2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 979-8-3503-6460-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/IPDPSW63119.2024.00116

FrameFeedback: A Closed-Loop Control System for Dynamic Offloading Real-Time Edge Inference

Matthew Jackson
Department of Computer Science
Virginia Tech
Blacksburg, VA
mnj98@vt.edu

Bo Ji
Department of Computer Science
Virginia Tech
Blacksburg, VA
boji@vt.edu

Dimitrios S. Nikolopoulos
Department of Computer Science
Virginia Tech
Blacksburg, VA
dsn@vt.edu

***Abstract*—Despite the demand for real-time deep learning applications such as video analytics at the edge, resource-constrained edge devices can largely not process video streams at their source frame rate. However, deep learning execution can be accelerated by offloading tasks to a nearby edge server equipped with a GPU. For a realistic edge system with variable network conditions and server load, we consider optimally partitioning the frames from a video stream between local processing and offloading to maximize the throughput of an edge device under a real-time deadline. To do this, we show that we can simplify the influences on processing latency into a single relevant metric and dynamically determine an appropriate offloading rate using a latency-based feedback control mechanism. Our controller settles on the optimal offloading rate without knowing network conditions, resource availability, or application computation cost. Our measurements show that our feedback controller balances sensitivity and overcorrection given a variety of network and load conditions set. We also show that our controller's Quality of Service outperforms state-of-the-art baselines and approaches.**

***Index Terms*—edge computing, adaptive offloading, real-time deep learning, feedback control**

I. Introduction

The intriguing opportunities for real-time deep learning (DL) applications at the edge clash with the harsh reality that computation, communication, and accuracy are limited with most edge devices. A wide variety of video analytics workloads in surveillance, industry, UAVs, IoT, and AR could effectively leverage advances in DL if not for the severe resource constraints faced with edge computation [1]–[5]. Real-time workloads are especially sensitive to these constraints due to tight end-to-end latencies often imposed on DL inference results. Offloading these tasks can mitigate the challenges of running these workloads on edge devices [6].

To maximize throughput, DL inference latency needs to be minimized. However, edge devices, characterized by limited resources [7], may not be able to process frames with a reasonable latency or frame rate. With a typical frame rate of 30 frames per second, there are about 33 ms between frames. However, depending on the DL model and the computing resources of the edge device, inference for a single frame can take hundreds of milliseconds to process [8].

A compelling solution to this problem is offloading [6], [9], [10] where we can leverage the resources of a nearby edge node or server. An edge server, less restricted by resource constraints, can accelerate DL inference throughput, especially when equipped with a GPU [11]. However, a single device's video stream may under-utilize modern hardware and fragment expensive hardware resources [12]. Leveraging multi-tenancy by allowing many devices to offload can efficiently utilize the server.

Real-time and multi-tenant constraints make offloading less flexible. When there is continuous demand for inference results, quality of service (QoS) becomes sensitive to end-to-end latency [13]. Multi-tenant offloading helps improve server resource utilization, but saturating server resources compromises QoS due to higher service latency [14]. Variable network conditions, device variability, and unpredictable system load make end-to-end offload latency unstable [15]. A reactive offloading policy for this domain must be developed and tuned to maximize frame rate without violating latency constraints [16].

Contributions

We propose a novel system that addresses the challenges of multi-tenancy, constrained local resources, limited offloading availability, and variable network conditions for real-time DL workloads that operate on fast and concurrent video streams:

To address these constraints, we use the rate of end-to-end latency violations as a critical metric in determining the suitability of an individual edge device's offloading policy. With little to no timeouts, offloading can be scaled up (and subsequently, as the offloading rate approaches the source frame rate, the local inference rate can scale down). We propose a novel real-time feedback control system, FrameFeedback, to regulate the offload rate. FrameFeedback can provide stable control under nominal conditions and reconfigure policies when QoS violations occur [17].

Our work makes the following contributions.

1) We identify the factors impacting the QoS in real-time DL applications at the edge and under multi-tenancy. We demonstrate that these factors are not taken into account in state-of-the-art approaches for total or partial offloading.
2) We use end-to-end latency violation rate as a new metric for evaluating offloading policies.

979-8-3503-6460-6/24/$31.00 ©2024 IEEE
DOI 10.1109/IPDPSW63119.2024.00116

TABLE I: Notation

Notation	Description
F_s	Source frame rate
P	Total inference processing rate
P_l	Local processing rate
P_o	Offloading rate
L	Maximum tolerable offloading latency
T	Rate of offloaded frames that time out
T_n	Rate of offloading timeouts due to network
T_l	Rate of offloading timeouts due to server load

TABLE II: P_l of our different Raspberry-Pis

	3B Rev. 1.2	4B Rev. 1.2	4B Rev. 1.4
CPUs	4	4	4
Speed	1200 MHz	1500 MHz	1800 MHz
Memory	909 Mi	3.7 Gi	7.6 Gi
MobileNetV3Small P_i	**5.5**	**13**	**13.4**
EfficientNetB0 P_i	**1.8**	**2.5**	**4.2**

3) We present FrameFeedback, a novel closed-loop control system that uses system feedback to set an optimal offload policy under current system constraints. FrameFeedback is available as open source software for Raspberry Pi and NVIDIA GPUs[1].
4) We evaluate FrameFeedback on a realistic edge system against baseline approaches and demonstrate its superiority under suboptimal server load and network conditions, where FrameFeedback outperforms a state-of-the-art system (DeepDecision) by more than a factor of two.

The rest of this paper is organized as follows. Section V reviews related literature. We describe the details of the FrameFeedback system and controller in Sections II and III with evaluation following in Section IV. We discuss future work in Section VI.

II. Background

In this section, we provide background for FrameFeedback and discuss the constraints that multi-tenant edge AI systems operate under. We also discuss our system configuration. We refer the reader to Figure 1 to visualize our system and the specific interactions between devices and servers. Table I provides the relevant notation.

A. System Model

We consider an environment that delivers AI services to users through user mobile edge devices, edge servers, and wireless networks. We seek resource allocation approaches that address the following combination of challenges and constraints:

1) Multi-Tenancy: We consider a system where users cannot have dedicated resources to offload AI tasks [18]. The combined load from all user devices also impacts the server's prediction time, so we cannot assume that our only source of offloading latency comes from networking. We also consider multiple classification workloads with different computational costs, latency, and quality requirements.

[1]https://github.com/mnj98/edge-inference.

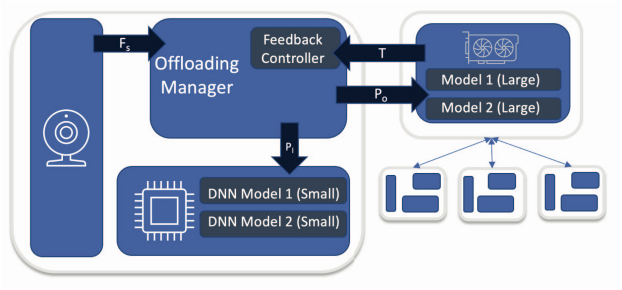

Fig. 1: A diagram of the system which shows edge devices (left & bottom right) and edge servers (top right). The feedback controller inside the edge device directs the system on how to react to varying conditions based on deadline violation rates. On the top right, the GPU-equipped Edge server supports offloading for multiple devices.

2) Constrained Local Resources: Edge devices are resource restricted, whether due to power limits, weak hardware, or lack of accelerators [19]. For the specific application domain that we study in this paper –object classification and identification from real-time video streams at high frame rates, Table II provides examples from our Raspberry-Pis running classification on frames of size 224×224. Referring to Table I, our system assumes that, in all user devices that capture video, $P_l < F_s$. In other words, the local processing rate, P_l, is slower than the source frame rate F_s.

3) Limited Offloading Availability: Specific workloads may saturate a server, thus causing QoS violations [20]. When the workload fully saturates the system, the system should respond by reducing offloading and distributing the available capacity fairly among clients. Our notation represents the load-induced time-out rate (and rejections) as T_l.

4) Variable Network Conditions: Most edge devices connect to the network wirelessly. Movement and sources of interference can make connections unreliable. Bandwidth limits, packet loss, or other network delays cause complex end-to-end offload latency patterns that can be difficult to predict [21]. Our notation includes T_n for network-induced timeouts that we consider in addition to T_l.

5) Quality of Service: QoS signifies how well the edge device can complete time-sensitive tasks enabled by DL inference results. Offloading is necessary to maximize task processing time (P), but our system assumptions cannot guarantee successful offloading. Therefore, when calculating P, the system must account for T and be careful not to offload excessively when T is high. Consider if $P_o = F_s$ and $T > F_s - P_l$ at some point. This means that the device attempts to offload all its frames, but the effective value of P is lower than P_l. The system should never fail to react to this scenario, and the controller should always strive to keep $P \geq P_l$.

TABLE III: Top-1 Model Accuracy [27], [28]

Model	Top-1 Accuracy
EfficientNetB0	77.1%
EfficientNetB4	82.9%
MobileNetV3Small	67.4%
MobileNetV3Large	75.2%

We neither measure nor optimize power usage with our QoS model, but note that, in general, effective offloading leads to lower power usage on edge devices [6], [22]. Our experiments show that Raspberry Pi CPU usage drops from 50.2% to 22.3% on average when transitioning from local execution to offloading.

B. Offloading Latency

We consider two categories of delay: multi-tenant contention and networking delay. The number of connected clients and their respective offloading request rates dominate the system load at the edge server. As the load increases, the time to execute each batch of DL inference increases, and interprocess communication increases. ATOMS [23] considers many of these factors but at the cost of an exhaustively complex system model that requires many different subsystems, including a device-server clock synchronization and a resource reservation system, to operate.

One benefit of FrameFeedback is that it does not need to directly measure or understand the sources of the delay, since it properly reacts when the delay arises. We make two observations: First, an offloaded inference task is successful if its result returns before its deadline. On a per-frame basis, the only relevant measure of QoS is binary. Second, the controller's reaction to T depends entirely on T"s value, and any changes made to P_o should be dependent on T.

In this work, we consider 250ms as a justifiable deadline for a real-world, real-time video processing system. This assumption aligns with recent work on video analytics [24], [25]. Furthermore, we consider pipelined offloading to overlap frame processing for higher throughput.

C. Classification Applications

We use Image Classification models from the Keras [26] framework to model our system. We chose MobileNetV3 [27] and EfficientNets [28] because they are fast, well documented and applicable to many domains.

D. Model Accuracy

Table III presents details on the accuracy of our selected models. These accuracies may be lower than those of non-constrained models, but are consistent with models previously used in edge computing [29].

One way to increase the accuracy of the model is to improve the information in the classified image. The default resolution with which all of these models have been pre-trained with is 224×224 except for EfficientNetB4, which accepts images with a resolution of 380× 380, and allows for a variable input size. Most modern cameras generate images with larger resolutions and resize them to fit as input to classification models, so using a larger resolution closer to the source could improve accuracy. When offloading images for classification, it is common to compress them [30], [31]. Using lighter compression can improve accuracy. Both techniques have a significant downside when offloading, because both increase the number of bytes per frame that need to be transferred.

III. Feedback Controller

In this section, we describe the FrameFeedback controller covering the foundations of the controller, showing our necessary modifications, and justifying the controller's settings. We adapt the basic theory of a closed-loop PID controller to show that a modified and tuned controller is a good choice for our problem domain.

A. A Feedback Controller for Real-Time Offloading

A closed-loop or feedback controller applies continuous control to a system by measuring a process variable (PV) and applying a correction such that PV converges to a set point (SP). A Proportional-Integral-Derivative (PID) controller computes an error $e(t)$ where:

$$e(t) = SP(t) - PV(t) \tag{1}$$

And a control function:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt} \tag{2}$$

where K_P, K_I, K_D are the proportional, integral, and derivative coefficients, respectively [32].

FrameFeedback uses a discrete feedback controller to determine a suitable offloading rate for each edge device using measured information such as P_o and T. When T is zero (no offloading latency violations or dropped requests), offloading will likely continue or increase.

1) PD Controller: For our controller, we observe that the integral term of conventional PID controllers is unnecessary. The characteristics of this term, the consideration of the past, and the variable external forces on PV, are not factors in our system. Since our controller's input is the average of T from the last few seconds, we already consider past data and show that PV does not experience variable forces.

Therefore, $K_I = 0$, and we modify Equation 2 to:

$$u(t) = K_P e(t) + K_D \frac{de(t)}{dt} \tag{3}$$

Our goal of maximizing the processing rate by maximizing available offloading means that our error function $e(t)$ needs to consider two cases:

1) If T is low ($\leq$ 10% of F_s), $e(t)$ needs to be positive
2) If T is high, $e(t)$ needs to be negative

These two cases are challenging to capture with a simple PV function. A balanced approach requires considering each case separately with a piecewise PV:

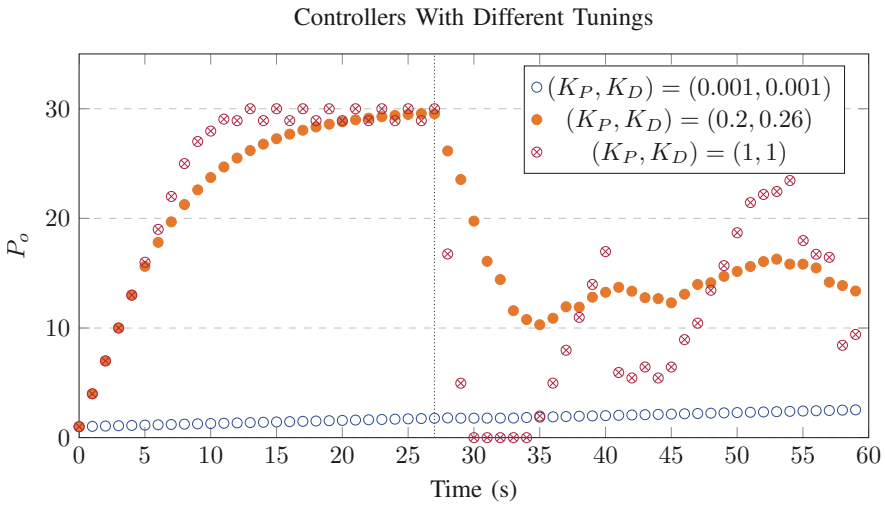

Fig. 2: Offloading rate P_o for controllers with different K_P and K_D coefficients. We introduce packet loss of 7% after 27 seconds.

$$PV = \begin{cases} P_o & T = 0 \\ T + 0.9F_s & T > 0 \end{cases} \tag{4}$$

In this approach, we set $SP = F_s$, so $e(t)$ is linear in both cases.

$$e(t) = \begin{cases} F_s - P_o & T = 0 \\ 0.1F_s - T & T > 0 \end{cases} \tag{5}$$

Our PD controller will drive the offloading rate to F_s, and in the presence of timeouts exceeding 10% of F_s, it will scale back the offloading. Note that $e(t) = 0$ when $T = 0.1F_s$, so P_o will stabilize to $0.1F_s$ when offloading always fails. This does not negatively impact QoS when compared to not trying to offload at all, but it does provide a constant measurement of offloading availability. Therefore, when good conditions return, offloading will immediately begin to increase.

B. Tuning

A traditional tuning approach, such as the Ziegler–Nichols method [33] offers good intuition for tuning our controller. Still, their exact method cannot be applied because it is designed for a full PID controller. Furthermore, due to the changing network and system load conditions, our PV function demonstrates more variance than those they considered.

The procedure to adjust our PD controller was to gradually increase K_P until the controller sensitivity was high and the PV oscillated under constant conditions. Next, we increased K_D to reduce the oscillations and stabilize the system. This follows from the wisdom that increasing K_P increases sensitivity while degrading stability, and increasing K_D decreases overshoot and improves stability [32].

To introduce an additional tuning layer into the controller, we limited $u(t)$ to a specific range of updates. To make the controller sensitive to when T is high, the minimum update to P_o is $\frac{-F_s}{2}$. However, we did not want the controller to react too quickly to increase P_o (the offloading rate), so we imposed a maximum update of $\frac{F_s}{10}$ when increasing P_o. These limits result in the controller improving QoS by reacting more forcefully to timeouts.

TABLE IV: PID Settings

Variable	Value
K_P	0.2
K_I	0
K_D	0.26
Update minimum	$-0.5 * F_s$
Update maximum	$0.1 * F_s$
Measure Frequency	1

Table IV displays the settings of our controller. Figure 2 shows the effect of different settings on the responsiveness and stability of our controller. At first, we see the controllers' behavior under ideal network conditions, and once we introduce packet loss, we can observe how the settings affect stability.

IV. Experiments

A. Testing Configuration

We used the Raspberry-Pis detailed in Table II as our edge devices. For the collection of the data shown in Figures 2, 3 and 4, we use the three Raspberry-Pi's concurrently sending streaming requests to our edge server and evaluated their total inference throughput. We use MobileNetV3 for these tests because it produces the smoothest results. To help reproduce results, we only used the same device and model for data collection. TThe Raspberry Pi runs Debian Linux 11 (bullseye) on kernel 5.15.84-v8+. It has four Cortex-A72 arm64 CPUs, a heat sink, and a fan. We wrote the entire Pi code in Python and used Python 3.9.13. Our Keras models run through

TensorFlow [34] 2.8.0, which we built specifically for the Pi's CPU architecture.

Our edge server runs on an Ubuntu 20.04.5 virtual machine inside of KVM. It has 16 EPYC 7251 CPUs. It has a Tesla V100 GPU running CUDA version 11.8 passed through from the host machine. We use Python 3.8.13 and Tensorflow version 2.9.1.

This configuration is realistic, as hardware and software are widely used in real-world deployments and have been used for evaluation in related work [23].

We did not observe significant differences in throughput or CPU usage when testing with an actual webcam versus sourcing frames from the ImageNet dataset. For convenience and reproducibility, we use ImageNet frames for evaluation.

Adaptive Batching Strategy

Batching can help maximize throughput and hardware utilization, especially when using GPUs with Image Classification models, which are notorious for their low hardware usage [35]. We adaptively vary our batch size based on the request volume to take advantage of batching. ATOMS [23] follows the same approach. Since we have a constant stream of incoming frames, our batching scheme can be simple: construct a batch using all frames (to a limit) that arrived while executing the previous batch. We maintain a request queue that is filled during the execution of a batch, and we fill the next batch with the contents of this queue. As noted, we cannot allow the batch size to grow too large, so we impose a limit of 15 frames for each batch, while rejecting the rest in the queue.

B. Controllers

In addition to FrameFeedback, we evaluate the following controllers:

1) Local Inference: The first of the baselines we compare our FrameFeedback controller with is the local execution only. This is an undesirable solution due to the low throughput and high power usage of computing Image Classification on Raspberry Pis.

2) Always Offload: This baseline is self-explanatory. At all times, we offload all frames to the edge server. Since we disregard any feedback, it is unlikely that this solution will be optimal unless the system conditions are perfect.

3) All-or-Nothing Intervals: Mimicking DeepDecision's [30] approach, we decide at each measurement step (1 second) whether to offload all frames in that interval or to classify frames locally. To make this decision, we follow DeepDecision's intuition and try to keep track of the system state by sending a heartbeat request to profile the latency. If the request is successful (returns before the deadline), we deem the conditions sufficient for offloading.

C. System Variables

We can alter two main system variables to induce timeouts and measure throughput. Changes in the network and server load can induce timeouts in different patterns, and we can change both in different ways.

1) Network: Our primary method for degrading the network connection between Pi and the server is by limiting the bandwidth and introducing packet loss, which has a realistic impact on real edge networks [21]. To artificially inject rate limits and loss, we used NetEm [36], a Linux network emulation tool that allows us to fine-grained control over the emulated network. We could have also used NetEm to add a latency delay to packets, but we believe that rate and loss are better tools to induce timeouts as they are more indirect. Additionally, NetEm delay calculations may not be accurate [36].

To ensure that the rate of deadline violations is not affected by excessively high or low rate and loss values, we decided to limit the bandwidth between 1–10 kbps and set the loss rate to 7% for this experiment. It should be noted that this loss rate may be low for some wireless networks, which can experience packet loss rates in the tens of percentage points [37].

2) Server Load: The second method of injecting latency is by using a high load on the server. While measuring on one device, we can use other devices to put a higher demand on the server. This allows us to measure the system behavior under high multi-tenancy.

Queuing delays, interprocess communication, and increased batch sizes increase the latency of each request. At some level of request volume, batch sizes will saturate the system, leading to rejections. However, batch size limits are set per model, so we hit both model types when measuring controller response under server load.

Combined Network and Server Measurements: Combining both sources of end-to-end latency largely works additively to create more unsuccessful offload requests. Combining the two sources of additional latency can generate various timeout patterns of interest. Due to space considerations, we do not discuss these patterns further here.

TABLE V: Network Variables

Time(s)	Bandwidth (kbps)	Loss (%)
0-30	10	0
30-45	4	0
45-60	1	0
60-90	10	0
90-105	10	7
105+	4	7

D. Network

To compare FrameFeedback to the baselines under ideal and degrading network conditions, we subjected each controller to the same tests. Generating a stream of 4,000 frames at 30 frames per second, we configure NetEm to alter the network conditions at predetermined intervals. Refer to Table V for specific values and to Figure 3 for data.

The first thing to address is that P, the successful inference rate, is noisy for all controllers. This is due to the discrete nature of the frames, which are generated at fixed intervals. Unpredictable and random timeout rates subtract from P, making it noisy even though an average trend is visible.

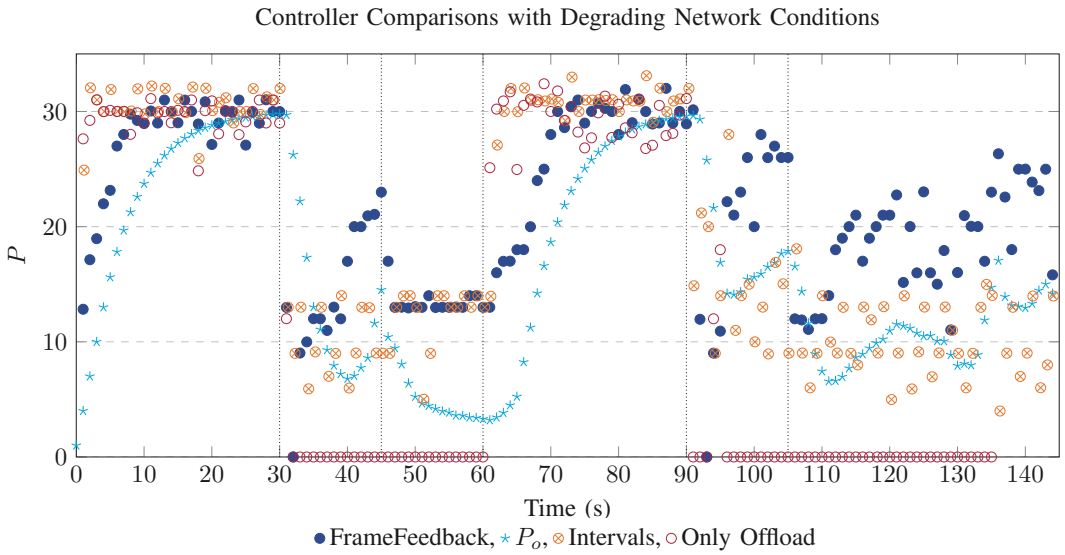

Fig. 3: Total inference throughput P for each of type of controller under the network conditions shown in Table V

TABLE VI: Server Load Configuration

Time(s)	Request Rate
0-10	0
10-20	90
20-35	120
35-50	135
50-60	150
60-75	130
75-90	120
90-100	90
100+	0

We include P_o for FrameFeedback to demonstrate its behavior. The dark blue dots represent $P_o + P_l - T$ and represent the throughput.

Under very high or low network quality periods, FrameFeedback and all-or-nothing intervals have equivalent throughput. However, we can see that under intermediate network conditions, FrameFeedback has a higher throughput because it can find an offload rate that the current conditions can support. Especially around 40 seconds and beyond 90 seconds, FrameFeedback has a better average P (between 50% and up to 3×) compared to the all-or-nothing approach. Clearly, the only-offloading strategy is suboptimal.

E. Server Load

Our testing structure for measuring the effects of multi-tenancy and server load follows our network testing protocol. We generate a stream, at 30 frames per second, of 4,000 frames, but instead of using NetEm to degrade the network, we use other devices to inject request volume. Refer to Table VI for details.

Figure 4 contains the same structure as Figure 3 in that we show the total throughput P for each baseline controller, and for FrameFeedback, we show P and P_o. Beginning with zero, we increase and then decrease the external offloading at identical intervals for each controller. Up until about 150 additional requests, our Pi can fit in some offloading when controlled by FrameFeedback. The other controllers have lower throughput due to their inability to adapt in a fine-grained way.

V. Related Works

In this Section, we review different approaches to real-time DL offloading at the edge.

A. Local Only Deep Learning

Initial works on running DL-based video analytics workloads solely on edge devices show poor performance. Most combinations of devices and models result in inference latencies of up to 2000 milliseconds [38] for data sets like those explored in this paper. This prevents meaningful service when processing real-time video streams.

B. Total Offloading

Some works rely completely on an external edge server to compute real-time DL results. In a single-tenant system, the work in [39] adapts the level of offload based on network conditions. Another single-tenant system presented in [8] describes an object tracking application where the detection stage uses off-loaded DL while computing frame-by-frame tracking locally. Multi-tenant offloading of multiple devices' video streams is explored in [40], but this work lacks exploration of QoS degradation due to networking or total system load.

Considering the constraints of a multi-tenant real-time offloading system, [23] describes a reservation, planning, and scheduling system called ATOMS. Since it requires computing time estimation and clock synchronization, the system is

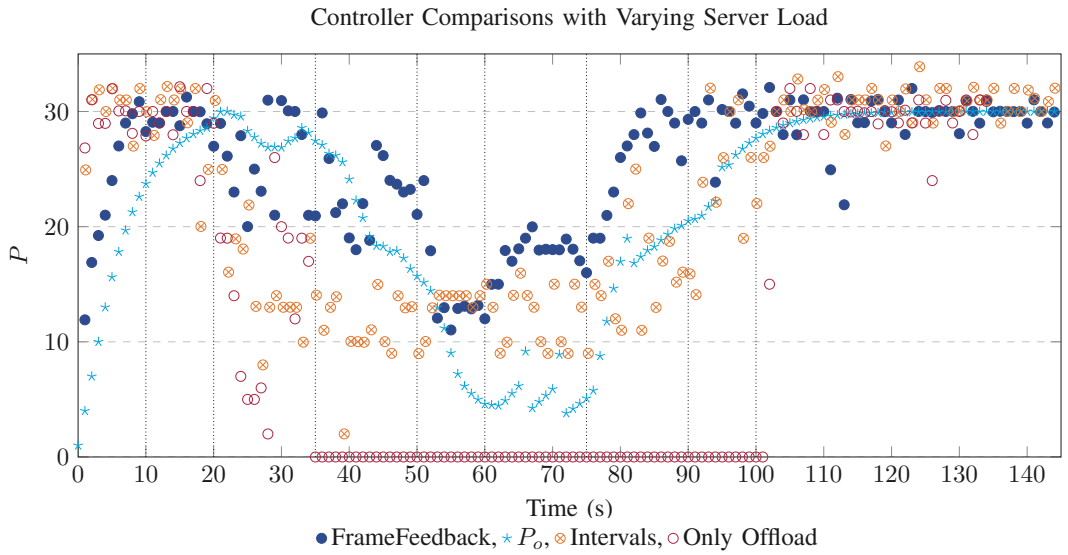

Fig. 4: Total inference throughput P for each of type of controller under the server load shown in Table VI.

complex but robust. ATOMS considers resource contention for CPU workloads and GPU-supported DL workloads, but it lacks rigorous evaluation under variable network conditions and is not an applicable solution for our system.

C. Partial Offloading

Adaptive partial offloading is a promising way to balance system constraints such as network bottlenecks and server load with local device processing. OsmoticGate [31] coordinates edge devices with edge nodes and can adaptively offload DL workloads to the cloud. Although local processing on edge devices is not actually performed, the constraints considered for edge node processing versus cloud offloading are similar to local versus edge node processing. OsmoticGate considers frame chunk size and rate for varying system conditions.

DeepDecision [30] uses all-or-nothing adaptive offloading, where it can offload all frames from a video stream to an edge server if it detects suitable conditions. If latency or power thresholds are exceeded, DeepDecision can adjust the offloading policy or compute the results on the device. Neither OsmoticGate nor DeepDecision adequately describes a way to generate an offloading policy given the constraints of our real-time and multi-tenant edge video analytics.

VI. Conclusion

We presented a system to improve the service of real-time video DL applications in a constrained and variable edge environment by adaptively adjusting offloading. FrameFeedback is an end-to-end latency-sensitive closed-loop feedback controller. When an edge server supports offloading for many resource-constrained devices, FrameFeedback's custom control function guides it toward maximum throughput. FrameFeedback reacts to degradation in system-wide offloading capacity better when compared to baseline approaches.

Acknowledgements:

This material is based upon work supported by the National Science Foundation under Grants No. 2315851 and 2106634, a Sony Faculty Innovation Award (Contract AG3ZURVF), and a Cisco Research Award (Contract 878201).

References

[1] S. P. Rachuri, F. Bronzino, and S. Jain, "Decentralized modular architecture for live video analytics at the edge," in *Proceedings of the 3rd ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges*, ser. HotEdgeVideo '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 13–18. [Online]. Available: https://doi.org/10.1145/3477083.3480153

[2] 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

[3] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3132211.3134459

[4] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, p. 58–67, jan 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.3641638

[5] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, "Edge computing with artificial intelligence: A machine learning perspective," *ACM Comput. Surv.*, vol. 55, no. 9, jan 2023. [Online]. Available: https://doi.org/10.1145/3555802

[6] G. Klas, "Edge computing and the role of cellular networks," *Computer*, vol. 50, no. 10, pp. 40–49, 2017.

[7] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[8] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 155–168. [Online]. Available: https://doi.org/10.1145/2809695.2809711

[9] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, feb 2019. [Online]. Available: https://doi.org/10.1145/3284387

[10] X. Wang, J. Ye, and J. C. Lui, "Decentralized task offloading in edge computing: A multi-user multi-armed bandit approach," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. IEEE Press, 2022, p. 1199–1208. [Online]. Available: https://doi.org/10.1109/INFOCOM48880.2022.9796961

[11] D. Steinkraus, I. Buck, and P. Simard, "Using gpus for machine learning algorithms," in *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, 2005, pp. 1115–1120 Vol. 2.

[12] D. Cao, J. Yoo, Z. Xu, E. Saurez, H. Gupta, T. Krishna, and U. Ramachandran, "Microedge: A multi-tenant edge cluster system architecture for scalable camera processing," in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, ser. Middleware '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 322–334. [Online]. Available: https://doi.org/10.1145/3528535.3565254

[13] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[14] S. Gazzaz and F. Nawab, "Collaborative edge-cloud and edge-edge video analytics," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 484. [Online]. Available: https://doi.org/10.1145/3357223.3366024

[15] A. J. Ben Ali, S. Semenova, and K. Dantu, "Platform variability in edge-cloud vision systems," in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 163. [Online]. Available: https://doi.org/10.1145/3301293.3309555

[16] X. Zhang and S. Debroy, "Resource management in mobile edge computing: A comprehensive survey," *ACM Comput. Surv.*, vol. 55, no. 13s, jul 2023. [Online]. Available: https://doi.org/10.1145/3589639

[17] D. Simon, A. Seuret, and O. Sename, "On real-time feedback control systems: Requirements, achievements and perspectives," in *2012 1st International Conference on Systems and Computer Science (ICSCS)*, 2012, pp. 1–6.

[18] N. Wang, M. Matthaiou, D. S. Nikolopoulos, and B. Varghese, "Dyverse: Dynamic vertical scaling in multi-tenant edge environments," *Future Generation Computer Systems*, vol. 108, pp. 598–612, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X19312403

[19] J. Hao, P. Subedi, I. K. Kim, and L. Ramaswamy, "Characterizing resource heterogeneity in edge devices for deep learning inferences," in *Proceedings of the 2021 on Systems and Network Telemetry and Analytics*, ser. SNTA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 21–24. [Online]. Available: https://doi.org/10.1145/3452411.3464446

[20] S. Bagchi, M.-B. Siddiqui, P. Wood, and H. Zhang, "Dependability in edge computing," *Commun. ACM*, vol. 63, no. 1, p. 58–66, dec 2019. [Online]. Available: https://doi.org/10.1145/3362068

[21] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive congestion control for unpredictable cellular networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 509–522. [Online]. Available: https://doi.org/10.1145/2785956.2787498

[22] X. Zhang and S. Debroy, "Energy efficient task offloading for compute-intensive mobile edge applications," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[23] Z. Fang, J.-H. Lin, M. B. Srivastava, and R. K. Gupta, "Multi-tenant mobile offloading systems for real-time computer vision applications," in *Proceedings of the 20th International Conference on Distributed Computing and Networking*, ser. ICDCN '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 21–30. [Online]. Available: https://doi.org/10.1145/3288599.3288634

[24] M. Liu, X. Ding, and W. Du, "Continuous, real-time object detection on mobile devices without offloading," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020, pp. 976–986.

[25] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3132211.3134458

[26] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[27] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," 2019. [Online]. Available: https://arxiv.org/abs/1905.02244

[28] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019. [Online]. Available: https://arxiv.org/abs/1905.11946

[29] E. Kristiani, C.-T. Yang, and C.-Y. Huang, "isec: An optimized deep learning model for image classification on edge computing," *IEEE Access*, vol. 8, pp. 27 267–27 276, 2020.

[30] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 1421–1429.

[31] B. Qian, Z. Wen, J. Tang, Y. Yuan, A. Y. Zomaya, and R. Ranjan, "Osmoticgate: Adaptive edge-based real-time video analytics for the internet of things," *IEEE Transactions on Computers*, pp. 1–14, 2022.

[32] K. H. Ang, G. Chong, and Y. Li, "Pid control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, 2005.

[33] J. G. Ziegler and N. B. Nichols, "Optimum Settings for Automatic Controllers," *Transactions of the American Society of Mechanical Engineers*, vol. 64, no. 8, pp. 759–765, 12 2022. [Online]. Available: https://doi.org/10.1115/1.4019264

[34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[35] J. Kosaian and A. Phanishayee, "A study on the intersection of gpu utilization and cnn inference," 2022. [Online]. Available: https://arxiv.org/abs/2212.07936

[36] A. Jurgelionis, J.-P. Laulajainen, M. Hirvonen, and A. I. Wang, "An empirical study of netem network emulation functionalities," in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, 2011, pp. 1–6.

[37] V. Sharma, K. Kar, K. K. Ramakrishnan, and S. Kalyanaraman, "A transport protocol to exploit multipath diversity in wireless networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 4, pp. 1024–1039, 2012.

[38] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," in *Proceedings of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, ser. LCTES 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 31–43. [Online]. Available: https://doi.org/10.1145/3211332.3211336

[39] 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

[40] P. Liu, B. Qi, and S. Banerjee, "Edgeeye: An edge service framework for real-time intelligent video analytics," in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–6. [Online]. Available: https://doi.org/10.1145/3213344.3213345