Nvidia Jetson TX2 and the Edge Device component is running on a laptop. Both physical devices are connected to the same WiFi network. The laptop represents any physical device with a sufficiently powerful CPU for the object tracking.

Scenarios are described as $\{model\}$-$\{mode\}$-$\{communication\}$-$\{*fusion\}$, where

1) $model \in \{faster\text{-}rcnn, swin\text{-}t\}$, representing the cloud model used.
2) $mode \in \{sync, async\}$, representing the mode used. For sync mode, a detection rate $d = 5$ is used. For async mode, a detection rate $d = 10$ is used.
3) $communication \in \{ipc, tcp\}$, representing whether IPC or TCP is used. IPC implies that Edge Device and Server are on the same physical device, whereas TCP implies that they are on different physical devices.
4) $fusion$ can optionally specify that the original edge-cloud fusion algorithm is used, referring to Algorithm 1 without the lines 18-20. If omitted, the enhanced edge-cloud fusion algorithm is used.

The results obtained by the scenarios are used for the evaluation of the experiments. 8 scenarios are tested: faster-rcnn-sync-ipc, swin-t-sync-ipc, faster-rcnn-sync-ipc-original-fusion, swin-t-sync-ipc-original-fusion, faster-rcnn-async-tcp, swin-t-async-tcp, faster-rcnn-sync-tcp, swin-t-sync-tcp

A. Experiment: VATE compared to REACT

The goal of this section is to compare VATE to REACT with regards to accuracy, where the enhanced edge-cloud fusion algorithm is the primary contribution of this work. To represent the approaches used by REACT, the *-sync-ipc-original-fusion scenarios are used. These represent REACTs architecture, coordination algorithm and fusion algorithm. The VisDrone-VID2019 dataset, which was used to evaluate REACT, is used here. faster-rcnn-sync-ipc-original-fusion achieves results comparable to the ones reported by REACT for the model combination MobileNetV2 SSD + Faster R-CNN.

Said scenarios are compared to the *-sync-ipc scenarios, where the enhanced fusion algorithm is used. As such, the enhancement of the fusion algorithm is evaluated in isolation. Figure 6 summarises the edge-cloud fusion algorithm results.

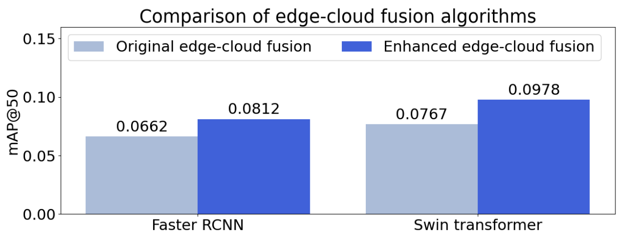

Fig. 6. Edge-cloud fusion: faster-rcnn-sync-ipc-original-fusion next to faster-rcnn-sync-ipc, swin-t-sync-ipc-original-fusion next to swin-t-sync-ipc

When using Faster R-CNN as the cloud model, the enhanced edge-cloud fusion leads to a 22.7% increase in accuracy, when compared to the original edge-cloud fusion. For Swin transformer, the increase in accuracy is 27.5%. As such, when comparing the approaches used by VATE and REACT using the same edge and cloud model, VATE achieves an up to 27.5% higher accuracy than REACT, attributed to the enhanced edge-cloud fusion algorithm.

The enhancement of the edge-cloud fusion algorithm is that a detected object from the edge model is kept even if there is no corresponding detected object from the cloud model. More weight is given to the edge model, which provides up-to-date, lower accuracy object detections. They are of lower accuracy compared to the cloud model, but they are not of low accuracy with current edge object detection models. This idea of giving more weight to the edge can be generalized and applied to tasks other than object detection. Approaches leveraging redundant computations on the edge and the cloud are likely applicable to other tasks, like human pose-estimation or instance segmentation [6].

By using different model combinations, REACT achieves higher accuracy, in particular when using more capable edge models which require more capable edge devices. Comparing the cloud models based on the results reported by REACT, Swin transformer lands between CenterNet [11] and RetinaNet [12] in terms of added accuracy for this use case.

B. Experiment: Impact of offloading edge object detection

The ability to offload edge object detection makes VATE support varying edge device capabilities and constraints like battery life. The goal of this section is to compare sync and async mode concerning the impact that offloading of edge object detection has on the time to process frames. Measuring said impact is done by measuring how often the frame rate drops below m, the desired frame rate. The focus is on videos recorded at a constant frame rate m. When displaying a video while VATE is processing it, such a frame rate drop is noticeable in the form of lags and is highly undesirable for low-latency object detection.

For the VisDrone-VID2019 dataset, $m = 24$ is assumed. An FPS value is calculated for every frame and the distribution of the FPS values is analysed. The FPS value v_i for a frame i is calculated as $v_i = 1/f$, where f represents the frame processing time in fractional seconds. FPS values v_i greater than m are set to m, as the desired frame rate is achieved. If this is not achieved, then the system cannot process frames at the desired rate, leading to lags. For each scenario, the standard deviation of these FPS values is calculated. As most frames are processed at or above the desired rate, a lower standard deviation indicates fewer lags in the video. Figure 7 summarises the edge object detection offloading results.

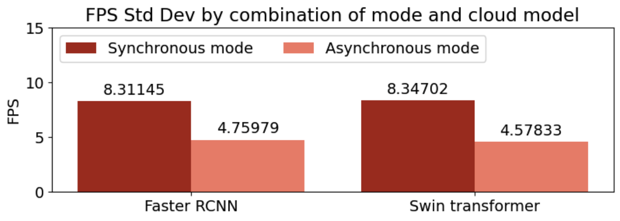

Fig. 7. FPS values: faster-rcnn-sync-tcp next to faster-rcnn-async-tcp, swin-t-sync-tcp next to swin-t-async-tcp

When offloading edge object detection and using Faster R-CNN as the cloud model, async mode leads to a 42.7% decrease in standard deviation, when compared to sync mode. For Swin transformer, the standard deviation decrease is 45.2%. The lag, which is noticeable in the video when using sync mode, disappears with async mode. The idea of providing edge devices with asynchronous edge and cloud object detections is demonstrated to work by VATE. The approach of offloading to edge and cloud servers can be applied to tasks other than object detection as well.

C. Experiment: Accuracy of VATE

The goal of this section is to evaluate the accuracy impact of offloading edge object detection, compared to doing on-device edge object detection, and to demonstrate that both sync and async modes benefit from more capable cloud models. The accuracy of VATE is evaluated by looking at different combinations of modes and models. Figure 8 summarises the accuracy results.

For sync mode, using Swin transformer as the cloud model results in a 20% higher accuracy compared to using Faster R-CNN. For async mode, a 14.5% accuracy increase is achieved. This shows that sync and async mode benefit from more capable cloud models. Said offloading comes with a loss in

TABLE I
VATE COMPARED TO RELATED SYSTEMS

Features	VATE	REACT [6]	Glimpse [13]	Marlin [14]	Edge-Assist. [15]	EC²Detect [16]
detection on edge device	Yes	Yes	No	Yes	No	No
offloading detection to edge or cloud server	Yes	Yes	Yes	No	Yes	Yes
offloading detection to edge and cloud server	Yes	No	No	No	No	No
redundant detection on edge and cloud server	Yes	Yes	No	No	No	No

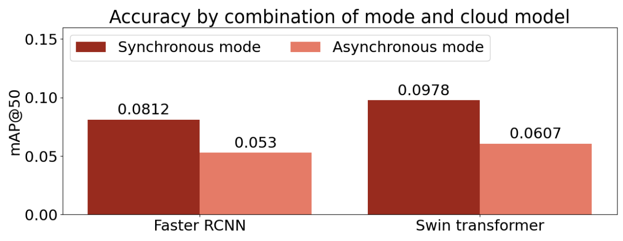

Fig. 8. Accuracy: faster-rcnn-sync-ipc, faster-rcnn-async-tcp, swin-t-sync-ipc and swin-t-async-tcp

accuracy, compared to doing on-device edge object detection, as the edge object detections are delayed due to the network and as more object tracking is required. Going from sync mode with IPC to async mode with TCP leads to a 34.7% decrease in accuracy when using Faster R-CNN. The decrease is 37.9% when using Swin transformer.

D. Experiment: System-related metrics of VATE

The CPU and network usage of VATE are evaluated next.

The approach of using CSRT trackers, as suggested by REACT, turned out to be problematic for VATE, as it is running on less capable hardware. The density of small objects in the dataset leads to a need for many single-object trackers. When using a multi-object tracker based on CSRT trackers, the average CPU utilization of VATE goes to $\geq 95\%$ on our edge infrastructure and the FPS drops below 1. By leveraging MOSSE trackers instead, VATE is able to achieve 24 FPS while running at $60\% - 70\%$ CPU utilization on average.

The network bandwidth used by the communication between Edge Device and Cloud Server is 291 KB/s on average, with the maximum value being 492 KB/s. When offloading edge object detection and using async mode, the network bandwidth used by the communication between Edge Device and Edge Server is 355 KB/s on average, with the maximum value being 736 KB/s. These numbers suggest that VATE can be used with many network technologies, like 4G and 3G.

VI. Related Work

Past works [17] show a trend that the most accurate models often come with the lowest FPS values. Currently, streaming video analytics systems have to choose a middle ground between model accuracy and achievable FPS. Low FPS is a severe issue when edge devices are used, often making streaming applications impractical.

The exact desired FPS for streaming video analytics use cases, like ones involving drones, varies. This work assumes it to be between 10 and 40 FPS, for drones specifically around 24 FPS (see for example [17]). The Nvidia Jetson TX2 is used to represent hardware in drones. In combination, this leads to a realistic restriction in the models which can be used.

Existing systems, apart from REACT [6], either run less accurate models which are fast enough on the edge device or run no GPU intensive models on the edge device at all, offloading this to an edge or cloud server. Marlin [14] opted for the former option, whereas Glimpse [13], EC²Detect [16] and Edge-Assist. [15] opted for the latter one.

REACT proposed a combination of both approaches, where redundant computations in the edge and cloud are leveraged to improve the accuracy of such systems. VATE is built on this idea of REACT and enhances it as described. Sync and async mode are introduced for edge object detection in VATE. Respective coordination algorithms are given by this work.

Significantly higher mAP values have been achieved on the VisDrone datasets (see for example [17]). To get these results, either more capable models and hardware are leveraged, low FPS is accepted or object detection is used on every frame, which is infeasible for many edge devices [6]. These aspects differentiate those approaches from our research.

Table I compares VATE with the related systems. Aspects from these related systems can be seen as complementary and used to improve the performance of VATE. This is the same for VATE as is described by REACT: VATE benefits from improved hardware in edge devices and this work is applicable as long as there exists a performance gap between the edge and cloud models. Models are continuously improving [1]. More capable edge models improve the accuracy of the system and decrease the impact of cloud unavailability. More capable cloud models and advances in multi-object tracking [18] further improve the accuracy of the system.

Furthermore, improvements in network latency and edge server infrastructure reduce the impact that offloading of edge object detection has on the accuracy of VATE.

VATE belongs to a subset of edge intelligence called edge video analytics (EVA) [3]. It fits the definition of EVA given in [3], as the hierarchy of end, edge, and cloud devices is leveraged by VATE to improve the accuracy and responsiveness of it as a video analytics system. In the context of edge intelligence [5], systems like VATE deal with challenges similar to the challenges of AI for edge research. The balance between optimality and efficiency is particularly prevalent. The goal to be optimized by the systems is Quality of Experience, determined by considering performance, cost, efficiency, reliability, and privacy. The current focus of such systems primarily lies on improving performance or cost (for example computation cost, energy consumption). For the VisDrone datasets in particular, performance remains a challenge, as is reflected by the mAP values [1] [2]. Video analytics systems have broad implications for privacy and must be used ethically. This is discussed in [19], which presents a privacy-preserving approach for video analytics queries.

VII. Conclusion & Future Work

This work introduced VATE, an edge-cloud system for object detection in real-time video streams inspired by REACT [6]. The system presents an enhanced edge-cloud fusion algorithm, a novel multi-object tracker, and coordination algorithms. It runs on less capable and cheaper edge hardware while achieving up to 27.5% higher accuracy than REACT when using the same edge and cloud model combination on the challenging VisDrone-VID2019 dataset [2]. VATE supports varying capabilities of edge devices. The offloading of edge object detection, in addition to cloud object detection, is enabled.

However, building systems to enable low-latency, high-accuracy object detection on a wide selection of edge devices remains a challenge. To push the accuracy further, usage of advanced multi-object tracking approaches, like StrongSORT [20], can be investigated on the appropriate edge devices. The privacy-related aspects can be incorporated into VATE to make it more suitable for real-life scenarios. Like with REACT, usage of the adaptive streaming perception [21] can be explored. Finally, we intend to investigate the serverless computing paradigm for efficient data processing in the edge-cloud continuum [22] [23].

Acknowledgments: This work is sponsored by the Austrian Research Promotion Agency (FFG), under the project No. 903884.

References

[1] D. Du, P. Zhu, L. Wen, X. Bian, H. Lin, Q. Hu *et al.*, "VisDrone-DET2019: The Vision Meets Drone Object Detection in Image Challenge Results," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 213–226.

[2] P. Zhu, D. Du, L. Wen, X. Bian, H. Ling, Q. Hu *et al.*, "VisDrone-VID2019: The Vision Meets Drone Object Detection in Video Challenge Results," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 227–235.

[3] R. Xu, S. Razavi, and R. Zheng, "Edge Video Analytics: A Survey on Applications, Systems and Enabling Techniques," *IEEE Communications Surveys and Tutorials*, vol. 25, no. 4, pp. 2951–2982, 2023.

[4] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang *et al.*, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2021, pp. 9992–10 002. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV48922.2021.00986

[5] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[6] A. Ghosh, S. Iyengar, S. Lee, A. Rathore, and V. N. Padmanabhan, "REACT: Streaming Video Analytics On The Edge With Asynchronous Cloud Support," in *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, ser. IoTDI '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 222–235. [Online]. Available: https://doi.org/10.1145/3576842.3582385

[7] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4555207

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 91–99.

[9] "SSD-based Object Detection in PyTorch," https://github.com/dusty-nv/pytorch-ssd, online, accessed 25-November-2023.

[10] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li *et al.*, "MMDetection: Open MMLab Detection Toolbox and Benchmark," *arXiv preprint arXiv:1906.07155*, 2019.

[11] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as Points," *ArXiv*, vol. abs/1904.07850, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:118714035

[12] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2017, pp. 2999–3007. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV.2017.324

[13] T. Y.-H. Chen, H. Balakrishnan, L. Ravindranath, and P. Bahl, "GLIMPSE: Continuous, Real-Time Object Recognition on Mobile Devices," *GetMobile: Mobile Comp. and Comm.*, vol. 20, no. 1, p. 26–29, jul 2016. [Online]. Available: https://doi.org/10.1145/2972413.2972423

[14] K. Apicharttrisorn, X. Ran, J. Chen, S. V. Krishnamurthy, and A. K. Roy-Chowdhury, "Frugal Following: Power Thrifty Object Detection and Tracking for Mobile Augmented Reality," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, ser. SenSys '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 96–109. [Online]. Available: https://doi.org/10.1145/3356250.3360044

[15] L. Liu, H. Li, and M. Gruteser, "Edge Assisted Real-Time Object Detection for Mobile Augmented Reality," in *The 25th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3300061.3300116

[16] S. Guo, C. Zhao, G. Wang, J. Yang, and S. Yang, "EC²Detect: Real-Time Online Video Object Detection in Edge-Cloud Collaborative IoT," *IEEE Internet of Things Journal*, vol. 9, no. 20, pp. 20 382–20 392, 2022.

[17] Z. Saeed, M. H. Yousaf, R. Ahmed, S. A. Velastin, and S. Viriri, "On-Board Small-Scale Object Detection for Unmanned Aerial Vehicles (UAVs)," *Drones*, vol. 7, no. 5, 2023. [Online]. Available: https://www.mdpi.com/2504-446X/7/5/310

[18] N. Dardagan, A. Brdjanin, D. Dzigal, and A. Akagic, "Multiple Object Trackers in OpenCV: A Benchmark," *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*, pp. 1–6, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:238582680

[19] F. Cangialosi, N. Agarwal, V. Arun, S. Narayana, A. Sarwate, and R. Netravali, "Privid: Practical, Privacy-Preserving Video Analytics Queries," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 209–228. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/cangialosi

[20] Y. Du, Z. Zhao, Y. Song, Y. Zhao, F. Su, T. Gong *et al.*, "Strongsort: Make deepsort great again," *IEEE Transactions on Multimedia*, 2023.

[21] A. Ghosh, A. U. Nambi, A. Singh, H. Yvs, and T. Ganu, "Adaptive Streaming Perception using Deep Reinforcement Learning," *ArXiv*, vol. abs/2106.05665, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235390415

[22] S. Nastic, S. Dustdar, R. Philipp, F. Alireza, and T. Pusztai, "A Serverless Computing Fabric for Edge & Cloud," in *4th IEEE International Conference on Cognitive Machine Intelligence (CogMi)*, 2022.

[23] P. Raith, S. Nastic, and S. Dustdar, "Serverless Edge Computing—Where We Are and What Lies Ahead," *IEEE Internet Computing*, vol. 27, no. 3, pp. 50–64, 2023.