On Balancing Latency and Quality of Edge-native Multi-view 3D Reconstruction

Xiaojie Zhang[1], Houchao Gan[2], Amitangshu Pal[3], Soumyabrata Dey[2], Saptarshi Debroy[1]
[1]City University of New York, USA; [2]Clarkson University, USA; [3]Indian Institute of Technology Kanpur, India
xzhang6@gradcenter.cuny.edu,ganh@clarkson.edu,amitangshu@cse.iitk.ac.in,sdey@calrkson.edu,saptarshi.debroy@hunter.cuny.edu

Abstract

Multi-view 3D reconstruction driven augmented, virtual, and mixed reality applications are becoming increasingly edge-native, due to factors such as, rapid reconstruction needs, security/privacy concerns, and lack of connectivity to cloud platforms. Managing edge-native 3D reconstruction, due to edge resource constraints and inherent dynamism of 'in the wild' 3D environments, involves striking a balance between conflicting objectives of achieving rapid reconstruction and satisfying minimum quality requirements. In this paper, we take a deeper dive into multi-view 3D reconstruction latency-quality trade-off, with an emphasis on reconstruction of dynamic 3D scenes. We propose data-level and task-level parallelization of 3D reconstruction pipelines, holistic edge system optimizations to reduce reconstruction latency, and long-term minimum reconstruction quality satisfaction. The proposed solutions are validated through collection of real-world 3D scenes with varying degree of dynamism that are used to perform experiments on hardware edge testbed. The results show that our solutions can achieve between 50% to 75% latency reduction without violating long term minimum quality requirements.

CCS Concepts: • **Computer systems organization** → **Real-time systems**; **Distributed architectures**; *Reliability*; • **Mathematics of computing** → *Network optimization*; • **Human-centered computing** → *Ubiquitous and mobile computing systems and tools.*

Keywords: 3D reconstruction, edge computing, openMVG, openMVS, latency optimization, quality satisfaction

ACM Reference Format:
Xiaojie Zhang[1], Houchao Gan[2], Amitangshu Pal[3], Soumyabrata Dey[2], Saptarshi Debroy[1]. 2023. On Balancing Latency and Quality of Edge-native Multi-view 3D Reconstruction. In *The Eighth ACM/IEEE Symposium on Edge Computing (SEC '23), December 6–9, 2023, Wilmington, DE, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3583740.3630267

1 Introduction

With the widespread adoption of video processing use cases, such as, robotic surveillance [12, 34], public safety [35–37], and tactical scenarios [4], creating reliable augmented/virtual/mixed reality [8] environments is becoming critical. For such applications, real world objects (often dynamic) captured from multiple vantage points (through camera enabled devices) are needed to be placed within a virtual 3D environment. Thus, multi-view 3D reconstruction is playing a critical role to efficiently create such reliable immersive environments, especially in aforementioned mission-critical use cases. Fig. 1 describes one such multi-view 3D reconstruction-driven fire rescue recon mission where ground first responders capture videos of a high-rise fire situation involving human beings and other living objects trapped in fire using camera equipped drones. The drones live stream the 2D video frames for rapid 3D reconstruction of the fire scene, before sending the reconstructed outcome to ground first responders. Carrying out such 3D reconstruction in cloud data centers (especially for mission-critical use cases) is considered impractical due to three key factors: 1) the substantial data transmission demands, 2) the risk of losing connectivity to data centers, and 3) privacy and security concerns. In contrast, edge computing can become an important enabler towards such rapid 3D reconstruction by bringing compute resources (e.g., CPU, GPU) closer to the video generation and consumption site(s). Fig. 1 shows adoption of such edge system where a ground fire truck equipped with communication and computational resources acts as 'on-premise edge servers' that runs 3D reconstruction algorithms and sends the reconstructed data to the hand-held devices of the first responders.

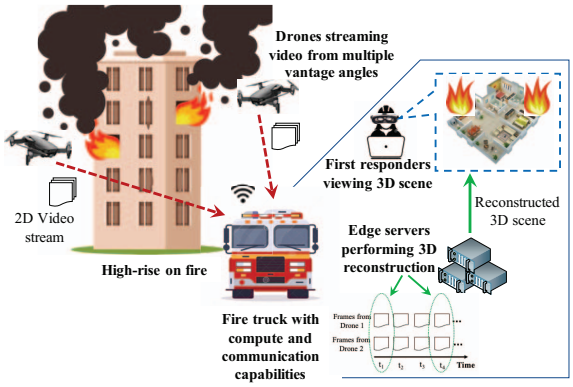


Figure 1. Fire rescue involving first responders viewing 3D constructed scene from 2D video streams captured by drones from multiple vantage angles

Traditionally, 3D reconstruction is achieved by photogrammetric algorithms, such as Structure from Motion (SfM), that compute image features and matchings across views from a set of unordered 2D images [15] which can then be intensified and textured by Multi-view Stereo (MVS) methods [5]. Unlike, simpler video processing applications, SfM+MVS pipeline based 3D reconstruction methods, such as, widely used openMVG/openMVS [5, 15, 24] are extremely compute-intensive, and thereby time-consuming, especially when reconstructing 'in the wild', i.e., unknown real-world scenes for critical use cases. Most SfM+MVS pipeline-based multi-view 3D reconstruction methods are designed to focus on reconstruction quality, and not on rapid processing. Thus, they are well suited to run on cloud environments with theoretically unlimited compute resources. Consequently, when run on computationally constrained typical edge severs, it is non-trivial to generate high-quality results that can successfully reconstruct all the dynamic objects inside a 3D

SEC '23, December 6–9, 2023, Wilmington, DE, USA

ACM ISBN 979-8-4007-0123-8/23/12…$15.00
https://doi.org/10.1145/3583740.3630267

scene within a short amount of time. E.g., a typical video stream of 80 sec. from 5 different cameras (with 2000 frames per camera @ 25 frames/sec) would typically need 9 hours to generate high quality reconstruction on typical low-cost edge servers. Any attempt to reduce the latency will severely impact the reconstruction quality. Such inability to produce high quality reconstruction can severely impact the success of the involved mission.

Therefore, one of the most critical challenges of edge-native 3D reconstruction is to ensure rapid processing with limited impact on their quality, i.e., striking a balance between reconstruction latency and quality. However, there exists limited work that deal with latency reduction of complex video processing applications, such as 3D reconstruction. Works, such as [10, 19, 28] propose edge-native frameworks for video analytics by jointly selecting of a variety of configurations. Others, such as [21, 32, 33] work on pipeline scheduling problems. However, both groups often ignore the necessity of application-specific optimization, thus are unsuitable for 3D reconstruction application. Furthermore, none of the existing works aim to capture the characteristics of video data content (captured from unknown real-world scenes) where components of the scene are often dynamic. 3D reconstruction of such dynamic scenes typically involves one or many moving objects with changing locations, poses, or shapes. Because of the presence of these dynamic (foreground) objects, the 3D information of the scene changes over time, requiring the reconstruction algorithm to recompute the 3D map iteratively. Inability to capture such dynamism efficiently and promptly may lead to massively redundant computation, resulting in prolonged latency. Contrarily, attempt to reduce reconstruction latency without addressing such dynamism can lead to considerably low quality and useless reconstruction.

To address these challenges, in this paper, we take a deeper dive into multi-view 3D reconstruction latency-quality inter-conflict, with an emphasis on reconstructing dynamic scenes. In particular:

- We address the lack of dynamic scene reconstruction in the current literature by setting up our own 3D scenes that mimic real-world scenarios and by collecting long sequence multi-view video datasets with different degrees of dynamism within the scene.
- We use these datasets to perform benchmarking experiments to demonstrate the need for intelligent orchestration of 3D reconstruction related data and resources to achieve optimal latency-quality trade-offs to sustain desired performance.
- We propose a hybrid approach of practical data-driven adaptations and more holistic system optimizations of edge resources. In particular, we model the entire process of reconstructing a dynamic 3D scene as a series of upcoming reconstruction tasks.
- We implement a difference detector to filter out consecutive tasks with high similarity and assign only a subset of tasks to the edge server for execution. This reduces computation redundancy.
- We establish analytical models, based on data-driven measurements, to characterize the relationship between system parameters impacting reconstruction latency and quality requirements.
- We propose a long-term and dynamic optimization problem that is solved with Lyapunov optimization with virtual queue that transforms the problem into multiple, per-epoch, and computationally tractable optimization problems.
- We propose a window-based pipeline that implements task execution, difference detection, and optimization in parallel through in-advance optimization and delayed-task assignment.

We implement and evaluate the proposed solutions on a hardware edge testbed using our own and publicly available datasets. Our evaluation methodology focuses on both long-term quality satisfaction and reduction of reconstruction latency of the entire video stream. The experiment results show that our solutions are able to flexibly balance long-term quality satisfaction and total reconstruction time. Moreover, the proposed pipeline parallelization approach executes much faster than the traditional approach with guaranteed quality bounds. Compared to the traditional approach, the reconstruction latency is reduced by 15% to 30% based on different system configurations. We also compare our solution with baseline strategies that do not account for the dynamic changes in the video content. Through experiment on real-world datasets, we demonstrate that our solution greatly (between 50% to 75%) reduces the total reconstruction latency (based on video content and the quality requirements) by eliminating unnecessary reconstructions.

The rest of the paper is organized as follows. Section 2 presents the background and related work. Section 3 discusses data collection and problem evidence analysis. Section 4 presents system model and problem formulation. Section 5 discusses Lyapunov optimization based solution and system design. Section 6 discusses implementation and evaluation. Section 7 concludes the paper.

2 Background and Related Work

In order to better articulate the detailed contributions of our work, we briefly explain the multi-view 3D reconstruction of dynamic scenes with particular emphasis on the most widely used SfM+MVS pipeline, along with the state-of-the-art in latency reduction of such.

2.1 Multi-view 3D reconstruction of Dynamic scenes

Multi-view 3D reconstruction is an extremely complex video processing application that is typically carried out by forming geometric relations of the image pixels through photogrammetric algorithms, such as Structure from Motion (SfM) [15, 23]. The SfM process estimates the 3D coordinates of a scene's feature points (captured by multiple cameras from their vantage points) belonging to both the dynamic (i.e., moving, e.g., human beings, moving objects) and static (i.e., stationary, e.g., furniture, walls) components, referred to as the sparse 3D point cloud, along with the camera poses (e.g., locations and orientations). This is followed by MVS dense point cloud estimation steps [27] that utilize the sparse point cloud and interpolation techniques to generate a realistic 3D scene. Dynamic scenes typically involve one of many moving objects with changing locations, poses, or shapes. Because of the presence of these dynamic (foreground) objects, the 3D information of the scene changes over time requiring the reconstruction algorithm to recompute the 3D map iteratively. Recently, a new branch of methods has emerged that uses deep learning models with single or multiple images for the 3D shape estimation of single or multiple objects, as well as 3D reconstruction of the whole scene [22]. Although these approaches are generally faster, SfM+MVS pipeline-based methods are still considered to be the golden standard for 3D reconstruction as deep learning based methods yield lower accuracy and do not generalize well for unknown objects and scenes [22] that are quite common for aforementioned mission-critical use cases. Among the SfM+MVS based multi-view reconstruction techniques that yield high quality 3D reconstruction of dynamic scenes, openMVG/openMVS [5, 15] is the most widely adopted because of its friendly coding style, modular design, and open-source nature [15],

Figure 2. Captured scenes from different view angle

making them easy to modify depending on the specific needs of particular applications. This provides the developers with complete control over the implemented functionalities [24]. For the purpose of this work, we analyze the 3D reconstruction latency-quality trade-off issues and propose solutions that are geared towards openMVG/openMVS. However, the proposed methods are equally effective for any SfM+MVS based 3D reconstruction pipelines.

2.2 Techniques to Optimize 3D Reconstruction Latency

SfM+MVS pipelines including openMVG/openMVS, like most other multi-view 3D reconstruction methods, are extremely computation-intensive and thereby time-consuming, especially when performed within a resource-constrained edge environment. This is particularly true for reconstructing large 'in the wild', i.e. unknown scenes (e.g., large public places, disaster-stricken areas, and enemy territories) that typically need a large number of high-resolution cameras to capture the target scene from different angles, involving many dynamic components. There exists only a few works in the current literature that seek to reduce 3D reconstruction processing latency. Among them, in [7], the authors directly use the RGB-D camera to obtain the depth information of the target scene, instead of running cumbersome algorithms to calculate the depth map from a regular RGB image set. In order to reduce the optimization time, the authors in [25] group several pixels into superpixels, i.e., a new point cloud after generating an one.

There exists even fewer methods that specifically work on latency reduction of SfM+MVS pipelines. Authors in [11] sort the input images based on the spatial orders of the cameras ensuring large overlaps between two subsequent images of the ordered set in order to reduce the computation cost in the feature-matching step. In [31], the authors optimize the densify point cloud step with a quasi-dense feature matching approach and achieved 9% improvement in latency. Authors in [30], group the sparse 3D points into different clusters and process each cluster separately for dense textured mesh generation, resulting in 13% reduction in total processing time. In our previous work [34], we separate a scene into foreground and background parts to perform 3D reconstruction separately before merging the outcomes at the end. However, reconstruction times of the dynamic scenes for most of the above solutions are still too long to be considered rapid 3D reconstruction (about 12 seconds per image set), which we seek to address.

3 Problem Evidence Analysis

In this section, we first discuss the motivation and details of lab based multi-view 3D dataset generation that are tailor-made for mission-critical use cases mentioned earlier. This is followed by 3D reconstruction latency-quality inter-conflict problem evidence analysis using the same datasets.

3.1 Multi-view Dataset Generation

As mentioned earlier, latency reduction of multi-view 3D reconstruction pipelines while satisfying minimum quality requirements is non-trivial for scenes with dynamic objects. Currently, only a few publicly available datasets (e.g., Dance1, Odzemok [16]) are available that can be used and tested for 3D reconstruction of such dynamic scenes. These datasets contain a limited number of sequential images and only a few subsets can generate high-quality 3D reconstruction results. Short sequence datasets are more likely to run out of images before any optimization can converge, leading to non-accurate optimization performance. Many recent works [17] advocate the need for long sequence of dynamic scenes to test and improve the robustness of 3D reconstruction related optimizations. Additionally, sound analysis of 3D reconstruction latency-quality inter-conflict and validation of optimization solutions for multi-view 3D reconstruction use cases (e.g., robotic surveillance, search and rescue, and tactical scenarios) that require rapid processing at high quality, warrant datasets of 3D scenes that are representative of such use cases in terms of scene complexity and dynamism. Unfortunately, none of the existing 3D datasets provide that.

Background scene setup: Thus, for our data collection, we aim to create an indoor scene that is colorful, full of diverse objects, and has different degrees of dynamism in terms of the moving objects. In the scene, toys and food items are selected because they come in multiple colors. Boxes are used to cover most empty space of the scene; a wooden frame with a white sheet is placed in the background; a black chair is placed in the middle of the scene; and a large multi-colored toy is placed at 45 degrees of the chair to increase the overall brightness of the scene. All the camera positions are adjusted based on the chair in the middle, as it is the focal point of the scene for different dynamic activities. The complete scene from different view points is shown in Fig. 2.

Acquisition system setup: A high-level representation of the acquisition system is shown in Fig. 3 where we implement a Python UDP-based script operating as client-server network structure. The socket program runs on a Host PC that acts as the server while 5 Raspberry pi boards act as clients. Similar to several other research works [1, 2, 14], we use Raspberry Pi Module 2 cameras and Model B boards due to their low price and ease of availability. Each Raspberry Pi camera (with highest resolution 1980×1080) connects to a Raspberry Pi board and with board connected to the Host PC through WiFi. The video capture (at a variable frame rate of 25-30 frames/sec.) starts when the host PC sends a 'start recording' command to all boards simultaneously. To finish video capture, the Host PC sends a 'stop recording' command to all boards. Once a board receives the stop command, it saves the captured videos locally as H264 files which are later transferred to the PC. The acquisition system also has the capability of live streaming the video to the server. We use OpenCV to extract all images from the H264 files which are then used for 3D reconstruction using openMVG/openMVS.

Based on the above background scene and acquisition system setups, we generate three multi-view datasets representing different degrees of indoor dynamic scenes, viz., *Pickup*, *Walk*, and *Handshake* as shown in Fig.4. In *Pickup*, a person sits on the middle chair and picks up a book from the ground. This scene represents low

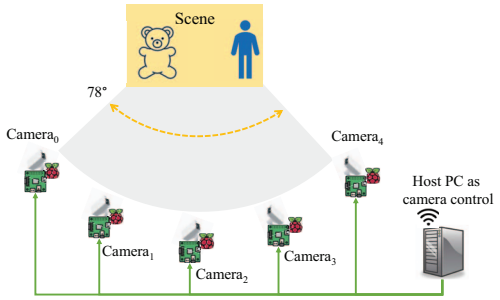


Figure 3. Acquisition system for capturing 3D scenes

(a) *Pickup* (b) *Walk* (c) *Handshake*

Figure 4. Image samples from the captured multi-view dataset

movement with the rest of the scene being static. In a comparatively more dynamic scene *Walk*, a person walks across the scene from one end to another. Finally, *Handshake* represents high level of dynamism where two people walk from two different side of the scene and shake hands in the middle. Using these three datasets, we next provide key evidences of non-triviality of latency-quality inter-conflict for 3D reconstruction. Later we will recreate these scenes for evaluating and validating our solutions in Section 6.

3.2 Problem Evidence Analyses

The objectives of problem evidence analyses are threefold: *first*, we show how the traditional SfM+MVS pipelines, without any optimization, fail to guarantee rapid 3D reconstruction; *second*, we demonstrate how 'quick-fix' strategies to reduce processing latency that work for other simpler video processing application, do not work for 3D reconstruction quite so well; and *third*, how 3D reconstruction of highly dynamic environments make the latency-quality inter-conflict even more challenging to address. Overall, these analyses results will motivate the need for optimizations to strike latency-quality trade-off. To this end, we use openMVG/openMVS on our collected *Pickup*, *Walk*, and *Handshake* datasets. The reconstructed 3D scenes are obtained by processing the video sequences on a Dell Gigabyte desktop with AMD Ryzen9 3900X @3.8GHz, 125GB RAM, and NVIDIA GeForce RTX 1080Ti, which is a typical configuration for a low-cost edge device used in various edge-native use cases.

Experiment results: From Table 1, it can be observed that the best quality 3D rendering is achieved when the highest resolution (i.e., setting scale=1) video frames/images are used for reconstruction. Here, for quality evaluation, we use widely accepted F-score metric proposed in [20]. From the table, we also observe that this baseline case requires a latency of 16.86 sec. to reconstruct one set of images (for *Pickup*). This means that a video steam consisting of 2000 frames/images from each camera, would need $2000 \times 16.86 \approx 9$ hours to complete reconstruction, which, by far, fails to meet the rapid reconstruction requirements of many critical use cases [29].

Now, one of the most obvious and popular 'quick fix strategies' among researchers [33] is to reduce the resolution of images collected from the cameras with the objective of reducing the total data size for processing. Table 1 illustrates that when 30% resolution reduction is applied (i.e., scale=0.7), the latency indeed reduces to 11.19 sec., but leads to significant degradation of reconstruction quality (i.e., from F-score 1.0 to 0.92). The same can be observed through qualitative analysis in Fig. 5, which shows that from a visual perspective, reconstructed scenes with 70% resolution lack the richness and many important details as compared to the best quality reconstructed scenes shown in Figs. 5(a) and 5(c). Such degradations can severely impair proper operation of critical use cases using 3D reconstruction. Both quantitative and qualitative results demonstrate that this trend of latency reduction at the cost of quality degradation is consistent across datasets.

We also observe that such quality degradations get magnified for scenes that have relatively pronounced dynamic components. E.g., in *Pickup*, the quality degradations from scale=1 to scale=0.7 and scale=0.5 are 7% and 15% respectively. Whereas, with more dynamism in the scene, such degradations increase to 9% and 18%, and to 11% and 25% (as shown in Table 1). Similar trends can be observed in Fig. 5, where more details are missing in reconstructed *Walk* scene than reconstructed *Pickup* scene. *These results together demonstrate the need for intelligent orchestration of 3D reconstruction application related data and resources, to achieve optimal latency-quality trade-offs for sustaining desired performance.*

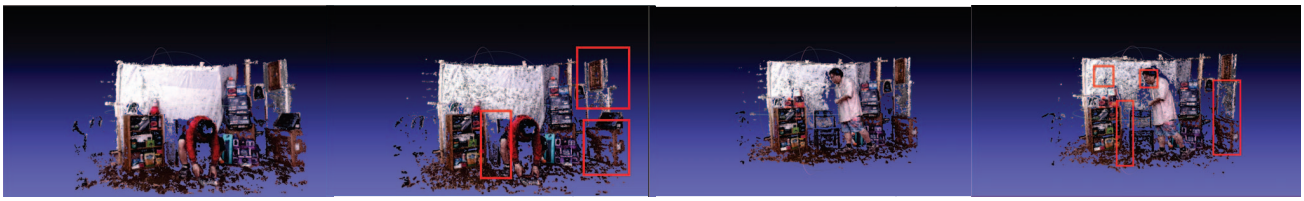
4 System Model and Problem Formulation

As multi-view SfM+MVS algorithms are extremely time-consuming, especially in use cases with dynamic scenes that need to continuously reconstruct 3D scenes for every upcoming set of 2D images (defined as a *task*), not only the reconstruction time of every single such task needs to be shortened, but the total reconstruction time of a sequence of tasks (i.e., the entire video stream) needs to be reduced. *In order to solve this fundamental problem of reducing 3D reconstruction latency at the edge with less than significant impact on reconstruction quality, we take a hybrid approach that implements practical data-driven adaptations, as well as more holistic system optimizations.* Thus, we seek to design and implement an approach that intelligently executes reconstruction tasks with different frequencies and frame resolutions based on the desired quality requirement.

For this work, we assume a typical edge-native real-world use case (such as, robotic surveillance, search and rescue, tactical scenarios) where the edge computing system comprises of a heterogeneous resource pool (in terms of available CPU and GPU units, RAM size) that has considerably more computational capacity than the camera-enabled devices themselves, but nothing close to the resources available at typical cloud data centers. In accordance with real-world scenarios, we assume that the resource components, i.e., computing units are connected with each other through high-bandwidth connections. Thus, the inter-communication delay for data offloading between such units is ignored. The computational units are primarily categorized into two: i) difference detector (DD) and ii) a set of servers $\{1, 2, ..., K\}$, both running continuously. The DD controls the task (as explained earlier, a set of 2D images to be reconstructed) execution frequency by filtering tasks with high similarity, while the servers run reconstruction tasks in batches. In this paper, we seek perform data-level and task-level parallelization of 3D reconstruction pipeline (as part of the *data-driven adaptations*) and optimize the following parameters (as part of the *holistic system optimizations*):

Table 1. 3D reconstruction quality (in F-score) and latency comparison with varying camera resolution

	(scale=1 *Pickup*)	(scale=0.7 *Pickup*)	(scale=0.5 *Pickup*)	(scale=1 *Walk*)	(scale=0.7 *Walk*)	(scale=0.5 *Walk*)	(scale=1 *Handshake*)	(scale=0.7 *Handshake*)	(scale=0.5 *Handshake*)
F-score	1.0	0.93	0.85	1.0	0.91	0.82	1.0	0.89	0.75
Latency	16.86	11.19	NA	16.43	10.6	NA	16.96	8.97	NA



(a) *Pickup* with scale=1 (b) *Pickup* with scale=0.7 (c) *Walk* with scale=1 (d) *Walk* with scale=0.7

Figure 5. Qualitative analysis of openMVG/openMVS pipeline with varying camera resolution

- *Difference threshold d*: The objective of DD is to filter out tasks with high similarity and thus eliminate unnecessary reconstructions (i.e., computations) that considerably lengthen the reconstruction process over time. However, achieving this for dynamic scenes is non-trivial. Our proposed DD computes 'percentage pixel-wise difference' between two image frames. Specifically, it computes the percentage difference between the current frame and the frame corresponding to the last reconstructed task, for each camera viewpoint. Subsequently, as the average percentage differences are estimated for all cameras, the current task is executed (i.e., reconstruction for the current set of frames is performed) only when such average difference exceeds a certain threshold, viz., difference threshold d. By adjusting d, the DD is able to adapt reconstruction frequency that can balance reconstruction latency and quality.
- *Image resizing ratio r*: As explained in Section 3.2, image resolution or image resizing ratio $r \in (0, 1]$ plays a key role in reducing reconstruction latency. However, this may come at a price of degraded quality. In particular, the choice of r also impacts the ability of proper reconstruction of dynamic objects, and failing to do so can severely impact the reconstruction quality. Thus, we choose r as a parameter for our optimization process.
- *Task assignment* $\mathbf{x}$: Finally, we define a task assignment/placement parameter $\mathbf{x} = \{x_1, ..., x_K\}$: x_k that indicates the number of reconstruction tasks assigned to the k-th server. Since servers are heterogeneous, the goal of task assignment is to minimize the maximum task completion time of all servers.

Overall, the system parameter tuple or policy $\{d, r, \mathbf{x}\}$ is optimized to balance reconstruction latency and quality. With the system parameters defined, we first discuss the 3D reconstruction pipeline parallelization, followed by the optimization problem formulation.

4.1 Reconstruction Pipeline Parallelization

In order to explain the pipeline parallelization, we first divide the timeline between the capture of first set of images for reconstruction and the completion of last reconstruction into successive optimization epochs $t \in \{1, 2, ..., T\}$. Without parallelization, the following steps that include the 3D reconstruction tasks as well as difference detection, will be executed in sequence at each epoch t: i) running DD for upcoming tasks at the beginning of epoch t, ii) choosing a batch of tasks for reconstruction by filtering other tasks with high similarity, iii) at the end of epoch t, letting the servers execute the selected tasks, and iv) making adjustments to the system parameters, particularly d and r based on observed quality (i.e., re-optimization). Here, the system parameters remain unchanged for the tasks within the window. In epoch t, a task window is defined (of size X^t) which consists of all the outstanding reconstruction tasks that were queued between two successive difference detection periods. Let this window and the set of tasks within this window be denoted as $\mathcal{I}^t = \{i, i+1, ..., i+X^t-1\}$. The DD selects a subset of the tasks $\mathcal{I}^t_{DD}$ to be executed for task window $\mathcal{I}^t$; subsequently, $\mathcal{I}^t_{-DD}$ denotes the set of tasks not selected. This relationship can be described as $\mathcal{DD} : \mathcal{I}^t \rightarrow \mathcal{I}^t_{DD}$. Important to note that $1 \leq |\mathcal{I}^t_{DD}| \leq X^t$ as the first task at the very beginning of the reconstruction process is always selected for obvious reasons.

Next, we define task computation period α^t which denotes the time taken to complete the execution of tasks in task window $\mathcal{I}^t_{DD}$. Also, if the speed of DD is b tasks per second, then the duration of DD period can be estimated by $\beta^t = X^t/b$. However, since DD needs to compute the average difference between consecutive image frames coming from all cameras, the computational overhead of DD increases linearly with the number of cameras. Therefore, the execution time of DD period β^t in epoch t cannot be ignored in practice. In order to address that challenge, a pipeline parallelization approach, as shown in Fig. 6 is deployed. The figure shows that when the selected tasks in $\mathcal{I}^t_{DD}$ are being executed, shown as the computation period α^t, the optimization process starts running DD for the next epoch $t+1$ (shown as β^{t+1}) as part of the task window $\mathcal{I}^{t+1}$. During this epoch, the optimization process (to be introduced in Section 5) that runs for a small period of time (as shown in Fig. 6) first determines the optimization parameters for the new epoch, i.e., difference threshold d^{t+1} and image resizing ratio r^{t+1}. The new difference threshold d^{t+1} is then used by DD to select new tasks to be executed, i.e., $\mathcal{I}^{t+1}_{DD}$ from the outstanding set of tasks in $\mathcal{I}^{t+1}$. The DD performs the task filtering process until all the selected tasks from the previous epoch t are finished execution, i.e., till the end of α^t. Given the speed for task filtering by DD, the size of the new task window in terms of number of outstanding tasks can be expressed as $X^{t+1} = \alpha^t \times b$. This pipeline runs until the last set of outstanding tasks are selected and executed (if needed). With this arrangement, the execution of selected tasks for the current epoch t and the execution of DD for the next epoch $t+1$ are run in parallel, resulting in $\beta^{t+1} = \alpha^t$. As a result, the computing units for DD and the servers are always fully utilized with the overall reconstruction time being considerably shortened.

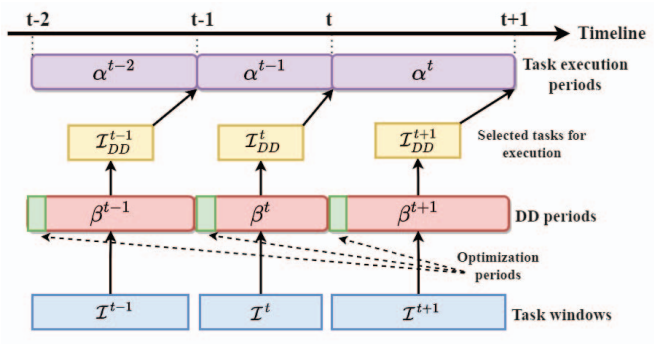


Figure 6. Parallel pipeline with task windows, task selection, and execution

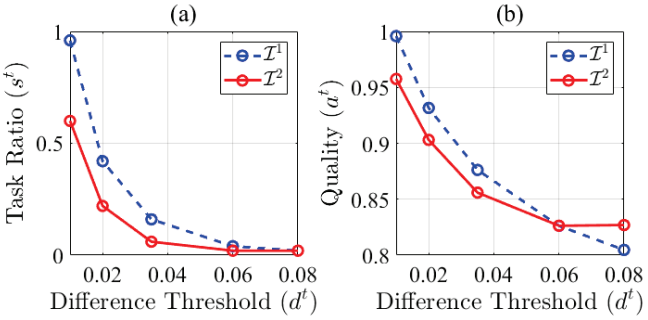


Figure 7. Scheduled task ratio and reconstruction quality against d^t for *Walk*

4.2 Task Selection by DD

With reconstruction pipeline parallelization in place, we next discuss the different aspects of holistic system optimization, starting with the task selection process by the DD. To this end, we define $s^t = \frac{|\mathcal{I}^t_{DD}|}{|\mathcal{I}^t|}$ as the ratio of selected tasks for task window $\mathcal{I}^t$, where $|\mathcal{I}^t| = X^t$. A higher s^t would mean longer α^t and vice versa. Based on such definition, s^t should be a function of the selected difference threshold d^t. On the other hand, the task selection by DD also depends on features of the images belonging to the tasks in the current task window in comparison to the previous window; more precisely, on the differences among the sequence of 2D images belonging to the task windows, especially when there are dynamic objects involved. Therefore, it is important to characterize the behavior of s^t against these two factors. In order to do that, we analyze how d^t affects s^t for different sequences of the same 3D scene through experiments performed on our own 3D dataset. To this end, we randomly select two task windows (viz., $\mathcal{I}^1$ and $\mathcal{I}^2$) of size 50 tasks (starting at different timestamps) in *Walk* dataset which signify moderate dynamism. For now, we do not resize the images; r^t is kept at 1, i.e., image resolution of 1920×1080. The experiment performs task selection through DD over these two sets of 50 tasks with different values of d^t, ranging from 0.01 to 0.08 (as experiments show that the entire range of the task ratio s^t, i.e., [0,1] is observed for this particular range of d^t).

In Fig. 7(a), we observe that the interactions between s^t and d^t vary for different task windows, viz., $\mathcal{I}^1$ and $\mathcal{I}^2$. This is caused by the different motion characteristics of the dynamic object within the scene at different time periods pertaining to the task windows. In our *Walk* dataset, the person moves slower during $\mathcal{I}^2$ than during $\mathcal{I}^1$; more precisely, sitting down on a chair versus walking across the room. As a result, more tasks are filtered out by DD during $\mathcal{I}^2$ than in $\mathcal{I}^1$, especially for small difference thresholds. However, irrespective of the task window and the relative motion of the dynamic object, with higher threshold, the task ratio s^t declines rapidly, signifying smaller amount of tasks being selected for reconstruction, which effectively decreases the end-to-end latency. Overall, the relationship between s^t and d^t seems to exhibit a long-tail distribution. Therefore, we characterize the relationship between s^t and d^t as $s^t = \mathcal{S}(d^t; \mathbf{w^t})$, where $\mathbf{w^t}$ are the weights of the function. One of the fitted examples of $\mathcal{S}(d^t; \mathbf{w^t})$ is $s^t = w_1 \times e^{-w_2 d^t} + w_3$. The parameters of the fitted functions can be obtained by using *curve_fit* function from *SciPy* [26] which is an open-source Python library that allows non-linear regression.

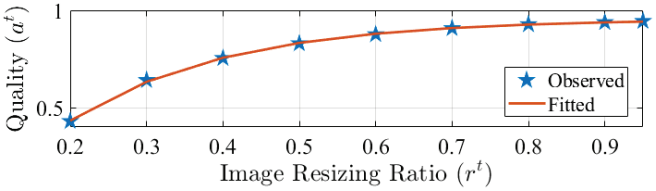


Figure 8. Reconstruction quality against r^t for a single task for *Walk*

Although the observation of reduced reconstruction latency with higher d^t is intuitive, there needs to be way to predict the potential impact of such adaptation of difference threshold on reconstruction quality. In Fig. 7(b), we analyze reconstruction quality in terms of F-scores (denoted by a^t) against the same range of d^t and for the same two task windows of the *Walk* dataset. Here, we observe similar temporal behavior (to that of s^t) for different sequence of tasks. Although both s^t and a^t reveal similar decreasing characteristic with respect to higher difference thresholds, there are subtle differences in the nature of a^t between task windows $\mathcal{I}^2$ and $\mathcal{I}^1$. Based on the observations, we define function $a^t = \mathcal{A}(d^t; \mathbf{j^t}) = j_1 \times e^{-j_2 d^t} + j_3$ to estimate the reconstruction quality at epoch t (with $r^t = 1$), where $\mathbf{j^t}$ is the weight vector of the function.

As mentioned earlier, both $\mathcal{S}(d^t; \mathbf{w^t})$ and $\mathcal{A}(d^t; \mathbf{j^t})$ functions assume task execution with original image resolution, i.e., with $r^t = 1$. Thus, we next observe how difference threshold d^t and image resizing ratio r^t jointly affect the functions, in particular quality a^t. We define the quality function of a single-task as $\mathcal{F}(r^t)$. In Fig. 8, we observe the characteristics of the quality function against r^t for different tasks in the same *Walk* dataset. $\mathcal{F}(r^t)$ exhibit stable behavior and can be perfectly fitted into a concave function of $\mathcal{F}(r^t) = 0.956 - 1.369 \times e^{-4.812 r^t}$ which provides a RMSE of 0.003. We argue that $\mathcal{F}(r^t)$ defines the upper bound of the reconstruction quality in epoch t (if the entire $\mathcal{I}^t$ is executed), while $\mathcal{A}(d^t; \mathbf{j^t})$ gradually increases or reduces the quality by adjusting the difference threshold d^t. Therefore, a reasonable assumption is that these two factors (i.e., d^t and r^t), affect the quality a^t independently. Thus, we estimate the average quality for tasks in the same epoch t by:

$$a^t \approx \mathcal{A}(d^t; \mathbf{j^t}) \times \mathcal{F}(r^t) \tag{1}$$

Nevertheless, such quality approximation exhibits about 0.04 to 0.10 error based on the sequence of images between task periods (as we have observed in $\mathcal{I}^1$ and $\mathcal{I}^2$). To address this error, we develop a mechanism that adjusts d^t and r^t appropriately to fill the gap between expected quality $a^t = \mathcal{A}(d^t; \mathbf{j^t}) \times \mathcal{F}(r^t)$ and the actual (i.e., observed) quality, say $\hat{a^t}$. This mechanism is discussed as a part of the holistic system optimization in Section 5. Important to note that although the models developed here are based on

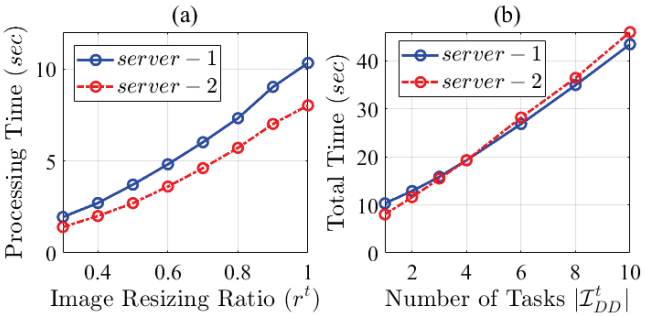


Figure 9. (a) The computation time of a single-task with different image resolution, (b) the total computation time of running multiple tasks in parallel

Walk dataset, the rationale and procedure behind such modeling remain universal for any arbitrary 3D scene with the help of run-time weight update method (Subsection 5.2), where we gradually generate these functions by collecting run-time observations.

4.3 Task Computation Model

For the next part of the holistic system optimization, we aim to find the relationship between task computation period α^t, image resizing ratio r^t, and the number of selected reconstruction tasks that needs to be processed in parallel (i.e., $|\mathcal{I}^t_{DD}|$), all for the time epoch t. Since such relationship is machine-specific, we run benchmarking experiments on machines with the configurations specified in Table 2. The computation/processing times of a single task against different r^t values are shown in Fig 9(a). Based on the nature of the curve, we fit and characterize the average single task computation time of the k-th server by $O_k(r^t;\mathbf{h^t}) = h_1 \times (r^t)^2 + h_2 \times r^t + h_3$, which is a monotonically increasing function (i.e., convex).

We next examine the capability of the edge servers to run multiple concurrent tasks, with $r^t = 1$. The results of such experiments, as shown in Fig. 9(b), show that when the number of tasks is less than 4, *server* − 2 computes faster than *server* − 1. However, beyond that, *server* − 1 becomes faster. This is due to *server* − 2's more powerful CPU but a weaker GPU resources in terms of *cuda* cores and memory size. We define function $\mathcal{P}_k(x^t_k;\mathbf{g^t})$ as the total computation time of k-th server running x^t_k tasks. Similar to the observations made in Section 4.2, we find that r^t and $|\mathcal{I}^t_{DD}|$ independently affect the total computation time. Thus, we define function $C_k(x^t_k, r^t)$ as the total computation time of k-th server handling x^t_k tasks at image resizing ratio r^t, which can be approximated by:

$$C_k(x^t_k, r^t) \approx \frac{O_k(r^t;\mathbf{h^t})}{O_k(1.0;\mathbf{h^t})} \times \mathcal{P}_k(x^t_k;\mathbf{g^t}) \tag{2}$$

Remark 1. *The shapes of $O_k(r^t;\mathbf{h^t})$ and $\mathcal{P}_k(x^t_k;\mathbf{g^t})$ are jointly determined by the machines' computation capacity and the 3D scene. Therefore, these two functions only need to be shaped once according to the use case.*

We also examine the average computation time per task in Fig. 10 while the total computation time is observed in Fig. 9(b). We observe that the average computation time decreases as more tasks are run in parallel. However, it tends to stabilize when the number of tasks is greater than 4 (due to resource limitation). Under such given resource conditions, we thus argue that our servers should run at least 4 tasks in parallel (if possible) at each epoch, in order to efficiently utilize the server resources. On the other hand, *server* − 1 and *server* − 2 should not run more than 10 and 8 tasks in parallel (respectively) to avoid over-utilization of their GPU memory (openMVS takes about 1GB of GPU memory). As shown in Fig. 10, we define such optimal number of tasks and the maximum number of tasks for the servers as x^{OPT}_k and x^{MAX}_k, respectively. These two markers are used to further optimize pipeline execution.

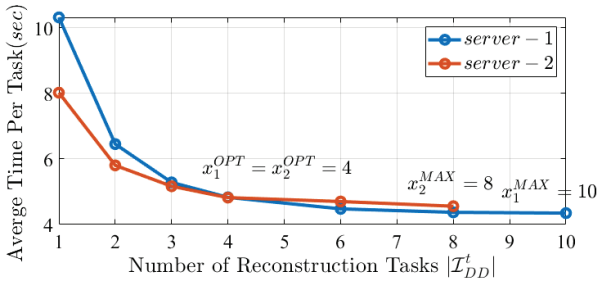


Figure 10. Average computation time of tasks when running in parallel

4.4 Holistic Optimization Problem Formulation

Next, we formulate the holistic optimization problem to balance reconstruction latency and quality. To that end, we define $C^t = \{C_1(x^t_1, r^t), C_2(x^t_2, r^t), ..., C_K(x^t_K, r^t)\}$ to be a collection of computation time of individual servers running the assigned x^t_k tasks with image resizing ratio r^t, at epoch t. Since the total computation time is determined by the slowest server, the objective of the optimal task assignment to servers is to minimize the maximum computation time of all servers, i.e., $\alpha^t = \max(C^t)$. Therefore, our long-term minimax problem is to find the optimal policy, i.e., system parameter tuple or policy $\{d^t, r^t, \mathbf{x}^t\}$ that minimizes the long-term sum of task computation time while satisfying the long-term quality requirement. The optimization problem thus formulated, can be stated as follows:

$$\min_{\mathbf{x}^t, r^t, d^t} \sum_{t=1}^{T} \max(C^t)$$

$$\text{s.t. } \mathbf{C1:} \sum_{k=1}^{K} x^t_k = \mathcal{S}(d^t;\mathbf{w}^t) \times X^t$$

$$\mathbf{C2:}\ r^{MIN} < r^t \leq 1$$

$$\mathbf{C3:}\ 0 \leq d^t \leq d^{MAX}$$

$$\mathbf{C4:} \sum_{t=1}^{T} \mathcal{A}(d^t;\mathbf{j}^t) \times \mathcal{F}(r^t) \times X^t \geq \sum_{t=1}^{T} (X^t \times A) \qquad \textbf{(P1)}$$

In problem (**P1**), **C1** illustrates the constraint of task assignment among K edge servers for an expected number of selected tasks with respect to d^t (i.e., $|\mathcal{I}^t_{DD}|$). Constraints **C2** and **C3** denote the range of image resizing ratios r^t and difference thresholds d^t, respectively. Constraint **C4** specifies that the long-term accumulated quality can not be lower than $\sum_{t=1}^{T}(X^t \times A)$, where $A \in (0, 1]$ is a pre-defined quality requirement for all tasks, while a^t is for one task period in epoch t. Overall, problem (**P1**) is a non-trivial problem to solve due the following challenges:

1. According to Figs. 7(a) and 7(b), having all time epochs share the same weights can lead to inaccurate predictions in $\mathcal{S}(d^t;\mathbf{w}^t)$ and $\mathcal{A}(d^t;\mathbf{j}^t)$. Therefore, we argue that the weights $\mathbf{w^t}$ and $\mathbf{j^t}$ are time-specific variables and need to be updated regularly.

Table 2. Machine Configuration

Machines	CPU details	GPU details
$server-1$	Intel(R) Xeon(R) Silver 4215R @3.20GHz×8	Nvidia RTX A4000 (6,144 CUDA cores), 16 GB Memory
$server-2$	Intel(R) Core(TM) i7-10700K @3.80GHz×8	Nvidia GeForce RTX 2060 SUPER (1,920 CUDA cores), 8 GB Memory
DD	Intel(R) Core(TM) i7-7700HQ CPU @2.80GHz×4	N/A

However, it is impractical to perform the same measurements as we did in Section 4.2 at beginning of each optimization epoch, since such weight updates inevitably prolong the optimization time, which is counter-productive.

2. Due to pipeline parallelization which makes DD and task computation to run in parallel (as shown in Fig. 6), problem (**P1**) is solved ahead of task computation period. Therefore, the value of X^t and the number of tasks actually selected by DD (i.e., $|\mathcal{I}^t_{DD}| = \mathcal{S}(d^t;\mathbf{w}^t) \times X^t$) may be different than expected. In this case, the system needs to perform task assignment again as constraint **C1** can be violated.
3. As mentioned in Section 4.2, the quality approximation defined in Eq. (1) is not 100% accurate. According to **C4**, the error in accumulated quality could be even larger. In fact, the accumulated error increases with X^t.
4. Finally, challenges 2 and 3 together make the prediction of long-term system performance even more challenging. If the system wants to strictly satisfy constraint **C4**, potential errors mentioned in challenges 2 and 3 must be addressed.

Since (**P1**) is a classic long-term optimization problem, we consider our proposed pipeline, as shown in Fig. 6, as a dynamic system where system parameters d^t, r^t, and $\mathbf{x}^t = \{x^t_1, x^t_2, ..., x^t_K\}$ are updated *on-demand* according to estimated quality defined in **C4**.

5 Optimization Solution and System Design

Due to the lack of availability of accurately fitted functions beforehand and the dynamism of 'in the wild' 3D scenes, we formulate the problem as a long-term average quality optimization, rather than independently optimizing in each epoch. Hence, we apply Lyapunov approach to create a virtual queue to alternatively express long-term constraint **C4** and the long-term objective function of (**P1**). Here, we aim to transform problem (**P1**) to an equivalent queue stability problem, where the queue length indicates the satisfaction of constraint **C4**. Our objective is to jointly stabilize the queue length and minimize the long-term computation time defined in problem (**P1**).

5.1 Optimization and Task Assignment

To this end, we define $Q^t \geq 0$ as the length of the virtual queue *at the beginning* of optimization epoch t. According to the queue dynamics, which is represented as:

$$Q^t = [Q^{t-1} - (\hat{a}^{t-1} - A) \times |\mathcal{I}^{t-1}|]^+ \tag{3}$$

However, due to proposed pipeline parallelization, as shown in Fig. 6, $\hat{a}^{t-1}$ (i.e., actual quality after epoch $t-1$) cannot be observed *at the beginning* of epoch t, since tasks $\mathcal{I}^{t-1}_{DD}$ are just being scheduled to be executed. Additionally, $\mathcal{I}^t$ and $\mathcal{I}^t_{DD}$ can only be observed by the end of DD period in epoch t. To perform such *in-advance optimization*, we use Eq. (1) to compute the expected value of $\hat{a}^{t-1}$ and use $X^t = \max(C^{t-1}) \times b$ to estimate the size of the window $\mathcal{I}^t$. The value of $\max(C^{t-1})$ is produced by the previous optimization.

Thereafter, we integrate Eq. (3) with the quadratic Lyapunov function and the Lyapunov drift function [6, 9] that transforms the single-objective and long-term optimization problem (**P1**) into a sequence of identical, one-shot, and multi-objective optimization problems. Essentially, $\forall\ t \in \{1, 2, ..., T\}$ we aim to solve:

$$\begin{aligned} \min_{\mathbf{x}^t, r^t, d^t}\ & V \times \max(C^t) - Q^t \times (\mathcal{A}(d^t;\mathbf{j}^t) \times \mathcal{F}(r^t) - A) \times X^t \\ \text{s.t. } & \mathbf{C1,C2,C3} \end{aligned} \tag{P2}$$

where the trade-off between latency reduction and queue stability is jointly addressed by a balancing factor V [18] and Q^t. As (**P2**) is a min-max problem, we introduce an auxiliary variable $Z^t \geq \max(C^t)$ which represents the upper bound of computation times in C^t. We also remove constants in objective function that reformulates (**P2**) as:

$$\begin{aligned} \min_{\mathbf{x}^t, r^t, d^t}\ & V \times Z^t - Q^t \times \hat{a}^t \times \max(C^{t-1}) \times b \\ \text{s.t. } & Z^t \geq C_k(x^t_k, r^t),\ \forall k \in \{1, 2, ..., K\} \\ & \mathbf{C1,C2,C3} \end{aligned} \tag{P3}$$

By the end of DD period, the solution $\mathbf{x}^t$ obtained by solving (**P3**) at beginning of epoch t may become invalid if $\mathcal{S}(d^t;\mathbf{w}^t) \times \max(C^{t-1}) \times b \neq |\mathcal{I}^t_{DD}|$. In that case, the system needs to perform task assignment again based on the actual value of $\mathcal{I}^t_{DD}$. Such delayed-task assignment is defined as follows:

$$\begin{aligned} \min_{\mathbf{x}^t}\ & V \times Z^t \\ \text{s.t. } & \mathbf{C1:}\ \sum_{k=1}^{K} x^t_k = |\mathcal{I}^t_{DD}| \\ & \mathbf{C2:}\ Z^t \geq C_k(x^t_k, r^t),\ \forall k \in \{1, 2, ..., K\} \end{aligned} \tag{P4}$$

Both (**P3**) and (**P4**) are solved by GEKKO [3].

5.2 Weights Update

In order to solve (**P3**) and (**P4**), one prerequisite is to perform the modeling of $\mathcal{S}(d^t;\mathbf{w}^t)$ and $\mathcal{A}(d^t;\mathbf{j}^t)$, similar to Sections 4.2 and 4.3. However, it is time-consuming to obtain sufficient measurement results (e.g., Fig. 7) at the beginning of each optimization epoch to update $\mathbf{w}^t$ and $\mathbf{j}^t$ according to upcoming tasks. To address this issue, we divide $[d^{MIN}, d^{MAX}]$ into several partitions and assign the partitions equally among K edge servers for obtaining an initial profiling, i.e., to obtain initial weights $\mathbf{w}^1$ and $\mathbf{j}^1$ using non-linear regression models. Once the initial weights are obtained, we gradually update $\mathbf{w}^t$ and $\mathbf{j}^t$ using observed results during run-time. While this strategy produces large prediction errors in the first few epochs, eventually this error becomes smaller when the system has enough observations. On the other hand, the errors will be captured and mitigated by the quality queuing mechanism and our proposed Lyapunov method. Therefore, instead of spending a lot of time on measurements, this approach can gain huge latency benefits by dynamically updating the model weights.

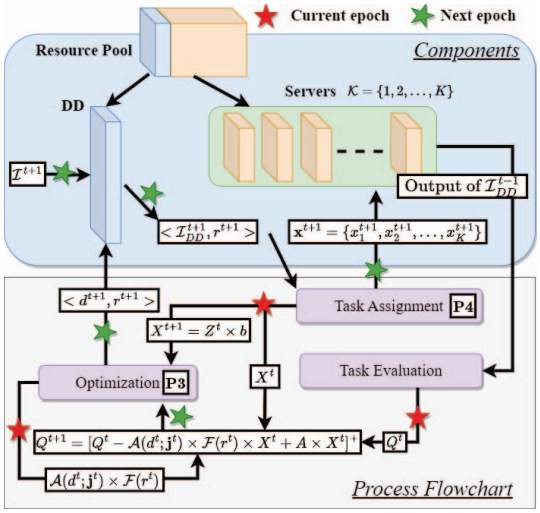


Figure 11. The system components and process flowchart

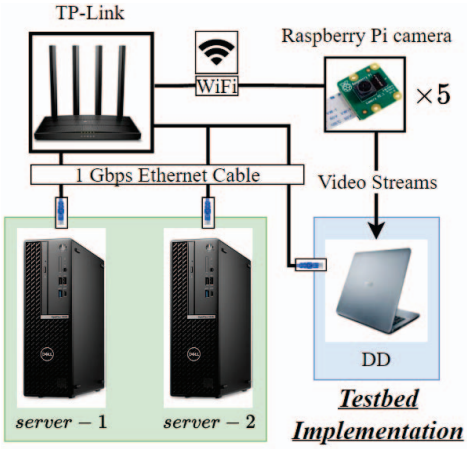


Figure 12. Hardware testbed implementation

5.3 Regulation on Window Size

Corollary 1. *If the task computation speed is greater than task selection speed, the window $\mathcal{I}^t$ decreases, and vice versa. When the two are equal, the window size if fixed.*

To improve resource-efficiency, it is important to find an appropriate control policy on $\mathcal{I}^t$ size that maximizes the server usage. In this work, we compare two different control strategies for task window size, viz., arbitrary window size and fixed window size.

5.3.1 Arbitrary window size. Under this policy, there are no regulations on the size of window $\mathcal{I}^t$. The execution of DD and selected tasks follow $\beta^t = \alpha^{t-1}$. The advantage of this strategy is that the edge servers and DD are always in use. According to **Corollary** 1, the window size changes according to the relationship between task compute speed in epoch $t-1$ and task selection speed in epoch t. The problem with this strategy is that the window is likely to increase or decrease monotonically. Therefore, it is either difficult to converge (if it keeps increasing) or computationally inefficient (if it keeps decreasing). The following optimizations are made to prevent the window from becoming too large or too small:

- To prevent the window from becoming too large, we limit the maximum number of tasks that can be added to $\mathcal{I}_{DD}$ as $|\mathcal{I}_{DD}| \leq \sum_{k=1}^{K} x_k^{MAX}$, where x_k^{MAX} is the maximum number of tasks that can be run in parallel on server k.
- To prevent the window from becoming too small, we set the minimum value for β^t as β^{MIN}, i.e., the DD will spend at least β^{MIN} amount of time on task selection.

The total task computation time with arbitrary window size can thus be stated as

$$C = \sum_{t=1}^{T} max(\alpha^t, \beta^{MIN}) \tag{4}$$

5.3.2 Fixed window size. Under this policy, the window size of $\mathcal{I}^t$ is limited to its maximum size, denoted by $|\mathcal{I}^t| \leq |\mathcal{I}|^m$. This strategy prevents the window from becoming too large or too small by cutting off the interaction between β^t and α^{t-1}. Since the length of the task period α^{t-1} varies between successive epochs, there will inevitably be idle times ($|\beta^t - \alpha^{t-1}| > 0$) between DD and task computation periods. Therefore, the system needs to find such an optimal $|\mathcal{I}|^m$ that guarantees resource-efficiency while: i) minimizing computation time and ii) satisfying quality requirement. *The optimal size of $\mathcal{I}^t$ is the one that minimizes the speed mismatch between task selection and task computation.* The constraint $|\mathcal{I}_{DD}| \leq \sum_{k=1}^{K} x_k^{MAX}$ is also applied to fixed window size policy. Compared to arbitrary window size policy (Eq. (4)), the total computation of fixed window size is computed by:

$$C = \sum_{t=1}^{T} \max(\alpha^t, \beta^t)$$

5.4 Overall System Description

The overall description of major components of the system optimization and process flowchart is shown in Fig. 11. By the end of task execution period in optimization epoch $t-1$ (which is also the end of DD period in epoch t, therefore $\mathcal{I}_{DD}^t$ can be obtained), the tasks in $\mathcal{I}^{t-1}$ are evaluated by computing the F-score (i.e., $\hat{a^{t-1}}$), while the servers process $\mathcal{I}_{DD}^t$. The quality Q^t can be updated according to the following expression:

$$Q^t = \left[Q^{t-1} - |\mathcal{I}^{t-1}| \times (\hat{a^{t-1}} - A) \right]^+$$

where $\hat{a^{t-1}}$ is the observed quality. Meanwhile, task assignment (**P4**) gives a completion time for finishing $\mathcal{I}_{DD}^t$, denoted by Z^t. Using the tuple $\{Z^t, Q^t, r^t, r^t\}$, the tuple $\{Q^{t+1}, X^{t+1}\}$ is approximated. Then, the optimization (**P3**) is performed that generates updated configurations $\{d^{t+1}, r^{t+1}\}$. While servers are processing the tasks in $\mathcal{I}_{DD}^t$, the DD period of epoch $t+1$ starts and a new optimization iteration begins.

6 Performance Evaluation

In this section, we evaluate the performance of the proposed data-driven adaptations and holistic system optimizations.

6.1 Testbed Implementation and Evaluation Methodology

Our testbed implementation mimicking a typical edge-native 3D reconstruction application is illustrated in Fig. 12. The five Raspberry

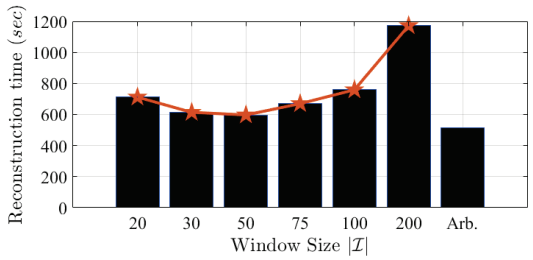


Figure 13. Total reconstruction time for different window size strategies

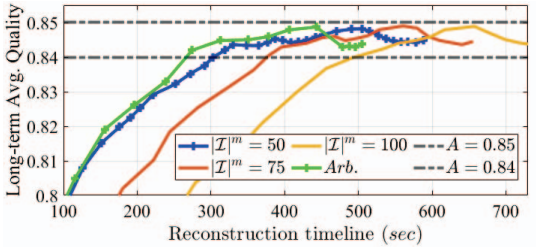


Figure 14. The convergence of fixed and arbitrary window size strategies

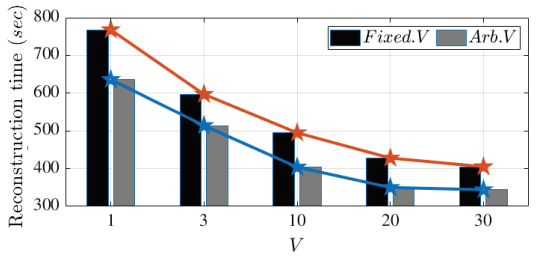


Figure 15. Total reconstruction times for fixed and arbitrary window strategies against different V

Pi cameras that are used for data collection (as shown in Fig. 3) are connected through WiFi to the TP-link router for video stream upload to the servers. The edge servers (i.e., *server-1* and *server-2*) and the laptop running DD are linked using high-speed 1 Gbps Ethernet cable. As mentioned earlier, the networking delay between components is ignored. The configurations for the machines are explained earlier in Table 2. The parallelized pipeline and system implementation within the hardware follows the process flowchart explained in Fig. 11.

The overall evaluation seeks to answer three overarching questions: 1) Can the long-term average quality be converged to the threshold A? 2) What is the speed of convergence? and 3) What is the total reconstruction time for a given sequence of task? For this, we use the distance between the obtained long-term average quality $\hat{A}^T$ (by the end of T optimization epochs) and the quality requirement A as the metric to evaluate how strictly the constraint **C4** of problem (**P1**) is satisfied. If we can get $|\hat{A}^T - A| \leq 0.01$, we say **C4** is *"strictly satisfied"*. In the experiment, the Raspberry Pi cameras continuously record 2000 timestamps (totally 2000×5 images) @ 25 frames/sec., generating video stream of 80 sec. and total $\sum_{t}^{T} |\mathcal{I}^t|$=2000 tasks. Apart from the cameras capturing data live, we use publicly available dataset Dance1 and Odzemok datasets [16] that are pre-loaded in the cameras. Unless stated otherwise, the following results present averages over all datasets and runs.

6.2 Impact of Window Size

Here, we evaluate the impact of window size $|\mathcal{I}^t|$ by setting A=0.85 and V=3, and using *Walk* dataset as an example of moderate dynamism. Fig. 13 shows that a window that is too small or too large increases the idle time ($|\beta^t - \alpha^{t-1}| > 0$) between DD and servers. The relationship can be treated as a convex function, where the optimal window size is obtained near the stationary point ($|\mathcal{I}|^m = 50$ and $t = 597$ seconds). Based on this evidence, we argue that the task computation speed is close to the task selection speed when the window size is 50. However, as the two speeds are never exactly the same, it is impossible to completely eliminate all idle times between DD and server computation. By contrast, based on server capacity, the arbitrary window strategy uses maximum and minimum limitations (explained in Section 5.3) to keep DD and servers running seamlessly. Compared to fixed window size of 50, the arbitrary strategy takes 514 seconds for computation, resulting in 14% latency reduction.

Next, we demonstrate the impact of window size on solution convergence speed in Fig. 14. Since the maximum number of optimization epochs T decreases as the task window size ($|\mathcal{I}^t|$) increases, larger windows lead to longer convergence time. This is a consequence of larger windows reducing the frequency of optimization and giving less opportunities to the system to tune its parameters, which is necessary to minimize the prediction error in $\mathcal{S}(d^t; \mathbf{w}^t)$ and $\mathcal{A}(d^t; \mathbf{J}^t) \times \mathcal{F}(d^t)$. Separately, the long term average quality of reconstruction fluctuates by a certain magnitude after convergence. This is predictable, as each time the convergence point is approached, the queue length Q^t is cleared or becomes very small. As explained in Section 5, problem (**P2**) then focuses on minimizing the computation time $max(C^t)$ by choosing a low image resolution and a large difference threshold, resulting in lower quality. To ensure that the fluctuation in quality is within an acceptable range (e.g., 0.01), the task window should not be too large.

6.3 Impact of Balance Factor V

Next, we evaluate the impact of factor V on minimizing the total reconstruction time and satisfaction of constraint **C4** in (**P1**). Here, we still consider $A = 0.85$ and *Walk* dataset. We denote fixed window and arbitrary window strategies as $Fixed.V$=$\mathcal{L}$ and $Arb.V$=$\mathcal{L}$ respectively, where V=$\mathcal{L}$ denotes that the value of V is set to $\mathcal{L}$ for that particular strategy. In $Fixed.V$ strategy, we set $|\mathcal{I}|^m = 50$. The time comparison between $Fixed.V$ and $Arb.V$ against different V are shown in Fig. 15. For both strategies, the results prove that there is a $[O(V), O(\frac{1}{V})]$ trade-off [13] between the objective function of (**P1**) and queue stability. The convergence results against different V's are shown in Fig. 16. Here, the long-term quality is hardly to converge to $A = 0.85$ with $V \geq 10$. As $V = 3$, the quality requirement is strictly satisfied after 300 seconds for both strategies. Overall, the proposed arbitrary window strategy reduces the total reconstruction time by 20% compared to fixed window strategy and achieves similar performance on quality satisfaction.

In Fig. 17, we explain the difference between different V in terms of quality. In particular, we show the comparison of how strictly the quality requirement is satisfied for different strategies. Overall, the distance $|\hat{A} - A|$ increases with increasing V. Strategies with $V = 1$ and $V = 3$ can strictly meet the quality requirement ($|\hat{A} - A| \leq 0.01$). However, $V = 1$ takes 20% extra time to finish all the tasks. Strategies with larger V fail to strictly satisfy the quality constraint **C4** of (**P1**). Therefore, we argue that $V = 3$ is a good strategy for

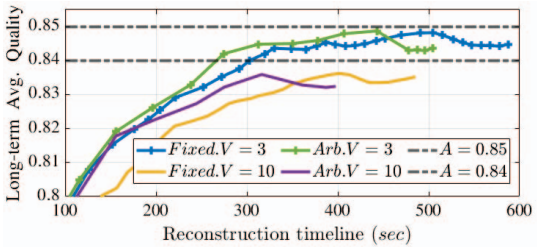


Figure 16. The convergence of fixed and arbitrary window strategies

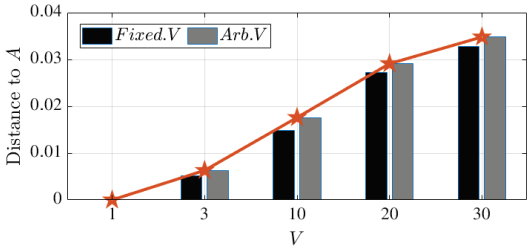


Figure 17. Quality requirement satisfaction for fixed and arbitrary window strategies against different V

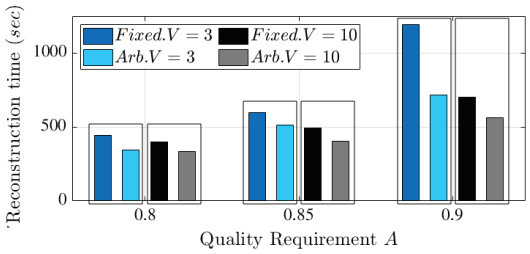


Figure 18. Reconstruction time for fixed and arbitrary window strategies

balancing the total computation time and quality satisfaction under these conditions.

6.4 Impact of Quality Constraint A

Here, we validate the proposed holistic optimization in addressing various quality constraints imposed by different use cases. We compare three scenarios: moderate quality requirement with $A = 0.8$, high quality requirement with $A = 0.85$, and extremely high quality requirement with $A = 0.9$. To address different preferences on reconstruction time and quality, two different values of V are applied, specifically $V = 3$ and $V = 10$. The quality constraint results are shown in Fig. 18 and Fig. 19 where the bars in the same box represent the reconstruction time or distance to A for strategies $Fixed.V$ ($|\mathcal{I}|^m = 50$) and $Arb.V$ under the same preference, e.g., $Fixed.V = 3$ and $Arb.V = 3$.

Fig. 18 shows that the average reconstruction time of all cases increased by 31.8% and 55.1% respectively when the quality requirement increased to $A = 0.85$ and $A = 0.9$ from $A = 0.8$. In Fig. 19, we observe that when long-term quality requirement is moderate (i.e., $A = 0.8$), all task window strategies can strictly meet the long-term quality requirement as the distances in these two boxes are $\leq 5 \times 10^{-3}$. However, at $A = 0.85$, only $V = 3$ can strictly meet the requirement (here, the two distances are 5.2×10^{-3} and 6.3×10^{-3} respectively, which are still acceptable). For extreme quality requirement like $A = 0.9$, even $V = 3$ can hardly meet (strictly) the long-term quality requirement. The average distance to A is 27×10^{-3} and 47×10^{-3} respectively for V=3 and V=10. Therefore, a lower V is desirable, especially for stricter quality scenarios.

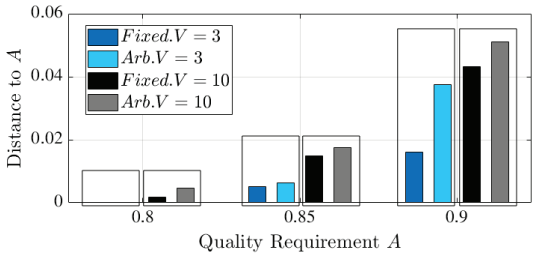


Figure 19. Quality requirement satisfaction for fixed and arbitrary window strategies against different A and V

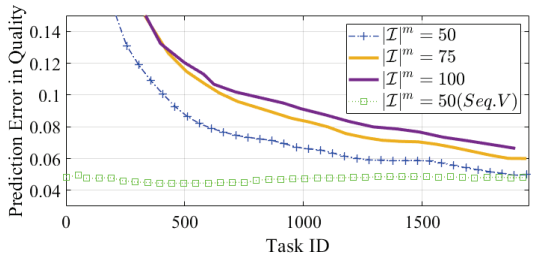


Figure 20. The comparison of prediction error in quality

6.5 Measurement Cost and Quality Prediction Error

In order to evaluate the measurement cost and prediction error in our experiment-driven analytic models, we compare our dynamic model update strategy with a greedy strategy $Seq.V$=$\mathcal{L}$, which performs measurement at the beginning of each optimization epoch to generate actual functions $\mathcal{A}(d^t; \mathbf{j^t})$ and $\mathcal{S}(d^t; \mathbf{w^t})$. The detailed comparisons of all the results are summarized in Table 3. The optimal window regulation strategy and preference V are selected based on similar analyses in previous subsection. In Table 3, the achieved time reductions are 31.6% (for A=0.8), 30.4% (for A=0.85), and 13.4% (for A=0.9), when compared to $Seq.V$.

In Fig. 20, we show how quality prediction error $|a^t - \hat{a}^t|$ is reduced with the weight update method proposed in subsection 5.2. We observe that although at beginning the system has relatively high prediction error, the error gradually decreases when we have more observations. Moreover, the prediction error reduces as window size decreases. This is because of the fact that a smaller window gives the system more observations to reshape the existing functions. For $Seq.V$, the prediction error is consistently hovers around 0.04, which provides us with an inherent error created by the definition of the function, i.e., $a^t \approx \mathcal{A}(d^t; \mathbf{j^t}) \times \mathcal{F}(r^t)$. Fig. 20 demonstrates that our final prediction error eventually approaches 0.04. Therefore, we argue that, our proposed dynamic model update solution is faster and guarantees minimum quality. Furthermore, the effect of quality prediction error $|a^t - \hat{a}^t|$ is mapped to the quality queue fluctuation $|Q^t - \hat{Q}^t|$, which is eventually addressed by the trade-off between computation time and queue stability as we defined in problem (**P2**). Therefore, the system convergence is always guaranteed.

*Combining all the above results, we conclude that, by adjusting the value of V, our system is able to provide a flexible balance between reconstruction latency and minimum quality satisfaction. More specifically, under the premise of satisfying the average quality constraint **C4**, the proposed solution is able to control the convergence rate by choosing an appropriate V based on the preference of total reconstruction time and quality.*

Table 3. Summary where δ is the profiling time for modeling $\mathcal{S}(d^t; \mathbf{w}^t)$ and $\mathcal{A}(d^t; \mathbf{j}^t)$; For parallel pipelines ($Fixed.V$ and $Arb.V$), the total profiling time is δ; For sequential pipeline ($Seq.V$), the total profiling time is $20 \times \delta$. Here, $\delta \approx 10$ seconds.

	Image Resizing Ratio		# Executed Tasks		Reconstruction Time		Quality satisfaction ($\lvert\hat{A} - A\rvert \leq 0.01$)	
Quality	Strategy	Value	Strategy	Value	Strategy	Value	Strategy	$\lvert\hat{A} - A\rvert$
	$Fixed.V = 3$	0.63	$Fixed.V = 3$	188	$Fixed.V = 3$	442 ($+1 \times \delta$)	$Fixed.V = 3$	0
$A = 0.8$	$Arb.V = 3$	**0.69**	$Arb.V = 3$	**131**	$Arb.V = 3$	**345** ($+1 \times \delta$)	$Arb.V = 3$	**0.0018**
(moderate	$Seq.V = 3$	0.66	$Seq.V = 3$	150	$Seq.V = 3$	305 ($+20 \times \delta$)	$Seq.V = 3$	0
quality	$Fixed.V = 10$	0.66	$Fixed.V = 10$	135	$Fixed.V = 10$	401 ($+1 \times \delta$)	$Fixed.V = 10$	0
req.)	$Arb.V = 10$	0.59	$Arb.V = 10$	177	$Arb.V = 10$	337 ($+1 \times \delta$)	$Arb.V = 10$	0.0046
	$Seq.V = 10$	0.65	$Seq.V = 10$	134	$Seq.V = 10$	266 ($+20 \times \delta$)	$Seq.V = 10$	0.0029
	$Fixed.V = 3$	0.80	$Fixed.V = 3$	257	$Fixed.V = 3$	597 ($+1 \times \delta$)	$Fixed.V = 3$	0.0052
$A = 0.85$	$Arb.V = 3$	**0.81**	$Arb.V = 3$	**255**	$Arb.V = 3$	**514** ($+1 \times \delta$)	$Arb.V = 3$	**0.0063**
(high	$Seq.V = 3$	0.78	$Seq.V = 3$	272	$Seq.V = 3$	553 ($+20 \times \delta$)	$Seq.V = 3$	0.0024
quality	$Fixed.V = 10$	0.78	$Fixed.V = 10$	209	$Fixed.V = 10$	495 ($+1 \times \delta$)	$Fixed.V = 10$	0.0148
req.)	$Arb.V = 10$	0.78	$Arb.V = 10$	148	$Arb.V = 10$	($+1 \times \delta$)	$Arb.V = 10$	0.0176
	$Seq.V = 10$	0.76	$Seq.V = 10$	207	$Seq.V = 10$	441 ($+20 \times \delta$)	$Seq.V = 10$	0.0135
	$Fixed.V = 1$	0.82	$Fixed.V = 1$	958	$Fixed.V = 1$	1805 ($+1 \times \delta$)	$Fixed.V = 1$	0.0050
	$Arb.V = 1$	**0.92**	$Arb.V = 1$	**650**	$Arb.V = 1$	**1481** ($+1 \times \delta$)	$Arb.V = 1$	**0.0091**
	$Seq.V = 1$	0.84	$Seq.V = 1$	849	$Seq.V = 1$	1677 ($+20 \times \delta$)	$Seq.V = 1$	0.0080
$A = 0.9$	$Fixed.V = 3$	0.86	$Fixed.V = 3$	559	$Fixed.V = 3$	1192 ($+1 \times \delta$)	$Fixed.V = 3$	0.0161
(extremely	$Arb.V = 3$	0.89	$Arb.V = 3$	321	$Arb.V = 3$	718 ($+1 \times \delta$)	$Arb.V = 3$	0.0375
high	$Seq.V = 3$	0.84	$Seq.V = 3$	606	$Seq.V = 3$	1188 ($+20 \times \delta$)	$Seq.V = 3$	0.0143
quality	$Fixed.V = 10$	0.85	$Fixed.V = 10$	306	$Fixed.V = 10$	702 ($+1 \times \delta$)	$Fixed.V = 10$	0.0432
req.)	$Arb.V = 10$	0.85	$Arb.V = 10$	252	$Arb.V = 10$	562 ($+1 \times \delta$)	$Arb.V = 10$	0.0511
	$Seq.V = 10$	0.81	$Seq.V = 10$	346	$Seq.V = 10$	707 ($+20 \times \delta$)	$Seq.V = 10$	0.0379

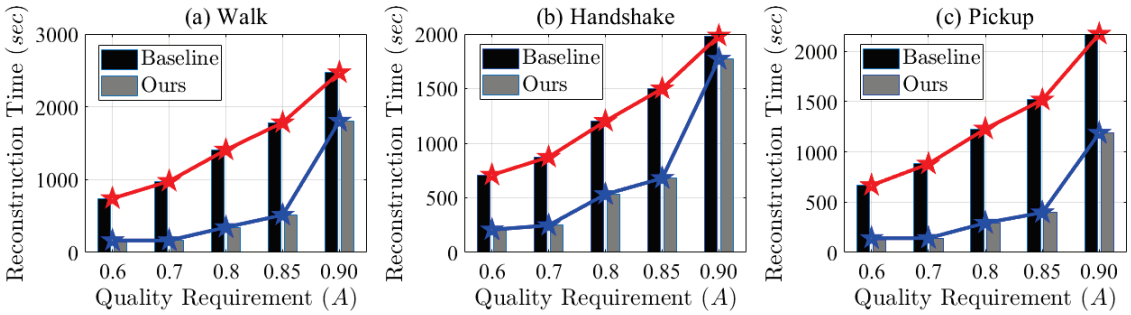


Figure 21. Comparison between the proposed window-based optimization strategy and the baseline strategies

6.6 Comparison Against Baseline Strategy

Finally, we compare our window-based strategy against a baseline strategy, where all tasks are performed at a specific resolution that just satisfies the quality requirement. This specific resolution is selected according to function $\mathcal{F}(r^t)$ that we obtained from Fig. 8. The results are summarized in Fig. 21 with quality requirements A ranging from 0.6 (very low) to 0.9 (extremely high). Overall, we observe that the total reconstruction time increases gradually with quality requirement. However, the time increases suddenly and significantly for high and extremely high quality requirements (i.e., for $A \geq 0.85$). This can be explained by the features described in Fig. 8 where the growth rate of reconstruction quality converges to 0 at such high values. Consequently, the system has to use much higher image resolutions for reconstruction; thus explaining longer reconstruction times. *Overall, compared to the baseline strategy, our proposed strategy which is based on the dynamics of video content and real-time quality changes eliminates unnecessary reconstruction, can greatly reduce the total reconstruction time (50% to 75%) without violating the quality requirements, irrespective of the 3D scene.*

Acknowledgments

This work supported by the National Science Foundation under Award Number:CNS-1943338.

7 Conclusions

In this paper, we proposed latency minimizing edge system adaptation and optimization solutions that can guarantee long-term 3D reconstruction quality satisfaction by eliminating redundant reconstructions. By exploiting data and task-level parallelism, our parallelized 3D reconstruction pipeline significantly reduced the total reconstruction time of the entire video stream. Moreover, a novel in-advance optimization and weight update mechanism that is driven by real-world measurements showed great advantages over traditional reconstruction pipelines and baseline strategies when running on an edge system. Our proposed flexible balancing mechanism can strike optimal trade-off between reconstruction time and quality by capturing the inherent dynamism of a 3D scene. The ideas, models, datasets, and results presented in this paper could be critical towards a broader paradigm shift that would dictate how edge resources are managed for complex video processing applications, adopted to support mission-critical use cases.

References

[1] Ali Hosseininaveh Ahmadabadian, Ali Karami, and Rouhallah Yazdan. 2019. An automatic 3D reconstruction system for texture-less objects. *Robotics and Autonomous Systems* 117 (2019), 29–39.

[2] Diego Bastias, Claudio A. Perez, Daniel P. Benalcazar, and Kevin W. Bowyer. 2017. A method for 3D iris reconstruction from multiple 2D near-infrared images. In *2017 IEEE International Joint Conference on Biometrics (IJCB)*. 503–509. https://doi.org/10.1109/BTAS.2017.8272735

[3] Logan DR Beal, Daniel C Hill, R Abraham Martin, and John D Hedengren. 2018. Gekko optimization suite. *Processes* 6, 8 (2018), 106.

[4] Leslie Bolick and Josh Harguess. 2016. A study of the effects of degraded imagery on tactical 3D model generation using structure-from-motion. In *Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications XIII*, Vol. 9828. International Society for Optics and Photonics, 98280F.

[5] Dan Cernea. 2020. OpenMVS: Multi-View Stereo Reconstruction Library. (2020). https://cdcseacave.github.io/openMVS

[6] Xiangyi Chen, Yuanguo Bi, Xueping Chen, Hai Zhao, Nan Cheng, Fuliang Li, and Wenlin Cheng. 2022. Dynamic Service Migration and Request Routing for Microservice in Multicell Mobile-Edge Computing. *IEEE Internet of Things Journal* 9, 15 (2022), 13126–13143.

[7] Yanping Fu, Qingan Yan, Long Yang, Jie Liao, and Chunxia Xiao. 2018. Texture mapping for 3d reconstruction with rgb-d sensor. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4645–4653.

[8] Xiao Li, Wen Yi, Hung-Lin Chi, Xiangyu Wang, and Albert PC Chan. 2018. A critical review of virtual and augmented reality (VR/AR) applications in construction safety. *Automation in Construction* 86 (2018), 150–162.

[9] Jie Lin, Lin Huang, Hanlin Zhang, Xinyu Yang, and Peng Zhao. 2022. A novel Lyapunov based dynamic resource allocation for UAVs-assisted edge computing. *Computer Networks* 205 (2022), 108710.

[10] Qiang Liu, Siqi Huang, Johnson Opadere, and Tao Han. 2018. An edge network orchestrator for mobile augmented reality. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 756–764.

[11] Lu Lou, Yonghuai Liu, Minglan Sheng, Jiwan Han, and John H Doonan. 2014. A cost-effective automatic 3D reconstruction pipeline for plants using multi-view images. In *Conference Towards Autonomous Robotic Systems*. Springer, 221–230.

[12] Qiang Lv, Huican Lin, Guosheng Wang, Heng Wei, and Yang Wang. 2017. ORB-SLAM-based tracing and 3D reconstruction for robot using Kinect 2.0. In *2017 29th Chinese Control And Decision Conference (CCDC)*. IEEE, 3319–3324.

[13] Yuyi Mao, Jun Zhang, SH Song, and Khaled B Letaief. 2017. Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. *IEEE Transactions on Wireless Communications* 16, 9 (2017), 5994–6009.

[14] Luca Mazzei, Lorenzo Corgnati, Simone Marini, Ennio Ottaviani, and Bruno Isoppo. 2015. Low cost stereo system for imaging and 3D reconstruction of underwater organisms. In *OCEANS 2015-Genova*. IEEE, 1–4.

[15] Pierre Moulon, Pascal Monasse, Romuald Perrot, and Renaud Marlet. 2016. Openmvg: Open multiple view geometry. In *International Workshop on Reproducible Research in Pattern Recognition*. Springer, 60–74.

[16] A. Mustafa, H. Kim, J.Y. Guillemaut, and A. Hilton. 2015. General dynamic scene reconstruction from wide-baseline views. In *ICCV*.

[17] Zihao Pan, Junyi Hou, and Lei Yu. 2023. Optimization RGB-D 3-D Reconstruction Algorithm Based on Dynamic SLAM. *IEEE Transactions on Instrumentation and Measurement* 72 (2023), 1–13.

[18] Xiaohan Qi, Jingzheng Chong, Qinyu Zhang, and Zhihua Yang. 2022. Collaborative Computation Offloading in the Multi-UAV Fleeted Mobile Edge Computing Network via Connected Dominating Set. *IEEE Transactions on Vehicular Technology* 71, 10 (2022), 10832–10848.

[19] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. 2018. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 1421–1429.

[20] Susana Ruano and Aljosa Smolic. [n.d.]. A Benchmark for 3D Reconstruction from Aerial Imagery in an Urban Environment. ([n. d.]).

[21] Yuvraj Sahni, Jiannong Cao, Lei Yang, and Yusheng Ji. 2020. Multihop Offloading of Multiple DAG Tasks in Collaborative Edge Computing. *IEEE Internet of Things Journal* 8, 6 (2020), 4893–4905.

[22] Taha Samavati and Mohsen Soryani. 2023. Deep learning-based 3D reconstruction: A survey. *Artificial Intelligence Review* (2023), 1–45.

[23] Johannes L Schonberger and Jan-Michael Frahm. 2016. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4104–4113.

[24] Elisavet Konstantina Stathopoulou and Fabio Remondino. 2019. Open-source image-based 3D reconstruction pipelines: Review, comparison and evaluation. In *6th International Workshop LowCost 3D–Sensors, Algorithms, Applications*. 331–338.

[25] Benjamin Ummenhofer and Thomas Brox. 2013. Point-based 3d reconstruction of thin objects. In *Proceedings of the IEEE International Conference on Computer Vision*. 969–976.

[26] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods* 17, 3 (2020), 261–272.

[27] Michael Waechter, Nils Moehrle, and Michael Goesele. 2014. Let there be color! Large-scale texturing of 3D reconstructions. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 836–850.

[28] Can Wang, Sheng Zhang, Yu Chen, Zhuzhong Qian, Jie Wu, and Mingjun Xiao. 2020. Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 257–266.

[29] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. 2018. Bandwidth-efficient live video analytics for drones via edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 159–173.

[30] Jiang-An Wang, Huang-Te Ma, Chun-Mei Wang, and Yong-Jie He. 2018. Fast 3D reconstruction method based on UAV photography. *ETRI Journal* 40, 6 (2018), 788–793.

[31] Lichun Wang, Ran Chen, and Dehui Kong. 2014. An improved patch based multi-view stereo (PMVS) algorithm. In *3rd International Conference on Computer Science and Service System*. Atlantis Press, 9–12.

[32] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. 2017. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. 1–13.

[33] Wuyang Zhang, Sugang Li, Luyang Liu, Zhenhua Jia, Yanyong Zhang, and Dipankar Raychaudhuri. 2019. Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1270–1278.

[34] Xiaojie Zhang, Mingjun Li, Andrew Hilton, Amitangshu Pal, Soumyabrata Dey, and Saptarshi Debroy. 2022. End-to-End Latency Optimization of Multi-view 3D Reconstruction for Disaster Response. In *2022 10th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. 17–24. https://doi.org/10.1109/MobileCloud55333.2022.00010

[35] Xiaojie Zhang, Motahare Mounesan, and Saptarshi Debroy. 2023. EFFECT-DNN: Energy-efficient Edge Framework for Real-time DNN Inference. In *2023 IEEE 24th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 10–20. https://doi.org/10.1109/WoWMoM57956.2023.00015

[36] Xiaojie Zhang, Amitangshu Pal, and Saptarshi Debroy. 2021. EFFECT: Energy-efficient Fog Computing Framework for Real-time Video Processing. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 493–503. https://doi.org/10.1109/CCGrid51090.2021.00059

[37] Xiaojie Zhang, Amitangshu Pal, and Saptarshi Debroy. 2021. EFFECT: Energy-efficient Fog Computing Framework for Real-time Video Processing. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 493–503. https://doi.org/10.1109/CCGrid51090.2021.00059