# AccuMO: Accuracy-Centric Multitask Offloading in Edge-Assisted Mobile Augmented Reality

Z. Jonny Kong[*]
Purdue University
West Lafayette, USA

Qiang Xu[*]
Purdue University
West Lafayette, USA

Jiayi Meng
Purdue University
West Lafayette, USA

Y. Charlie Hu
Purdue University
West Lafayette, USA

## Abstract

Immersive applications such as Augmented Reality (AR) and Mixed Reality (MR) often need to perform multiple latency-critical tasks on every frame captured by the camera, which all require results to be available within the current frame interval. While such tasks are increasingly supported by Deep Neural Networks (DNNs) offloaded to edge servers due to their high accuracy but heavy computation, prior work has largely focused on offloading one task at a time. Compared to offloading a single task, where more frequent offloading directly translates into higher task accuracy, offloading of multiple tasks competes for shared edge server resources, and hence faces the additional challenge of balancing the offloading frequencies of different tasks to maximize the overall accuracy and hence app QoE.

In this paper, we formulate this *accuracy-centric multitask offloading problem*, and present a framework that dynamically schedules the offloading of multiple DNN tasks from a mobile device to an edge server while optimizing the overall accuracy across tasks. Our design employs two novel ideas: (1) task-specific lightweight models that predict offloading accuracy drop as a function of offloading frequency and frame content, and (2) a general two-level control feedback loop that concurrently balances offloading among tasks and adapts between offloading and using local algorithms for each task. Evaluation results show that our framework improves the overall accuracy significantly in jointly offloading two core tasks in AR — depth estimation and odometry — by on average 7.6%–14.3% over the best baselines under different accuracy weight ratios.

## CCS Concepts

• **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; • **Computer systems organization** → *Real-time system architecture.*

## Keywords

Mobile Augmented Reality, Edge Computing, DNN Offloading, Multitask Offloading, Depth Estimation, Odometry

**ACM Reference Format:**
Z. Jonny Kong, Qiang Xu, Jiayi Meng, and Y. Charlie Hu. 2023. AccuMO: Accuracy-Centric Multitask Offloading in Edge-Assisted Mobile Augmented Reality. In *The 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23), October 2–6, 2023, Madrid, Spain.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3570361.3592531

## 1 Introduction

Fueled by the rise of metaverse, immersive mobile apps such as Augmented Reality (AR) often need to perform a number of challenging tasks to provide enhanced user experience. Consider Pokemon Go, a representative AR app, where players use their mobile devices to capture, train and battle with virtual creatures called Pokemon which appear as if they exist in the real-world environment. Supporting realistic, interactive, and immersive user experience such as allowing Pokemon to hide behind a tree or jump on a tree branch requires performing several essential computer vision tasks for each frame, at the high frame rate of the camera capture, *e.g.,* every 16.7 ms: (1) **Odometry**: While a player moves within her real-world surroundings, the mobile device needs to track the camera pose to render virtual objects (*e.g.,* Pokemon) at the correctly perceived locations; (2) **Depth estimation**: The mobile device needs to process the distance information between the camera and physical objects, in order to correctly render virtual objects into the physical environment, *e.g.,* allowing Pokemon to hide behind a tree; (3) **Object detection**: Being able to identify physical objects in each camera frame enhances the virtual objects with spatial and contextual awareness of their surrounding physical world and allows them to interact accordingly, *e.g.,* if a tree branch is nearby, the Pokemon jumps onto it.

Equally importantly, all of the tasks of such an AR app performed on each frame are latency-critical; the results for the current frame need to be available in the *current frame interval*, as otherwise they will miss the rendering for the current frame, as dictated by the stringent QoE of AR

[*] Both authors contributed equally to the paper.

*ACM MobiCom '23, October 2–6, 2023, Madrid, Spain*

ACM ISBN 978-1-4503-9990-6/23/10.
https://doi.org/10.1145/3570361.3592531

apps, *e.g.,* 60 FPS [47]. We note this is a critical difference from latency-sensitive applications such as video analytics pipelines where frames of surveillance cameras are uploaded to the cloud for real-time analytics which can tolerate a delay of hundreds of milliseconds [45]. To clearly distinguish the two scenarios, we denote the accuracy returned by AR tasks as the *current-frame accuracy* (CFA), and that for video analytics pipelines as *delayed-frame accuracy* (DFA).

Deep Neural Network (DNN) models have been increasingly used to support these complex AR tasks due to their high accuracy (*e.g.,* [13, 20, 28, 34, 53, 54]). In contrast, traditional AR frameworks, *e.g.,* ARCore, have several known limitations including low resolution and only getting accurate results for up to 5 meters for depth estimation [8]. However, the high-accuracy, DNN-based solutions are generally computationally heavy. Running state-of-the-art DNN models on commodity mobile devices could take hundreds of milliseconds or even seconds [47, 50, 63]. To attain high accuracy results on resource-constrained devices, offloading (also known as edge-assisted solutions) has been proposed [6, 47, 49], where camera frames are uploaded to a cloud or edge server for DNN inference.

Despite the importance of offloading multiple tasks of an AR app, the large amount of recent work on edge-assisted AR have focused on offloading a single task at a time, in particular, object detection (*e.g.,* [18, 39, 41, 66]), assuming a user (AR app) is allocated a dedicated edge server or GPU. Such solutions can potentially be applied to offloading multiple tasks of an AR app of a user by allocating each offloaded task a dedicated GPU. However, such an approach is not cost-effective and not scalable. In fact, to control the server cost, in a shared edge cloud, a user may only be allocated a slice of a GPU [1, 16, 56]. For example, in Amazon Elastic Inference, GPUs are partitioned and priced per TFLOPS [56].

Offloading multiple tasks of a latency-critical app faces a new design objective beyond that for offloading a single task. In offloading a single task, when the end-to-end offloading latency is longer than a frame interval, which happens often due to server inference and frame transfer delay, the task result for the current frame is generated by either directly returning the last server-returned result (*e.g.,* [50]) or by applying some local tracking technique to that result (*e.g.,* [17, 18, 47, 50]). In both cases, the current frame accuracy suffers from staleness of the last server-returned result. Hence, in offloading a single task, the primary goal is to reduce the end-to-end offloading latency, as the lower the end-to-end offloading latency, the less stale the last server-returned result, and the higher the task accuracy.

In contrast, when an app needs to offload multiple tasks, which compete for shared resources allocated to a user (*e.g.,* edge server GPU), offloading of different tasks have to be interleaved in some manner which increases the staleness of the last returned server inference result for each task and adversely affects the accuracy of all tasks. Since the accuracy of different tasks can affect the app QoE differently, in addition to reducing the end-to-end offloading latency for each task, multitask offloading faces a new design goal: *how to balance the offloading of different tasks to maximize the overall accuracy that ultimately determines the app QoE.* Existing works on multi-DNN inference scheduling (*e.g.,* [19, 30, 42, 71, 76]) only focus on maximizing throughput or reducing SLO violation, without consideration for accuracy.

In general, the combination of task accuracies that optimize the app QoE is app specific and is beyond the scope of this paper. We assume such a function $accuracy\_merge()$ that merges the accuracies of all app tasks into a single *overall accuracy* metric is given, *e.g.,* by the app developer. We formally state the **accuracy-centric multitask offloading (AccuMO)** problem:

*Given the function $accuracy\_merge()$ for an app that offloads multiple tasks $t_1, \dots, t_k$ and the server resource constraints, how to schedule the offloading of the tasks to maximize the overall accuracy $accuracy\_merge(t_1, \dots, t_k)$?*

Tackling the above multitask offloading scheduling problem faces two challenges: (1) How to estimate the accuracy or accuracy drop if a task is offloaded under a candidate schedule? (2) How should the scheduler balance offloading of different tasks in an online manner given the interdependence between current and future offloading decisions, while maximizing the overall accuracy?

In this paper, we present a framework called AccuMO that dynamically schedules offloading of multiple compute-intensive DNN tasks of an AR app from a mobile device to an edge server while optimizing the overall accuracy across the tasks. Our design is motivated by two key insights we made about AR tasks: (1) local tracking accuracy drop rates are content-dependent, and different tasks may be affected by different content features; (2) although offloading solutions achieve higher accuracies than directly running a local algorithm on the device, *e.g.,* a lite model, on average, the local algorithm may perform better on selected frames, *e.g.,* when the camera is moving fast and local tracking suffers from low continuity across consecutive frames.

In addition to maximizing the overall accuracy of the tasks, an additional design goal of the AccuMO framework is to easily support any mix of app tasks. To achieve this, we develop a modular design that runs two concurrent control loops that both adapt according to frame content dynamics: a low-level control loop is per task and adapts between local tracking and using the local algorithm for each frame, and a global control loop runs model predictive control (MPC) [15] to dynamically balance offloading of the tasks.

We have implemented the AccuMO framework and two core tasks of immersive AR apps — depth estimation and odometry — on commodity Android phones and GPU servers. Our evaluation using a large set of videos with diverse frame content and camera movement show that in jointly offloading the two core tasks in AR on commodity devices, AccuMO improves the overall task accuracy over the best baseline by 2.3%–11.9% (avg. 7.6%), 6.7%–14.6% (avg. 10.1%), 10.8%–16.7% (avg. 14.3%), 2.3%–15.2% (avg. 11.2%), and -2.8%–23.9% (avg. 11.2%) under 5 diverse accuracy weight ratios. In particular, AccuMO improves the accuracies of both tasks simultaneously over round-robin scheduling by up to 13.4% for depth estimation and up to 33.8% for odometry.

In summary, our main contributions are as follows:

- We present, to our best knowledge, the first accuracy-centric multitask offloading framework that jointly optimizes the overall accuracy of multiple tasks of an AR app running on a mobile device to an edge server.
- We present a novel two-level control feedback loop design that allows for easily adding new tasks while optimizing the overall accuracy across the tasks.
- We also present the first, complete edge-assisted offloading design for two core AR tasks, depth estimation and odometry, which includes local trackers, local algorithms, and novel accuracy models, which are used to dynamically select between them.
- We implement and experimentally validate our AccuMO framework design by comparing it with various static offloading schemes.

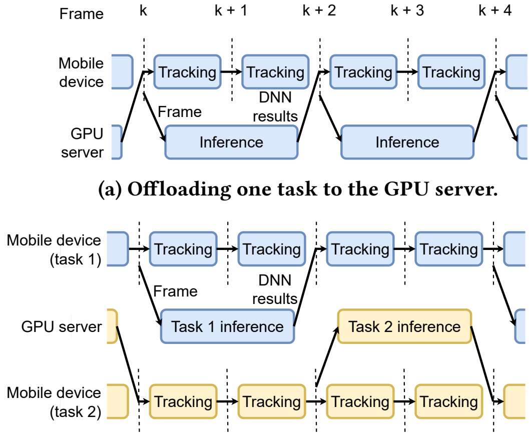
We have open-sourced the AccuMO framework to stimulate further research on multitask offloading in AR.[1]

## 2 Background: The Offloading + Local Tracking Paradigm

In this section, we give a brief background on the popular offloading + local tracking paradigm.

In edge-assisted AR, even with powerful GPUs, a typical DNN inference still takes tens of milliseconds, failing to return the result within the same frame interval. For example, models in Meta's object detection model zoo [60] have a median inference time of 52.5 ms on Tesla V100, much longer than the per-frame time of 16.7 ms needed by an AR app running at 60 FPS [47]. In such cases, the result of an offloaded frame may come back several frame intervals later, and stale server-returned results have to be used for the interim frames, resulting in reduced accuracy.

To mitigate the staleness of server DNN inference results, *local tracking* has been proposed to generate more accurate task results for the current frames than simply using the last returned result from the server [5, 7, 17, 18, 25, 47, 48, 50, 64, 65, 67, 70, 74]. Specially, a local tracker runs on the mobile device and adjusts the DNN inference results for the last offloaded frame $f_l$ sent back by the server to generate refined results for the current frame $f_c$, by analyzing the changes between the stale frame $f_l$ and the current frame $f_c$, as shown in Figure 1a. We denote edge-assisted solutions that exploit local tracking as the *offloading + local tracking paradigm*. Local trackers are fast and can typically finish within one frame time. They are also task-specific and often custom-designed for each type of tasks. For example, for object detection, local trackers use motion vectors to estimate the movement of an object within a bounding box, and adjust the bounding box accordingly [47, 70].



**(a) Offloading one task to the GPU server.**

**(b) Offloading two tasks to the GPU server, the average tracking stride increases from 2.5 to 3.5.**

**Figure 1: The offloading + local tracking paradigm.**

While local tracking improves the accuracy of the result for the current frame $f_c$ (compared to directly reusing the last server returned result), the gap between its accuracy and that of running the server DNN model (if we could), denoted as *accuracy drop*, still widens with *tracking stride*, defined as the frame distance between $f_l$ and $f_c$, due to increased staleness of the results for frame $f_l$. For example, in Figure 1a, the tracking stride is 2 for frame $k + 2$, and 3 for frame $k + 3$, since the local tracker has to use the last returned result for frame $k$ in performing local tracking for these two frames.

Prior work also examined other optimizations for DNN offloading for AR, *e.g.,* pipelining [47, 50] and frame compression [17, 18, 47]. However, under the network conditions we consider, *e.g.,* 802.11ac, pipelining is less effective as DNN inference latency dominates network delay (§8.3), while frame transmission time saved by compression is offset by the compression overhead. To our knowledge, local tracking is the only design option (apart from on-device lite models) that ensures the results for the current frame are available in the current frame interval.

## 3 Motivation

### 3.1 Accuracy Impact of Multitask Offloading

Compared to single-task offlaoding, multitask offloading impacts the task accuracy and hence the app QoE in two ways: (1) it reduces per-task accuracy, and (2) it can reduce accuracy for different tasks by different amount. To quantify these impact, we conducted a measurement study using depth estimation and odometry as the two example tasks. Offloading each task follows the offloading + local tracking paradigm discussed above. The server runs DNN models AdaBins [10] and DAVO [40] for the two tasks, respectively,

[1] https://github.com/JonnyKong/AccuMO

**Table 1: Task accuracies under different offloading schedules.**

| | Single task | RR | WRR 1:2 |
|---|---|---|---|
| Depth estimation (AbsRel ↓) | 0.166 | 0.192 | 0.213 |
| Odometry ($t_{err}$ ↓) | 0.038 | 0.094 | 0.066 |
| Overall accuracy w/ (0.5, 0.5) weights | N/A | 0.143 | 0.140 |
| Overall accuracy w/ (0.7, 0.3) weights | N/A | 0.163 | 0.169 |

where a single inference of the two models takes 49 ms and 54 ms on an NVIDIA RTX 2080 Ti GPU. The client runs warping and Kalman filter (see §6 for details) as the local trackers for the two tasks. The client and the server are connected with 802.11ac. We report the absolute relative error (AbsRel) for depth estimation and the KITTI odometry metric ($t_{err}$) for odometry, averaging over all videos in the dataset. The detailed setup, metrics, and dataset can be found in §8.1. In single-task offloading, each task is offloaded back to back. In offloading both tasks, we used the simple round-robin and several other static schemes.

***Impact on per-task accuracy.*** Table 1 shows the average accuracies across all videos in our dataset. Compared to single-task offloading, *i.e.,* only running and evaluating one of the tasks, the accuracies of the two tasks when offloaded under round-robin drop by 15.7% and 166.7%, respectively. Intuitively, the reason for the accuracy drop in concurrent offloading is the reduced offloading frequency, and hence increased local tracking stride. Increased local tracking stride affects the average local tracking accuracy in two ways: (1) frame $f_l$ which corresponds to the latest offloading result becomes more stale and hence less similar compared to the current frame $f_c$, which leads to less accurate local tracking; (2) more frames, *e.g.,* all frames between $f_c$ and $f_{c+stride-1}$, will be using the stale result (from the server) for $f_l$ in local tracking. The average depth estimation tracking stride increases from 6.68 when offloaded alone to 8.75 when offloaded along with the second task in a round-robin manner.

***Impact on relative accuracy drop.*** We repeated the two-task offloading experiment by changing round-robin to repeatedly offloading depth estimation once followed by offloading odometry twice, denoted as the "WRR 1:2" scheme. Table 1 shows compared to round-robin, such an offload schedule improves the accuracy of odometry by 29.8% at the cost of reducing the accuracy of depth estimation by 10.9%. Assume a hypothetical accuracy merge function that calculate the overall accuracy as the weighted accuracy of the two tasks. Table 1 shows that under (0.5, 0.5) weights, WRR 1:2 achieves 2.4% higher overall accuracy than RR, while under (0.7, 0.3), RR achieves 3.7% higher accuracy than WRR 1:2.

## 3.2 Prior Work on DNN Offloading

The large amount of prior work on DNN offloading have primarily focused on single-task offloading, and the few exceptions on multi-task offloading did not consider optimizing the current frame accuracy (CFA) of the offloaded tasks.

***DNN offloading for a single task.*** Researchers have designed many DNN offloading systems for single AR tasks such as object detection [17, 18, 47], human pose estimation [47], and depth estimation [50]. Glimpse [18] sends key frames to the server side to perform object detection, and employs local tracking on the client side to mask the offloading latency. Liu *et al.* [47] additionally employ optimizations like pipelined streaming and inference as well as region-of-interest encoding. In addition to visual features, MARVEL [17] utilizes IMU data to track the objects. Meng *et al.* [50] examines the performance of offloading the depth estimation task under different setups, *e.g.,* with or without local tracking and under different bandwidths.

**Table 2: Comparison between AccuMO and other works on multitask offloading.**

| | Application | Objective |
|---|---|---|
| MCDNN [33] | Video analytics | Max. DFA |
| LinkShare [35] | Low frame rate apps | Min. deadline violation |
| AccuMO | AR / latency-critical apps | Max. CFA |

There have also been much work on DNN offloading with a single DNN task with relaxed latency constraints, in particular, for video analytics. These work propose several techniques to optimize the offloading latency. (1) Frame compression algorithms specifically designed for DNN offloading were proposed to improve the compression ratio and task accuracy over standard image/video compression algorithms [23, 62, 66]. (2) Offloading regions of interest [49] or key frames [45] also reduces frame transmission latency. (3) Partial offloading splits the DNN model between the mobile device and the server, at an intermediate DNN layer that is significantly smaller than the input [9, 24, 36, 39, 43, 44]. Researchers have also studied special cases of the offloading problem, *e.g.,* for high resolution frames [75] and when multiple devices offload similar data [22, 31].

***Offloading multiple DNN tasks.*** While most works on optimizing DNN inference for mobile devices have focused on one task at a time, few works studied offloading multiple DNN tasks. Table 2 compares existing multitask offloading works with AccuMO. A major difference is that both MCDNN [33] and LinkShare [35] target applications with relaxed latency requirements such as video analytics, while AccuMO focuses on latency-critical applications like AR. LinkShare does not optimize overall accuracy. MCDNN tries to maximize the overall accuracy, but it only focuses on delayed frame accuracy (DFA). In contrast, AccuMO maximizes overall current-frame accuracy (CFA) for AR apps.

***Local tracking.*** Local trackers have been developed for a variety of computer vision tasks — these techniques are orthogonal and can benefit both single-task and multitask DNN offloading. Local trackers for object detection are based on feature point extraction and matching [7, 18], correlation filters [48, 64, 65], motion vectors [47], optical flow [17], and/or inertial data [17]. Human object detection and super-resolution local trackers both utilize motion vectors [25, 47, 67, 74], while the local tracker for depth estimation relies on

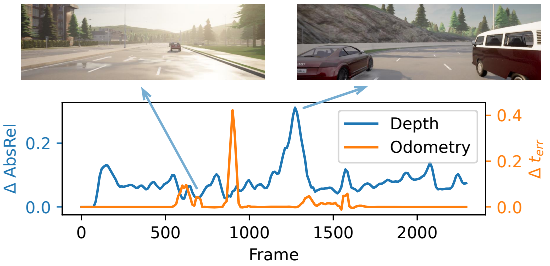


**Figure 2: Accuracy drop timeline for depth estimation and odometry on a sample video when increasing the stride from 6 to 12.**

warping [50].

***Optimizing multiple tasks on-device.*** For on-device DNN inference, Potluck [32] enables computation reuse across applications on the same mobile device via caching, Heimdall [68] coordinates DNN execution and rendering by partitioning and scheduling DNNs in blocks, and RT-mDL [46] schedules a series of compressed models across mobile CPU and GPU based on their accuracy and latency requirements. These systems are all designed for non-real-time applications, and do not meet the latency requirements of AR.

## 4 Key Insights

Intuitively, as the task accuracy is correlated with its offloading frequency, the key to multitask offloading scheduling is to balance offloading of multiple tasks under the constraint of edge server resource, *e.g.*, the GPU. If the accuracy-offloading frequency correlation is static, the optimal offloading schedule can be derived offline. However, we observe that frame content can affect the offloading accuracy. In this section, we discuss two key observations that motivate our adaptive online multitask offloading design.

**(1) Offloading frequency-accuracy correlation is frame-dependent.** Figure 2 shows the local tracking accuracy drop timeline of the two tasks on a sample video when increasing the stride from 6 to 12. From the timeline, we see that local tracking accuracy drops for both tasks change over time. After comparing the accuracy drops with the input frames side by side, we observe that the accuracy drop rate (accuracy drop per frame delay) is affected by the frame content. For example, Figure 2 annotates two data points with different depth estimation accuracy drops with their corresponding frames. The one corresponding to high depth estimation accuracy drop has larger regions of cars, because warping — the local tracker for depth estimation — cannot accurately handle regions with moving objects (see §6.1). Furthermore, Figure 2 shows that there is minimal correlation between the local tracking accuracy drops of the two tasks. In fact, the accuracy drop rate of odometry is instead affected by the angular velocity of the camera (see §6.2.3). We thus make our first observation: *(K1) Local tracking accuracy drop rates are content-dependent, and different tasks may be affected by different content features.*

***Design challenges.*** This observation suggests that instead of offloading the tasks following some static schedule, *e.g.*, round-robin, adaptively increasing (decreasing) a task's offloading frequency when its local tracking accuracy drop rate goes up (down) can potentially result in improved overall accuracy for all tasks. Designing a framework to exploit this observation faces several challenges: (i) how to estimate the local tracking accuracy drop rates during runtime? (ii) how to make the optimal offloading decisions?

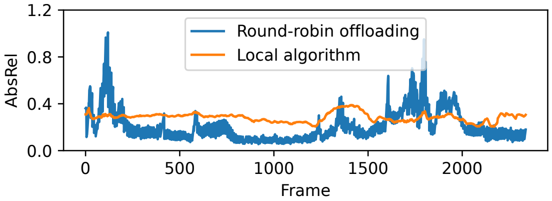


**Figure 3: Depth estimation accuracy timeline of round-robin offloading vs. the local algorithm FastDepth on a sample video. Round-robin offloading has the same setup as Table 1.**

**(2) Is local tracking the best thing to do on-device?** We experimentally compare offloading + local tracking with running *local algorithms*, which are either lite DNN models or conventional algorithms that can finish within one frame interval on the mobile device. Figure 3 compares the depth estimation accuracy obtained from round-robin offloading to that of a lite model FastDepth [59] on a sample video. We see that while offloading outperforms the lite model on average (0.212 vs. 0.284), its accuracy is much worse for some frames, *e.g.*, frames 1600 to 2000. We make our second key observation: *(K2) Even though the accuracies of the local algorithms are significantly worse than the offloading solutions on average across the frames of a video, the local algorithms can achieve better accuracies for some individual frames.* In particular, this happens in two situations: (1) when the content changes very fast, and (2) when the local tracking stride is large enough, *i.e.,* due to other offloading tasks occupying the server GPU resource. Further, we observe little correlation between offloading and lite model accuracies. This is because while the local tracker is affected by content changes, the lite model, which is a DNN model taking a single frame as input, is more likely affected by static features within a frame.

***Design challenges.*** Exploiting K2 in multitask offloading faces several challenges: (i) how to estimate the local algorithm accuracies during runtime? (ii) how to incorporate the accuracy estimates in making offloading decisions?

## 5 AccuMO Design

Motivated by the above key insights, we design a content-aware adaptive multitask offloading framework called AccuMO that jointly optimizes the accuracies of multiple tasks by dynamically controlling each task's offloading frequency and switching between offloading and local algorithms.

### 5.1 Design Goals

We design the multitask offloading framework to achieve the following goals:

**Support for different application needs.** The system

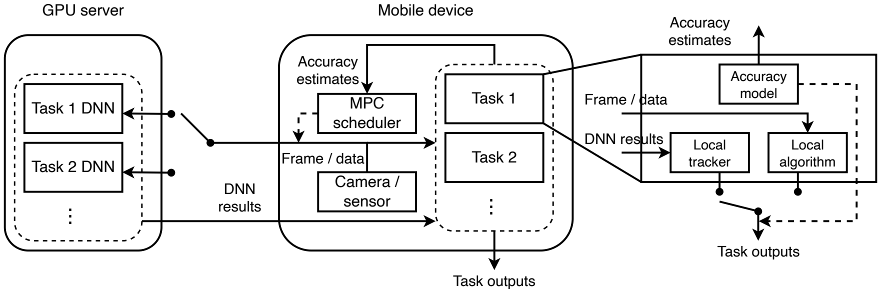


**Figure 4: AccuMO architecture. Solid arrows represent data flows, while dashed arrows represent control flows. Inputs to accuracy models are task-specific and not shown.**

should accommodate different high-level application needs, in particular different needs for balancing task accuracies.

**Optimized overall task accuracy.** The system should optimize the overall accuracy across the tasks.

**Extensible for different tasks.** The system can easily support varying numbers of tasks. Furthermore, it should accommodate different task designs, *e.g.,* with or without a local algorithm.

**Real-time.** The system should produce results in real-time, *e.g.,* 60 FPS, to support latency-critical applications like AR.

## 5.2 Architecture Overview

***Design rational.*** As shown in Figure 4, AccuMO employs a modular design to support different numbers of tasks. There is one task component for each task. To meet the real-time requirement, we employ the offloading + tracking paradigm for each task, and thus each task component requires a high accuracy DNN model running on the server GPU and a local tracker that runs in real-time. Furthermore, to exploit our second key insight (K2), each task component may also contain a local algorithm. To dynamically switch between the two, an accuracy model is required to estimate their accuracies based on the frame content.

Exploiting our first key insight (K1), we employ a global scheduler to control when and for which task to offload each frame captured by the camera and/or other sensors based on each task's accuracy estimates. We design our global scheduler using model predictive control (MPC) [15], a control theory optimization algorithm, rather than machine learning algorithms, to meet our flexible accuracy and extensibility design goals. Machine learning algorithms would require hardcoded task number, task design, and optimization goals which then have to be trained offline, while control theory-based algorithms are more flexible in dynamically adapting to different tasks and application needs, *e.g.,* how to merge the task accuracies.

Our MPC scheduler only decides on the offloading schedule, while leaving the decision of switching between offloading and local algorithms to the individual tasks. While an algorithm that makes both decisions jointly might produce better schedules, we decide to decouple the decisions to accommodate diverse task component designs. For example, a task might not have a local algorithm, or a single design may serve as both the local tracker and the local algorithm and the "local tracker vs. local algorithm" decision is made internally (see §6.2.2). Our MPC scheduler accommodates both of these cases by simply not considering the local algorithm accuracy drop for that task.

***Control loops.*** We develop a modular design that runs two concurrent control loops that both adapt according to frame content dynamics: a low-level control loop is per task and adapts between local tracking and using the local algorithm for each frame; and a global control loop runs MPC to dynamically balance offloading of the tasks. We observe that accuracy models may be time consuming to run, or they can only estimate task accuracies on certain key frames (see §6.1). To meet the real time and extensibility goals, we run accuracy models concurrently with the other two control loops. Effectively, the two control loops make decisions based on the most recent accuracy estimates, exploiting the temporal locality. In summary, the framework consists of two control loops, in addition to the accuracy models, running concurrently and continuously:

- **MPC scheduler:** When a new frame becomes available, and there is currently no unfinished offloading, the MPC scheduler determines the next task to offload for that frame based on the latest accuracy estimates by the task-specific accuracy models.
- **Local selector:** For each frame, either the local tracker or the local algorithm is executed for each task, depending on which one has higher estimated accuracy.
- **Accuracy model:** The local tracker and local algorithm accuracy models for each task are executed repeatedly on the most recent frame, *i.e.,* best effort. The most recent accuracy estimates are used by the two control loops.

***Adding new tasks.*** Our modular design simplifies the addition of new tasks. To offload a task under any framework under the offloading + local tracking paradigm, the developer already needs to choose and prepare a local tracker to run on the client, and a DNN model to run on the server. To add such a task in AccuMO, the developer (1) can reuse the chosen local tracker and server DNN model, (2) optionally uses an off-the-shelf local algorithm for the task (*e.g.,* a lightweight DNN model), and (3) only needs to develop an accuracy model for the new task. We provide a set of easy-to-follow guidelines for developing accuracy models in §5.4, and we show how to apply them to derive the accuracy models with a case study in §6.1 and §6.2.3.

## 5.3 Local Tracker and Local Algorithm

As described in §3, local trackers adjust the stale results from the server DNN models for the current frame. They need to run fast in order to be executed for every frame. For many tasks, off-the-shelf algorithms, possibly designed for other purposes originally, could be used readily [50, 64, 70].

In contrast to the local tracker, a local algorithm directly produces task results from camera or sensor data, without relying on server DNN results. It needs to be fast to be executed for every frame. The local algorithm could be a lite DNN model designed for mobile devices, but it could also be a conventional algorithm without any learning components.

## 5.4 Accuracy Model

The accuracy models for each task estimate the accuracies for both offloading (*i.e.,* the local tracker) and the local algorithm. First, to model the accuracy for offloading, since local trackers introduce accuracy drops in adjusting the stale server DNN results, we only need to estimate the accuracy drop (w.r.t. that of running the server DNN model) or accuracy drop rate (per frame delay). We propose a set of principles that can be easily followed in developing accuracy models for different tasks. (1) We start by identifying limitations of the local tracker, *i.e.,* what kind of inputs lead to bad tracking results. (2) Next, we identify and extract features that quantify good inputs vs. bad inputs. (3) Finally, we derive the correlation between the extracted features and accuracy drops or accuracy drop rates.

Second, for modeling the accuracy of local algorithms, since they typically have much worse accuracy compared to the server DNN models, an effective method to approximate their accuracy is by treating the server DNN results as the ground truth [47, 73]. To make the accuracy modeling comparable to local trackers' estimated accuracy drops (as opposed to accuracy), we subtract the local algorithm accuracy estimate by the server DNN model's average accuracy (calculated offline), which transforms it into the accuracy drop relative to the DNN model. §6.1 shows such an accuracy estimation method on the depth estimation lite model works well. However, other task-specific methods may also be used for local algorithm accuracy estimation.

## 5.5 Model Predictive Control Scheduler

Ideally, given perfect knowledge of offloading and local algorithm accuracies over an entire video, the optimal offloading plan can be calculated offline.[2] While perfect accuracy information is not available in practice, it is possible to estimate the current and future accuracies during a short horizon of $N$ frames $[k, k + N]$ using the accuracy models described before. With the estimated accuracies, we can calculate the optimal offloading plan in this horizon, apply the first step of the plan (*i.e.,* for the current frame), and move the horizon forward to $[k + 1, k + N + 1]$. This scheme is known as model predictive control (MPC) [15, 69].

**Algorithm 1:** Offloading scheduling using MPC

```
input: the list of tasks $T$ with offloading histories
       latest accuracy estimates $A$ for all tasks
       horizon length $N$
       overall accuracy function $accuracy\_merge()$
$d^* = \infty, t^* = t_{\text{default}}$;
for $p$ in ValidOffloadPlans($T, N$) do
    $d$ = SimulatePlan($p, T, A, accuracy\_merge()$);
    if $d < d^*$ then
        $d^* = d, t^* = p[0]$;
    end
end
offload the current frame for task $t^*$;
```

***The MPC algorithm.*** Algorithm 1 shows the MPC offloading scheduling algorithm for deciding the next task to offload the frame/sensor data for. The algorithm is executed to decide the offloading task whenever a new frame is to be offloaded, *i.e.,* when the last offloaded result has been received by the mobile device. It essentially populates and simulates all valid offloading plans within a horizon of $N$ frame intervals, calculates the overall accuracy drop of each plan using the accuracy estimates from the accuracy models, and picks the first offloaded task in the plan that has the lowest overall accuracy drop in the horizon for offloading.

Since the maximum offloading frequency is constrained by the task offloading latencies, only the plans that offload the next frame after the last offloading results have come back are valid. This drastically reduces the number of plans to be simulated and allows the MPC algorithm to run in real-time. Further, we prune heavily unbalanced plans where one task is offloaded much more often than the other, *e.g.,* one task is offloaded four times while the other is offloaded only once, which leads to overly unbalanced task accuracies. Each valid plan is simulated by starting with the current task offloading status and rolling out the steps in the plan. For each frame, we choose the lower between the local tracker's accuracy drop and the local algorithm's accuracy drop. Finally, we compute the accumulated overall accuracy drop.

***An example.*** Consider an application with two offloading tasks, and the offloading latencies are two frame times for both tasks. With a horizon $N = 4$, the valid plans are [1, 0, 1, 0], [1, 0, 2, 0], [2, 0, 1, 0], and [2, 0, 2, 0], where 1 and 2 represents offloading task 1 and task 2 respectively, and 0 means no offloading. Assuming the local tracker accuracy drop rates per frame are $r_1$ and $r_2$ for the two tasks, the local algorithm accuracy drops are $d_1$ and $d_2$, and frame $k - 4$ was offloaded for task 1 while frame $k - 2$ was offloaded for task

[2] We assume the network transmission delay is relatively stable, which holds under the network conditions and frame sizes we consider (§8.1).

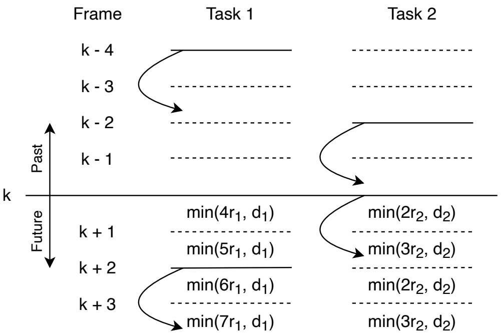


**Figure 5: Simulating an example MPC plan [2, 0, 1, 0]. A curved arrow points from a frame offloaded for a task, to the frame interval when the offloading result comes back.**

2. Figure 5 shows the simulation of the plan [2, 0, 1, 0]. The local tracker accuracy drop is calculated by multiplying the tracking stride with the local tracker accuracy drop rate, *e.g.,* $4r_1$ in $\min(4r_1, d_1)$. Finally, the plan's overall accuracy drop is obtained by merging the accuracy drops of all tasks across all frames in the plan according to *accuracy_merge()*.

## 6 Case Study: Offloading AR Tasks

In this section, we present a case study of how the modular design of our AccuMO framework can easily support multiple tasks, using two representative tasks from a complete AR app — depth estimation and visual odometry. Since the MPC-based offloading scheduler is generic, applying AccuMO in our case study boils down to designing the local tracker, the local algorithm, and their accuracy models for each task.

### 6.1 Depth Estimation

***Background.*** Depth estimation infers the depth map for a given image, containing the distance between the camera and surrounding environment represented by each pixel. The depth estimation results can be used for many tasks, *e.g.,* rendering occlusion between virtual and physical objects in AR, and perception in self driving. In this paper, we focus on monocular depth estimation, since most smartphones are equipped with monocular cameras. Recently, several monocular depth estimation DNNs have been proposed (*e.g.,* [10, 51, 58]). While accurate, such DNN models are compute-intensive and require offloading.

***Depth local tracker.*** We use warping [50], a lightweight geometry-based algorithm, as the local tracker for the depth estimation task. It takes as input the depth map of the last frame $D_1$, and the relative pose change between the last frame and the current frame $p_{12}$, and outputs the depth map of the current frame $D_2'$ as an approximation for the unknown $D_2$. Specifically, given the intrinsics parameters of the camera, warping first converts the depth map into a point cloud in the last frame's camera coordinate system. Then, using $p_{12}$, it transforms the point cloud into the current frame's camera coordinate system. Finally, it projects the point cloud onto the camera plane and performs nearest interpolation to generate $D_2'$, the estimated depth map of the current frame.

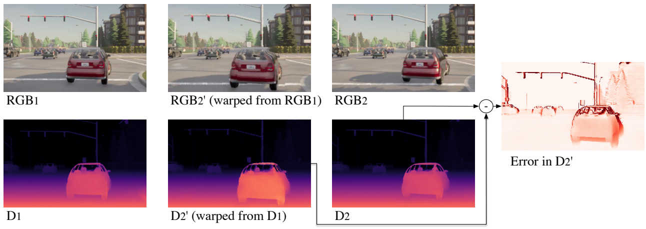


**Figure 6: Moving objects cause errors for depth local tracker.**

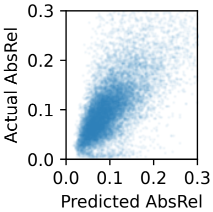


**Figure 7: Predicted and actual depth estimation tracker error.**

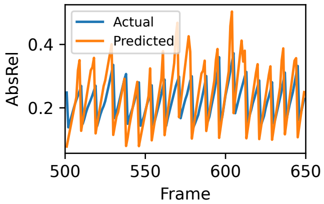


**Figure 8: Example timeline of the accuracy model. Sharp drops are due to receiving offloaded results.**

***Accuracy model for depth local tracker.*** The depth accuracy model estimates the error increase caused by the local tracker, *i.e.,* warping, relative to if the server DNN model could be used. Since warping assumes the objects in the scene are static, dynamic objects will make warping inaccurate. This is shown in Figure 6, where a second vehicle is moving in front of the ego vehicle where the camera sits. Warping assumes that the second vehicle is stationary relative to the ego vehicle and hence the camera, and thus the vehicle in $D_2'$ is incorrectly predicted as being closer to the ego vehicle. This results in high errors around the second vehicle, as shown in the error map. We make a key observation that the magnitude of the error is determined by the relative depth between the vehicle's surface and the background, and hence the error of $D_2'$ can be estimated by (1) identifying regions in the frame that are affected by moving objects; and (2) accounting for the relative depth difference between the moved object and the background.

Based on the observation, we propose a novel accuracy model for estimating the warping error: (1) Warp $RGB_1$ to $RGB_2'$. Since warping assumes objects are static, $RGB_2'$ differs from the real $RGB_2$ on moving objects. (2) Capture moving objects by calculating the optical flow $\mathbb{F}$ from $RGB_2'$ to $RGB_2$, and (3) compute $D_2''$ by translating the pixels of $D_2'$ based on $\mathbb{F}$, where the resulting $D_2''$ accounts for the motion of moving objects. (4) Finally, calculate the absolute relative error (AbsRel) between $D_2'$ and $D_2''$ as an estimate for the error caused by warping. Note that we are using AbsRel between $D_2'$ and $D_2''$ as a proxy for the AbsRel between $D_2'$ and the ground truth. Assuming the warping error increases at a constant rate, we divide the estimated error by the warping stride between the two input frames, to get the error increase rate $r$ (the average error increase per frame).

Figure 7 shows the predicted and actual depth estimation errors exhibit a strong correlation using the proposed depth

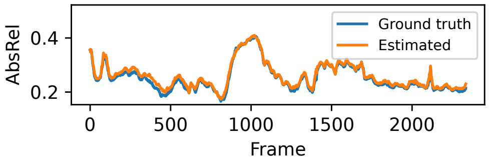


**Figure 9: Example timeline of estimated error vs. ground-truth error of the depth estimation local algorithm.**

accuracy model. Figure 8 plots an example timeline which shows the predicted error follows the actual error closely.

***Local algorithm and its accuracy model.*** Since the above depth estimation DNN models [10, 51, 58] are compute-intensive, several lightweight depth estimation DNN models have been proposed to run on mobile phones and embedded devices in real-time [52, 59], with compromised accuracy.

To support adaption between local tracking and lite model, we estimate the error of the lite model following the method described in §5.4. The client estimates the error of the lite model's output against the server returned depth map for the same frame. Since the server DNN model is much more accurate, the estimated error against server DNN output is close to that against the ground truth, as shown in Figure 9.

## 6.2 Odometry

**6.2.1 Background.** Odometry is the use of various sensor data to estimate the ego-pose (location and orientation) of the device relative to a starting position. The input sensor data can be monocular or stereo camera frames, depth maps (*e.g.,* from Lidar), IMU, GPS, or a combination of them. We focus on commodity smartphones in indoor or urban areas, where only monocular camera frames and IMU data are available. For camera frames, a pair of consecutive frames $f_1$ and $f_2$ are taken as inputs, and the algorithm estimates the relative pose change between $f_1$ and $f_2$. The current pose is estimated by accumulating the sequence of pose changes from start. In offloading, $f_1$ and $f_2$ are offloaded frames, which could be non-consecutive. The poses of the frames in-between are given by the local tracker, as described below.

**6.2.2 Kalman Filter as Local Tracker and Local Algorithm.** We use a Kalman filter [57] together with the IMU data as both local tracker and local algorithm for odometry. Since IMU only reports accelerations and angular velocities, the Kalman filter first integrates them to obtain translations and rotations. Secondly, the Kalman filter maintains internal states including position, velocity, orientation, and their estimated error covariance, which helps to reduce IMU sensor noise. For example, if the velocity covariance is small yet the acceleration reported by IMU is huge, it is likely that the acceleration data contain noise. Lastly, the server DNN pose estimations can be similarly fused with the Kalman filter's internal states. Hence the Kalman filter with IMU data can estimate ego-pose with or without server DNN results, and we use it as both a local tracker (when with server DNN results) and a local algorithm (when without) in our framework. Furthermore, since it calculates the relative importance between server DNN results and IMU data based on the error covariance, it is also used as an accuracy model for choosing between offloading and local algorithms.

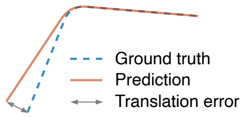


**Figure 10: Translation error between trajectory segments.**

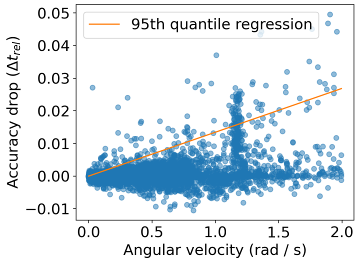


**Figure 11: Scatter plot of accuracy drop vs. angular velocity.**

**6.2.3 Accuracy Model.** Since the Kalman filter does not explicitly estimate the local algorithm accuracy drop, we will design an offloading accuracy model for use in the MPC scheduler. We first introduce the odometry accuracy metric.

***Accuracy metric.*** The commonly used KITTI odometry metric [26, 72], denoted as $t_{err}$, is calculated in 3 steps: (1) extract all pairs of segments of (100, 200, ..., 800) meters long from both ground truth and prediction trajectories, (2) compute the translation error between each pair (see Figure 10) and divide it by the segment length, and (3) average the error over all segment pairs.

The above odometry metric $t_{err}$ is calculated offline with segment as the basic unit, which cannot be used by the MPC scheduler in our framework which plans online at the frame level. To this end, we adapt $t_{err}$ for the MPC scheduler and calculate the accuracy drop caused by a different offloading schedule on a sequence of frames by calculating $t_{err}$ only over the segments that contain the frames, while keeping the offloading schedule for other frames unchanged. We use this frame-level accuracy as a proxy for the offline $t_{err}$ in the MPC scheduler instead.

***Camera rotation amplifies translation error.*** By comparing the trajectories produced by different offloading schedules, we observe that they mainly deviate when the camera rotates (*i.e.,* with high angular velocity, see Figure 16 for an example). The reasons are two-fold. First, camera rotation changes the view more drastically compared to translation, and thus it is more difficult to match the two input frames as the stride increases, where the stride for the odometry task is the difference between the frame IDs of the two consecutive offloaded frames. Second, as shown in Figure 10, the translation error of a frame is amplified when it is further away from the camera rotation point, causing higher $t_{err}$.

***The accuracy model.*** We draw the scatter plot between angular velocity and accuracy drop in Figure 11 to quantify the correlation between them. The accuracy drop is calculated when increasing the inter-frame stride from 4 to 8 for a sequence of 32 frames while keeping the rest at stride 4. The angular velocity is calculated on the same 32 frames.

The data points are mainly distributed at the bottom region, rather than forming a line, as accuracy drop depends on not only the 32 frames, but also all the frames in the affected segments. Nonetheless, Figure 11 indicates that angular velocity can be used to estimate the upper bound of accuracy drop.

Furthermore, we argue that it is imperative to optimize all the frames with high accuracy drops, as these frames cause high errors to all segments that contain them, which spread the impact of suboptimal offloading schedule to longer distances, which was not captured by the proxy accuracy drop calculation (see §8.5). For this reason, we estimate the upper bound of the accuracy drops. Since the upper bound can be affected by outliers, we perform the 95th quantile regression [11] which corresponds to the orange line in Figure 11.

During runtime, the angular velocity is calculated from recent poses, and the upper bound of offloading accuracy drop is estimated based on the model discussed above, which is derived offline. Similarly, we assume linear accuracy drop with increasing strides in estimating the accuracy drop rate.

# 7 Implementation

We implement our multitask offloading framework AccuMO for the case study on depth estimation and odometry.

***Client.*** We implement the client, including the MPC scheduler, task-specific local trackers, local algorithms, and accuracy models as an Android application in 2K lines of Java and C++ code. The control loops and the MPC scheduler are implemented in Java. For depth estimation, the local tracker (warping) is implemented in C++ and runs on the phone CPU. We use FastDepth [59], a state-of-the-art depth estimation DNN designed for embedded devices, as the local algorithm, and run it on the phone CPU on top of the ncnn [3] DNN inference framework. We use FlowNet2S [37], a lightweight DNN, to estimate the optical flow used by the accuracy model, and it runs on the phone GPU utilizing TensorFlow Lite [29]. For odometry, the Kalman filter implementation adopts the `insfilterErrorState` object in the MATLAB Sensor Fusion and Tracking Toolbox [4], and is converted to C using the MATLAB Coder [2] to run on mobile devices.

***Server.*** We implemented the server in about 300 lines of Python code. We use AdaBins [10] (implemented in PyTorch) as the server DNN model for depth estimation and DAVO [40] (implemented in TensorFlow) for odometry. Both DNN models are loaded upon server startup.

The client and the server communicates over TCP. Camera frames are uploaded to the server in raw YUV420 format (the default Android camera format) with resolution 448×128, while depth maps are sent back in raw 16-bit bitmaps.

# 8 Evaluation

## 8.1 Methodology

***Evaluation setup.*** Our client app runs on a Samsung Galaxy Note20 Ultra 5G phone with a Qualcomm Snapdragon 865+ SoC, which has eight Kyro 585 CPU cores and an Adreno 650 GPU. We use two different servers with GPUs of different tiers, an NVIDIA GeForce RTX 2080 Ti and a more powerful NVIDIA A40. We evaluate AccuMO under 802.11ac for indoor scenarios and 5G mmWave for outdoor scenarios. For 802.11ac, we connect the phone and the server to the same access point (550 Mbps for both uplink and downlink, 3 ms RTT). We emulate the 5G mmWave network using the tc tool on top of the 802.11ac setup, based on recent 5G mmWave measurements (152 Mbps uplink, 1715 Mbps downlink, 14 ms RTT) [27]. However, note that we are only able to emulate up to 550 Mbps for 5G mmWave downlink due to the 802.11ac downlink bandwidth limit. When not specified, we default to the 2080 Ti GPU and 802.11ac network.

***Dataset.*** To evaluate our case study of performing depth estimation and odometry for AR, we need a dataset that concurrently provides IMU data and ground truths for both depth estimation and odometry, and has a camera frame rate of at least 60 FPS. To the best of our knowledge, we are not aware of any existing dataset that satisfies all these requirements. To this end, we resort to the same methodology as in prior works [38, 55] — we use CARLA [21], an autonomous driving simulator, to generate a synthetic dataset with sufficient sensor data and ground truth labels. We generated videos of 40 seconds long with the camera mounted on top of the ego-vehicle capturing frames of size 832×256 at 60 FPS. As the simulator only outputs ground truth IMU data, we added noise to the IMU data based on typical IMU data sheets [12].

To analyze the impact of video content, we further classify videos into "easy" ones vs. "hard" ones. In §6, we show that depth estimation offloading accuracy is affected by moving objects, while odometry is affected by camera rotation. To this end, we categorize the videos based on (1) the percentage of pixels occupied by dynamic objects identified by a semantic segmentation DNN [61], and (2) the average camera angular velocity using ground truth pose. We classify each video to have high (H) / low (L) dynamic object percentage / angular velocity, and select 20 videos from each of the 4 combinations, *i.e.,* L/L, L/H, H/L, and H/H. We generated 20 additional videos to train the DNN models (see §7).

***Metrics.*** We evaluate depth estimation using the popular absolute relative error (AbsRel) [14]. We evaluate odometry using the KITTI odometry metric ($t_{err}$), as discussed in §6.2.3.

## 8.2 Baselines

We compare AccuMO with the following baselines:

**Local algorithms (LA).** In this setup, all task results are produced by the local algorithms running on the phone without offloading.

**Round-robin (RR).** This setup offloads the tasks in a round-robin manner (Figure 1b).

**Weighted round-robin (WRR).** This setup is similar to RR, except that some tasks are offloaded more frequently than others. We denote a specific configuration of WRR as WRR $x$:$y$, where the first task (depth estimation) is offloaded

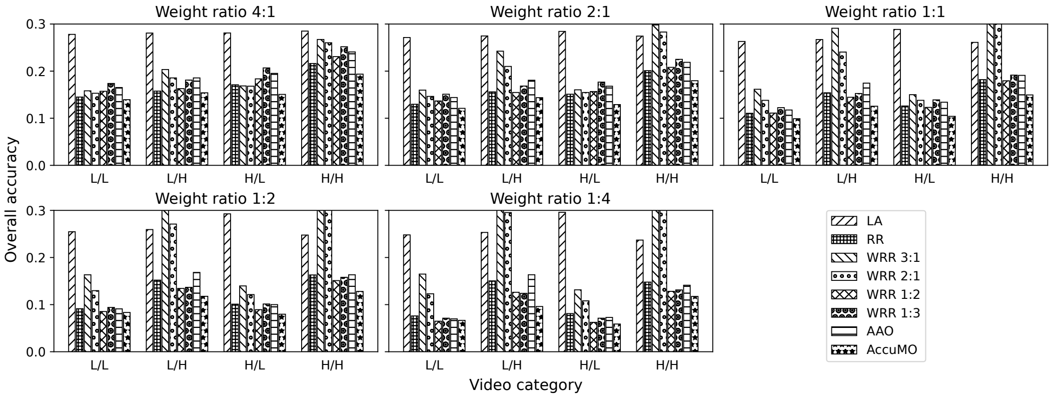


**Figure 12: The overall accuracies (lower is better) under different video categories and varying accuracy weight ratios.**

$x$ times and then the second task (odometry) is offloaded $y$ times. We study WRR 3:1, 2:1, 1:2, and 1:3.

**All at once (AAO).** In this setup, for each offloaded frame, DNN models for all tasks will be executed.

In RR, WRR, and AAO, the offloading results are processed by the local trackers, but local algorithms are not used. We will evaluate the impact of local algorithms in §8.5.

We do not provide direct comparisons to MCDNN [33] and LinkShare [35], as they target different applications and have different design objectives, and their mechanisms are not applicable to AR applications, as discussed in §3.2.

## 8.3 Overall Performance

We compare the performance of AccuMO with all baselines under the default environment setup and a number of accuracy merge functions that calculate weighted average of the accuracies for depth estimation and odometry tasks with varying weights 4:1, 2:1, 1:1, 1:2, and 1:4. We set the MPC horizon to be 30 frames, and restrict the maximum task offloading ratio to be 3:1, *i.e.,* the other task will be offloaded next time if one task has been offloaded three times in a row.

***Comparison with baselines.*** We compare the overall accuracy of our framework, which is the weighted average of the task accuracies, with other baselines under different accuracy weights. As shown in Figure 12, AccuMO improves over the best baseline (the one with the best average overall accuracy across the four video categories) under each weight ratio 4:1, 2:1, 1:1, 1:2, and 1:4 by 2.3%–11.9% (avg. 7.6%), 6.7%–14.6% (avg. 10.1%), 10.8%–16.7% (avg. 14.3%), 2.3%–15.2% (avg. 11.2%), and -2.8%–23.9% (avg. 11.2%) across different video categories. In particular, AccuMO performs on par with the best baseline, *i.e.,* WRR 1:2, in worst cases (-2.8% improvement on L/L videos for weight ratio 1:4), while it outperforms the best baseline significantly in best cases (23.9% improvement on L/H videos for weight ratio 1:4).

***Individual task accuracies.*** While the objective of AccuMO is to optimize the overall accuracy, individual task accuracies help us understand why a scheduler has good or bad overall accuracy. Table 3 shows the accuracies of individual tasks under weight ratio 1:1, along with the overall accuracy. We make the following observations. (1) The accuracies of LA for both tasks are among the worst across all algorithms, indicating the need for offloading. (2) For the RR and WRR schedulers, as shown in Figure 13, offloading one task more frequently improves the accuracy of that task, but hurts the accuracy of the other task. (3) In contrast, AccuMO consistently improves over RR, obtaining 4.3%–13.4% accuracy improvements for depth estimation and 18.9%–33.8% accuracy improvements for odometry, depending on the video categories. (4) Finally, AAO represents a different kind of tradeoff compared to the RR and WRR schedulers. The depth estimation accuracy drops due to increased tracking stride, while the odometry accuracy improves due to decreased stride between the two odometry DNN input frames. However, our framework still outperforms AAO, as our dynamic offloading schedule can provide even smaller strides for frames with high accuracy drop rates.

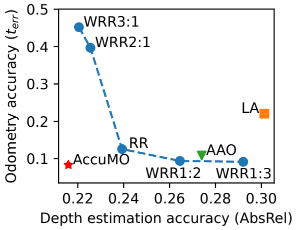


**Figure 13: The accuracy tradeoffs between the two tasks on H/H videos.**

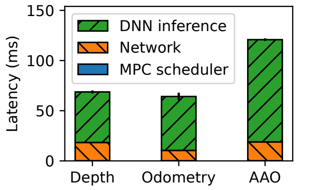


**Figure 14: The offloading latency breakdown of the two tasks, and AAO.**

***Impact of video content.*** From Table 3, we observe that the content or the difficulty of the videos have huge impacts on both task accuracies and task accuracy drop rates. Videos with higher percentage of dynamic objects lead to overall worse depth estimation accuracies, and the accuracy differences among the RR and WRR schedulers are also higher. The same relationship applies to videos' average angular velocity and the odometry accuracy. For both tasks, our framework achieves higher accuracy improvements over RR on more difficult videos due to the higher accuracy drop rates. For example, for depth estimation, our framework improves over RR by 7.7% on L/L videos but by 13.4% on H/L videos. In general, a task is offloaded more frequently when its accuracy drop rate is high and the other task's is low. However, we

**Table 3: Average depth (estimation) error (AbsRel), odom(etry) error ($t_{err}$) under weight ratio 1:1, along with overall accuracies (lower is better). The best accuracies achieved under each video category are highlighted in bold.**

| | L/L | | | L/H | | | H/L | | | H/H | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Depth | Odom | Overall | Depth | Odom | Overall | Depth | Odom | Overall | Depth | Odom | Overall |
| LA | 0.288 | 0.238 | 0.263 | 0.290 | 0.244 | 0.267 | 0.276 | 0.301 | 0.289 | 0.301 | 0.221 | 0.261 |
| RR | 0.168 | 0.053 | 0.111 | 0.160 | 0.148 | 0.153 | 0.201 | 0.051 | 0.126 | 0.239 | 0.125 | 0.182 |
| WRR 3:1 | 0.156 | 0.167 | 0.161 | **0.145** | 0.437 | 0.290 | 0.181 | 0.119 | 0.150 | 0.221 | 0.452 | 0.336 |
| WRR 2:1 | 0.163 | 0.113 | 0.138 | 0.149 | 0.332 | 0.241 | 0.188 | 0.088 | 0.138 | 0.226 | 0.397 | 0.311 |
| WRR 1:2 | 0.188 | **0.034** | 0.111 | 0.175 | 0.114 | 0.145 | 0.224 | **0.022** | 0.123 | 0.265 | 0.094 | 0.179 |
| WRR 1:3 | 0.208 | 0.037 | 0.123 | 0.200 | 0.105 | 0.152 | 0.252 | 0.026 | 0.139 | 0.292 | 0.091 | 0.192 |
| AAO | 0.197 | 0.038 | 0.117 | 0.193 | 0.156 | 0.174 | 0.236 | 0.032 | 0.134 | 0.274 | 0.108 | 0.191 |
| AccuMO | **0.155** | 0.043 | **0.099** | 0.153 | **0.098** | **0.126** | **0.174** | 0.034 | **0.104** | **0.216** | **0.083** | **0.149** |

note that our framework is still able to improve the accuracies of both tasks by 9.6% and 33.6% respectively on the H/H videos. As we will see in §8.4, the reason is two-fold. First, the offloading accuracy drop rates for both tasks still change over time, and they change independently. Secondly, the local algorithms can be used instead when both tasks have high offloading accuracy drop rates, which mitigates the contention on server resources.

***Latency breakdown.*** Figure 14 breaks down the end-to-end offloading latency under AccuMO for both tasks and when performing AAO offloading. Offloading a frame for depth estimation and odometry takes 70.98 ms and 61.05 ms respectively, which translates to 5 and 4 frame times. For both tasks, server DNN inference (50.39 ms and 53.73 ms) takes up a major portion of the total latency. For network transmission, the majority of the time for odometry is spent uploading the frame to the server, while for depth estimation, more time is needed to send the depth map back. We also note that our MPC scheduler is fast (0.2 ms) and has minimal impact on offloading latency. Finally, the AAO offloading latency is much longer than offloading a single task, as both server DNN models need to be executed.

As for other components in the case study, the depth estimation local tracker (warping) takes 15.04 ms, the depth estimation local algorithm (FastDepth) takes 13.42 ms, and the odometry local tracker/local algorithm (Kalman filter) takes 0.03 ms, *i.e.,* all finish within one frame time, making it possible to support real-time applications like AR. For the accuracy models, while the odometry accuracy model (extracting angular velocities) and the depth estimation local algorithm accuracy model (comparing the local algorithm depth map with the server DNN depth map) take minimal time, the depth estimation offloading accuracy model (estimating the optical flow using FlowNet) takes 66.15 ms, and thus it has to run asynchronously, as discussed in §5.2.

## 8.4 Scheduler Behavior Analysis

To understand how the estimated accuracy drops affect AccuMO's scheduling decisions, and how the scheduling decisions in turn affect the task accuracies, in Figure 15 we plot all the relevant information in the same figure for a sample H/H video.

Similar to what is shown in Figure 2, the estimated accuracy drop rates of the two tasks are largely uncorrelated (1st figure). The MPC scheduler makes offloading decisions based on the estimated accuracy drop rates. The 3rd figure plots the depth estimation vs. odometry offloading frequency ratio within a sliding window of 8 offloaded frames. As we restrict the maximum offloading ratio to 3:1, the ratio ranges from 1/3 (when odometry is offloaded frequently) to 3/1 (when depth estimation is offloaded frequently). We see that a task is offloaded more frequently when its accuracy drop rate is relatively high and the other task's is relatively low. Furthermore, we notice that the offloading ratio stays at 3/1 frequently, which means that odometry is offloaded less frequently than depth estimation. However, the odometry accuracy is still better than that of RR, which we explain in §8.5. When the depth estimation local algorithm gives better accuracy over offloading as shown in the 2nd figure, *e.g.,* for frames 1200–1400, the framework switches to local algorithm instead for depth estimation (3rd figure). Meanwhile, the offloading opportunities are saved for odometry, as indicated by the close to 1/3 scheduling ratio in the 3rd figure. The 4th figure shows that the depth estimation accuracy, which is the moving average of 16 frames, improves over RR when the offloading frequency increases, *e.g.,* for frames 200–600, or when the local algorithm is used, *e.g.,* for frames 1200–1400, while the accuracy is worse than RR when the offloading opportunity is saved for odometry, *e.g.,* for frames 900–1000. On average,

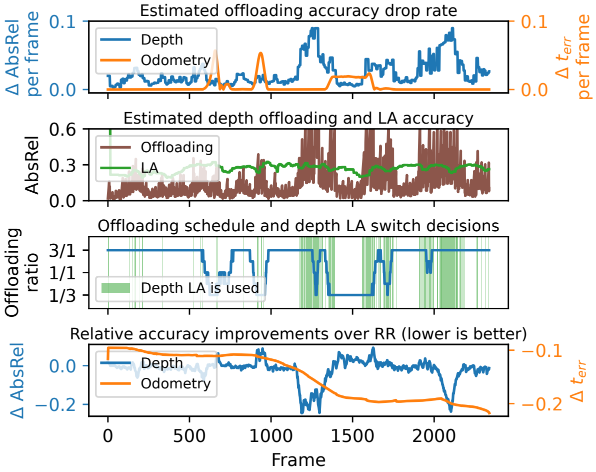


**Figure 15: The timeline for a sample H/H video.**

**Table 4: Ablation study on the contribution of offloading scheduling and local algorithm to depth estimation accuracy.**

| Scheduler | AbsRel ↓ | Imp. over RR |
|---|---|---|
| RR | 0.192 | - |
| RR w/ LA | 0.180 | 6.3% |
| AccuMO w/o LA | 0.183 | 4.7% |
| AccuMO | 0.175 | 8.9% |

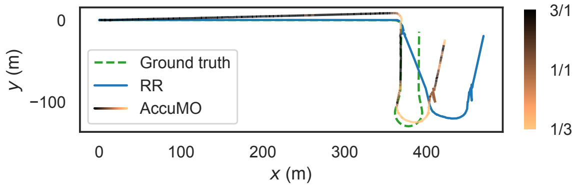


**Figure 16: Odometry trajectories of a sample video.** AccuMO**'s offloading schedule is overlayed on its trajectory.**

the depth estimation accuracy is better than that of RR.

***Impact of accuracy model.*** The optimality of AccuMO's scheduling decisions could be affected by the accuracy of the accuracy models. To estimate the impact, we test AccuMO with accuracy model outputs replaced by actual accuracy drops. The overall accuracy on H/H videos under weight ratio 1:1 is 0.148, on par with that of AccuMO backed by accuracy models (0.149). This is because AccuMO only requires accuracy models to have moderate accuracy levels that are sufficient for the MPC scheduler to determine the right offloading ratios (3rd figure of Figure 15), and the experiment shows that our accuracy models are adequate to support AccuMO's scheduling decisions.

## 8.5 Task-Specific Analysis

***Depth estimation.*** Figure 15 has shown that the depth estimation accuracy improvement over RR is contributed by two factors: (1) the depth estimation task is offloaded more frequently when its estimated accuracy drop rate is high, and (2) the local algorithm is instead used when its accuracy is estimated to be better than offloading. To estimate the contribution of the two factors, we perform an ablation study where we run AccuMO without the depth estimation local algorithm (AccuMO w/o LA), and run RR additionally with the depth estimation local algorithm and the accuracy model (RR w/ LA), so that the scheduler switches to the local algorithm based on the accuracy estimates. Table 4 shows that RR w/ LA and AccuMO w/o LA each contribute about 60% and 40% of the total accuracy improvements, indicating that both are needed to attain high accuracy improvements.

***Odometry.*** Since the odometry accuracy metric is calculated with segments, rather than frames, as the basic unit, in Figure 15 we calculate the accuracy of each frame by averaging over the accuracies of all segments containing the frame (see §6.2.3). We notice that for many frames, *e.g.*, frames 1800–2000, the per-frame accuracy is better than under RR even though the task is offloaded less frequently around these frames. This is because with high-accuracy-drop-rate frames (*e.g.*, frames 1400–1600) offloaded more often, the accuracies of all segments containing those frames are improved, which in turn improves the accuracies of less frequently offloaded frames (*e.g.*, frames 1800–2000) contained in these segments. As another example, in Figure 16, the RR trajectory mainly deviates from the ground-truth at places where the camera rotates. AccuMO offloads the odometry task more often at these frames, resulting in a trajectory that is closer to the ground truth. The extended impact of these high-accuracy-drop-rate frames also supports our accuracy model design of estimating the accuracy drop upper bound (§6.2.3).

**Table 5: Task accuracies under different network and server configurations and scheduling algorithms on H/H videos.**

| | RR | | AccuMO | |
|---|---|---|---|---|
| | Depth ↓ | Odom ↓ | Depth ↓ | Odom ↓ |
| 802.11ac + 2080 Ti | 0.239 | 0.125 | 0.216 | 0.083 |
| 5G + 2080 Ti | 0.252 | 0.181 | 0.228 | 0.162 |
| 802.11ac + A40 | 0.217 | 0.105 | 0.200 | 0.078 |

## 8.6 Impact of Network and Server Configuration

In Table 5, we analyze the performance of our framework under the emulated 5G mmWave network. Furthermore, we also evaluate our framework against a more powerful server GPU — NVIDIA A40. The accuracies of both tasks become worse when switching from 802.11ac to 5G, mainly due to the longer transmission time caused by the higher RTT (14 ms vs. 3 ms for 802.11ac). On the other hand, the task accuracies improve when switching to the A40 server GPU due to faster DNN inference (from 50.39 ms to 39.66 ms for depth estimation and from 53.73 ms to 50.56 ms for odometry). Under all configuration combinations, our framework consistently improves over RR by 7.8%–9.6% for depth estimation and 10.5%–33.6% for odometry, indicating that our framework is generalizable to different network and server configurations.

## 9 Conclusion

In this paper, we presented to our knowledge the first framework that dynamically schedules offloading of multiple compute-intensive DNN tasks of an AR app from a mobile device while optimizing the overall DNN inference accuracy across the tasks. We designed our framework to easily support diverse tasks by employing a general, two-level control feedback loop that adapts between alternative local execution options within each task module and globally balancing offloading among multiple tasks using MPC. Our evaluation results show that our framework can improve the overall task accuracy by on average 7.6%–14.3% over the best baseline under different accuracy weight ratios for depth estimation and odometry in edge-assisted AR.

## Acknowledgments

We thank the anonymous reviewers and shepherd for their helpful comments. This work is supported in part by NSF grant 2112778 and by Cisco Research.

## References

[1] 2019. Cloud-native computing platform. https://puzl.ee/resources/gpu.

[2] MathWorks 2022. *MATLAB Coder*. MathWorks, Natick, USA.

[3] Tencent 2022. *ncnn: a high-performance neural network inference computing framework optimized for mobile platforms*. Tencent, Shenzhen, China.

[4] MathWorks 2022. *Sensor Fusion and Tracking Toolbox*. MathWorks, Natick, USA.

[5] Adel Ahmadyan and Tingbo Hou. 2020. *Real-Time 3D Object Detection on Mobile Devices with MediaPipe*. Retrieved July 13, 2022 from https://ai.googleblog.com/2020/03/real-time-3d-object-detection-on-mobile.html

[6] Mario Almeida, Stefanos Laskaridis, Abhinav Mehrotra, Lukasz Dudziak, Ilias Leontiadis, and Nicholas D. Lane. 2021. Smart at What Cost? Characterising Mobile Deep Neural Networks in the Wild. In *Proceedings of the 21st ACM Internet Measurement Conference* (Virtual Event) *(IMC '21)*. Association for Computing Machinery, New York, NY, USA, 658–672. https://doi.org/10.1145/3487552.3487863

[7] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V Krishnamurthy, and Amit K Roy-Chowdhury. 2019. Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 96–109.

[8] arcoredepth 2022. *Depth adds realism*. Retrieved Jan 25, 2023 from https://developers.google.com/ar/develop/depth

[9] Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. 2021. Auto-Split: A General Framework of Collaborative Edge-Cloud AI. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 2543–2553. https://doi.org/10.1145/3447548.3467078

[10] Shariq Farooq Bhat, Ibraheem Alhashim, and Peter Wonka. 2021. Adabins: Depth estimation using adaptive bins. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4009–4018.

[11] D. Birkes and Y. Dodge. 2011. *Alternative Methods of Regression*. Wiley. https://books.google.com/books?id=CIedErj0HKcC

[12] BMI263 2022. *Inertial Measurement Unit BMI263*.

[13] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv preprint arXiv:2004.10934* (2020).

[14] Cesar Cadena, Yasir Latif, and Ian D. Reid. 2016. Measuring the performance of single image depth estimation methods. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 4150–4157. https://doi.org/10.1109/IROS.2016.7759611

[15] E.F. Camacho and C.B. Alba. 2013. *Model Predictive Control*. Springer London. https://books.google.com/books?id=tXZDAAAAQBAJ

[16] Shankar Chandrasekaran. 2020. *Ride the Fast Lane to AI Productivity with Multi-Instance GPUs*. Retrieved April 23, 2022 from https://blogs.nvidia.com/blog/2020/05/14/multi-instance-gpus/

[17] Kaifei Chen, Tong Li, Hyung-Sin Kim, David E. Culler, and Randy H. Katz. 2018. MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems* (Shenzhen, China) *(SenSys '18)*. Association for Computing Machinery, New York, NY, USA, 292–304. https://doi.org/10.1145/3274783.3274834

[18] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. 155–168.

[19] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 199–216. https://www.usenix.org/conference/atc22/presentation/choi-seungbeom

[20] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. 2016. R-FCN: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*. 379–387.

[21] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16.

[22] Utsav Drolia, Katherine Guo, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. 2017. Cachier: Edge-caching for recognition applications. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 276–286.

[23] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. 2020. Server-driven video streaming for deep learning inference. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 557–570.

[24] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2019. JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing* 20, 2 (2019), 565–576.

[25] Chengsi Gao, Ying Wang, Weiwei Chen, and Lei Zhang. 2021. An Intelligent Video Processing Architecture for Edge-cloud Video Streaming. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 415–420. https://doi.org/10.1109/DAC18074.2021.9586328

[26] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 3354–3361. https://doi.org/10.1109/CVPR.2012.6248074

[27] Moinak Ghoshal, Pranab Dash, Zhaoning Kong, Qiang Xu, Y. Charlie Hu, Dimitrios Koutsonikolas, and Yuanjie Li. 2022. Can 5G mmWave Support Multi-user AR?. In *Passive and Active Measurement*, Oliver Hohlfeld, Giovane Moura, and Cristel Pelsser (Eds.). Springer International Publishing, Cham, 180–196.

[28] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.

[29] Google Brain Team. 2022. *TensorFlow Lite*. Mountain View, USA.

[30] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, USA, Article 25, 20 pages.

[31] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. 2018. Foggycache: Cross-device approximate computation reuse. In *Proceedings of the 24th annual international conference on mobile computing and networking*. 19–34.

[32] Peizhen Guo and Wenjun Hu. 2018. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 271–284.

[33] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. 123–136.

[34] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on*

*computer vision*. 2961–2969.

[35] Bo Hu and Wenjun Hu. 2019. Linkshare: Device-centric control for concurrent and continuous mobile-cloud interactions. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. 15–29.

[36] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1423–1431.

[37] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. 2017. FlowNet 2.0: Evolution of Optical Flow Estimation With Deep Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[38] Sagar Jha, Youjie Li, Shadi Noghabi, Vaishnavi Ranganathan, Peeyush Kumar, Andrew Nelson, Michael Toelle, Sudipta Sinha, Ranveer Chandra, and Anirudh Badam. 2021. Visage: Enabling Timely Analytics for Drone Imagery. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking* (New Orleans, Louisiana) *(MobiCom '21)*. Association for Computing Machinery, New York, NY, USA, 789–803. https://doi.org/10.1145/3447993.3483273

[39] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.

[40] Xin-Yu Kuo, Chien Liu, Kai-Chen Lin, Evan Luo, Yu-Wen Chen, and Chun-Yi Lee. 2020. Dynamic Attention-based Visual Odometry. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 5753–5760. https://doi.org/10.1109/IROS45743.2020.9340890

[41] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–15.

[42] Yunseong Lee, Alberto Scolari, Byung-Gon Chun, Marco Domenico Santambrogio, Markus Weimer, and Matteo Interlandi. 2018. Pretzel: Opening the Black Box of Machine Learning Prediction Serving Systems. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, USA, 611–626.

[43] En Li, Zhi Zhou, and Xu Chen. 2018. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*. 31–36.

[44] Hongshan Li, Chenghao Hu, Jingyan Jiang, Zhi Wang, Yonggang Wen, and Wenwu Zhu. 2018. Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution. In *2018 IEEE 24th international conference on parallel and distributed systems (ICPADS)*. IEEE, 671–678.

[45] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 359–376.

[46] Neiwen Ling, Kai Wang, Yuze He, Guoliang Xing, and Daqi Xie. 2021. RT-mDL: Supporting Real-Time Mixed Deep Learning Tasks on Edge Platforms. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 1–14.

[47] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge Assisted Real-Time Object Detection for Mobile Augmented Reality. In *The 25th Annual International Conference on Mobile Computing and Networking* (Los Cabos, Mexico) *(MobiCom '19)*. Association for Computing Machinery, New York, NY, USA, Article 25, 16 pages. https://doi.org/10.1145/3300061.3300116

[48] Ruoyang Liu, Lu Zhang, Jingyu Wang, Huazhong Yang, and Yongpan Liu. 2021. PETRI: Reducing Bandwidth Requirement in Smart Surveillance by Edge-Cloud Collaborative Adaptive Frame Clustering and Pipelined Bidirectional Tracking. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 421–426. https://doi.org/10.1109/DAC18074.2021.9586088

[49] Jiachen Mao, Qing Yang, Ang Li, Hai Li, and Yiran Chen. 2019. MobiEye: An Efficient Cloud-Based Video Detection System for Real-Time Mobile Applications. In *Proceedings of the 56th Annual Design Automation Conference 2019* (Las Vegas, NV, USA) *(DAC '19)*. Association for Computing Machinery, New York, NY, USA, Article 102, 6 pages. https://doi.org/10.1145/3316781.3317865

[50] Jiayi Meng, Zhaoning Kong, Qiang Xu, and Y. Charlie Hu. 2021. Do Larger (More Accurate) Deep Neural Network Models Help in Edge-Assisted Augmented Reality?. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration* (Virtual Event, USA) *(NAI'21)*. Association for Computing Machinery, New York, NY, USA, 47–52. https://doi.org/10.1145/3472727.3472807

[51] Michaël Ramamonjisoa, Michael Firman, Jamie Watson, Vincent Lepetit, and Daniyar Turmukhambetov. 2021. Single image depth prediction with wavelet decomposition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11089–11098.

[52] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. 2020. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE transactions on pattern analysis and machine intelligence* (2020).

[53] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7263–7271.

[54] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

[55] Shuyao Shi, Jiahe Cui, Zhehao Jiang, Zhenyu Yan, Guoliang Xing, Jianwei Niu, and Zhenchao Ouyang. 2022. VIPS: Real-Time Perception Fusion for Infrastructure-Assisted Autonomous Driving. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking* (Sydney, NSW, Australia) *(MobiCom '22)*. Association for Computing Machinery, New York, NY, USA, 133–146. https://doi.org/10.1145/3495243.3560539

[56] Julien Simon. 2018. *Amazon Elastic Inference – GPU-Powered Deep Learning Inference Acceleration.* Retrieved July 14, 2022 from https://aws.amazon.com/blogs/aws/amazon-elastic-inference-gpu-powered-deep-learning-inference-acceleration/

[57] Joan Solà. 2017. Quaternion kinematics for the error-state Kalman filter. https://doi.org/10.48550/ARXIV.1711.02508

[58] Jamie Watson, Oisin Mac Aodha, Victor Prisacariu, Gabriel Brostow, and Michael Firman. 2021. The temporal opportunist: Self-supervised multi-frame monocular depth. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1164–1174.

[59] Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. 2019. Fastdepth: Fast monocular depth estimation on embedded systems. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 6101–6108.

[60] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. 2019. Detectron2. https://github.com/facebookresearch/detectron2.

[61] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. 2021. SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 12077–12090. https://proceedings.neurips.cc/paper/2021/file/64f1f27bf1b4ec22924fd0acb550c235-Paper.pdf

[62] Xiufeng Xie and Kyu-Han Kim. 2019. Source Compression with Bounded DNN Perception Loss for IoT Edge Computer Vision. In *The 25th Annual International Conference on Mobile Computing and Networking* (Los Cabos, Mexico) *(MobiCom '19)*. Association for Com-

puting Machinery, New York, NY, USA, Article 47, 16 pages. https://doi.org/10.1145/3300061.3345448

[63] Ran Xu, Jayoung Lee, Pengcheng Wang, Saurabh Bagchi, Yin Li, and Somali Chaterji. 2022. LiteReconfig: Cost and Content Aware Reconfiguration of Video Object Detection Systems for Mobile GPUs. In *Proceedings of the Seventeenth European Conference on Computer Systems* (Rennes, France) *(EuroSys '22)*. Association for Computing Machinery, New York, NY, USA, 334–351. https://doi.org/10.1145/3492321.3519577

[64] Ran Xu, Chen-lin Zhang, Pengcheng Wang, Jayoung Lee, Subrata Mitra, Somali Chaterji, Yin Li, and Saurabh Bagchi. 2020. ApproxDet: Content and Contention-Aware Approximate Object Detection for Mobiles. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems* (Virtual Event, Japan) *(SenSys '20)*. Association for Computing Machinery, New York, NY, USA, 449–462. https://doi.org/10.1145/3384419.3431159

[65] Chun-Han Yao, Chen Fang, Xiaohui Shen, Yangyue Wan, and Ming-Hsuan Yang. 2020. Video Object Detection via Object-Level Temporal Aggregation. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 160–177.

[66] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 476–488.

[67] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2020. NEMO: Enabling Neural-Enhanced Video Streaming on Commodity Mobile Devices. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (London, United Kingdom) *(MobiCom '20)*. Association for Computing Machinery, New York, NY, USA, Article 28, 14 pages. https://doi.org/10.1145/3372224.3419185

[68] Juheon Yi and Youngki Lee. 2020. Heimdall: mobile gpu coordination platform for augmented reality applications. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.

[69] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 325–338. https://doi.org/10.1145/2785956.2787486

[70] Ming Guang Yong. 2019. *Object Detection and Tracking using MediaPipe*. Retrieved April 22, 2022 from https://developers.googleblog.com/2019/12/object-detection-and-tracking-using-mediapipe.html

[71] Fuxun Yu, Shawn Bray, Di Wang, Longfei Shangguan, Xulong Tang, Chenchen Liu, and Xiang Chen. 2021. Automated Runtime-Aware Scheduling for Multi-Tenant DNN Inference on GPU. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9. https://doi.org/10.1109/ICCAD51958.2021.9643501

[72] Huangying Zhan, Chamara Saroj Weerasekera, Jia-Wang Bian, and Ian Reid. 2020. Visual Odometry Revisited: What Should Be Learnt?. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 4203–4210. https://doi.org/10.1109/ICRA40945.2020.9197374

[73] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. 2018. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 236–252.

[74] Jinrui Zhang, Deyu Zhang, Xiaohui Xu, Fucheng Jia, Yunxin Liu, Xuanzhe Liu, Ju Ren, and Yaoxue Zhang. 2020. MobiPose: Real-Time Multi-Person Pose Estimation on Mobile Devices. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems* (Virtual Event, Japan) *(SenSys '20)*. Association for Computing Machinery, New York, NY, USA, 136–149. https://doi.org/10.1145/3384419.3430726

[75] Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. 2021. Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 201–214.

[76] Husheng Zhou, Soroush Bateni, and Cong Liu. 2018. S3̂DNN: Supervised Streaming and Scheduling for GPU-Accelerated Real-Time DNN Workloads. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 190–201. https://doi.org/10.1109/RTAS.2018.00028