

Uirapuru: Timely Video Analytics for High-Resolution Steerable Cameras on Edge Devices

Guilherme H. Apostolo
Vrije Universiteit Amsterdam
g.apostolo@vu.nl

Henri E. Bal
Vrije Universiteit Amsterdam
h.e.bal@vu.nl

Pablo Bauszat
Vrije Universiteit Amsterdam
pablo.bauszat@gmail.com

Vinod Nigade
Vrije Universiteit Amsterdam
vinod.nigade@gmail.com

© Guilherme H. Apostolo 2025. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in the 31st Annual International Conference on Mobile Computing and Networking (ACM MOBICOM '25), November 4–8, 2025, Hong Kong, China, <http://dx.doi.org/10.1145/3680207.3765260>.

ABSTRACT

Real-time video analytics on high-resolution cameras has become a popular technology for various intelligent services like traffic control and crowd monitoring. While extensive work has been done on improving analytics accuracy with timing guarantees, virtually all of them target static viewpoint cameras. In this paper, we present Uirapuru, a novel framework for real-time, edge-based video analytics on high-resolution *steerable* cameras. The actuation performed by those cameras brings significant dynamism to the scene, presenting a critical challenge to existing popular approaches such as frame tiling. To address this problem, Uirapuru incorporates a comprehensive understanding of camera actuation into the system design paired with fast adaptive tiling at a per-frame level. We evaluate Uirapuru on a high-resolution video dataset, augmented by pan-tilt-zoom (PTZ) movements typical for steerable cameras and on real-world videos collected from an actual PTZ camera. Our experimental results show that Uirapuru provides up to $1.45\times$ improvement in accuracy while respecting specified latency budgets or reaches up to $4.53\times$ inference speedup with on-par accuracy compared to state-of-the-art static camera approaches.

CCS CONCEPTS

- Computer systems organization → Embedded systems;
- Computing methodologies → Computer vision; Neural networks.

Lin Wang
Paderborn University
lin.wang@uni-paderborn.de

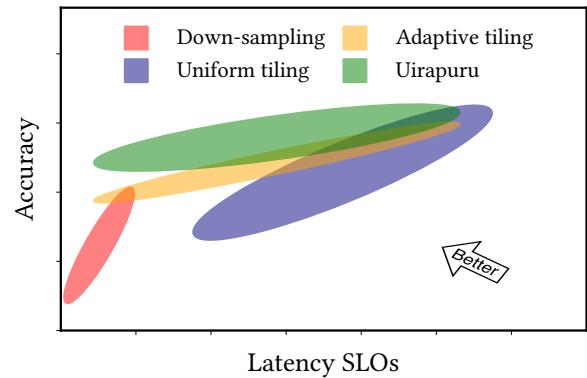


Figure 1: A comparison of Uirapuru to previous approaches in terms of accuracy and latency SLOs.

KEYWORDS

Video Analytics, Edge Computing, Object Detection, Steerable Cameras, Latency SLO

1 INTRODUCTION

Recent technological advances have introduced high-resolution cameras (e.g., 4K, 8K, and higher) that are rapidly spreading around public places like traffic intersections, airports, and nursing homes [20, 23, 43, 53]. Public and private organizations utilize these cameras for tasks like traffic control, crowd monitoring, and security and safety [13, 19, 52]. The high-resolution video streams generated by these cameras are typically processed by machine learning models such as deep neural networks, performing video analytics tasks including object detection, tracking, and activity recognition [3, 15, 44]. These applications typically impose strict real-time requirements on video analytics, a.k.a. service-level objectives (SLOs), calling for a camera-local solution given the high variability of the communication network for cloud-based alternatives [13, 21, 23, 32, 46]. Security and data privacy concerns further reinforce this need [6].

Meeting the stringent timing requirements with on-device processing poses a significant challenge due to the mismatch

between the required intensive computation and the limited available resources [47]. One straightforward approach for dealing with this issue is to use small and fast models and down-sample the video frames. However, down-sampling will most likely degrade the quality of the analytics due to the loss of details, thus defeating the purpose of high-resolution cameras. A more involved approach is *tiling*, where a high-resolution video frame is broken down into smaller tiles which are processed individually with smaller models [21, 32, 36, 43]. Adaptive tiling exploits the *spatial and temporal locality* in the object distribution of a scene and often provides improved accuracy over simple uniform tiling. By customizing the tile layout (e.g., using fine-grained tiles only for crowded regions) and applying scene-specific optimizations (e.g., skipping tiles that do not contain objects of interest), video analytics can be accelerated significantly.

Many of the cameras deployed today are *steerable* [8, 18, 39]. Steerable cameras can perform human-instructed actuation such as pan-tilt-zoom (PTZ) [25, 51]. The actuation performed by a steerable camera present significant challenges to tiling-based video analytics due to the high dynamism that is introduced. More specifically, the object distribution in the video stream of a steerable camera is constantly changing, making it hard to leverage spatial and temporal locality.

We propose **Uirapuru**¹, a novel framework for real-time video analytics on high-resolution steerable cameras. Uirapuru addresses the aforementioned challenges to provide high-performance video analytics for steerable cameras while guaranteeing latency SLOs. To account for camera actuation, Uirapuru first introduces the concepts of a “global” view perspective that encompasses the whole scene of observation and a “local” view perspective that reflects the current view of the camera. Uirapuru builds the global view with historical frames and later, at runtime, transforms the objects from the global view to the local view according to the camera’s actuation. This allows Uirapuru to estimate the object distribution of each frame more accurately in real-time. Based on these distributions, Uirapuru dynamically generates a tile plan per frame that assigns more accurate models to regions of interest, using a novel fast tiling algorithm based on dynamic programming. Uirapuru also features a new model profile that better encompasses the high variability of object sizes introduced by camera actuation. As shown in Fig. 1, this novel design allows Uirapuru to surpass previous state-of-the-art solutions in video analytics scenarios with high-resolution steerable cameras.

Overall, we make the following contributions:

- We present Uirapuru, a framework for edge-based, latency-critical video analytics for high-resolution steerable cameras that incorporates an understanding of camera actuation and addresses profiling biases in its design (§3).
- We propose a fast and optimal tiling algorithm, named **depth-first post-order dynamic programming** (DP-DP), that efficiently generates high-quality tile plans (§3.2.2).
- We demonstrate the effectiveness of our prototype by conducting extensive experiments with a high-resolution object detection dataset retrofitted with actuation by steerable cameras and on a real-world steerable camera deployment (§4). Our results show an improvement in accuracy up to 1.4x while meeting user-specified latency SLOs, and reach an up to 4.53× inference speedup with on-par accuracy compared to the baseline.

2 BACKGROUND AND RELATED WORK

Deep learning has become the de-facto standard for many video analytics tasks. To address the computational challenge of deep learning models, existing works have explored offloading model inference to a cloud platform [4, 12, 19, 24, 28, 48, 50, 54]. Yet, such offloading-based approaches face major challenges centered around SLO guarantee (due to network variability in the wide area) and data privacy. In this paper, we focus on video analytics solely on edge devices, specifically on high-resolution steerable cameras.

2.1 Edge-based Video Analytics

Performing analytics with deep learning on the original full-resolution video data (in 4K, 8K, or even higher) on edge devices is prohibitively expensive and impractical, considering the limited onboard resources [21, 43, 46, 47]. Popular approaches in the literature that address this challenge are *down-sampling* and *tiling*.

Given a model tailored for a target edge device, down-sampling reduces the frame resolution to match the model’s input size. This provides opportunities for latency guarantees, albeit at the cost of significant accuracy losses due to drastic reductions in object sizes in the input. To improve accuracy, one must increase the model size or use more sophisticated architectures [15, 35, 41, 42], provided that latency requirements of the application and memory constraints of the edge device are still met. However, the capacities of the edge device impose strict limitations that make it generally hard to scale up the models. For example, popular mobile system-on-chips such as Snapdragon 855 [34] and Kirin 970 [16] can run the EfficientDet family [42] only from D0 up to D5 [21]. At the same time, the Nvidia Jetson AGX [29] can run up to D7 but with a high latency of over 1.5 seconds per frame.

Alternatively, tiling has been utilized to perform high-resolution video analytics on the edge. Tiling breaks down

¹Uirapuru (/wi.ra'pu.ru/) or Musician Wren is a native bird of the Amazon rainforest. The bird has a complex and varied song, with locals claiming it can sing for minutes without repeating a single note in its melody (just like our system that can create a new plan for every frame).



Figure 2: The figure shows video frames from a static camera (top) and a steerable camera executing a sequence of actuation (bottom) on the left side. The right plot shows the average size of objects in each frame of the sequence.

the frame into smaller tiles, allowing the use of smaller models based on the tile size [36, 43]. Existing tiling solutions can be roughly categorized into *uniform* and *adaptive*. Uniform tiling generates equally-sized tiles whose resolution roughly matches the expected input size of a specific model. Despite achieving higher accuracy, uniform tiling suffers from high latency since the per-frame inference time is the sum of all tile inference times. Moreover, uniform tiling fails to provide a fine-grained trade-off between accuracy and latency. To meet latency SLOs between the execution time of two models of different sizes, we are forced to choose the smaller model with degraded accuracy since using the larger model would cause SLO violations.

Adaptive tiling approaches [7, 32, 48] allow tiles of varying sizes and shapes, as well as tile-specific model selection. For example, tiles with higher importance can be allocated more of the time budget and computing resources, and be processed with more sophisticated models to improve accuracy. However, such flexibility comes at the cost of higher complexity in the tile generation. Remix is a prominent example of adaptive tiling [21] for high-resolution static videos and guides its tile plan generation with historical information collected from video frames during a bootstrap phase. While achieving superior accuracy compared to down-sampling and uniform tiling-based approaches, Remix assumes strong temporal correlations in the video stream for its optimizations to work, which may not hold in dynamic scenarios.

2.2 Challenges with Steerable Cameras

Steerable cameras that support actuation (i.e., sequence of movements) like pan-tilt-zoom (PTZ) [8, 51] have been increasingly deployed. Such cameras enable essential use cases, such as close inspection of specific regions in the camera view by applying user-instructed actuation, but bring additional challenges to high-resolution video analytics due to the significantly increased dynamism.

Existing solutions based on adaptive tiling typically leverage spatial and temporal locality in video frames for tile generation. However, videos from steerable cameras may lack

such a locality. Fig. 2 depicts the frames from a static camera (top row) and a steerable camera executing a sequence of actuation (bottom row). On the right side of the figure, the plot shows how the average object size (relative to the frame resolution) changes in every frame. For the static camera case, the average object size and density of objects in the scene remain almost constant. Previous approaches, such as Remix, leverage such spatial and temporal locality and rely on a stable object distribution when generating tiles. In contrast, frames captured by steerable cameras have diverging view points due to the actuation applied to the camera and exhibit drastic changes in object sizes and locations as seen in Fig. 2. This renders existing approaches for tile generation based on historical object distributions impractical.

When comparing the object distribution of a frame to the historical objects collected from a global viewpoint during an early bootstrap phase, changes typically occur in two ways: (1) changing object density in particular regions of the scene and (2) changing object size. We can see examples of the first case in Fig. 2(B) with a pan movement from left to right and Fig. 2(D) with a tilt movement to the top. For the second case, we can observe changes in object sizes in Fig. 2(A, C) caused by a zoom movement and in Fig. 2(D) due to a tilt movement. In extreme cases, an object can become even bigger than the tile containing it, likely leading to the model's complete failure. Existing adaptive tiling approaches assume relatively static object distributions, so tiles that are anticipated to contain few objects are processed with less accurate models, and conversely, tiles that are assumed to cover densely populated areas are processed with highly accurate models. Mismatches in the anticipated number of objects and their sizes can lead to accuracy degradation (when sparsely populated areas become crowded with essential objects) and resource waste (when previously crowded regions become empty). This shows that tile planning cannot purely rely on historical object information in the case of steerable cameras.

Moreover, the rapid changes and high range of object sizes bring extra challenges to the performance profiling of models [17, 21, 27]. Previous approaches do already distinguish model performance by object size, but typically categorize

objects into uniform ranges using absolute pixel values. For example, Remix [21] uses a profile with 12 uniform accuracy bins, the smallest being between 0 – 64 pixels and the biggest between $196^2 - \infty$. When applying models to tiles that are only a subset of the image (where objects are now relatively enlarged), they propose using a correcting scaling factor to address the difference in resolution and view. Their results demonstrate that such a profile is effective enough for the static camera scenario where object sizes are roughly constant and bounded. However, in the steerable camera scenario, sizes vary significantly in terms of absolute pixel values and over a much larger range. For example, an object that was far away before and minuscule can be a quarter of the image’s area size after a zoom actuation. Additionally, there are cases where zooming can cause objects to become even larger than the model inputs themselves. In such cases, tiling can be detrimental, and down-sampling would be a better solution. Such variations have the effect that objects are quickly projected to belong to the largest bin, eliminating any ability for distinction when comparing models. *Therefore, it is integral to create model profiles that are well suited for rapid, highly variable size changes and understand how changes in the relative sizes of objects affect model accuracy.*

3 UIRAPURU SYSTEM DESIGN

The main goal of our **Uirapuru** system is to maximize object detection accuracy for steerable cameras within the available resources of an edge device given a per-frame latency SLO. To achieve this, Uirapuru must deal with rapidly changing object distributions (i.e., locations and sizes) arising from camera actuation. Our key idea is to keep track of these actuation over time in order to understand the camera’s view transformation in each frame. Initially, Uirapuru collects a historical object distribution from a set of frames, denoted as the historical frames, that views the scene from a global, all-encompassing perspective. In each frame, we then extract and transform the relevant historical objects utilizing the state of the camera to form a *local* object distribution (“local” here contrasts a frame’s view in relation to the “global” view perspective). The local distribution identifies regions of interest much more timely and accurately and is then utilized for adaptive tiling.

Previous approaches that perform adaptive tiling create tile plans during an offline phase, typically of very high quality [21]. However, these plans do not adapt well when cameras are steered. We argue that a timely plan is more beneficial than an optimal one as long as it still provides a sufficient level of quality. Accordingly, we propose an approximate but fast *online* adaptive tiling strategy that efficiently generates a fresh tile layout for each frame.

Our system’s overall design and components are illustrated in Fig. 3. Uirapuru operates in two phases: an offline bootstrap phase (§3.1) and a runtime phase (§3.2). The former is done as a pre-process to generate model profiles and extract objects from historical frames for a global overview. The latter processes the incoming camera frames through tiling and performs the actual model inference.

3.1 Bootstrap Phase

During the bootstrap phase, Uirapuru first creates a performance profile for each model in the provided family of models using a profiling dataset (§3.1.1). Each profile contains the model’s characteristics in terms of inference latency and detection accuracy for different object sizes. It is later used during the tile plan creation to decide on the most suitable models. Afterward, the objects in the historical frames are extracted (§3.1.2). These objects provide a global overview of where and in which size objects are to be expected. They will be used later, at runtime, to estimate the local object distribution in each frame. Finally, as generating a new tile plan in each frame consumes a part of the available latency budget, we also use the historical frames to estimate the algorithmic runtime overhead of our plan creation method (§3.1.3). This guarantees that during the runtime phase, the system understands the available *inference* budget that already accounts for the overhead from the plan creation algorithm and does not violate the per-frame SLO. The inputs to the bootstrap phase are: 1) a family of models for the object detection task, 2) a profiling dataset, 3) the historical frames taken from the global view perspective, and 4) the per-frame latency SLO.

3.1.1 Model Profiler. For each model M in the family, we create a model profile that stores the models’ mean and 99th percentile inference latencies, L_M^{mean} , and L_M^{99} , and an accuracy vector with b bins for different size ranges, A_M :

$$A_M = \langle \alpha_{S_0}, \alpha_{S_1}, \dots, \alpha_{S_{b-1}} \rangle$$

The inference latencies are measured by collecting the mean and 99th percentile of the inference time across multiple executions of the models on the edge device. We will only use one of them during runtime depending on the working mode of Uirapuru that defines the strictness of the SLO guarantee that the system should provide (see §3.2.2). The accuracy vector A_M represents the model’s estimated accuracy for various ranges of object sizes. Models tend to perform differently depending on the size of objects, e.g., small models are generally good at detecting relatively large objects while tiny objects are often missed [43]. Hence, it is beneficial to estimate the mean accuracy of a model over all objects, which allows for more fine-grained distinction based on object size.

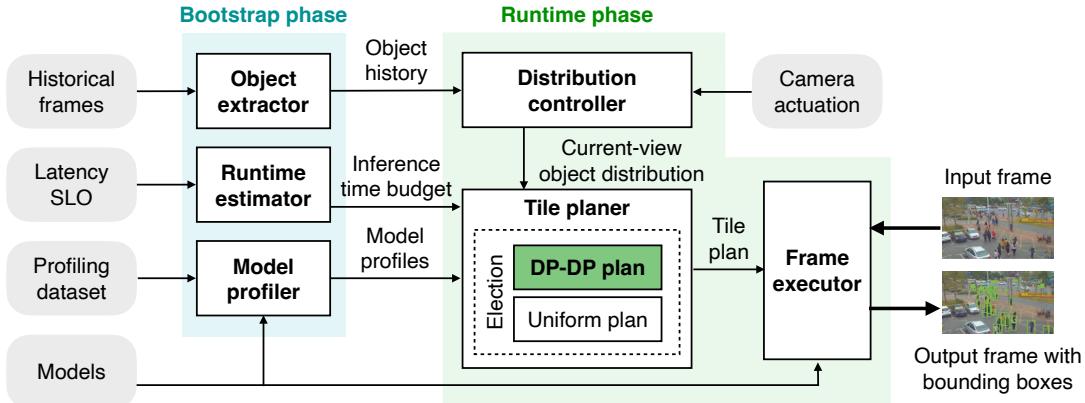


Figure 3: Uirapuru system overview.

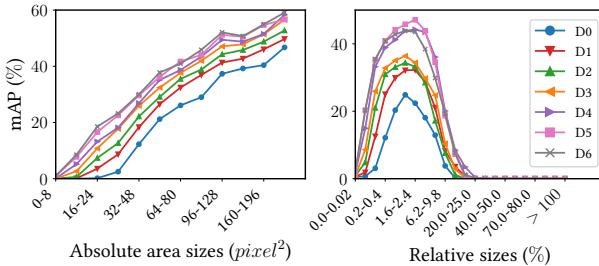


Figure 4: Accuracy profiles using Remix's absolute pixel sizes (left) vs. Uirapuru's relative sizes (right).

In the steerable camera scenario, sizes vary significantly in terms of absolute pixel values and are over a much larger range. Such variations have the effect that objects are quickly projected to belong to the largest bin, eliminating any ability for distinction when comparing models. To tackle those issues, we propose using a non-linear, relative size model profile to address both problems and better represent model performance for wide-ranging object sizes. Using relative sizes, our model profile is easily transferable between different views and tile sizes without the need for a correcting scaling factor. Further, using a non-linear profile allows fine-grained distinction for small sizes, which is required for down-sampling, while still covering a wide range of sizes, as is required for tiling. Our profile consists of 22 bins. The profile can use any non-linear function to define its bins. For ease of comparison with the other baseline in §4.3.1, the ranges for the first 12 bins are computed by taking the 12 absolute value ranges from Remix divided by the resolution of Efficientdet-D6, while bins 13 and 14 ranges from 20% – 25% and 25% – 30% respectively. The following eight bins are chosen from a linear range of 10% until > 100%. Fig. 4 shows Remix and Uirapuru profiles over the training data. Fig. 4 shows that the Remix profile captures only one side of the profile curve, limiting its ability to evaluate models when actuation and tiling make objects too large. For instance,

an object with an absolute area size of 600 on our profile will be profiled into α at 21.9% on Efficientdet-D6, while on Efficientdet-D0, it exceeds $\alpha > 100\%$.

Our model profiler generates sub-images for each input image from the profiling dataset, ensuring that the models see various content and sizes. The collection used to profile each model consists of original images and sub-images, created by uniformly partitioning the original ones for each available model resolution.

3.1.2 Historical Frames Object Extraction. Uirapuru needs to identify regions of interest and the sizes of objects in these regions in each frame to create a tile plan. We assume that the object distribution of the historical frames, provided or collected during the bootstrap phase, is a good overall representative of how objects will appear in the scene. Further, we assume that, during these historical frames, the camera transformation is in an *identity* state. This means that the historical frames are either viewed from a global perspective, representing the widest angle of view that encapsulates all the scenes of interest, or a collection of frames with smaller angles that can represent the global perspective when combined. The collection of frames with smaller angles does not need to be captured simultaneously but rather in a short period. The historical frames need to represent the distribution of objects in the scene correctly. Therefore, as long as the capture of those frames is done in a short period of time, this does not pose challenges to Uirapuru performance as demonstrated later in §4.3.3. As the camera deployer typically controls the placement and understands the scene, we believe this is a reasonable assumption for this to be feasible.

Uirapuru collects all objects from the historical frames during the bootstrap phase. The best way to extract those objects initially would be to use an Oracle model; however, as this is not possible for real-case scenarios, we use the best strategy available to us. In our case, we employ the uniform partition strategy using the most accurate model possible.

Previous approaches demonstrate that such choice, while approximate, is enough to provide reliable results [21, 54].

3.1.3 Plan Runtime Estimation. Uirapuru is designed to generate plans rapidly so every frame is executed with a new tile plan. Theoretically, one would have to update the tile plan only when a camera actuation happens. In practice, however, the camera is often constantly in motion (e.g., panning or tilting slowly across a region of interest) so we chose to re-compute a new plan each frame. Although our algorithm is fast, it still introduces a non-negligible overhead that we need to subtract from the total per-frame latency SLO to get to the actual budget for inference. As the plan creation might vary depending on the number of objects in a frame, we use the historical frames to estimate the introduced overhead. We first collect the time to execute the tiling algorithm in each historical frame using the latency SLO. Then, we use the 99th percentile of the times, plus 10% of its value as a safety tolerance, as the cost for plan creation.

3.2 Runtime Phase

Uirapuru’s runtime phase generates a tile plan, runs inference on the tiles, and merges the detection results for every incoming frame. First, camera transformations triggered by actuation are tracked using the distribution controller, and the current camera state is used to extract the historical objects relevant to the current frame’s view (§3.2.1). Using these objects, the tile planer generates a fresh tiling layout that maximizes accuracy while respecting the available latency budget (§3.2.2). Uirapuru has two working modes that depend on the chosen SLO guarantee: a more stringent, conservative mode (for a per-frame latency guarantee) and a relaxed, non-conservative mode (for an average inference time guarantee). Finally, each tile is inferred with its associated model, and the individual detections are merged to produce the final results (§3.2.3). The inputs to the runtime phase are: 1) a family of models with its corresponding profiles, 2) the historical global view objects, 3) the current camera frame with its corresponding actuation, 4) the latency budget, and 5) the desired SLO working mode.

3.2.1 Distribution Controller. The object distributor controller has two functions: 1) transform the objects extracted from the historical frames in the global view coordinates into local coordinates, and 2) generate the tile object distribution for each tile position requested by the tile planner. As the camera operator applies an actuation, the objects in the view will change in size and appearance. Therefore, the system adapts historical objects by applying the same transformations, ensuring consistency between global and local distributions. This design allows Uirapuru to execute the transformation only for objects detected in historical

frames and not for objects detected by the camera during the runtime phase, which avoids problems with discontinuity or fast-moving objects such as cars. For example, fast-moving cars will be captured over several historical frames, their locations and sizes identified in the global view distribution, and then later transformed to the local distribution if the street appears in it. As long as Uirapuru collects enough historical frames, the distribution controller will be able to identify the distributions of objects in all camera regions correctly, providing reliable input for adaptive tiling.

The distribution controller tracks the camera transformation T over time. We assume that the camera initially starts in a rest position, i.e., T is a 3D identity transformation that corresponds to the global scene perspective with a camera view V_{global} . Each actuation by the camera operator causes a camera movement, potentially extending over multiple frames, that introduces a change to the transformation ΔT . This change ΔT is passed as additional input to the controller together with each image frame. The goal is then to extract the relevant objects from the global historical frames and transform their coordinates into the local coordinates of the current frame.

Since we do not have full access to the 3D coordinates of the historical objects, but only their 2D bounding boxes, we can only approximately estimate their new 2D locations using re-projection [14, 26]. The distribution controller computes the local 3D transformation of a frame, T_{frame} , by tracking and accumulating ΔT . Assuming a simple pinhole camera model, we compute the local view of a frame, V_{frame} , by applying T_{frame} to the original view of the camera V_{global} . Afterward, we re-project the 2D coordinates of the bounding boxes of the historical objects from the plane of V_{global} onto the local plane defined by V_{frame} . Objects that lie outside of V_{frame} are removed as they have no impact on the tile planning. While our approach is based on an approximate estimation and may be potentially affected by issues such as object occlusions and distortions, it achieves a good trade-off between computational efficiency and practical accuracy in our scenario. We acknowledge the use of more sophisticated computer graphics and vision techniques (e.g., motion estimation [38], view synthesis [33], image-based rendering [40], or deep learning [51]) as interesting avenues for future work.

3.2.2 Tile Planer. The tile planner slices the high-resolution frame into a collection of non-overlapping image regions (tiles) and selects a model for each tile in such a way that it achieves the highest possible (estimated) accuracy while guaranteeing execution within the required latency. Exploring all the possible tile layouts and model combinations is a time-consuming task that can take up to minutes [21]. One reason for that is that previous approaches typically link the arrangement of tiles directly with the model selection, i.e.,

tiles are sized according to the available models. Here, our main insight is that, for fast performance, we have to **separate the tile arrangement from the model selection** to reduce the search space of combinations significantly. We propose to first perform the division of the image independently using some form of hierarchical space partitioning, disregarding the available model sizes. This reduces the possible combinations of tile arrangements while allowing for a reasonable amount of space exploration and adaptation. Afterward, model selection is performed, but no longer influences the chosen regions. Instead, tiles are fit to models through resizing (up- or down-sampling). While resizing can introduce some accuracy degradation, these effects are implicitly integrated into the model selection with our new profile. Even with a reduced number of possible layouts, however, brute-force searching for the best combination of tiles and models still requires an infeasible amount of time. Therefore, we propose a recursive dynamic programming solution to solve the problem more efficiently.

Hierarchical Space Partitioning. First, we chose the quad-tree as a simple way of hierarchically partitioning the image into regions [37], but other hierarchical tree structures would also work. The quad-tree is a tree structure where each node has four children that subdivide the space of the parent into four non-overlapping quadrants. Each leaf node in a quad-tree corresponds to a tile region that needs to be inferred. We select a maximum tree depth based on the resolution of the input frame (in our case we use a depth of 3 for 4K input images), and explore all possible topologies as options for tile layouts.

Tile Plan. For each node, we can compute its distribution of object sizes by first finding the objects falling into the node's region W from the set of local objects 0, and then define a vector D_0^W with bb elements, where each element represents the normalized number of objects (ϕ) for each of the pre-defined sizes:

$$D_0^W = \langle \phi S_0, \phi S_1, \dots, \phi S_{b-1} \rangle$$

The object distribution of a node can then be combined with a model profile to generate a regionally-adaptive estimate of that model's accuracy:

$$eAP^W = D_0^W * A_M$$

The choice of tree topology and model selection in each leaf node constitute a tile plan. The estimated accuracy of a plan, $PeAP$, is simply the sum of all the tile's estimated accuracy weighted by the fraction of objects λ_W in each tile:

$$PeAP = \sum_{W \in W_p} eAP^W \cdot \lambda_W \quad (1)$$

DP-DP Algorithm. To optimally select a set of nodes (tiles) from a quad-tree topology and assign models to those

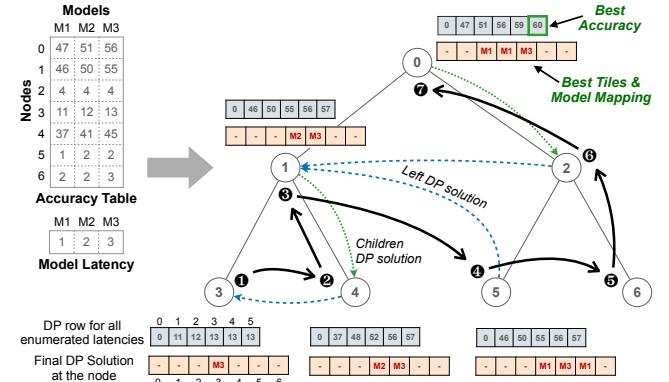


Figure 5: Illustration of the depth-first post-order dynamic programming algorithm for a binary tree of depth 3.

nodes, we observe that this combinatorial problem can be formulated as a 0-1 knapsack problem [22] with additional constraints. In this context, the items in the “knapsack” are tuples consisting of nodes and models, where the weight of the item (tuple) is the model's inference latency, and the value of the item is the model's accuracy for that corresponding tile. The total inference latency budget represents the capacity of the knapsack. There are two additional constraints: (C1) a node cannot be selected if any of its descendant nodes are chosen, as this would create overlapping areas and lead to redundant execution; (C2) only one model can be selected for a node, so two tuples with the same tile cannot be chosen. To solve this problem optimally in (pseudo-)polynomial time, we propose a dynamic programming-based (DP) solution that traverses the items (nodes) and builds the DP solution in a specific order to avoid constraint violation. We refer to this algorithm as the **depth-first post-order dynamic programming (DP-DP) algorithm**.

The algorithm first determines each node's optimal assignment to find the model that maximizes its accuracy. The algorithm then uses a depth-first post-order recursion to build the partial and optimal DP solution at each node. This traversal order ensures that optimal solutions for child nodes are available at parent nodes and that partial optimal solutions for the left sub-tree or left sibling are available when we reach the right sub-tree nodes. The algorithm ultimately returns an array with the optimal mapping of nodes to models, where every node is either assigned a model or marked as empty. Appendix A presents the recursive pseudo-code for the DP-DP algorithm.

Example. Fig. 5 illustrates a simple example of the DP-DP algorithm applied to a binary tree with three levels. The accuracy table contains the estimated accuracy for 7 nodes and 3 models. Note that the accuracy values differ across tiles (nodes), even for the same model, due to variations in

object distribution and sizes. The latency vector indicates the inference latency of the three models. Our DP-DP algorithm starts ❶ by finding the DP solution at Node 3. At Node 3, with a maximum latency budget of 5, the algorithm selects Model M1, which has an inference latency of 3 and accuracy of 13. Moving ❷ to the second node in the depth-first post-order traversal, Node 4, the algorithm selects the best combination of both the tiles (Node 3 and 4) and chooses Model M2 and M3, respectively, resulting in a total latency of 5 and aggregate accuracy of 57. At the parent node 1, the algorithm selects ❸ the best solution between Nodes 1 and 4, retaining the solution from Node 4. At Node 5, the algorithm selects ❹ the optimal combination of the solution at Node 4 and the best model (if any) at Node 5. Finally, following the recursive process to the root node, the algorithm determines ❺ the optimal node-to-model mapping: Node 2 with Model M1, Node 3 with Model M1, and Node 4 with Model M3, achieving the highest accuracy of 60.

In the worst case, the latency interval is 1 unit when discretizing the latency values. Therefore, the asymptotic complexity of our algorithm becomes $O(\#Tiles \cdot |\mathbb{M}| \cdot L_{slo})$, where L_{slo} is the inference latency budget. For proof of the optimality of the DP-DP algorithm, we refer to the supplemental material found in Appendix B.

Working Modes. The latency of a plan is simply estimated by summing up the inference times for all its tiles. However, Uirapuru has two working modes, non-conservative and conservative, that depends on which latency estimate from the model profile is chosen. Uirapuru’s non-conservative mode uses the mean inference time of the model profile, L_M^{mean} . While this allows plans to generally select larger models, it also makes tile plans more prone to underestimate a frame’s real execution time. Ultimately, the system will converge towards an average execution time over the execution of a sequence of frames, but a per-frame latency is not guaranteed. Uirapuru’s conservative mode guarantees that a certain percentile of frames fulfills the specified latency SLO by using L_M^{99} , the more strict estimate of model inference time. This forces the plans to select models more conservatively and makes them less prone to underestimate the real execution time. By default, Uirapuru’s conservative profile uses the 99th percentile inference time estimate, and so the system should estimate the time spent on inference accurately for at least 99% of frames, but other values could be chosen depending on the required SLO.

Plan Election. Uirapuru additionally creates uniform plans that simply slice the high-resolution image into non-overlapping tiles for each model in the family. The final tile plan for a frame is the most accurate between the adaptive and the best uniform plan.

Table 1: Parameters of EfficientDet models.

Variant	D0	D1	D2	D3	D4	D5	D6
Input Size	512 ²	640 ²	768 ²	896 ²	1024 ²	1280 ²	1280 ²
Size (MB)	61.3	92.2	112	154	235	359	503

3.2.3 Frame Inference. Once the tile plan is established, the actual model inference can be performed. The frame execution module slices the high-resolution image into tiles according to the plan layout and runs inference on each tile using the designated model. Padding is added before a tile is passed to a model to reduce the chance of missing objects at tile boundaries. Padding is a common practice, and there are many ways to decide the size of the padding [21, 43, 48]. We chose a simple uniform padding, which provided sufficient results with minimum overhead. The individual detection results from the tiles are merged using Non-Maximum Suppression [9] to avoid object duplicates.

4 EVALUATION

We first evaluate Uirapuru’s performance in terms of real-time inference latency guarantees and accuracy and compare it to existing baselines. We then perform an ablation study to examine the individual components’ performance. Furthermore, we conduct a case study using Uirapuru in a real-world PTZ camera. Finally, we conducted an overhead analysis. We then perform an ablation study to examine the individual components’ performance.

4.1 Experimental Setup

Hardware and Implementation. We evaluate our approach using two series of edge device hardware: the Jetson AGX Xavier [29] and the Jetson AGX Orin [30] series. Jetson AGX Xavier has a 512-core NVIDIA Volta architecture GPU with 64 Tensor Cores GPU and an 8-core NVIDIA Carmel Arm®v8.2 64-bit CPU. In contrast, the Jetson AGX Orin has a 2048-core NVIDIA Ampere architecture GPU with 64 Tensor Cores and a 12-core Arm Cortex 64-bit CPU. Our Uirapuru system prototype is implemented using Python 3.8. The models are built on top of Google’s TensorFlow [10] framework, and we use OpenCV [31] for image processing.

Dataset and Models. We use the EfficientDet [42] family as our models for inference to be consistent with the Remix baseline. We use these variants ranging from D0 to D6 for all experiments. Each variant has different input and memory sizes (see Tab. 1). The models are fine-tuned using the training and test images from the PANDA Image dataset [45] using 600 epochs.

While existing datasets in the literature have some of the specific requirements of our scenario [2, 45, 49], none could fulfill all of them: high-resolution, steerable videos with a

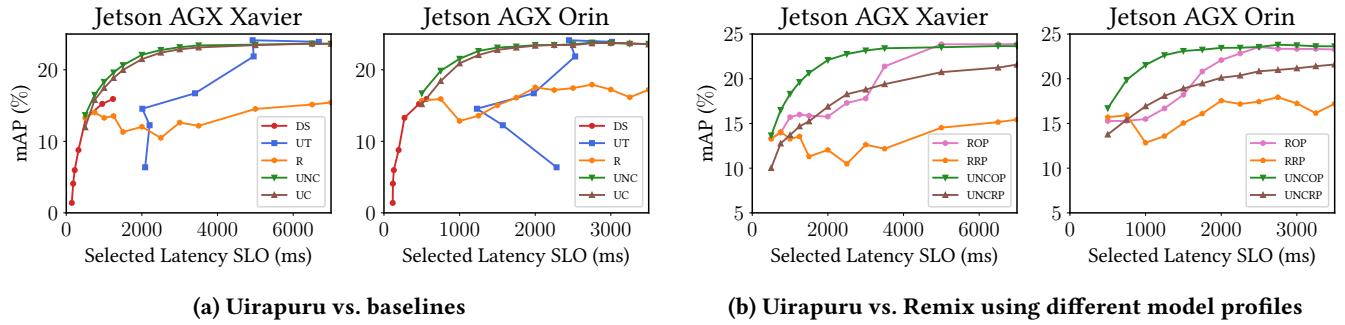


Figure 6: mAP performance of Uirapuru on the steerable PANDA dataset across different hardware.

high density of objects, and pre-defined/controlled movements. As a result of such limitation, we created a steerable video dataset based on the PANDA dataset [45] for our evaluation, exploiting the fact that we have an extremely high image resolution available (see example in Appendix D). The original resolution of the video is 26,753x15,052 pixels, allowing us to create smaller 4K cut-out videos that virtually represent transformations (i.e., pan, tilt, and zoom movements) that a steerable camera would apply. In total, the PANDA dataset has 10 video sequences. We generate a random PTZ sequence for each of the 10 sequences, where each randomly generated sequence contains at least one of the three types of actuation and, in some cases, more than one. The type and parameters of each actuation (i.e., direction, magnitude, start frame, and frame count) are randomly selected, but image boundaries are respected. The duration of each actuation is randomly selected from a range of 15 and a maximum of 50 frames. There is a 15-frame pause (no movement) between the actuation.

Baselines. We compare Uirapuru against the results obtained by three baseline approaches: down-sampling (**DS**), uniform tiling (**UT**), and Remix (**R**).

Evaluation Parameters. We evaluate 12 latency SLOs for Xavier: 500, 750, 1000, 1250, 1500, 2000, 2500, 3000, 3500, 5000, 6500, and 7000 ms, and 13 SLOs for Orin: 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 2750, 3000, 3250, and 3500 ms, covering from the smallest inference time with down-sampling to the largest uniform tiling for both hardware. We used the mean averaged precision (mAP) and the average precision at IoU=0.50 (AP50), which are widely used metrics to evaluate video analysis tasks in industry [5].

4.2 Overall Performance

We first evaluate the overall performance of Uirapuru’s non-conservative (**UNC**) and conservative (**UC**) mode in terms of accuracy in Fig. 6a. Each point represents the achieved mAP for a latency SLO on the steerable PANDA dataset. Here, we only report mAP as it is the most stringent of our metrics. Uirapuru’s non-conservative version has an mAP

improvement factor between 0.97-1.45× compared to the closest baseline for both Xavier and Orin, outperforming especially in the SLO range from 500ms until 3500ms, with the highest improvement of 1.45× being achieved at the 3000ms SLO on the Xavier. For the Orin, **UNC** has a better mAP than the baselines in the range from 500ms to 2250ms, with the most significant improvement for an SLO of 1500ms. Uirapuru’s conservative version performs very similarly, closely following the trend of its non-conservative counterpart. The slight performance decrease is explained by the fact that Uirapuru’s conservative version has more stringent constraints on the models that can be selected. As the latency SLO grows, both versions have more options available, bringing the performance of Uirapuru’s conservative version closer to the non-conservative one. Above the 5000ms SLO for the Xavier and 2500ms for the Orin, Uirapuru performance roughly converges towards the most expensive and accurate choice, which is uniform tiling (**UT**) with EfficientDet-D5 and D6.

Overall, Uirapuru achieves higher mAP in a shorter time than the baselines. Uirapuru is up to 4.53× faster in terms of latency on the Jetson AGX Xavier hardware in reaching a certain mAP value compared to similar mAP values from the baselines within 1% of tolerance. For example, for the Xavier, the **UNC** mode achieves 16.49% mAP at 750ms, while uniform tiling using EfficientDet-D3 achieves 16.71% mAP requiring 3404ms. Similarly, **UNC** reaches up to 2.53× inference speedup while achieving on-par mAP for the Orin.

4.2.1 Average Latency vs Per-frame SLO. Remix and Uirapuru’s non-conservative mode both provide an average latency execution convergence, i.e., the average execution time per frame will ultimately become close to the latency SLO. However, this means that occasionally, the inference time of a frame might exceed the specified SLO. Uirapuru’s conservative mode provides a more strict per-frame latency guarantee. In Tab. 2, we show the miss rate (i.e., the percentage of frames with a higher execution time than the allowed SLO) in percentage values for Remix (**R**) and both our modes. It can be seen that **UNC** delivers more than 99% of the frames within

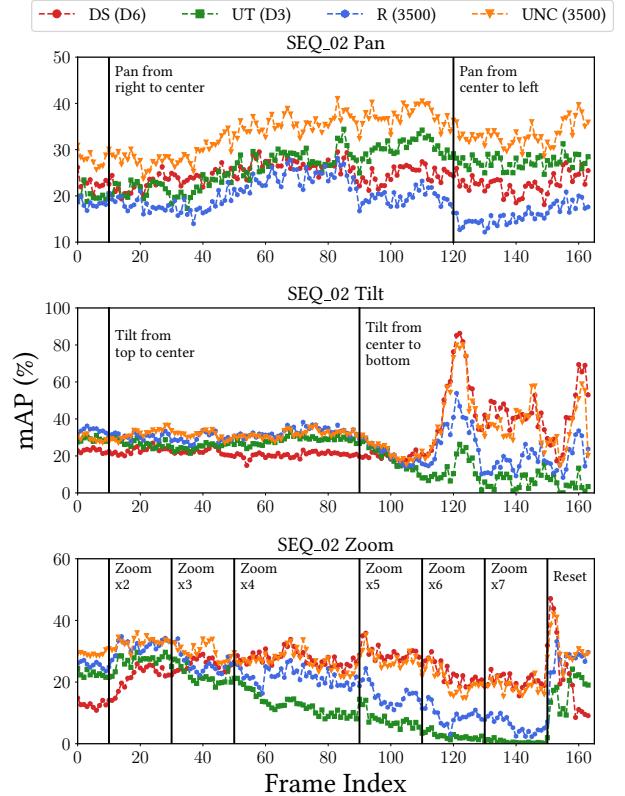
Table 2: Miss rate (%) for Remix (R), Uirapuru Non-Conservative (UNC), and Uirapuru Conservative (UC).

SLO	AGX Xavier			AGX Orin		
	R	UNC	UC	R	UNC	UC
500	0.61	7.04	0.49	64.48	3.90	0
750	40.51	6.56	0.14	0	1.05	0.27
1000	36.95	5.16	0	16.46	4.60	0.34
1250	0	1.19	0.07	21.72	0.35	0
1500	2.73	2.23	0.07	27.59	0.28	0.06
1750	-	-	-	25.68	0.07	0
2000	8.33	2.65	0.14	23.49	0.21	0.06
2250	-	-	-	23.29	0	0
2500	22.68	0.14	0.07	27.80	0	0.13
2750	-	-	-	26.02	0	0.13
3000	20.97	0.49	0	25.88	0	0
3250	-	-	-	21.58	0.21	0
3500	18.65	0.14	0	24.65	0	0.13
5000	8.40	0	0	-	-	-
6500	13.73	0	0	-	-	-
7000	0	0.07	0.07	-	-	-

the desired latency SLO. Uirapuru’s non-conservative miss rate can be explained by its profile that takes the average inference time for models into account instead of the more stringent *99th* percentile. While Uirapuru’s non-conservative cannot provide such stringent guarantees, it still provides significantly lower miss rates than Remix.

4.2.2 PTZ Breakdown. To emphasize Uirapuru’s understanding of PTZ actuation and changing object sizes, we created three sequences out of the second sequence (SEQ_02) from the original PANDA dataset where only one type of actuation (pan, tilt, or zoom) is applied at customized moments. In Fig. 7, we show a per-frame performance breakdown for several strategies on these sequences for a medium latency SLO of 3500ms. Here, we compare UNC to Remix with the same SLO (**R**), the EfficientDet-D6 variant as the best down-sampling option (**DS D6**), and uniform tiling with Efficientdet-D3 as the uniform choice that runs in roughly the same time (**UT D3**). Note that Uirapuru is always on top of the mAP curve for each frame, independently of the PTZ actuation used.

While the performance for the pan case exhibits less variance (object sizes remain rather constant), variations are more pronounced and sudden for the tilt and zoom cases. For the tilt sequence (Fig. 7, middle), we can see that when the camera tilts towards the bottom part of the global view and objects become larger as they approach the camera, the performance of Remix and the uniform approach substantially drops while the down-sample approach benefits from these larger object sizes. Similar behavior can be seen for the zoom case (Fig. 7, bottom). Uirapuru correctly identifies these situations and rapidly adapts by either choosing plans closer

**Figure 7: mAP performance per frame of Uirapuru for individual actuations on the Jetson AGX Xavier.**

to down-sampling or uniform tiling, whichever is beneficial in the situation.

4.3 Ablation Studies

In this section, we evaluate how the parameters and components of Uirapuru contribute to its performance.

4.3.1 Model Profiling. We first evaluate in Fig. 6b how our non-linear, relative profile significantly improves the performance of adaptive strategies for the steerable case. The figure shows the mAP achieved by Uirapuru in non-conservative mode using both our profile (UNCOP) and Remix’s absolute size profile (UNCRP) as well as Remix using our profile (ROP) or theirs (RRP). As it can be seen, both strategies benefit greatly from the use of our new profile. Besides the improved performance of Remix with our profile, Uirapuru still provides an improvement in mAP between 0.98-1.40× in comparison for the Xavier and 1.1-1.35× for the Orin.

Uirapuru performs better for all SLOs while offering comparable mAP results for the others, indicating that the profile, while beneficial, is not the sole reason behind Uirapuru’s improved performance.

Table 3: AP50 results achieved by Uirapuru and the Baselines for the different hours in our real PTZ data.

AP50	Down-sampling D6	Uniform Partition D2	Remix SLO 2000	Uirapuru SLO 2000	Uirapuru's Difference to	
					Uniform	Remix
10:00	35.74%	70.05%	70.41%	74.63%	4.58%	4.22%
13:00	26.69%	67.90%	69.96%	73.19%	5.29%	3.22%
16:00	28.62%	72.12%	70.87%	77.14%	5.02%	6.27%
19:00	25.84%	63.29%	65.89%	68.88%	5.59%	2.99%

4.3.2 Tile Planning. Most algorithm time is spent on tile planning, so we analyze it in detail. We measured how often Uirapuru selects plans from the DP-DP algorithm or the uniform tiler. Uirapuru’s non-conservative mode only chose DP-DP plans for SLOs below 6500ms. As the SLOs increase, more frames select uniform plans. On the Xavier, DP-DP plans were chosen for 76.64% and 76.84% of frames under 6500ms and 7000ms SLOs. On the Orin, DP-DP plans were used exclusively for SLOs below 2750ms, and for 75.81%, 76.43%, 76.63%, and 76.84% of the frames under 2750, 3000, 3250, and 3500ms SLOs, respectively. This shows that uniform tiling supplements DP-DP when needed, while DP-DP still generates highly accurate plans within a limited search space.

4.3.3 Object Extraction. We evaluate how historical frames impact Uirapuru’s accuracy. Experiments showed that varying the number of historical frames had little effect on mAP across different SLOs. Using from 10% to 30% of frames results in no significant accuracy differences, indicating that Uirapuru does not need a high number of historical frames. As long as the object distribution is well-represented, Uirapuru accurately predicts object locations and sizes. Additionally, using uniform tiling with the best model to extract objects performs similarly to using ground truth data, therefore being a sufficient object extraction strategy.

4.4 Case Study

In addition to experiments with our created steerable PANDA dataset, we also evaluate Uirapuru in a real-world setup. We collected 4K video sequences (around 6,400 frames in total) from a steerable camera in a square at different hours of the day². The sequences were collected on a sunny day at 10:00, 13:00, 16:00, and 19:00, covering different lighting conditions and exhibiting average person densities of 73, 211, 233, and 147, respectively.

The steerable camera performs several PTZ actuations in a fixed loop. The actuation loop is pre-defined by the camera operator, without any interference from us, and it was always exactly the same throughout the different times of the day. The high degree of Pan and Tilt causes objects to occlude others constantly. When Zoom is applied, the camera

²The use of the data has been approved by our university ethics committee.

Table 4: AP50 values of Uirapuru when using the plans generated with the historical frames of different times.

Evaluation at	AP50			
	10:00	13:00	16:00	19:00
10:00	74.63%	-	-	-
13:00	73.91%	73.19%	-	-
16:00	77.09%	76.30%	77.14%	-
19:00	69.81%	68.26%	68.93%	68.88%

executes abrupt actuation, but it is limited in duration (less than 7% of the loop sequence) and magnification (only a 2× zoom factor). Each sequence consists of three consecutive actuation loops. We use the first loop as our historical frames for each sequence and the other two for evaluation. Similarly to Remix [21], we generate labels using the best possible strategy as our oracle model (Uniform Partition with EfficientDet D6) due to the lack of ground-truth labels for the collected sequences.

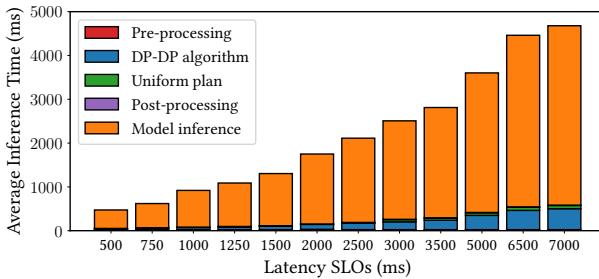
4.4.1 Overall Performance. For this analysis, we set the latency SLO to 2,000ms. We compared Uirapuru against Remix, the best down-sampling approach (EfficientDet D6), and Uniform Partition using EfficientDet D2 since it has the execution time closest to the SLO. Like Remix [21], we use the AP50 in our case study. However, complete results for the mAP metric, which confirm the observed trends, can be found in Appendix C.

Tab. 3 shows a comparison against the baselines for different times of the day. Uirapuru achieves the highest AP50 at all times. Remix and Uniform Partition have similar results, but Uniform Partition performs better at 16:00. As detailed in §4.2.2, Remix becomes suboptimal under strong Pan and Tilt motion: plans that should prioritize dense or complex areas are instead executed in easier or empty regions. Frequent camera motion also makes frame skipping harder, further limiting Remix’s ability to adapt. When objects are more evenly distributed, Remix can perform worse than Uniform Partition, as observed at 16:00. In contrast, Uirapuru can address both issues with its per-frame plan creation and improve object detection in the scene. Additionally, larger Zooms can drastically degrade the performance of Remix and Uniform while Uirapuru remains robust, widening the performance gap. We can see that Uirapuru offers tiling solutions that can cope with those changes and maintain higher mAP performance. However, Zoom duration and magnification were limited by the operator in our case study, so these effects are less prominent than they would be in scenarios that require closer inspection.

4.4.2 Performance Under Different Update Rates. As the object distribution changes over time, Uirapuru may need to update its object history by collecting new frames.

Table 5: AP50 values achieved by Uirapuru under different weather conditions.

AP50	Uirapuru	Remix	Uniform	Downscale
Sunny	73.19%	69.96%	67.90%	26.69%
Dark	68.88%	65.89%	63.29%	25.84%
Cloudy	75.46%	71.91%	71.33%	29.11%

**Figure 8: Per-frame break-down of the time spent on pre-processing, tile planning, post-processing, and model inference across different latency SLOs.**

Tab. 4 shows how historical frames collected at different times might affect Uirapuru. The results show that staleness in historical frames has minimal impact, with performance differences under 1%. As long as object distribution remains similar, Uirapuru does not need to re-run its bootstrap phase. Uirapuru’s runtime estimator (in §3.1.3) calculates the inference time SLO using the model profiles and the object history. At 10:00, the number of objects is smaller, creating less overhead in the system. Therefore, the historical frames from 10:00 yield the best performance, even 9 hours later. This overhead will be discussed in §4.5.

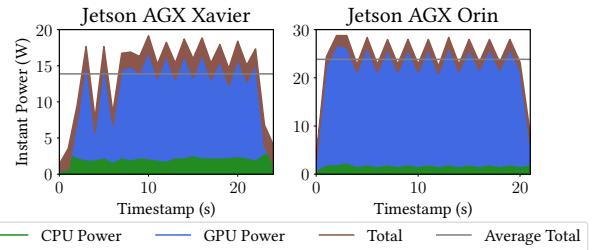
4.4.3 Performance Under Different Weather Conditions. The collected 4K videos also exhibit different lighting and weather conditions (namely sunny, cloudy, and dark) (see in Appendix D). The sunny and cloudy sequences were recorded at 13:00, while the dark sequence was captured at 19:00.

Tab. 5 shows that Uirapuru performs best, independent of these conditions. We can also see that the cloudy condition gives better results than the sunny and dark ones for all approaches. This is mainly due to even lighting under the cloudy condition, eliminating shadows, flares, and brightness changes that hinder object detection. When it is dark at night, artificial lighting intensifies these issues, further reducing accuracy.

4.5 Overhead Analysis

Next, we analyze the extra overhead introduced by Uirapuru.

4.5.1 Execution Time. Fig. 8 shows the average execution time of each step on the Jetson Xavier. Pre-processing, which consists of extracting and transforming historical objects,

**Figure 9: Energy consumption trace when processing ten frames with Jetson Xavier (left) and Orin (right).**

computing the object distribution for each quad-tree node, and estimating the expected accuracy for each node-model pair, introduces minimal overhead, consuming only 5.14% of the available SLO at 500ms and 0.51% at 7000ms. Post-processing, which merges detections using Non-Maximum Suppression (NMS), also has low overhead, ranging from 0.59% at 750ms to 0.15% at 7000ms. The uniform plan computation is fast, contributing 0.24-2.02% of the SLO, while DP-DP introduces the largest overhead, from 4.37% at 500ms to 10.22% at 7000ms. Generally, Uirapuru’s overhead remains low (less than 12.5% for all SLOs), with most time spent on model inference (87% on Xavier, 89% on Orin).

In our case study in §4.4.1, the pre-processing stage can be affected by the number of objects in the historical frames. As object numbers grow, extraction, transformation, and distribution computations take longer, reducing the inference time and affecting accuracy. Furthermore, Uirapuru generates a new plan every frame, even when actuation is minimal, leading to redundant computations.

4.5.2 Memory Footprint. Uirapuru can process each frame using multiple models. Thus, all models (from D0 to D6) must be loaded and initialized into memory during the bootstrap phase. Our measurements show that Uirapuru consumes a maximum of 12.2 GB memory during inference on both Jetsons. Approximately half of this memory is used by D5 and D6, which consume nearly 6.1 GB of memory combined.

4.5.3 Energy Consumption. Fig. 9 shows the power consumption of Uirapuru on both hardware. We used Jetson’s built-in tegrastats monitor to collect the instant energy consumption traces. We executed 10 frames using an SLO of 2000 ms. The total power consumption per frame is, on average, about 13.9 W for the Jetson Xavier and 23.8 W for the Orin. The GPU consumes most of the power in both, with an average of 11.7 W in the Xavier and 22 W in the Orin. The power consumption of Uirapuru matches those achieved by Remix [21], and the total average consumption represents less than half of the nominal power of the devices.

5 DISCUSSION AND LIMITATIONS

In the following, we'd like to discuss the limitations of our approach and possible opportunities for further exploration.

Steerable Camera Parameters. Uirapuru assumes that the steerable camera is mounted in a fixed position, as PTZ cameras are. This is necessary for the distribution controller to process the object history and create the current-view object distribution. For example, video sequences from unmanned aerial vehicles (drones) that are not mounted in a fixed position would imply more complex actuation. This would require a much more robust and time-consuming version of the distribution controller.

Historical Objects. Selecting historical frames for high-quality partition plans remains an open question. Extensive object histories increase pre-processing overhead. One solution is to reduce the sampling rate when collecting objects. Another solution would be to group objects in regions and store pre-processed distributions. While this would reduce the overall overhead linked to the number of objects in the object history, it would also introduce a new challenge: deciding how to merge these objects. Therefore, we leave these directions for future work.

Reducing Runtime Overhead. Uirapuru uses fixed parameters in the DP-DP algorithm, which could be optimized to reduce the overhead as the latency budget grows. Dynamically adjusting parameters, such as increasing the latency interval (step size) to shrink the inference latency budget array, could lower the runtime without impacting accuracy. Additionally, skipping tile plan creation for frames with minimal or no actuation could reduce overhead, allowing better plans to be used when updates are unnecessary. Finally, more effort is required to optimize memory/energy usage.

Memory Bottleneck. Uirapuru's performance is influenced by a trade-off between memory capacity and the available model pool. Larger models provide higher detection accuracy, while smaller ones enable faster inference. A diverse pool of models would enrich the optimization space of Uirapuru, leading to better performance. However, loading more models into memory is not always possible and might exceed the capacity of many edge platforms (e.g., Jetson Nano). To address this, Uirapuru can rely on a smaller pool of models or utilize a more optimized framework for deploying machine learning models on resource-constrained devices like LiteRT [11]. Such reductions come at the cost of accuracy, even though reducing the pool of models equally affects baseline accuracy. While we use EfficientDet as an example throughout this paper, Uirapuru is agnostic to the underlying models and can operate with any compatible neural networks.

Skipping. Currently, tile skipping is only done implicitly in our DP-DP algorithm, but not explicitly during the inference in the runtime phase (as done e.g., in Remix where tiles are marked as empty if they were empty in previous frames). Skipping on a steerable camera scenario is less trivial than in the static case; actuation can quickly change object distributions between frames. However, more latency budget could be gained when performed effectively. Developing a skipping mechanism that works in the steerable context could be a promising avenue for future work.

6 CONCLUSION

We introduce Uirapuru, a framework for latency-constrained object detection for **high-resolution steerable** cameras. Uirapuru tackles the challenges introduced by persistent camera actuation and changing viewpoints by incorporating an understanding of that movement into its system design. Additionally, Uirapuru proposes a new efficient way of generating adaptive tile plans quickly for each video frame. Our evaluation with synthetic PTZ video datasets shows that Uirapuru improves accuracy by up to $1.45\times$ for the same inference time and produces results with similar accuracy with a $4.53\times$ inference speedup compared to previously existing approaches. Our experiments with real-world videos collected from an actual PTZ camera further confirm the significant improvements brought by Uirapuru.

ACKNOWLEDGMENTS

This work is part of the Real-Time Video Surveillance Search project (grant number 18038), financed by the Dutch Research Council (NWO).

REFERENCES

- [1] Richard Bellman. 1954. The theory of dynamic programming. *Bull. Amer. Math. Soc.* 60, 6 (1954).
- [2] Yaru Cao, Zhijian He, Lujia Wang, Wenguan Wang, Yixuan Yuan, Dingwen Zhang, Jinglin Zhang, Pengfei Zhu, Luc Van Gool, Junwei Han, Steven C. H. Hoi, Qinghua Hu, Ming Liu, Chong Cheng, Fanfan Liu, Guojin Cao, Guozhen Li, Hongkai Wang, Jianye He, Junfeng Wan, Qi Wan, Qi Zhao, Shuchang Lyu, Wenzhe Zhao, Xiaoqiang Lu, Xingkui Zhu, Yingjie Liu, Yixuan Lv, Yujing Ma, Yuting Yang, Zhe Wang, Zhenyu Xu, Zhipeng Luo, Zhimin Zhang, Zhiqiang Zhang, Zihao Li, and Zixiao Zhang. 2021. VisDrone-DET2021: The Vision Meets Drone Object detection Challenge Results. In *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*. 2847–2854.
- [3] João Carreira and Andrew Zisserman. 2017. Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. In *IEEE CVPR*. 4724–4733.
- [4] Tiffany Yu-Han Chen, Hari Balakrishnan, Lenin Ravindranath, and Paramvir Bahl. 2016. GLIMPSE: Continuous, Real-Time Object Recognition on Mobile Devices. *GetMobile Mob. Comput. Commun.* 20, 1 (2016), 26–29.
- [5] COCO Consortium. 2024. Common Objects in Context. <https://cocodataset.org/home>. Accessed: 2024-03-15.
- [6] Sandeep D'Souza, Victor Bahl, Lixiang Ao, and Landon P. Cox. 2020. Amadeus: Scalable, Privacy-Preserving Live Video Analytics. *CoRR*

- abs/2011.05163 (2020). arXiv:2011.05163 <https://arxiv.org/abs/2011.05163>
- [7] Chengzhen Duan, Zhiwei Wei, Chi Zhang, Siying Qu, and Hongpeng Wang. 2021. Coarse-grained Density Map Guided Object Detection in Aerial Images. In *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*. 2789–2798.
- [8] Abdulrahman Fahim, Evangelos Papalexakis, Srikanth V. Krishnamurthy, Amit K. Roy Chowdhury, Lance Kaplan, and Tarek Abdelzaher. 2023. AcTrak: Controlling a Steerable Surveillance Camera using Reinforcement Learning. *ACM Trans. Cyber-Phys. Syst.* 7, 2, Article 14 (apr 2023), 27 pages. <https://doi.org/10.1145/3585316>
- [9] Ross B. Girshick. 2015. Fast R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*. 1440–1448.
- [10] Google. 2024. Tensorflow. <https://www.tensorflow.org/>. Accessed: 2024-09-05.
- [11] Google. 2025. LiteRT. <https://ai.google.dev/edge/liter/>. Accessed: 2025-08-13.
- [12] Giulio Grassi, Kyle Jamieson, Paramvir Bahl, and Giovanni Pau. 2017. Parkmaster: an in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments. In *ACM/IEEE Symposium on Edge Computing (SEC)*. 16:1–16:14.
- [13] Hongpeng Guo, Shuochao Yao, Zhe Yang, Qian Zhou, and Klara Nahrstedt. 2021. CrossRoI: cross-camera region of interest optimization for efficient real time video analytics at scale. In *ACM Multimedia Systems Conference (MMSys)*. ACM, 186–199.
- [14] Richard Hartley and Andrew Zisserman. 2004. *Multiple View Geometry in Computer Vision*. Cambridge University Press.
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE CVPR*. 770–778.
- [16] hisilicon. 2024. Kirin 970. <https://www.hisilicon.com/en/products/Kirin/Kirin-flagship-chips/Kirin-970>. Accessed: 2024-06-22.
- [17] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 115–131.
- [18] Shubham Jain, Viet Nguyen, Marco Gruteser, and Paramvir Bahl. 2017. Panoptes: servicing multiple applications simultaneously using steerable cameras. In *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. ACM, 119–130.
- [19] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. 2020. Spatula: Efficient cross-camera video analytics on large camera networks. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 110–124.
- [20] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *ACM SIGCOMM*. 253–266.
- [21] Shiqi Jiang, Zhiqi Lin, Yuanchun Li, Yuanchao Shu, and Yunxin Liu. 2021. Flexible high-resolution object detection on edge devices with tunable latency. In *ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*. 559–572.
- [22] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *Multidimensional Knapsack Problems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 235–283. https://doi.org/10.1007/978-3-540-24777-7_9
- [23] Jingzong Li, Libin Liu, Hong Xu, Shudeng Wu, and Chun Jason Xue. 2023. Cross-Camera Inference on the Constrained Edge. In *IEEE Conference on Computer Communications (INFOCOM)*. 1–10.
- [24] Xiaochen Liu, Pradipta Ghosh, Oytun Ulutan, B. S. Manjunath, Kevin S. Chan, and Ramesh Govindan. 2019. Caesar: cross-camera complex activity recognition. In *ACM SenSys*. 232–244.
- [25] Christian Micheloni, Bernhard Rinner, and Gian Luca Foresti. 2010. Video Analysis in Pan-Tilt-Zoom Camera Networks. *IEEE Signal Process. Mag.* 27, 5 (2010), 78–90.
- [26] Camille Morlighem, Anna Labetski, and Hugo Ledoux. 2022. Reconstructing historical 3D city models. *Urban Inform.* 1, 1 (2022).
- [27] Vinod Nigade, Pablo Bauszat, Henri E. Bal, and Lin Wang. 2022. Jellyfish: Timely Inference Serving for Dynamic Edge Networks. In *IEEE Real-Time Systems Symposium (RTSS)*. 277–290.
- [28] Vinod Nigade, Lin Wang, and Henri E. Bal. 2020. Clownfish: Edge and Cloud Symbiosis for Video Stream Analytics. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 55–69.
- [29] Nvidia. 2024. Nvidia Jetson AGX Xavier. <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/>. Accessed: 2024-03-15.
- [30] Nvidia. 2024. Nvidia Jetson Orin. <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/>. Accessed: 2024-09-01.
- [31] OpenCV org. 2024. OpenCV. <https://opencv.org/>. Accessed: 2024-09-05.
- [32] George Plastiras, Christos Kyrou, and Theocharis Theοcharides. 2018. Efficient ConvNet-based Object Detection for Unmanned Aerial Vehicles by Selective Tile Processing. In *International Conference on Distributed Smart Cameras (ICDSC)*. 3:1–3:6.
- [33] Sergi Pujades, Frederic Devernay, and Bastian Goldluecke. 2014. Bayesian View Synthesis and Image-Based Rendering Principles. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3906–3913.
- [34] Qualcomm. 2024. Snapdragon 855 Mobile Platform. <https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-855-mobile-platform>. Accessed: 2024-06-22.
- [35] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 6 (2017), 1137–1149.
- [36] Vit Ruzicka and Franz Franchetti. 2018. Fast and accurate object detection in high resolution 4K and 8K video using GPUs. In *IEEE High Performance Extreme Computing Conference (HPEC)*. 1–7.
- [37] Hanan Samet. 1984. The Quadtree and Related Hierarchical Data Structures. *ACM Comput. Surv.* 16, 2 (1984), 187–260.
- [38] Davide Scaramuzza and Friedrich Fraundorfer. 2011. Visual Odometry [Tutorial]. *IEEE Robotics Autom. Mag.* 18, 4 (2011), 80–92.
- [39] Navin Sharma, David E. Irwin, Prashant J. Shenoy, and Michael Zink. 2011. MultiSense: fine-grained multiplexing for steerable camera sensor networks. In *ACM SIGMM Conference on Multimedia Systems (MMSys)*. 23–34.
- [40] Harry Shum and Sing Bing Kang. 2000. Review of image-based rendering techniques. In *Visual Communications and Image Processing*, Vol. 4067. SPIE, 2–13.
- [41] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning (ICML)*, Vol. 97. 6105–6114.
- [42] Mingxing Tan, Ruoming Pang, and Quoc V. Le. 2020. EfficientDet: Scalable and Efficient Object Detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 10778–10787.
- [43] F. Ozge Unel, Burak Oguz Özkalayci, and Cevahir Çigla. 2019. The Power of Tiling for Small Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR)*. 582–591.
- [44] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip H. S. Torr. 2019. Fast Online Object Tracking and Segmentation: A Unifying Approach. In *IEEE CVPR*. 1328–1338.
- [45] Xueyang Wang, Xiya Zhang, Yinheng Zhu, Yuchen Guo, Xiaoyun Yuan, Liuyu Xiang, Zerun Wang, Guiguang Ding, David J. Brady, Qionghai Dai, and Lu Fang. 2020. PANDA: A Gigapixel-Level Human-Centric Video Dataset. In *IEEE/CVF Conference on Computer Vision and Pattern*

- Recognition (CVPR).* 3265–3275.
- [46] 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 *ACM European Conference on Computer Systems (EuroSys)*. 334–351.
 - [47] Kichang Yang, Juheon Yi, Kyungjin Lee, and Youngki Lee. 2022. FlexPatch: Fast and Accurate Object Detection for On-device High-Resolution Live Video Analytics. In *IEEE Conference on Computer Communications (INFOCOM)*. 1898–1907.
 - [48] Zheng Yang, Xu Wang, Jiahang Wu, Yi Zhao, Qiang Ma, Xin Miao, Li Zhang, and Zimu Zhou. 2023. EdgeDuet: Tiling Small Object Detection for Edge Assisted Autonomous Mobile Vision. *IEEE/ACM Trans. Netw.* 31, 4 (2023), 1765–1778.
 - [49] Hongyang Yu, Guorong Li, Weigang Zhang, Qingming Huang, Dawei Du, Qi Tian, and Nicu Sebe. 2020. The Unmanned Aerial Vehicle Benchmark: Object Detection, Tracking and Baseline. *Int. J. Comput. Vis.* 128, 5 (2020), 1141–1159.
 - [50] Xiao Zeng, Biyi Fang, Haichen Shen, and Mi Zhang. 2020. Distream: scaling live video analytics with workload-adaptive distributed edge intelligence. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 409–421.
 - [51] Chaoning Zhang, Francois Rameau, Junsik Kim, Dawit Mureja Argaw, Jean-Charles Bazin, and In So Kweon. 2020. DeepPTZ: Deep Self-Calibration for PTZ Cameras. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*.
 - [52] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Phlipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *USENIX NSDI*. 377–392.
 - [53] Qingyang Zhang, Hui Sun, Xiaopei Wu, and Hong Zhong. 2019. Edge Video Analytics for Public Safety: A Review. *Proc. IEEE* 107, 8 (2019), 1675–1696.
 - [54] 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 *ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*. 201–214.

APPENDIX

A PSEUDO CODE

Alg. 1 presents the recursive pseudo-code for the DP-DP algorithm, initiated with the call `DP-DP(root_node, 0, 0)`. The algorithm determines the optimal model assignment for each node across all discrete latency values (lines 9-15). These latency values range from zero to the maximum latency budget (i.e., the knapsack capacity) in steps defined by a latency interval. The algorithm then starts recursively constructing the partial and optimal DP solution at each node in a depth-first post-order manner (lines 6-8). At each parent node, the algorithm compares the local optimal solution with the children's optimal solution to choose the best one (lines 19-21), satisfying constraint **C1**. For each node and latency value, there are $|\mathbb{M}|$ model choices, and the algorithm selects only the best model, satisfying constraint **C2** (lines 10-15). Note that all dynamic programming algorithms depend on previous partial and optimal solutions. In our algorithm, the previous solution may come from the left sibling or the left sub-tree. If a node lacks a left sibling or left sub-tree, we use an array of zeros. Additionally, while traversing, we keep track of which models are assigned to which nodes. The algorithm ultimately returns an array with the optimal mapping of nodes to models, where every node is either assigned a model or marked as empty.

B OPTIMALITY PROOF

Dynamic programming framework is based on the *principle of optimality* [1]. It divides the problem into smaller sub-problems, finds their optimal solutions, and iteratively combines these to form the optimal solution to the original problem.

B.1 Recurrence Relation

In our context, we iterate over the tree (representing image space partitioning) to compute the list of nodes and their corresponding models from the model set \mathbb{M} that together achieve the optimal value (aggregate estimated accuracy) for a given latency budget L_{slo} . We construct the value (aggregate estimated accuracy) matrix $DP(n, l)$ at each node n for discrete latency budgets l by traversing the tree in a depth-first post-order manner. The *recurrence* relation is defined as follows:

$$DP(n, l) = \max_{m \in \mathbb{M} \text{ and } l \geq l_m} DP(n_{left}, l - l_m) + A(n, m) \quad (2a)$$

$$DP(n, l) = \max\{DP(n, l), DP(n_{left}, l), DP(n_{right}, l)\} \quad (2b)$$

Here, $A(n, m)$ represents the estimated accuracy when model m is selected for node (or tile) n , and l_m denotes the inference latency of model m . In Equation 2a, the recurrence function first computes $DP(n, l)$ based on the previous sub-solution at node n_{left} , which is the nearest left sibling node

Algorithm 1: Tiles Selection and Model Mapping

Input: A node n with optimal accuracy values DP_{left} and mapping solution SOL_{left} from the left subtree or sibling
Output: DP array with optimal accuracy values and model mapping solution SOL for every latency value at node n

Data: Matrix A representing node accuracy per model, array L storing latency values for each model, latency SLO L_{slo} , set \mathbb{N} of nodes, and set \mathbb{M} of models

```

1 Function DP-DP( $n, DP_{left}, SOL_{left}$ ):  

2    $DP(i) \leftarrow 0, \forall i \in [0, L_{slo}]$   

3    $SOL(i, j) \leftarrow 0, \forall i \in [0, L_{slo}], \forall j \in [0, |\mathbb{N}| - 1]$   

4   // Traverse all the children from left to right  

5    $C \leftarrow \text{CHILDREN}(n)$   

6    $DP_{temp} \leftarrow DP_{left}, SOL_{temp} \leftarrow SOL_{left}$   

7   foreach  $c \in C$  do  

8      $DP_{child}, SOL_{child} \leftarrow \text{DP-DP}(c, DP_{temp}, SOL_{temp})$   

9      $DP_{temp} \leftarrow DP_{child}, SOL_{temp} \leftarrow SOL_{child}$   

10    // Compute DP array values (accuracy) for this node  

11    for  $i \leftarrow 1$  to  $L_{slo}$  do  

12      for  $m \leftarrow 0$  to  $|\mathbb{M}| - 1$  do // Select the best model  

13         $j \leftarrow i - L(m)$   

14        if  $j \geq 0$  and  $(A(n, m) + DP_{left}(j)) > DP(i)$  then  

15           $DP(i) \leftarrow A(n, m) + DP_{left}(j)$   

16           $SOL(i, k) \leftarrow SOL_{left}(j, k), \forall k \in [0, |\mathbb{N}| - 1]$   

17           $SOL(i, n) \leftarrow m + 1$   

18        // Select the left subtree/sibling solution if better  

19        if  $DP_{left}(i) > DP(i)$  then  

20           $DP(i) \leftarrow DP_{left}(i)$   

21           $SOL(i, k) \leftarrow SOL_{left}(i, k), \forall k \in [0, |\mathbb{N}| - 1]$   

22        // Select the children solution if better  

23        if  $C \neq \emptyset$  and  $DP_{lastchild}(i) > DP(i)$  then  

24           $DP(i) \leftarrow DP_{lastchild}(i)$   

25           $SOL(i, k) \leftarrow SOL_{lastchild}(i, k), \forall k \in [0, |\mathbb{N}| - 1]$   

26    return  $DP, SOL$ 

```

on the ancestor path from node n to the root. The aim is to select the model that gives the best accuracy. Then, in Equation 2b, we exclusively select the optimal sub-solution from the current sub-solution, the previous sub-solution, and the sub-solution of the rightmost child node (satisfying Constraint C1, see §3.2.2).

B.2 Proof of Correctness

We now consider the cases one by one for different types of nodes in the tree and then use mathematical induction on the tree structure to establish the correctness of the algorithm.

Base Case. Since the root node does not have a nearest left sibling node on its ancestor path, we set $DP(n_{left}, l)$ at the root node to zero for latency budget l . This means $DP(n_{left}, l)$ will be zero for all nodes along the leftmost path. Similarly, $DP(n_{right}, l)$ will be zero for all leaf nodes for any latency budget l , as leaf nodes have no children.

Case 1: Leftmost Leaf Node. This node is the first visited node in our depth-first post-order traversal. It has no n_{left} or n_{right} . Following Equation 2a and the base case, the DP value for this node simply depends on the model that maximizes

the accuracy under latency budget l , given by $DP(n, l) = \max_{m \in \mathbb{M} \text{ and } l \geq l_m} A(n, m)$. Since there are no other nodes to consider, the DP value at this node is correctly initialized to the best achievable accuracy within the latency budget.

Case 2: Right Sibling of the Above Node. This is the second node visited. It has n_{left} (the above node) but no n_{right} . Equation 2a ensures that the best possible model m is considered in combination with the optimal solution from its left sibling n_{left} for a given latency budget l . Equation 2b helps the algorithm to explore whether including this node with a specific model gives a better solution than considering n_{left} alone. This guarantees optimality by evaluating all possible combinations between the two nodes.

Case 3: Parent of the Above Two Nodes. For simplicity and without loss of generality, we assume that the tree is binary. Hence, this node is the third node visited. It has no n_{left} but has n_{right} . Equation 2a computes the DP value by choosing the model that maximizes accuracy under the latency budget l , given by $DP(n, l) = \max_{m \in \mathbb{M} \text{ and } l \geq l_m} A(n, m)$. Since this node has children, Equation 2b ensures that the final optimal DP value results from either considering this node's contribution or a better solution from the combination of child nodes stored at the rightmost child, n_{right} .

Case 4: Leaf Nodes without Immediate Left Sibling. These nodes lack an immediate left sibling but have a nearest left sibling on the ancestor path. Equation 2a ensures that we consider the contribution of this node n with the best model in combination with the optimal solution at n_{left} . Equation 2b ensures the final optimal DP value for node n is the maximum of either including the contribution from node n or only the optimal solution from n_{left} . This case is similar to Case 1.

Case 5: Nodes with Left Sibling and Children. This is the most general case. These nodes have both a left sibling and children. Equation 2a ensures that we select the best model m in combination with the optimal solution at n_{left} . Equation 2b ensures the DP value is optimal by accounting for all contributions from its left sibling, its children (the solution that is built on top of the left sibling solution), and the node itself.

Our depth-first post-order traversal ensures that for any node n , the optimal solution at its rightmost child considers the combination of all the descendant nodes that is included on top of the optimal solution at the left sibling. This approach guarantees that choosing the best solution between the children (descendants) and the node itself satisfies Constraint C1.

We now present the proof of correctness by induction. Let $n \in [0, N]$ represent the index in the ordered sequence of tree traversal, where $n = 0$ corresponds to the case of no nodes. We define $(n, l) < (n', l')$ if $n < n'$ and $l \leq l'$.

Induction Hypothesis. The algorithm computes the value of $DP(n, l)$ correctly for all $(n, l) < (n', l')$, meaning all node indices from 0 to n are correctly processed.

Base Case. $DP(n, l) = 0$ for $n = 0$ and $\forall l \in [0, L_{slo}]$. This is trivially correct since there are no nodes. Also, $DP(n, l) = 0$ for all n and $l < \min_{m \in M} l_m$ because we cannot select any model without violating the latency constraint.

Induction Step. To compute $DP(n', l')$, we need the values of $DP(i, l')$, $DP(i, l' - l_m)$ for all $m \in M$, and $DP(j, l')$ as per the recurrence relation. Here, i represents the index of the nearest left sibling node on the ancestor path, where $i = n' - 1$ if the node lacks children, and j represents the index of the rightmost child, where $j = n' - 1$ if the node has children, otherwise 0. The depth-first post-order traversal guarantees both $i < n'$ and $j < n'$. Thus, by the inductive hypothesis, the values $DP(i, l')$, $DP(i, l' - l_m)$ for all $m \in M$, and $DP(j, l')$ are available and computed *correctly*.

The algorithm determines the optimal value for $DP(n', l')$ by comparing three options: using $DP(i, l' - l_m)$ if adding this node to the solution list is beneficial, using $DP(j, l')$ if the combination of child nodes is better, or retaining the previous solution $DP(i, l')$ if adding any node in the subtree rooted at this node is not beneficial. Therefore, the value for $DP(n', l')$ is computed *correctly*.

Overall, our DP-DP algorithm provides an optimal solution for the following reasons: (1) Principle of Optimality [1]: The solution is constructed by combining solutions to subproblems, whose correctness is proven above. (2) Exhaustive Search in Subproblems: The recurrence relation considers all possible models m at each node n and all possible ways to partition the latency budget between the node and its subproblems. (3) No Overlapping Subproblems: Each subproblem is computed exactly once and does not incur redundant computations.

C CASE STUDY MAP METRICS

Here are the results of additional metrics for our case study analysis. Tab. 6 shows the mAP results achieved by Uirapuru and Remix using a budget of 2000 ms; Uniform Partition using EfficientDet-D2, which is the closest Uniform Partition network approach to the budget in terms of execution time; and down-sampling using Efficientdet D6, which is the best down-sampling approach available.

Tab. 7 shows the mAP results reached by Uirapuru when the object history is updated at different moments than the evaluation and the evaluation moment itself. As it can be seen, Uirapuru reaches the highest mAP results while using the object history generated at 10:00. These results are on par with the ones seen in 4.4.2. The only difference happens at 16:00, where the result using the object history from 10:00 is

Table 6: mAP results achieved by Uirapuru and the Baselines for the different hours in our real PTZ data.

mAP	Corresponding Best		Budget 2000		Uirapuru's Difference to		
	Hour	Down-sampling	Uniform	Remix	Uirapuru	Uniform	Remix
10:00:00		18.54%	41.45%	7.34%	50.17%	8.72%	2.83%
13:00:00		12.10%	38.32%	3.46%	45.42%	7.10%	1.96%
16:00:00		13.33%	2.14%	5.33%	49.00%	6.86%	3.67%
19:00:00		12.28%	35.08%	0.67%	42.30%	7.22%	1.64%

Table 7: mAP values of Uirapuru when using the plans generated with the historical frames of different times.

mAP	Update at				
	Evaluation at	10:00	13:00	16:00	19:00
10:00		50.17%	-	-	-
13:00		45.97%	45.42%	-	-
16:00		49.03%	48.77%	49.00%	-
19:00		42.81%	42.31%	42.28%	42.30%

Table 8: mAP values achieved by Uirapuru under different lighting conditions.

mAP	Sunny	Dark	Cloudy
Uirapuru	45.42%	42.30%	47.52%
Remix	43.46%	40.67%	45.62%
Uniform	38.32%	35.08%	41.72%
Downscale	12.10%	12.28%	12.78%

now higher than the one using the history of 16:00. However, the difference is only 0.03%.

Tab. 8 shows the mAP values achieved by Uirapuru, Remix, down-sampling, and Uniform Partition under different lighting conditions. The trends here are the same as in 4.4.3.

D DATASET INSPECTION

Our scenario requires a dataset of high-resolution video frames with a high density of objects generated by a steerable camera, ground truth labels, and pre-defined/controlled camera movements. Datasets in the literature lack one or more of those requirements. The PANDA dataset [45] is the one that gets closest to fulfilling all requirements, but it lacks pre-defined/controlled camera movements like all others. We created a video dataset with movements based on the PANDA dataset [45]. Fig. 10 shows how we generated new frames that simulate Pan, Tilt, Zoom actuation, or a combination of all three. Note that the camera transformation consists of rotation and scaling (zoom) and does not contain translation (we assume the camera is mounted in a fixed position).

We also conducted a case study using a real steerable camera. The camera is in a city square, and Fig. 11 shows some of the frames captured by steerable cameras in different weather/light conditions. Each frame in the row was captured with a different actuation. Persons, cars, windows, ads, and other objects are redacted for privacy preservation.



Figure 10: Original Panda sequence frames (top row) and its generated sequences examples for Pan (second row), Tilt (third row), Zoom (third row), and a random actuation combining Pan, Tilt and Zoom (bottom row).



Figure 11: Case study frame examples at different weather/lighting conditions. Top row show frames captured during a sunny day, middle row shows frames captured at a cloudy day, and bottom row shows frames captured at night when it is dark.