PMOTHS: Efficient Multi-Object Tracking for Multi-Accelerator Mobile Systems

Abstract

Multi-object tracking (MOT) is a critical task in computer vision, essential in various fields of mobile computing such as sensor networks, smartphones and surveillance. It involves detecting and tracking multiple objects over time and making informed decisions. State-of-the-art MOT systems rely on deep neural networks for object detection, which can be computationally intensive and energy-consuming, especially in performance-limited and energy-efficient mobile computing environments. This paper addresses the challenge of improving MOT execution efficiency on heterogeneous system-on-chips (SoC) that integrate multiple accelerators, including GPUs, and domain-specific accelerators.

We introduce *PMOTHS*, a Priority-based Multi-object Tracking approach for Heterogeneous SoCs. Our method leverages a novel multi-model, multi-accelerator execution strategy to improve latency and energy consumption without compromising critical operational accuracy. By identifying the locations of high-priority and low-priority objects in the frame, *PMOTHS* can dynamically allocate computational resources to balance the detection of objects across multiple accelerators and sub-regions of the frame. This approach significantly reduces the size of input data and allows energy-efficient accelerators to be utilized. Evaluations on the MOT17 dataset demonstrate a reduction of up to 2.2x in latency, 3.8x in energy, and 2.2x in power draw, while preserving more than 95% of recall and 99% of detection accuracy, showcasing the effectiveness of *PMOTHS* in real world scenarios.

1 Introduction

Multi-object tracking (MOT) is a principle computer vision task utilized in various fields of mobile computing, including sensor networks, augmented reality, smartphones, autonomous vehicles, surveillance, and robotics [7, 12, 20, 22, 23]. It involves identifying and following multiple objects within a scene over time, which is essential to understand dynamic environments and making informed decisions. MOT systems typically consist of two main components: *detection*, which uses deep neural networks (DNNs) to identify objects in individual frames, and *tracking*, which associates these detections across frames to maintain consistent object identities [11, 33–36, 38]. In this work, we focus on real-time MOT on energy-efficient and performance-limited computing environments.

Many mobile systems rely on multi-accelerator, *i.e.*, *heterogeneous*, system-on-chips (SoC) to efficiently execute various workloads such as MOT, rendering, video analytics, facial recognition, or other DNN-based tasks [5]. In addition to CPUs and general purpose GPUs, these SoCs embed domain-specific processors such as deep learning accelerators (DLA) and programmable vision accelerators (PVA) [24]. Domain-specific accelerators are capable of running a specific set of functions in the domain much more efficiently, *i.e.*, less power spent for a unit of computation, making them highly favorable for energy- and power- constrained platforms. Figure 1 shows the breakdown of the total time spent processing a single frame with a MOT pipeline using a DNN-based object detector. The tracking algorithm is run on the CPU and accounts for only 14.5% of the frame processing; whereas the DNN model for detection, although it runs on the much faster GPU, takes the remainder of the 26 ms execution time, while consuming around 95% of all the energy spent. An important realization is that a mobile system usually has other GPU-based workloads, such as rendering, video encoding/decoding, video analytics, or facial recognition. scheduled to run in parallel or in series with the MOT pipeline [12, 16, 22]. As such, focusing on detection within a MOT pipeline has the greatest potential to improve power efficiency and reduce overall system latency.

In this paper, we explore the following question: *How could we improve MOT execution efficiency by reducing either the latency or energy spent per frame, without sacrificing critical requirements of operational accuracy?* In an attempt to find an answer, we come up with two important observations regarding MOT execution on multi-accelerator mobile SoCs: **(i)** A comparison of inference time, power draw and energy consumption of the YOLOX [11] object detection (OD) model is shown in Figure 2. We run various parameter sizes of the YOLO model on the GPU and DLA of the popular NVIDIA Orin AGX mobile platform. Using DLA, the total energy spent on inference can be almost halved at the cost of a 1.42x increase in inference latency. Importantly, the GPU and DLA

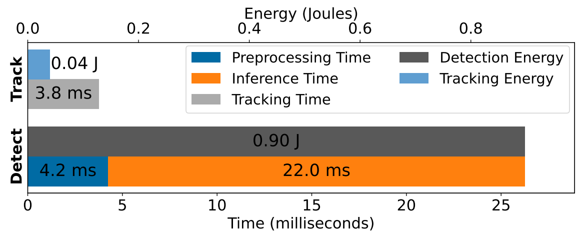

Figure 1: Comparison of the runtimes of YOLOv10 [33] M with 1280x1280 input size and ByteTrack [38] on an Orin AGX 64GB using an off-the-shelf single-threaded implementation.

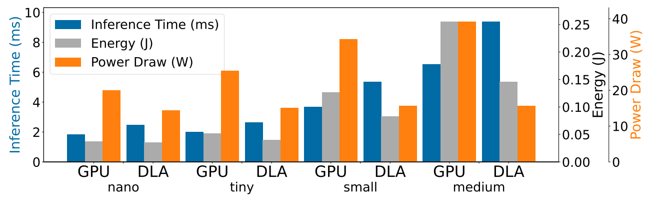

Figure 2: Comparison of GPU vs. DLA inference metrics on NVIDIA Orin AGX 64GB. Each version of YOLOX (nano, tiny, small, medium) uses the default input size as defined by the maintainers being 416, 416, 640, and 640 respectively. DLA based models use int8 precision with calibration performed using the COCO17 validation set.

can be run concurrently to enable collaborative execution and effectively parallelize DNN-based workloads. *Current MOT approaches do not consider collaborative execution, because the execution pipeline is not parallel.* Using multiple accelerators would provide performance and energy benefits if there were a method to parallelize the MOT pipeline.

(ii) While the state-of-the-art practice is to feed the entire camera input to the detection model, the portion of the input image that is large enough to perform accurate OD may be much smaller than the entire input. Moreover, *not all tracked objects may be of critical importance.* Figure 3 depicts the smallest region that contains bounding boxes of the most important objects in red, *e.g.*, cars or pedestrians, for a variety of videos in the MOT17 dataset [23]. Additionally, the regions containing objects of lesser importance, shown in blue, indicate situations where detection results matter less, such as when objects are stationary or far from the path of travel. The region containing all objects is shown in green and represents a typical region-of-interest (ROI) [3] based approach. Figure 4 shows a heatmap of the average number of objects per frame in various MOT sequences. In most scenarios, there are long sequences of frames with large empty regions which do not contain any objects. Thus, a significant amount of computational cycles is wasted by processing these regions. Processing only the relevant data from the sequence can save latency and energy. Figure 5 expands on this and shows how the occurrence of objects in real-life sequences is often cyclic, as objects move in and out of view. Notably, there are spikes in the amount of coverage when

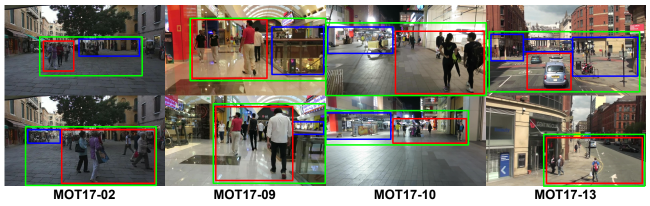

Figure 3: MOT17 scenarios and corresponding example regions of high priority (red), low priority (blue), and overall region-of-interests (green) detection regions.

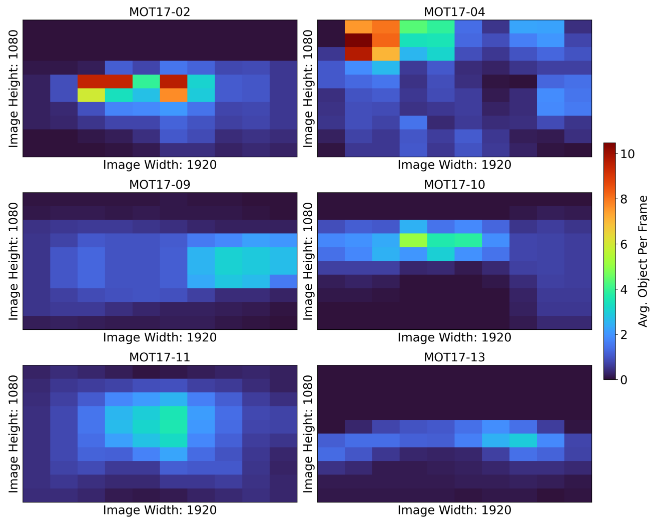

Figure 4: Average number of intersecting objects for each gridcell on a frame-by-frame basis for MOT sequences. A 10 by 10 grid a pixel size determined by the frame size is used to compute the object intersections.

new objects appear far away from the existing object cluster, demonstrating the importance of being context-aware and not focusing only on existing detections. In particular, MOT scenarios 02, 09, 10, 11, and 13 have significant portions of the overall sequence where only a subset of a frame is sufficient to detect all the objects with the same accuracy that full frame processing would provide. Meanwhile, MOT scenarios 04 and 05 (the latter is not shown) represent sequences in which the spatial information and context of the objects cannot be used to optimize performance. Most of the time, only a small portion of the input contains all the relevant information to perform detection on the critical objects of interest.

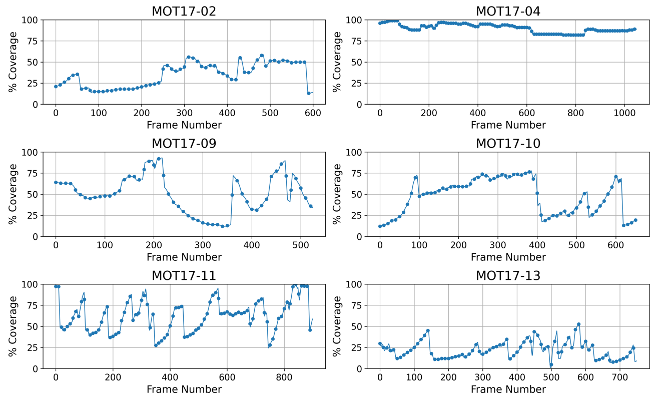

Figure 5: Percentage of each frame which is occupied by objects. The object coverage is computed by taking the bounding box of all object bounding boxes, thus this coverage will be an overestimate including empty space between objects.

By separating each frame into multiple regions, we can distribute the detection for a single frame across several accelerators, enabling parallel execution and potentially improving the overall efficiency of the system. This strategy is based on three key observations: objects within a frame are often clustered; the clusters tend to cover only a fraction of the full image; and the spatial distribution of objects can change rapidly as new objects enter or leave the scene. However, several challenges arise with this approach.

C1: Determining how to efficiently divide frames into regions while minimizing computational overhead.
C2: Identifying when new objects appear outside the established regions.
C3: Estimating current DNN accuracy based only on the limited available context.
C4: Managing multi and concurrent DNN execution across multiple accelerators without additional overhead.

Based on these observations and challenges we propose a new approach to improve the efficiency of MOT execution.

We introduce *PMOTHS*, a Priority-based Multi-object Tracking approach for Heterogeneous mobile SoCs. We devise a novel, multi-model, multi-accelerator execution methodology that significantly improves the latency and energy consumption of OD in MOT pipelines. *PMOTHS* achieves this by treating user-defined high-priority (HP) and low-priority (LP) classes of objects separately. During runtime, we process the region of the frame containing all HP objects using the original detection model (or an equivalent) on a reduced input to improve latency. For the remainder of the frame, we employ faster and/or lower-accuracy models, which can run in parallel on another accelerator. The detection accuracy for LP objects is lower-bounded by a user specified knob.

The contributions of our work are as follows:

- We propose a novel MOT approach that improves latency, energy-consumption, and power-draw in mobile platforms by reducing the size of the input data without sacrificing the detection accuracy for high-priority objects.
- We efficiently split captured frames into high- and low-priority regions according to a set of user defined class priorities.
- We utilize energy-efficient accelerators to detect low-priority objects collaboratively while meeting a user-determined detection accuracy threshold.
- We present a unique methodology for merging the output of multiple object detection models and derive the current context and assessment of the model performance.
- We integrate a very-low-overhead scheduler capable of handling the large state-space of possible model selections and accelerator mappings.
- We evaluate the efficiency of our approach on the industry-standard MOT17 dataset and show a reduction of up to **2.2x** in latency, **3.8x** in energy, and **2.2x** in power draw, while operating at a higher recall and detection accuracy than 95% and 99%, respectively.

2 Related Work

Multi-object tracking (MOT): State-of-the-art MOT solutions [9, 17, 25, 26, 30, 35, 36, 38] often utilize a two-stage detection process followed by tracking, with a single-pass DNN using a GPU as the primary detector. The tracking portion can either consist of traditional CPU-based implementations to label individual detections with unique IDs [17, 26] or include additional DNN inferences to improve tracking accuracy [9, 30]. Recently, ByteTrack [38] demonstrates that optimizing the underlying detection algorithm can increase tracking accuracy. However, its detection model relies on a heavyweight DNN, specifically a custom YOLOX model with an input size of 1440×800. All such MOT works [9, 17, 25, 26, 30, 38] do not consider the role of dynamically allocating detection resources and how they can be augmented, instead treating the detection stage as unchangeable during runtime. *PMOTHS instead focuses on improving detection latency while utilizing existing tracking algorithms.*

Concurrent DNN execution: Several studies [4, 13, 27, 32] focus on optimizing the concurrent execution of DNNs. Hax-CoNN [4] optimizes the throughput of multiple DNNs in a layer-by-layer fashion across multiple accelerators while accounting for the shared-memory slowdown. Although this method increases throughput, the layer-by-layer scheduling process can be easily interrupted on systems running many tasks, such as mobile devices. CARin [27] optimizes service-level objectives in multi-DNN workloads. Similar to Hax-CoNN [4], CARin [27] considers multiple accelerators, but instead of utilizing fine-grained techniques to optimize throughput, it focuses on optimizing high-level objectives such as fairness and system-level throughput. While Hax-CoNN [4] focuses on fine-grained optimizations, CARin [27] targets multi-application level objectives, which may introduce more overhead than required to enable concurrent utilization within a single application. *PMOTHS considers concurrent execution, but, contrary to other methods, does not require fine-grained modeling or the overhead of considering arbitrary workloads throughout the system.*

Scheduling for latency, energy, and accuracy: Scheduling DNN workloads to improve latency, energy efficiency, or accuracy has been an open research area for more than a decade [1, 2, 5, 10, 19, 28]. BigLittle [28] proposes using either a large or small classifier, running the small classifier on every frame but activating the large classifier based on a confidence score threshold to save energy and minimize latency. This approach lacks fine-tuned flexibility in selecting which models to use. Additionally, in continuous scenarios, it may incur high penalties from repeatedly performing two inferences. NestDNN [10] focuses on splitting a single DNN model into multiple subgraphs and pruning such graphs to balance accuracy, latency, and energy trade-offs. This approach requires significant offline resources, demands that users have fine-grained knowledge of the detection models, and assumes that training data is accessible to the user. *PMOTHS, instead of focusing on improving concurrent execution or latency through advanced low-level methodologies, demonstrates that system-level scheduling, scalable input sizes, and input partitioning are sufficient to manage multi-DNN inference schemes while accounting for additional computational requirements in real-world scenarios.*

Context-aware DNN Inference: Only a small set of studies [6, 29] offer context-aware approaches to enhance the use of DNN models. CACTUS [29] splits the classification into many smaller problems by creating a set of micro-classifiers, where each classifier handles a subset of classes. The inference time is minimized by identifying the context of the current image and performing only a few micro-classifications. However, this method introduces high complexity during context changes and requires custom training schemes. Furthermore, since CACTUS [29] is designed for classification models, context identification is strictly tied to the entire output and image, meaning that when context changes occur, the methodology must run additional models. SHIFT [6] focuses on switching between different DNNs at runtime based on the confidence score output, as well as the differences between frames and detected bounding boxes. SHIFT switches between DNN families (e.g., YOLO, SSD) to exploit non-monotonic latency/accuracy trade-offs. However, it is limited to single-class and single-object detection scenarios, which means it does not generalize to other datasets. *PMOTHS reduces runtime latency by scaling the input size based on the spatial context of object detections and can consider more than a single-class / single-object scenario during context identification.*

Region-of-Interest for Object Detection: Methodologies using a system-level region-of-interest (ROI) to constrain OD typically rely on manual or domain-specific automated computer vision algorithms [8, 37]. In FlexPatch [37], an aggregated image of known regions with detections and regions that may contain detections is created to reduce communication time to the server running OD. The algorithm for determining candidate regions is computationally expensive and limits the usability of this method in applications without offloading. In Dhonde [8], a YOLO-based DNN is used to assess what items a customer is purchasing at a self-checkout kiosk. An ROI region is determined based on additional DNN inferences to reduce the amount of data passed to the primary detection model, enabling more accurate inference. However, requiring additional DNN inferences significantly increases overall latency, restricting applications to those with offloading capability or substantial computational resources. *PMOTHS utilizes a generic automatic ROI-finding method that does not require additional computation, operates solely on existing OD outputs, and does not require additional runtime input, unlike previous methods.*

Related Work / **Feature**	HaxCoNN [4]	CARin [27]	BigLittle [28]	NestDNN [10]	CACTUS [29]	SHIFT [6]	FlexPatch [37]	***PMOTHS***
Context Aware	✗	✗	✓	✗	✓	✓	✓	✓
Accuracy Predictions	✗	✗	✗	✗	✗	✓	✗	✓
Concurrent DNNs	✓	✓	✗	✗	✗	✗	✗	✓
Non-GPU Accelerators	✓	✓	✗	✗	✗	✓	✗	✓
Object Detection	✗	✗	✗	✗	✗	✓	✓	✓
No Additional Training	✓	✓	✓	✗	✗	✓	✓	✓
ROI Focusing	✗	✗	✗	✗	✗	✗	✓	✓
Low Overhead	✓	✗	✗	✓	✗	✓	✗	✓

Table 1: Comparison of the features offered by related works.

Table 1 provides an overview of the features offered by the most relevant works and *PMOTHS*: (i) the capability to detect contextual information embedded in the input data stream, (ii) the ability to predict accuracy as a metric in their methodology, (iii) concurrent execution of DNNs, (iv) utilization of domain-specific accelerators, (v) targeting object detection workloads, (vi) no requirement for additional training, pruning, or fine-tuning to achieve performance, (vii) the ability to target specific data within an input stream by utilizing ROIs, and (viii) a focus on low runtime overhead for computation without offloading. *Safe and efficient operation of MOT pipelines with an optimized detection stage requires a holistic consideration of all these features. To the best of our knowledge, only PMOTHS achieves this goal.*

3 Proposed Approach: *PMOTHS*

PMOTHS relies on four main components to enable a priority-based MOT scheme and save latency, power, and energy without losing accuracy on the HP objects: (i) *High and low priority splitting algorithm* which enables *PMOTHS* to split a single frame into parallel regions with varying accuracy goals, (ii) *the frame packing algorithm* which enables low priority objects to be represented with less data, (iii) *accuracy and latency models* to predict which model to use for LP and HP detection, and (iv) *the runtime scheduler* which dynamically maps models to accelerators (*i.e.*, GPU and DLA) so that the accuracy constraints are met while minimizing overall runtime, *i.e.*, frame latency. An overview of the data flow and components is given in Figure 6.

3.1 Identification of High Priority Regions

High priority regions are determined based on a set of user defined priority class labels. For a set of detections on the entire image, the detections with class labels in the priority set are isolated. Then, the bounding box encasing all isolated

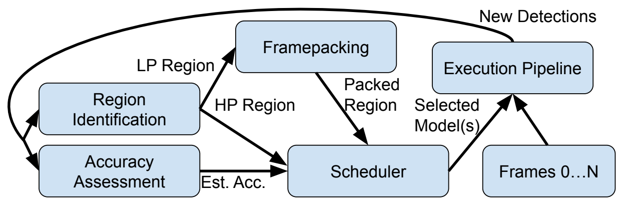

Figure 6: Overview of the *PMOTHS* setup and how each component connects. Region identification, framepacking, accuracy/latency modeling, and scheduling are outlined in Section 3.1, Section 3.2, Section 3.3, and Section 3.4 respectively.

detections is generated. This new bounding box is used as the *HP region* for the next frame. The remainder of the frame is the LP region and is processed further by the frame packing algorithm described next in Section 3.2. Since MOT is designed for continuous frame sequences, there is an implicit assumption that objects will not significantly change position frame-to-frame. This assumption is used in MOT algorithms such as SORT, DeepSORT, and ByteTrack [35, 36, 38]. Thus, when an HP region is identified in the current frame, it will still be present in the next frame. A small amount of padding is added to account for object movement between frames to ensure objects do not get cut off. This is a standard practice for methodologies that utilize regions of interest (ROIs) in an OD context [8, 37]. By assessing the priority based on a set of user-defined class labels, the relevant HP region can be determined with almost no overhead and requires only a single iteration over the set of detections, addressing challenge **C1**.

It is still possible that the LP region will contain objects of the HP class in one of these conditions: 1) the HP/LP model cannot detect the object, 2) the HP model loses detection of the object and the object moves to LP region, 3) the movement of camera results in new objects appearing in the frames, and 4) objects move into the scene when camera is static. In such cases, the LP model will detect objects with HP classes, and the HP area will be redefined to include them.

All objects in the HP region, regardless of their class, will be detected by the HP model. Therefore, corresponding pixels in the HP region should be excluded from the LP model's inputs to save computational resources. A naive solution to this problem is to black out the HP region and then perform inference on the remaining regions. By this way, duplicate detections do not need to be filtered on the CPU side post inference, providing a potential optimization. However, this method wastes computation since the LP model will still process the blacked-out region. A more efficient solution is to create a new image for the LP region which has been reorganized to completely exclude the HP region. This method would have three benefits; i) there will be no wasted computation, ii) the data size will be smaller, allowing for a smaller input size or less sub-sampling during inference, and iii) we will be able to use the DLA to parallelize computation and save energy.

In addition to reorganizing the data in the rest of the frame, we could further improve the execution of the LP model. As observed in Figures 4 and 5, most of a frame can be devoid of objects. For example, in the MOT17-13 scenario, the maximum coverage observed is 60%, while the average coverage is significantly lower. Since most of each frame is empty, excluding regions without objects can allow reduced model input size and lower latency.

Based on the insights detailed above, we introduce a novel frame packing algorithm to efficiently process LP regions.

3.2 Frame Packing for Low Priority Regions

Our frame packing algorithm is based on simulated annealing and 2D bin packing: First, regions that are not likely to contain detectable objects are removed, reducing the overall data size. Then, regions are bin-packed into a new image, minimizing the difference between the height and width, which in turn minimizes distortion and resolution loss during preprocessing for the OD DNN.

Formulation: To reduce the computational complexity of determining the importance of regions of the input stream, a predetermined grid is maintained, where each cell maps to a fixed square region of the frame. Each grid cell tracks the number of detections within itself. Once detections are generated for a frame, each cell of the grid has its counter incremented by the number of objects that intersect that cell. All cells that did not have any intersecting detections have their counter decremented. Using this simple counting methodology, the relevance of each cell can be determined. The simulated annealing algorithm assigns a probability to include the cell using the current frame number and the detection counter. The probability is defined in Equation 1,

$$P_{explore} = \min\left\{1, \max\left\{p_{\min}, e^{-\alpha c} + \min\left\{1, \frac{d}{c+1}\right\}\right\}\right\} \quad (1)$$

where c is the global frame counter, d is current detection counter for the cell, α is the cooling rate, and p_{min} is the minimum probability. The detection tracking counter and the frames captured since initialization provide enough contextual information to accurately identify regions that can be excluded from the LP model inference. Importantly, it is possible that no cells are selected by simulated annealing, meaning that no additional inference pass is required for that frame and only the HP region gets processed.

Once a set of grid cells has been generated by simulated annealing, the corresponding image regions represented by these cells are packed into a new image. Since the cells are on a structured grid pattern, some of the cells can be grouped together to form connected components. By forming groups of cells, more spatial contextual information can be preserved. This prevents objects that span multiple cells from

Algorithm 1 Frame Packing via Shelf Bin Packing

```
Input: Image $I$, Groups $G$
Output: Packed Image $P$
 1: if $G = \emptyset$ then
 2:     return Empty Image
 3: end if
 4: $G \leftarrow \text{sort}(G)$                                  ▷ Sort descending by size
 5: $ts \leftarrow \text{ceil}(\text{sqrt}(\text{sum}(\text{size}(g) \forall g \in G)))$   ▷ Target size
 6: $S \leftarrow \emptyset$                                     ▷ Set of shelves
 7: for all $g \in G$ do
 8:     for all $s \in S$ do
 9:         if $g_{\text{width}} \neq s_{\text{width}} \vee g_{\text{height}} + s_{\text{height}} > ts$ then
10:             continue
11:         end if
12:         $s \leftarrow s \cap \{g\}$
13:         break
14:     end for
15:     $S \leftarrow S \cup \{g\}$
16: end for
17: $h \leftarrow \max(s_{\text{height}} \forall s \in S)$
18: $w \leftarrow \text{sum}(s_{\text{width}} \forall s \in S)$
19: $P = image[h, w]$
20: $frontier \leftarrow 0$
21: for all $s, s_w, s_h \in S$ do
22:     for all $i \in s_h$ do
23:         for all $j \in s_w$ do
24:             $\text{bbox} \leftarrow s[i \times s_w + j]_{\text{bbox}}$          ▷ Get bbox from cell
25:             $\text{bbox}_{\text{new}} \leftarrow \text{getBbox}(i, j)$        ▷ Get new bbox
26:             $P[\text{bbox}_{\text{new}}] \leftarrow I[\text{bbox}]$             ▷ Update image
27:         end for
28:     end for
29:     $frontier \leftarrow frontier + s_w$
30: end for
31: return $P$
```

being split into smaller pieces and not being detected. To make re-packing of the groups easier, only connected component groups which are dense, *i.e.*, are fully filled rectangles, are considered. Since groups of cells will have many different shapes, forming a new image from these groups is NP-Complete since it can be represented as 2D bin packing problem [15]. As such, we use the shelf-based 2D bin packing heuristic algorithm with a minor modification to attempt to pack the regions into a square image. A square input is important, as the standard configurations of object detection models such as YOLO [11, 33] are trained on square images.

Algorithm: The procedure for the bin-packing algorithm is presented in Algorithm 1. The input image I is the current frame from the sequence and the groups G is the set of all connected component groups or standalone cells. The algorithm works as follows: *(1-3):* If there are no groups, return an empty image. *(4):* Sort the groups by descending size to pack the largest groups first. *(5):* Define the target size of the ideal square image. *(6):* Define the set of shelves.

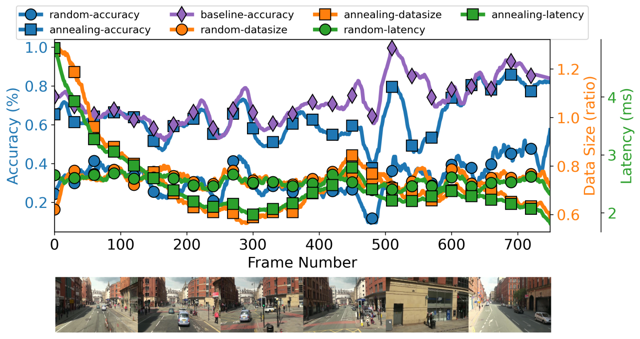

Figure 7: Analysis of frame packing reduction ratio, latency, and the maximum recoverable accuracy using the baseline YOLOv10 M 1280x1280 DNN on the MOT17-13 sequence.

(7-16): For all groups, attempt to greedily place the group on the first shelf with the same width and available height, otherwise create a new shelf. *(17-19):* Create the new image based on the shelf sizes. *(20-31):* Iterate over each shelf, then iterate over each cell –all exist since width matching and dense groups were required–, and copy each cell from I into P to fill in the new image.

Once the new LP region image is formed, it is passed to the LP model for detection. To convert detections from the packed LP image back to the real frame coordinates, the bin-packing algorithm saves a 2D set of offsets. Then, each bounding box is converted to cell coordinates in the packed image. These cell coordinates are used to get the offset to transform the bounding box to the original frame.

Verification: In order to verify that the frame packing methodology is able to keep the relevant regions of the image within the packed representation, we evaluate its recall against the baseline model defined in Section 4. The experiment is set up as follows: 1) the entire frame is given as the input frame to be packed instead of just a subset; 2) the detections used to update the grid are the detections of the baseline model on the entire frame; and 3) the recall of the baseline model on the packed image is the displayed accuracy. As can be observed in Figure 7, the baseline model can recover most of the detections while removing up to 40% of the image. The minimum probability of the annealing function was set to 10% with the grid size to be equal to one tenth of the larger dimension of the images. A random sampling method was also created that will sample 75% of the image to be packed. Both methods utilize the bin packing methodology to create the new packed image. As can be observed, the random methodology does not recover the same accuracy as the annealing based methodology, showcasing sampling cells with simulated annealing can adapt to the scenario sufficiently after a small warm-up period. A visual example of the frame-packing process is presented in Figure 8. As shown, within 300 frames (or 10 seconds), the simulated annealing and bin-packing algorithms work together to reduce the LP data to

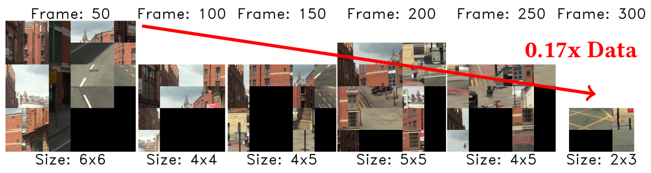

Figure 8: Comparison of the packed grid cells after blocks of frames have been processed via simulated annealing. Detection results are from the baseline YOLOv10 M 1280x1280 DNN running on the entire image.

be processed to just 7% of the overall image resolution (10% when accounting for inefficiently packed space). Having such small input sizes significantly increases the optimization capability of a parallel inference for LP.

Using fast heuristics to create the LP region for detection, *PMOTHS* is able to effectively solve challenge **C2**.

3.3 Predicting Model Accuracy and Latency

To effectively schedule object detection DNNs at runtime, an estimation of their accuracy and latency needs to be modeled beforehand so that quick scheduling decisions could be made. Ideally, such a model should be built using ground truths for the best representation of accuracy. However, ground truth data will not be available for the frames encountered during runtime, and the closest metric reported by the DNN execution is the confidence scores, which do not directly represent the accuracy of the model. To quantify accuracy, we use the F1 score, the geometric mean between precision and recall, as the accuracy metric [31]. The higher the F1 score, the more precise the model and the more objects the model can recall. During runtime, the only metric available to quantify accuracy is the confidence score. However, since the confidence score primarily reflects the certainty of positive detections, it is an incomplete metric. Consider the case where a model generates high confidence values but misses many objects. In this scenario, the recall will suffer significantly, but the precision of the model remains high. This creates a non-linear relationship which will not be accurately predicted at runtime. Thus, the F1 score is used to model the hidden relationship between confidence score, precision, and recall.

In order to generate runtime F1 score estimates, we develop a 2D setup of linear estimators. For each model, an estimator is trained to predict the F1 score of itself and every other model based on the mean confidence value of the set of detections on a given frame. To train the estimators, we use the validation set of the dataset on which the object detection DNN was trained. Importantly, the difficulty of the datasets may be different, *i.e.*, the F1 score distribution can change depending on whether the DNN is running on COCO17 [18] images (with which YOLO was trained) or MOT17 [23] frames (which we use in our experiments). To account for this, a scaling factor is calculated based on the

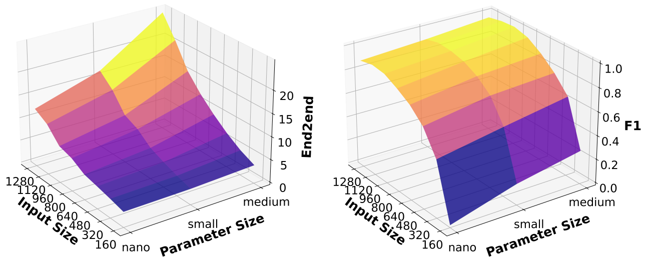

Figure 9: On the left and right, respectively: (a) YOLOv10 end-to-end inference time and (b) normalized F1 score characterization for various model variations.

median F1 score that will be used along with the user-defined knobs in the runtime scheduler, described in Section 3.4.

A visualization of the characterizations for end-to-end inference time and the F1 score can be found in Figure 9. As expected, increasing computational resources results in higher F1 scores. We observe that reductions in parameter sizes can be offset by proportionally increasing input sizes, effectively recovering performance. These relationships are captured by the 2D linear estimator setup, enabling dynamic runtime adjustments to balance input and parameter sizes.

Using a lightweight set of linear estimators trained to predict the F1 score of all models from the confidence score of a single model, *PMOTHS* is able to predict accuracy without adding significant latency, hence solving challenge **C3**.

3.4 Runtime Scheduler

The runtime scheduler determines which models to execute for the HP and LP regions. HP regions are always executed on the GPU, as they are the critical execution path. Based on the estimated latency, the scheduler decides whether to execute an LP region on the DLA (in parallel) or on the GPU (in series), while maintaining a user defined accuracy threshold for the LP region. The scheduler accomplishes this using a combination of offline and online approaches.

Offline: We characterize all possible GPU(HP)+GPU(LP) and GPU(HP)+DLA(LP) combinations based on the set of all available OD models. Schedules will be evaluated based on runtime characteristics including pre-processing, inference, post-processing, and end-to-end latency.

Online: The scheduler will first generate the HP and LP regions for each frame using the methods described in Section 3.1. Then, the LP regions will be packed into a new image based on the current detection context outlined in Section 3.2. Once the HP and LP regions are ready to be detected, the scheduler uses the routine described in Algorithm 2 to choose a new schedule of OD models.

The scheduler works as follows: *(1-3):* If there are no previous detections or no HP region, do not change the schedule. *(4):* Compute the bounding box for the HP region. *(5-17):* Predict current F1 scores based on prior detections of the HP models, then generate a set of possible new candidates for the

Algorithm 2 Runtime Scheduler

```
Input: Previous Detections $D$, LP accuracy goal $\gamma_{lp}$, Possible Schedules $S$, HP Bounding box $B$
Output: New schedule configuration $C \in S$
 1: if $\mathcal{D} = \emptyset$ or $B$ is undefined then
 2:     return $C_{\text{baseline}}$                              ▷ Exit without changes
 3: end if
 4: Compute HP bounding box area $A_{\text{hp}} \leftarrow \text{area}(B)$
 5: Compute HP Confidence $c_{\text{hp}} \leftarrow$ mean confidence in $D_{\text{hp}}$
 6: Predict HP F1 $P_{hp} \leftarrow \text{predict}(D, c_{hp})$
 7: Get baseline scaling ratio $s_b \leftarrow b_m / A_{hp}$
 8: Get median bounding box size $s_{\text{bbox}} \leftarrow \text{medianSize}(D)$
 9: Compute scale $b_s \leftarrow 1.0 - (s_{\text{bbox}} / \max(\text{img}_{\text{height}}, \text{img}_{\text{width}}))$
10: Initialize $\mathcal{M}_{\text{hp}}^{\text{valid}} \leftarrow \emptyset$
11: for all $m \in \mathcal{M}_{\text{hp}}$ do
12:     Retrieve model area $A_m \leftarrow$ input area of $m$
13:     Compute scaling ratio $s_m \leftarrow A_m / A_{\text{hp}}$
14:     if $s_m \leq s_b * b_s$ & $P_{\text{hp}}[m] \geq P_{\text{hp}}[\text{baseline}]$ then
15:         $\mathcal{M}_{\text{hp}}^{\text{valid}} \leftarrow \mathcal{M}_{\text{hp}}^{\text{valid}} \cup \{m\}$
16:     end if
17: end for
18: if $\mathcal{M}_{\text{hp}}^{\text{valid}} \neq \emptyset$ then
19:     $m_{\text{hp}}^{\text{new}} \leftarrow \min_{m \in \mathcal{M}_{\text{hp}}^{\text{valid}}} \text{latency}(m)$
20: else
21:     return $C_{baseline}$
22: end if
23: $\mathcal{S}_{\text{filtered}} \leftarrow \{s \in S : \text{HP model in } s = m_{\text{hp}}^{\text{new}}\}$
24: Compute LP Confidence $c_{\text{lp}} \leftarrow$ mean confidence in $D_{\text{lp}}$
25: Predict LP F1 $P_{\text{lp}} \leftarrow \text{predict}(D, c_{\text{lp}})$
26: Initialize $\mathcal{S}_{\text{final}} \leftarrow \emptyset$
27: for all $s \in \mathcal{S}_{\text{filtered}}$ do
28:     Retrieve LP model $m_{\text{lp}} \leftarrow$ LP model in $s$
29:     if $P_{\text{lp}}[m_{\text{lp}}] \geq \gamma_{lp}$ & $\text{latency}(m_{\text{lp}}) + \text{latency}(m_{\text{hp}}) < \text{latency}(m_{\text{baseline}})$ then
30:         $\mathcal{S}_{\text{final}} \leftarrow \mathcal{S}_{\text{final}} \cup \{s\}$
31:     end if
32: end for
33: if $\mathcal{S}_{\text{final}} = \emptyset$ then
34:     return $C_{\text{baseline}}$
35: else
36:     $s_{\text{new}} \leftarrow \min_{s \in \mathcal{S}_{\text{final}}} \text{latency}(s)$
37: end if
38: return $s_{\text{new}}$
```

HP model: (i) that are estimated to maintain the accuracy of the baseline model and (ii) that can perform inference on the HP region without scaling the data down more than the baseline model multiplied by a scaling factor s_b. s_b is the ratio of the median bounding box dimension to the maximum image dimension and is used to optimize edge cases where objects are large relative to total image size. *(18-22):* If there are models that can run the HP region with scaling down more than baseline, use the fastest one, otherwise use the most accurate of all models. *(23):* Enumerate possible schedules using the model determined for the HP region. *(24-25):* Predict current F1 scores based on previous detections, *i.e.*, recalls, from the LP models, *(26-32):* Filter out the schedules having an LP model that does not meet the accuracy threshold. *(33-38):* If there are schedules which meet the LP accuracy threshold, use the fastest one, otherwise use the most accurate schedule.

Once a desired schedule is found, the corresponding detection algorithms are executed on designated accelerators. The locations of the detected objects are then fed into the ByteTrack [38] tracking algorithm. Please note that the scheduler falls back to the baseline YOLO model for the entire frame if it realizes that the HP-LP approach would not provide latency benefits. By avoiding fine-grained optimizations and using heuristics tuned to the MOT problem, *PMOTHS* is able to effectively consider concurrent schedules and model swaps at runtime with low overhead, solving challenge **C4**.

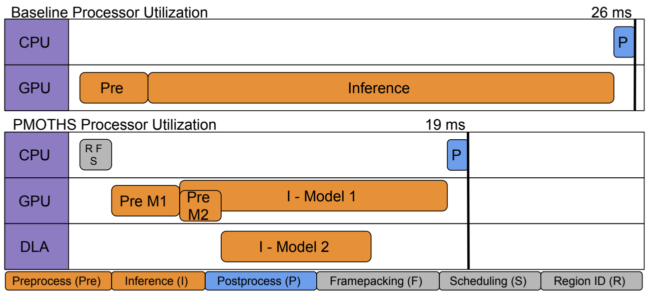

Figure 10: Overview of the processor utilization of the baseline object detection DNN pipeline and the proposed *PMOTHS* computational pipeline. Computational blocks with a hardware acceleration requirement are shown in orange, CPU-based computation is shown in grey, and additional computational for *PMOTHS* is shown in blue.

3.5 Summary

The final integration of the *PMOTHS* methodology is summarized in Figure 6. From this overview figure, the image is first analyzed to identify the HP and LP regions based on the HP object classes defined by the user as in Section 3.1. The HP region is used directly, while the LP region is processed using the frame-packing algorithms outlined in Section 3.2. Once the frame packing algorithms have processed the LP region data, if a packed image representation exists, it is passed to the scheduler along with the existing HP image. The scheduler then generates a set of accuracy estimates according to the linear estimation methodology in Section 3.3. The scheduler, described in Section 3.4, optimizes the latency while adhering to the input size and LP accuracy constraints.

Figure 10 shows an example of how the proposed *PMOTHS* OD pipeline redistributes the computational load across available hardware resources. While the baseline approach processes the entire frame with a single large pass, our approach leverages multiple accelerators (GPU for HP and DLA for LP). The example illustrates that, even though additional computation is introduced for region identification, frame packing, and scheduling, the additional overhead is far less than the saved latency, resulting in a lower overall latency. **By splitting frames into multiple detection passes, the utilization of the system can be improved, allowing for lower latency and energy usage due to the use of DLAs.**

4 Experimental Setup

Object detection models: In our implementation, we use various parameterizations of the YOLOv10 [33] and YOLO-X [11] models. YOLOv10 is the state-of-the-art model based on the original YOLO single-pass object detection methodology. We select YOLOv10 version due to its low inference time and since it does not require non-max-suppression to be run on the outputs. However, v10 is not compatible with NVIDIA DLA since the DLA does not support many of the activation functions and layers used in the modern architecture. As such, YOLOX models are used onboard the DLA since they use simpler layers, which are mostly supported. Non-max-suppression layers and transpose operations are still needed for YOLOX and these layers are run on the GPU since they are not supported on DLA. The GPU + DLA concurrent inference is implemented by using separate CUDA streams for each model and running them asynchronously. We characterize performance by measuring the host-side latency for each stream per the details given in Section 3.4.

On the GPU, YOLOv10 *nano*, *small*, and *medium* are used with possible input sizes between 160 and 1280 with increments of 160, resulting in 24 possible combinations. These models are quantized to float-16 precision in order to achieve state-of-the-art inference times. On the DLA, YOLOX small and medium are used with input sizes 320, 480 and 480, 640 respectively. All possible input sizes can not be utilized at once, since the DLA has a hard limit on the number of compiled models which can be loaded in memory at once. Since each YOLOX instance still requires running a small amount of layers at the beginning and end of inference on the GPU, each model contains multiple compiled sub-engines. As such, a limit of four possible YOLOX instances can be utilized at once. An overview of all YOLOv10 and YOLOX models we use is presented in Table 5 in Appendix A. All metrics are measured on the MOT17 [23] dataset and averaged over all sequences.

The chosen input size and parameter count pairs represent a distribution of latencies allowing a range of latency/accuracy tradeoffs to be targeted with DLA based schedules. The DLA models are quantized to integer-8 precision using the COCO17 validation images because the DLA is optimized for integer-8 computation only. The accuracy changes are modeled according to section 3.3.

For tracking, we utilized ByteTrack [38], which is an industry standard tracking solution which does not require additional DNN inferences to produce good tracking results.
Datasets: We use the MOT17 dataset [23] to evaluate our methodology. This data set was chosen because it contains a variety of scenarios that encapsulate driving, navigating through pedestrians, and static camera setups. It also has a high variation in the density, size, and location of objects that must be detected and tracked.
Hardware: The mobile device on which we evaluate *PMOTHS* is an NVIDIA Orin AGX 64GB development kit (Orin), a popular multi-accelarator platform used in mobile, edge, and autonomous computing. The Orin is equipped with 8 Arm Cortex-A78AE CPU cores, an Ampere generation GPU with 2048 CUDA cores and 64 tensor cores, a pair of second-generation NVIDIA DLA cores, a PVA, and 64GB of LPDDR5 shared memory. We utilize only a single DLA core at a time, due to the restrictions deployed by the vendor runtime.
Evaluation setup: We establish the baseline to use a single YOLOv10 medium model for object detection with an input size of 1280x1280. This configuration achieves the maximum possible accuracy under 30ms *end-to-end latency*, which includes data transfer, pre-processing, inference, and post-processing of the output data. Post-processing accounts for less than 1% of end-to-end latency, and it is the same across all DNNs. To allow for larger image sizes on the Orin, we implement a custom CUDA kernel to pre-process the frames, avoiding any CPU bottlenecks. *The ground-truth high priority* regions are the regions determined from the detection output of the baseline model. The accuracy, reported as the recall, is computed based on the annotations provided by MOT17.
Comparisons: We compare *PMOTHS* with the baseline model (*i.e.*, single YOLOv10), and our adaptations of BigLittle [28] and SHIFT [6] (see Section 2 for details). We select these two studies because they both aim to achieve energy-efficient object detection on compute-limited systems, hence they are the closest state-of-the-art approaches whose implementations are available.

BigLittle [28] is adapted from the initial classification task, by using the mean confidence score for a given image to act as the differentiating factor. The confidence threshold to determine when to use the larger model is found using the COCO17 validation dataset [18]. The two DNNs we use for this approach are the baseline YOLOv10 medium model with 1280x1280 input size and YOLOv10 nano with 640x640.

SHIFT [6] is adapted from the single-class single-object method, considering the aggregate confidence values when constructing their accuracy estimation method. As such, for any given frame, instead of using the top confidence score of the desired class, the mean confidence score of all detections is utilized. COCO17 validation data [18] is again used to generate the offline characteristics required for its runtime.

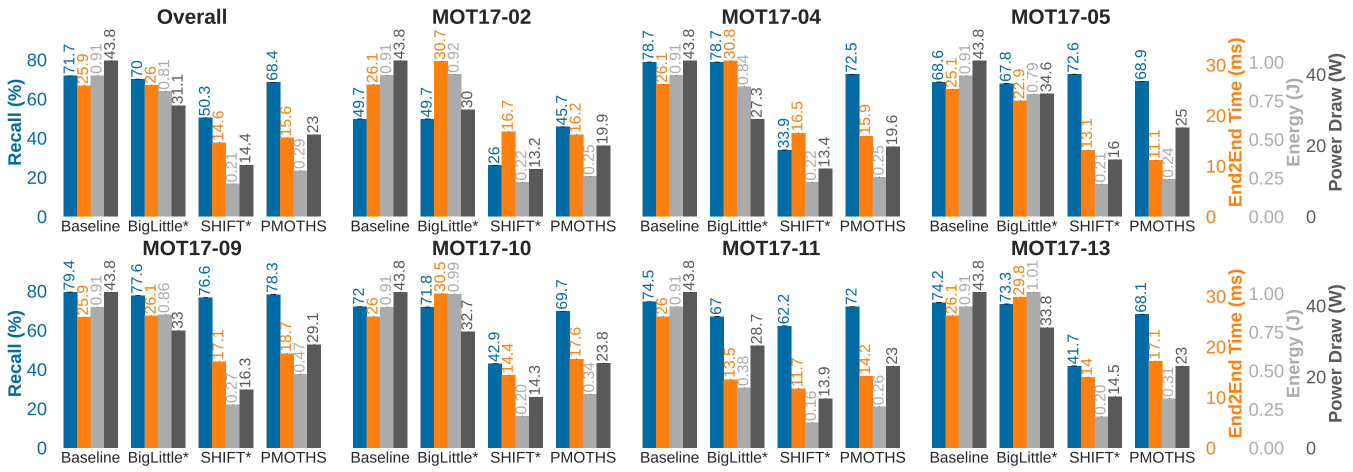

Figure 11: Comparison of baseline YOLOv10 [33] and BigLittle [28] with *PMOTHS*. The low priority accuracy knob was set to 0.5 for all scenarios and the priority detection classes used were people and vehicles with recall computed using those class labels.

Since both implementations require modification, we will label them as BigLittle* and SHIFT* in the results in Section 5.

5 Results

The primary results on the MOT17[23] dataset are shown in Figure 11. We report accuracy (*i.e.*, *Recall*), latency (*i.e.*, *End2End Time*), total *Energy* spent and average *Power Draw* for each MOT17-* scenario individually and also present an *Overall* corresponding to the mean of the values in all scenarios. The reported end-to-end latencies include all overheads such as scheduling and preprocessing. Recall corresponds to the portion of ground truth objects detected, important for understanding the capability of each method [31].

5.1 Scenes with Single-class HP Objects

We first examine the scenarios in which only a single class of detection, *i.e.*, *person*, is present.

MOT17-02 is a static camera scenario in which people are moving across a large plaza square. The people are mainly small and occupy a horizontal strip of the scene, see Figure 4. SHIFT [6] conserves computational resources, but suffers a severe degradation in recall. Since confidence values are high, it continuously lowers the OD model and ends up losing many objects. BigLittle [28] spends more resources, since the single confidence score threshold is consistently below the value deemed acceptable for shortcut detection. Thus, it ends up running both DNNs for many frames. We observe that *PMOTHS* is able to opportunistically schedule detection models which use fewer resources and achieve the same accuracy. This is because *PMOTHS* is able to conserve resources by focusing on a smaller subset of each frame. In particular, *PMOTHS* uses more energy and power consumption within the same latency envelope, showcasing how the utilization of additional inferences and parallel hardware is key to improving accuracy.

MOT17-09 is another static camera scene, with people walking across a pedestrian street area. The frame coverage of people varies between fully covered and half covered, alternating. In this scene, the primary challenge is to recover detection on the entire scene once the HP region is smaller. As seen in Figure 4, people are mainly clustered in the middle right portion, while Figure 5 shows how people move throughout the sequence. SHIFT [6] is capable of maintaining detection recall while achieving speedup compared to baseline, as the majority of objects are large and the use of inferior OD models does not significantly affect recall. BigLittle [28] sees a slight degradation in accuracy and latency since it spends extra resources running a small inference for every frame, and there were not enough small detection frames to make up for the latency. As observed in Figure 11, *PMOTHS* is able to achieve baseline accuracy while saving latency / energy / power consumption, but not as significantly as other scenes. This is because there are few exploitable subsequences where HP and LP splits are consistent.

MOT17-11 is a scenario with a moving camera with people occupying a wide variety of areas inside of a shopping mall. The HP region changes frequently, meaning that LP detection must adequately find objects moving in the scene. Figure 4 shows that people occupy primarily a single region, but Figure 5 shows the cyclic nature of the area considered as people traverse the camera path. In this scenario, we observe that BigLittle [28] is able to make gains in all metrics due to a clear correlation in confidence score and accuracy. In particular, SHIFT [6] makes similar improvements in overall performance since it also changes behavior based on runtime confidence scores. *PMOTHS* maintains the accuracy while achieving savings relative to the baseline model. The savings are not as significant as BigLittle [28] or SHIFT [6] because

PMOTHS mainly exploits the spatial characteristics of the detection boxes instead of the confidence values.

5.2 Scenes with Multi-class HP Objects

We now examine the scenarios in which two classes of objects, *i.e.*, *person* and *vehicle*, is present.

MOT17-04 is filmed using a static elevated camera on a busy street. As observed in Figure 4 and Figure 5 the frames are close to fully occupied by high-priority detection classes, meaning there exists little room for optimizing detection without sacrificing accuracy. Notably, BigLittle [28] requires the use of two DNNs when confidence is low, resulting in a high end-to-end latency. SHIFT [6] attempts to perform too much optimization on the object detection model latency, resulting in severe accuracy degradation. We observe that *PMOTHS* has some accuracy loss, but with a decrease in overall processing latency primarily due to the input size scaling heuristic.

MOT17-05 is shot by a camera moving through people on the street. In the beginning of the scene, there are pedestrians and vehicles moving along the roadway. The majority of early frames have the high-priority bounding region taking up the entire frame, but for only a few large objects making the scenario easy to detect. Large and easy-to-detect objects result in SHIFT [6] being able to make a significant improvement in runtime latency. However, despite BigLittle [28] also using confidence score values to look for performance gains, the simple threshold method cannot make significant improvements. Meanwhile, *PMOTHS* was able to make significant improvements, as there were either consistent LP regions that could be exploited, or the input size scaling heuristic allowed more efficient computation.

MOT17-10 is a scenario in which the camera moves through a less crowded street. There are few pedestrians, which occupy a small region, and vehicles to the opposite side in the early frames. The distribution of objects can be seen in Figure 4. Both BigLittle [28] and SHIFT [6] face challenges in achieving performance gains. This is either due to the necessity of running both models –since the small model's substantial inferiority limits its usefulness– or because the confidence values fail to align effectively with the complexity of the detection task. *PMOTHS* is able to maintain 97% the accuracy of the baseline, while achieving a speedup of 1.48x since it does not rely on confidence values as the only method with identified optimization potential.

MOT17-13 is a camera mounted on a vehicle that moves through busy streets occupied by both vehicles and people. This scenario is the hardest, since the objects vary in position and size along a thin horizon, and the scene itself changes significantly from start to end. The distribution of objects along the horizon can be visualized in Figure 4 and the complex nature of the scene can be assessed from Figure 5. Similarly to MOT17-10, BigLittle [28] and SHIFT [6] are not able to improve performance in a notable way because both methods strictly use confidence values. However, *PMOTHS* can achieve significant savings in latency, energy, power draw, but misses some portions of the detections, resulting in slightly lower accuracy. The missed portions are due to a delay in identifying the new objects once the scene changes. Performance gains occur because the relevant regions of the data allow reducing computational resource requirements.

Overall: BigLittle [28] improves latency, energy, and power draw in two of the seven sequences, while maintaining accuracy within 90% of the baseline in 7/7. BigLittle fails because it optimizes for the best computation time when the detection task is easy, but when the detection task is difficult or changes, it results in a runtime worse than the baseline model it utilizes. SHIFT [6] improves latency, energy, and power draw in all sequences, but only achieves the accuracy margin of 90% in 2/7 sequences. SHIFT fails because it aggressively attempts to optimize performance together with accuracy by altering model parameter size, input size, and accelerator target based only on the mean output confidence score. This results in poor detection accuracy, since a model can produce high-confidence results when only detecting a few objects. ***PMOTHS* achieves an improvement in all performance metrics and achieves accuracy comparable to baseline in all scenarios**, by considering the detection task as a parallelization problem within MOT and allocating faster computational resources for high priority objects in the scene.

5.3 MOT Metrics and Validation

To validate whether the detection accuracies achieved by *PMOTHS* translate into tracking accuracy, we evaluate the YOLOv10 [33] baseline, BigLittle [28], SHIFT [6] and *PMOTHS* using MOT17 ground truths for pedestrian tracking. All methods use ByteTrack [38] as the tracking algorithm with default configurations from the authors' original

Methodology	DetA (%)	HOTA (%)	MOTA (%)	IDF1 (%)	Latency (ms)
YOLOv10 M 1280x1280	41.82	11.64	6.45	9.21	25.91
YOLOv10 S 1280x1280	39.69	11.81	7.80	9.16	16.66
YOLOv10 N 1280x1280	37.35	11.40	7.89	9.97	12.22
BigLittle [28]*	41.46	11.52	7.06	9.05	25.99
SHIFT [6]*	19.01	11.11	8.44	9.49	14.64
***PMOTHS** People*	41.70	11.78	7.76	9.28	15.65

Table 2: Metrics calculated against the pedestrian class on the MOT17 dataset. Evaluation is conducted using the official MOT17 evaluation software [14]. DetA is the detection accuracy, HOTA is higher order tracking accuracy [21], MOTA is multi-object tracking accuracy, and IDF1 is identity F1 score.

Schedule	17-02	17-04	17-05	17-09	17-10	17-11	17-13
Baseline	0.17%	0.10%	0.12%	20.9%	0.15%	3.67%	0.13%
GPU+GPU	63.3%	74.4%	72.0%	45.1%	62.8%	58.4%	60.1%
GPU+DLA	36.5%	25.5%	27.8%	33.9%	37.0%	37.9%	39.7%

Table 3: Scheduling decision distribution across GPU+GPU, GPU+DLA executions, and baseline single model fallback on each MOT17 scenario.

implementations to ensure fairness. The results presented in Table 2, show that *PMOTHS*, with the high-priority class set to people, achieves **99. 7%** of the baseline detection accuracy while providing a **1.66x** speed-up in the overall data set. Furthermore, *PMOTHS*, outperforms the baseline in all tracking metrics (HOTA, MOTA, IDF1). Of the mean latency reported in Table 2, 85.1% (13.34ms) is attributed to frame pre-processing and OD model inference, while total overhead, including scheduling and image stitching, accounts for 14.9% (2.32ms) per frame, with 1.29ms being spent on frame-packing and 1.03ms being spent on scheduling on average.

Figure 12 presents the range of energy consumption and latency across the frames in each sequence while Table 3 shows the distribution of schedules utilized. On sequences with less optimization potential, such as MOT17-09, *PMOTHS* often falls back to the baseline. In contrast, for heavily optimizable sequences like MOT17-13, baseline usage is minimal, with the highest DLA-based schedules. Depending on the position and density of objects, *PMOTHS* is able to produce a range of schedules that optimize the accuracy-latency-energy trade-off.

5.4 LP Accuracy Goal Knob

To ensure that the estimation methodology has the intended effect on the accuracy performance of the system, we evaluate *PMOTHS* using multiple settings for the low-priority accuracy knob, which is provided to our scheduler given in Algorithm 2 with the *LP accuracy goal* parameter, γ_{lp}. The results of this sensitivity analysis are presented in Table 4. As expected, when the LP knob is increased, more compute-intensive DNNs are scheduled. The nonlinear relationship on power draw is due to whether DLA-based schedules are used

Metric	0.4	0.5	0.6	0.7	0.8	0.9
Time (ms)	15.2	15.6	16.8	17.8	19.3	20.3
Energy (J)	0.29	0.30	0.36	0.41	0.48	0.54
Powerdraw (W)	23.3	23.0	24.9	26.4	28.4	29.8
DLA Usage	66.3%	33.5%	13.4%	9.5%	7.5%	5.5%
Baseline Usage	2.8%	2.8%	13.6%	23.9%	41.5%	53.7%

Table 4: Varying LP accuracy goal values and the effect on runtime metrics and schedules.

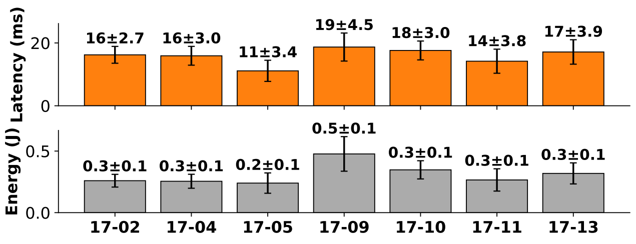

Figure 12: Range in latency and energy usage during runtime of *PMOTHS* on each sequence in the MOT17 dataset.

or not. If the LP knob is set to lower than or equal to 0.5 (below the median accuracy allowed), GPU-DLA schedules become more balanced. If the knob is set higher, the precision limitations of the DLA-based models do not meet the desired accuracy, hence resulting in schedules that do not use DLA.

5.5 Stability Analysis

To assess whether the stochastic nature of simulated annealing has a significant effect on the average performance of runtime, a stability analysis was performed in which *PMOTHS* was run ten times for each sequence in the MOT17 dataset with a unique random seed. The standard deviations are as follows: recall: 0.06%, precision: 0.01%, f1: 0.06%, latency 0.13 ms, energy: 0.005 J and power draw: 0.23 W. Analyzing the produced schedules, there is a 0.4% standard deviation in the amount of GPU-GPU vs. GPU-DLA sub-schedules utilized. **The low variation in run-to-run accuracy and performance metrics demonstrates that *PMOTHS* consistently provides the desired accuracy for HP objects while producing precise computational schedules that meet the constraints.**

5.6 Ablation Study

To better understand the individual contributions of several optimizations we employ in *PMOTHS*, we provide an ablation study in Appendix B.

6 Conclusion

We introduce *PMOTHS* to address the challenge of improving the efficiency of multi-object tracking (MOT) on heterogeneous system-on-chips (SoCs). By grouping objects based on assigned priority classes, *PMOTHS* optimizes the use of multiple accelerators to enhance latency and energy consumption without compromising accuracy. Through the use of two separate contextual regions based on object priority, *PMOTHS* can allocate computation dynamically reducing latency and saving energy. The effectiveness of *PMOTHS* is demonstrated on the MOT17 dataset where it achieves a reduction of up to 2.2x in latency, 3.8x in energy, and 2.2x in power draw while maintaining 99.7% of detection accuracy of high-priority detections.

References

[1] Md Adnan Arefeen, Sumaiya Tabassum Nimi, and Md Yusuf Sarwar Uddin. Framehopper: Selective processing of video frames in detection-driven real-time video analytics. In *DCOSS'22*.

[2] Alejandro Cartas, Martin Kocour, Aravindh Raman, Ilias Leontiadis, Jordi Luque, Nishanth Sastry, Jose Nuñez Martinez, Diego Perino, and Carlos Segura. A reality check on inference at mobile networks edge. In *EdgeSys '19*, 2019.

[3] Jack Chen, Hen-Wei Huang, Philipp Rupp, Anjali Sinha, Claas Ehmke, and Giovanni Traverso. Closed-loop region of interest enabling high spatial and temporal resolutions in object detection and tracking via wireless camera. *IEEE Access*, 9:87340–87350, 2021.

[4] Ismet Dagli and Mehmet E. Belviranli. Shared memory-contention-aware concurrent dnn execution for diversely heterogeneous system-on-chips. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '24)*, 2024.

[5] Ismet Dagli, Alexander Cieslewicz, Jedidiah McClurg, and Mehmet E Belviranli. Axonn: Energy-aware execution of neural network inference on multi-accelerator heterogeneous socs. In *DAC*, pages 1069–1074, 2022.

[6] Justin Davis and Mehmet E Belviranli. Context-aware multi-model object detection for diversely heterogeneous compute systems. In *DATE*, pages 1–6. IEEE, 2024.

[7] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé. Mot20: A benchmark for multi object tracking in crowded scenes. *arXiv:2003.09003[cs]*, March 2020. arXiv: 2003.09003.

[8] Anudeep Dhonde, Prabhudev Guntur, and Vinitha Palani. Adaptive roi with pretrained models for automated retail checkout. In *CVPR Workshops*, pages 5507–5510, June 2023.

[9] Yunhao Du, Zhicheng Zhao, Yang Song, Yanyun Zhao, Fei Su, Tao Gong, and Hongying Meng. Strongsort: Make deepsort great again. *IEEE Transactions on Multimedia*, 25:8725–8737, 2023.

[10] Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *MobiCom*, pages 115–127, 2018.

[11] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

[12] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. Deepmix: mobility-aware, lightweight, and hybrid 3d object detection for headsets. In *MobiSys'22*, pages 28–41, 2022.

[13] Lixiang Han, Zimu Zhou, and Zhenjiang Li. Pantheon: Preemptible multi-dnn inference on mobile edge gpus. In *MobiSys'24*, MOBISYS '24, page 465–478, New York, NY, USA, 2024. Association for Computing Machinery.

[14] Arne Hoffhues Jonathon Luiten. Trackeval. https://github.com/JonathonLuiten/TrackEval, 2020.

[15] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2007.

[16] Royson Lee, Stylianos I. Venieris, and Nicholas D. Lane. Deep neural network–based enhancement for image and video streaming systems: A survey and future directions. *ACM Comput. Surv.*, October 2021.

[17] Chao Liang, Zhipeng Zhang, Xue Zhou, Bing Li, Shuyuan Zhu, and Weiming Hu. Rethinking the competition between detection and reid in multiobject tracking. *IEEE Transactions on Image Processing*, 31:3182–3196, 2022.

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[19] Miaomiao Liu, Xianzhong Ding, and Wan Du. Continuous, real-time object detection on mobile devices without offloading. In *ICDCS'20*.

[20] Chen Liyan. Monte carlo multi-object tracking in wireless sensor networks. In *2010 International Conference on Challenges in Environmental Science and Computer Engineering*, volume 2, pages 162–165. IEEE, 2010.

[21] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. Hota: A higher order metric for evaluating multi-object tracking. *International Journal of Computer Vision*, pages 1–31, 2020.

[22] Ignacio Martinez-Alpiste, Gelayol Golcarenarenji, Qi Wang, and Jose Maria Alcaraz-Calero. Smartphone-based real-time object recognition architecture for portable and constrained systems. *Journal of Real-Time Image Processing*, 19(1):103–115, 2022.

[23] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. MOT16: A benchmark for multi-object tracking. *arXiv:1603.00831 [cs]*, March 2016. arXiv: 1603.00831.

[24] NVIDIA. Next-level ai performance for next-gen robotics | nvidia jetson orin agx. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, 2024. (accessed on 11/19/2024).

[25] Sankar K Pal, Anima Pramanik, Jhareswar Maiti, and Pabitra Mitra. Deep learning in multi-object detection and tracking: state of the art. *Applied Intelligence*, 51:6400–6429, 2021.

[26] Jiangmiao Pang, Linlu Qiu, Xia Li, Haofeng Chen, Qi Li, Trevor Darrell, and Fisher Yu. Quasi-dense similarity learning for multiple object tracking. In *CVPR*, pages 164–173, 2021.

[27] Ioannis Panopoulos, Stylianos Venieris, and Iakovos Venieris. Carin: Constraint-aware and responsive inference on heterogeneous devices for single-and multi-dnn workloads. *TECS*, 2024.

[28] Eunhyeok Park, Dongyoung Kim, Soobeom Kim, Yong-Deok Kim, Gunhee Kim, Sungroh Yoon, and Sungjoo Yoo. Big/little deep neural network for ultra low power inference. In *CODES+ISSS*, pages 124–132, 2015.

[29] Mohammad Mehdi Rastikerdar, Jin Huang, Shiwei Fang, Hui Guan, and Deepak Ganesan. Cactus: Dynamically switchable context-aware micro-classifiers for efficient iot inference. In *MobiSys'22*, page 505–518. Association for Computing Machinery, 2024.

[30] Ratheesh Ravindran, Michael J Santora, and Mohsin M Jamali. Multi-object detection and tracking, based on dnn, for autonomous vehicles: A review. *IEEE Sensors Journal*, 21(5):5668–5677, 2020.

[31] Kevin Smith, Daniel Gatica-Perez, J Odobez, and Sileye Ba. Evaluating multi-object tracking. In *CVPR workshops*. IEEE, 2005.

[32] Stylianos I. Venieris, Ioannis Panopoulos, and Iakovos S. Venieris. Oodin: An optimised on-device inference framework for heterogeneous mobile devices. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8, 2021.

[33] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10: Real-time end-to-end object detection. *arXiv preprint arXiv:2405.14458*, 2024.

[34] Chien-Yao Wang and Hong-Yuan Mark Liao. YOLOv9: Learning what you want to learn using programmable gradient information. 2024.

[35] Nicolai Wojke and Alex Bewley. Deep cosine metric learning for person re-identification. In *WACV*. IEEE, 2018.

[36] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *ICIP*, pages 3645–3649. IEEE, 2017.

[37] Kichang Yang, Juheon Yi, Kyungjin Lee, and Youngki Lee. Flexpatch: Fast and accurate object detection for on-device high-resolution live video analytics. In *INFOCOM'22*.

[38] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box. In *ECCV'21*.

A Comparison of YOLO Models Used in Experiments

Model Info				Metrics	Input Size							
Model	**Param**	**Device**	**Precision**		**160**	**320**	**480**	**640**	**800**	**960**	**1120**	**1280**
YOLOv10[33]	M	GPU	FP16	Recall	0.4x	0.64x	0.75x	0.83x	0.91x	0.96x	0.99x	**1.0x**
				Latency	3.48ms	4.97ms	6.98ms	9.45ms	12.31ms	15.84ms	21.21ms	**25.91ms**
				Energy	0.04J	0.07J	0.14J	0.24J	0.37J	0.54J	0.73J	**0.91J**
				Power	17.65W	24.52W	31.29W	35.57W	40.2W	42.0W	43.73W	**43.8W**
	S	GPU	FP16	Recall	0.36x	0.59x	0.72x	0.8x	0.87x	0.92x	0.96x	0.98x
				Latency	2.94ms	3.88ms	4.99ms	6.31ms	8.69ms	10.04ms	14.01ms	16.66ms
				Energy	0.02J	0.04J	0.07J	0.11J	0.18J	0.25J	0.34J	0.44J
				Power	15.13W	20.26W	25.27W	29.23W	32.11W	35.51W	36.08W	37.91W
	N	GPU	FP16	Recall	0.28x	0.53x	0.66x	0.75x	0.82x	0.87x	0.92x	0.93x
				Latency	2.82ms	3.63ms	4.35ms	5.02ms	6.68ms	7.51ms	10.59ms	12.22ms
				Energy	0.02J	0.03J	0.04J	0.06J	0.09J	0.13J	0.18J	0.23J
				Power	13.08W	16.41W	20.44W	23.1W	26.24W	28.45W	29.87W	31.66W
YOLOX [11]	M	DLA	INT8	Recall	0.54x	0.97x	0.99x	1.0x	0.95x	0.96x	0.86x	0.82x
				Latency	3.18ms	4.51ms	7.08ms	11.21ms	16.42ms	22.87ms	30.46ms	40.05ms
				Energy	0.03J	0.04J	0.08J	0.13J	0.21J	0.29J	0.4J	0.55J
				Power	12.72W	13.4W	14.67W	14.79W	14.74W	14.69W	14.75W	15.24W
	S	DLA	INT8	Recall	0.53x	0.95x	0.92x	0.94x	0.95x	0.43x	0.34x	0.24x
				Latency	2.69ms	3.42ms	4.91ms	7.41ms	10.52ms	14.23ms	18.9ms	23.76ms
				Energy	0.02J	0.03J	0.05J	0.08J	0.12J	0.17J	0.23J	0.3J
				Power	11.96W	12.71W	14.66W	15.18W	15.08W	15.15W	15.3W	15.32W

Table 5: Performance comparision of YOLOv10 and YOLOX models across all MOT17 sequence. All metrics are reported as the average for each frame across all sequences. Power draw is reported as the total system draw to run the model.

B Ablation Study

In order to understand the individual contributions of several optimizations we deploy in *PMOTHS*, we conduct an ablation study. We iteratively add features into *PMOTHS* and compare the resulting accuracy, latency, and energy usage over the entire dataset. The key features to be assessed are the impact of 1) LP detection and frame packing and 2) the utilization of parallel hardware to augment the performance of the system.

Baseline: The YOLOv10M [33] model with an input size of 1280x1280 representing the standard detection performance.

ROI: A region-of-interest (ROI) strategy that processes only the bounding box of previous HP detections with some padding. If no prior detections exist, the full frame is used. This evaluates accuracy trade-offs from focusing only on relevant regions.

Dynamic: A dynamic model selection approach where we pick a YOLOv10M version with lower resolution input size that matches the ROI bounding box. This approach improves the ROI method and represents the lower bound on latency, energy and power needed to process the HP region using the baseline model (*i.e.*, YOLOv10M).

OnlyGPU: This version is based on *PMOTHS*'s HP/LP-based MOT execution strategy, but the scheduler only uses GPU and not the DLA. It also considers the impact of LP detection on accuracy and uses the same scheduling mechanism as *PMOTHS* but switches back to the baseline model unless HP/LP-based processing is expected to be faster.

PMOTHS: The full *PMOTHS* system, incorporating GPU-DLA scheduling for optimal performance.

Each method is evaluated on MOT17 as detailed in Section 4. Figure 13 presents the relative speedup of each version for different metrics, in comparison to the previous version.

ROI: ROI strategy provides minimal benefits in latency, energy and power draw at the expense of reduced recall. Accuracy loss occurs when detections fall outside the selected region, leading to rapid over-focusing. Latency gains come primarily from reduced data transfer between CPU and GPU before preprocessing.

Dynamic: Dynamically selecting the model with smaller input size based on the ROI size improves performance, but further reduces recall. Moving objects or cameras can cause excessive focusing, and hence omitting important information. While dynamic scaling achieves nearly 1.25x speedup over the baseline, its accuracy-latency trade-off remains inefficient, with significant accuracy losses in highly dynamic scenarios. In contrast, *PMOTHS* provides a 1.6x speedup with greater than 3.0x gains in energy usage while maintaining nearly all recall.

OnlyGPU: This variant showcases the benefits of *PMOTHS*'s novel LP identification and binpacking algorithms with the exception that it does not take advantage of multiple accelerators (*i.e.*, DLAs in addition to GPUs). Therefore, the resulting schedules serialize HP and LP processing on the same GPU. *OnlyGPU* improves accuracy over *ROI* and *Dynamic* methods, while also improving latency, energy, and power draw since it can exploit smaller models for LP regions.

PMOTHS: Using additional hardware, *PMOTHS* improves latency more than the *GPU-only* schedules and thus maximizes gains in latency and energy usage (Figure 13). However, compared to GPU-only schedules, the power draw is slightly higher since both accelerators are being used at once. Multi-accelerator execution further optimizes performance, underscoring the importance of LP region processing and full hardware utilization.

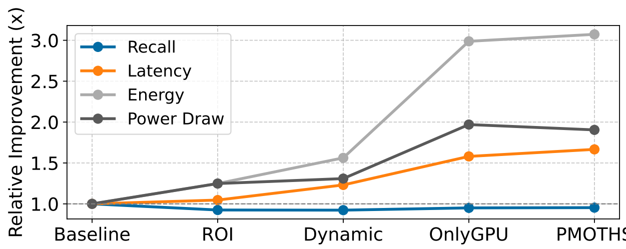

Figure 13: Comparison of various simple methods and portions of the *PMOTHS* methodology to identify the important contributions. Evaluated on the MOT17 dataset with results averaged across all video sequences.