TAPAS: Throughput-adaptive Perception for Autonomous Systems

Aman Vyas, Vasistha Kodumagulla, Zain Taufique, Pasi Liljeberg, Anil Kanduri
Department of Computing, University of Turku, Finland
Email: {amvyas, vakodu, zatauf, pakrli, spakan}@utu.fi

***Abstract*—Autonomous systems rely on a perception module to navigate through dynamic environments. In real-world scenarios, the perception module's throughput requirements vary at runtime due to changes in scene complexity. However, existing perception strategies assume a fixed FPS and static model-to-cluster mapping, resulting in either over/under provision of throughput requirements or unnecessary energy consumption across diverse scenes. Addressing this challenge requires tightly coupled *scene complexity awareness* to estimate an appropriate FPS target and *dynamic model-to-cluster mapping* to deliver the required throughput at minimum energy. We propose a throughput-adaptive perception strategy for mobile/edge platforms, enabling intelligent runtime resource allocation based on varying FPS targets. We use Reinforcement Learning (RL) with RRM (Reward Reasoning Model) and a GRU (Gated Recurrent Unit) agent to orchestrate perception tasks across heterogeneous mobile/edge platforms. We evaluate TAPAS on Jetson Orin NX across KITTI and unseen nuScenes. On the *KITTI* dataset's test sequences, TAPAS achieves 93-100% throughput met rate while saving energy by 76%. On the unseen *nuScenes* dataset, TAPAS maintains 97% throughput met rate with 64% lower energy compared to *SOTA* approaches, proving its robustness.**

I. INTRODUCTION

Autonomous systems (AS) such as self-driving vehicles, drones, robots, etc., rely on a perception stack for navigation, planning, and control [1]. Modern perception pipelines include deep learning based vision tasks for object detection, semantic segmentation, visual odometry, and obstacle avoidance [2], [3]. Throughput of the perception module (typically expressed in frames-per-second (FPS) [4], [5]) is crucial for efficient navigation of AS [6]. The throughput of the perception module depends on the complexity of the scene in a given frame [5]. Complex heterogeneous scenes (e.g., dense urban environments) require processing at a higher FPS for efficient navigation, while processing at a lower FPS suffices for sparse homogeneous scenes [5]–[9].

In real-world navigation, AS encounter a mixture of both simple and complex scenes, creating variable FPS requirements over time. Yet, existing perception modules operate at a single FPS target that is fixed at design time, irrespective of the dynamic scene complexity [4], [6]. Setting a conservatively higher FPS target offers robust navigation in dense scenes. However, this approach delivers unnecessarily higher FPS in sparse environments [5], [6], resulting in significant computational and energy wastage that is particularly detrimental to battery-powered mobile autonomous systems. Thus, AS operating in unstructured and dynamic environments requires a perception module that can adapt to variable FPS targets for energy efficiency.

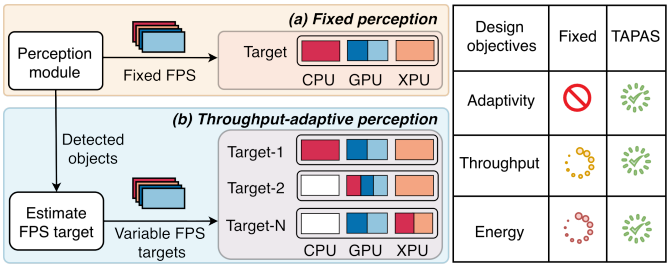

Figure 1. Overview of static and adaptive perception modules: SOTA approaches (top) employ fixed FPS and static mapping; TAPAS (bottom region) introduces estimating FPS targets and runtime mapping.

AS are increasingly employing Heterogeneous Multi-core Processings (HMPs) with asymmetric CPUs, GPUs, and custom deep learning accelerators [10]. Core-level heterogeneity offers flexibility in running the perception pipeline at different FPS targets and energy budgets. Despite using accelerator-rich hardware, existing perception pipelines employ static model-to-cluster mapping approaches [11], [12], restricting their throughput adaptivity. State-of-the-art inference optimization techniques for HMPs [13]–[15] cannot be directly repurposed for the perception pipeline, since they: (i) lack high-level domain semantic awareness (e.g., scene complexity), (ii) are confined to a single class of vision tasks (such as object detection), as opposed to concurrently running multiple classes of vision modules in a modern perception pipeline, and (iii) are not designed for multi-objective throughput adaptivity.

Adapting to varying throughput targets requires tightly coupled *scene complexity awareness* – to estimate an appropriate FPS target, and *dynamic model-to-cluster mapping* – to deliver the scene complexity-aware throughput target with the lowest energy consumption. Figure 1 (top) illustrates a representation of existing perception modules with a fixed FPS target and static model-to-cluster mapping. Without scene awareness and run-time adaptability, such strategies can provide only a predetermined throughput across diverse scenes, draining compute and energy resources. Figure 1 (bottom) demonstrates *throughput-adaptive perception*, where variable FPS targets (Target 1, 2,..N per scene 1, 2,..N) are estimated through scene awareness (Section II-B1 [16]). The scene-derived FPS target is parsed to update the model-to-cluster mapping at run-time, ensuring that only the required FPS target is delivered, significantly minimizing energy consumption.

Our goal is to design a throughput-adaptive perception

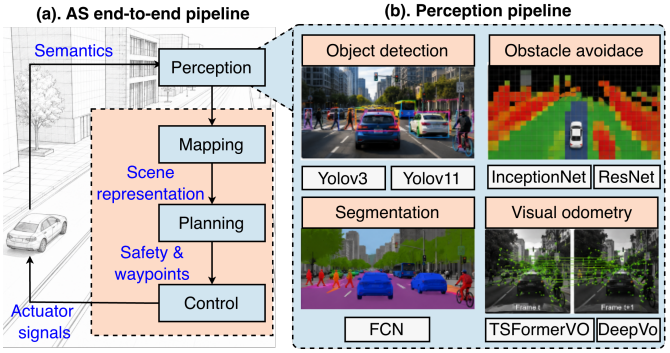

Figure 2. End-to-end and perception pipeline for AS.

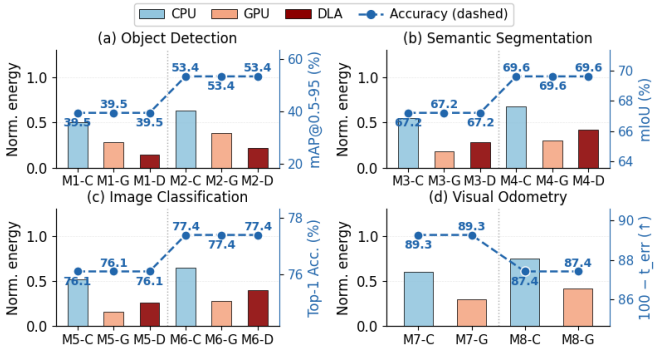

Figure 3. Impact of hardware vs. algorithmic knobs on energy and accuracy of perception pipeline

framework for AS running on heterogeneous mobile/edge platforms. We propose *TAPAS*, a framework that estimates FPS targets based on scene complexity and dynamically configures compute resources to meet these targets while minimizing energy consumption. *TAPAS* uses *spatial entropy* computed from perception pipeline outputs using Shannon's entropy (Section III-C1) to estimate scene-specific FPS targets. This lightweight mechanism quantifies scene complexity by directly mapping entropy levels to FPS targets. Configuring compute resources to meet variable FPS targets presents a complex design space exploration challenge due to application diversity (multiple DNN and transformer models in the perception pipeline) and hardware diversity (different energy-latency characteristics and variable deep learning operator support across clusters). Navigating such complex design space through exhaustive search is infeasible, heuristic methods lack adaptivity beyond design-time assumptions, and supervised learning cannot generalize beyond labeled data. In this context, Reinforcement Learning (RL) enables adaptation to variable FPS demands through system-level reward signals without labeled training data, by learning policies that maximize long-term returns across diverse HMP configurations. This enables TAPAS to make zero-shot predictions across unseen scenes and perception pipelines. We adopt Proximal Policy Optimization (PPO) [17] RL, which provides stable policy updates via clipped surrogate objectives and sample-efficient learning (Section III-C2) compared to heuristic-based methods. Despite RL's advantages, two challenges remain open. First, scene complexity exhibits temporal dependencies that memory-less agents [14], [18] cannot exploit. We address this using a Gated Recurrent Unit (GRU) agent (Section III-C3) that leverages variable FPS, entropy trajectories, and workload variations to perform model-to-cluster mapping. Second, joint throughput-energy optimization involves conflicting objectives that simple weighted-sum or heuristic reward functions cannot resolve [14], [18]. We address this through Reward Reasoning Model (RRM) [19] (Section III-C4) that provides structured reasoning to balance throughput met rate and energy efficiency under variable FPS demands. Our contributions are:

- TAPAS framework for orchestrating throughput-adaptive perception on mobile HMPs, by coupling scene-awareness with system-level resource allocation.
- Quantifying scene dynamicity with spatial entropy to estimate run-time variable throughput targets for throughput-adaptivity.
- Designing a GRU-based RL agent with RRM for capturing temporal context while balancing energy-throughput objectives, to determine model-to-cluster mapping under varying throughput targets.
- Deployment and evaluation of TAPAS on Jetson Orin NX platform, demonstrating up to 93–100% throughput-met rate with up to 76% energy savings on KITTI sequences, and generalizes to unseen nuScenes data with 97% throughput-met rate and 64% lower energy.

II. Background and Motivation

A. Background

1) Autonomous systems pipeline and perception module overview: As presented in figure 2(a), AS contains functionally independent yet semantically coupled modules: Perception, Mapping, Planning, and Control. At the task level, each module operates independently and can be designed, profiled, and optimized in isolation. However, at the functional level, these modules create a strict dependency chain: mapping consumes perception outputs to build scene representations, planning relies on the map to reason about waypoints and safety, and control translates planned trajectories into real-time actuator signals. As highlighted in figure 2(b), perception module encompasses four concurrent vision tasks: object detection, obstacle avoidance, semantic segmentation, and visual odometry. Each perception task is implemented by a dedicated DNN and/or transformer. The perception module constitutes the most compute and energy-intensive stage of the AS stack. The sole responsibility of perception is to extract scene semantics from sensor data, while safety and motion planning are handled by downstream planning modules. This separation decouples perception from safety-critical reasoning, enabling enhanced scene understanding while allowing downstream modules to independently manage navigation and safety decisions.

2) Significance of run-time system for adaptive perception: With the increasing diversity of perception workload and the increasing heterogeneity of edge platforms, efficiently deploying perception pipelines on HMP has become a critical challenge. *TAPAS* addresses this through a portable, cross-platform middleware that provides runtime hardware-level adaptation, avoiding the need for application-specific model optimization.

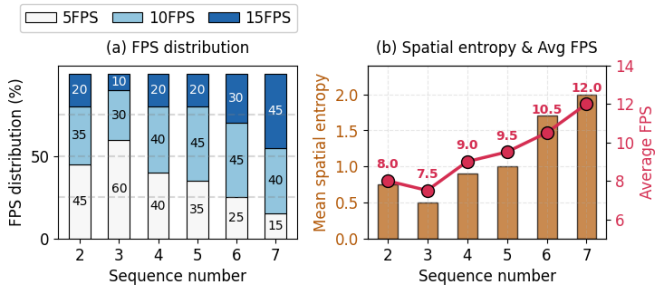

Figure 4. (a) Distribution of FPS targets across KITTI sequences. (b) Mean spatial entropy-throughput correlation.

Consequently, it generalizes across diverse shallow and deep perception pipelines for autonomous systems. An alternative approach is to tune the perception pipeline through algorithmic knobs, such as model approximation, quantization, and pruning, which typically trade accuracy for improvements in throughput and energy efficiency. Figure 3 depicts the resulting energy–accuracy trade-off space enabled by algorithmic and hardware adaptation mechanisms. We characterize the adaptivity on HMP platform (Jetson Orin NX) comprising CPU (C), GPU (G), and DLA (D), from both algorithmic and hardware perspectives. Our study considers perception models, including YOLOv11-n/l (M1–M2), FCN-ResNet50/101 (M3–M4), ResNet50/101 (M5–M6), and TSformerVO-1/2 (M7–M8), executed on the Jetson Orin NX platform. We evaluate object detection using mAP, semantic segmentation using mIoU, image classification using Top-1 accuracy, and visual odometry using translation error. Typically, algorithmic knobs achieve adaptivity by trading accuracy to meet varying FPS targets. An algorithmic knob, such as model approximation, performs model selection (e.g., (M-1 $\leftrightarrow$ M-2), (M-3 $\leftrightarrow$ M-4), (M-5 $\leftrightarrow$ M-6), and (M-7 $\leftrightarrow$ M-8)) to meet varying FPS targets. In the case of model approximation, perception quality is degraded by switching between models of different accuracies to achieve the desired FPS target. While this improves adaptivity, it can compromise the safety of the AS in dynamic environments due to reduced perception quality. Furthermore, since *TAPAS* relies on object-detection outputs to estimate spatial entropy and derive FPS targets, runtime model switching can alter detection quality and consequently affect the fidelity of entropy estimation itself. In contrast, hardware knobs such as cluster selection and task migration treat model accuracy as a fixed system parameter. Different clusters exhibit distinct performance–energy characteristics, and for variable FPS targets this heterogeneity can be leveraged to improve efficiency without degrading perception quality. Specifically, tasks can be migrated based on performance-to-cluster affinity by exploiting the diversity in compute and energy profiles across clusters. This approach provides adaptivity through throughput-energy trade-offs without degrading perception quality. Therefore, in *TAPAS*, we leverage hardware knobs (cluster mapping and task migration) to achieve adaptive and energy-efficient operation while preserving the accuracy of the throughput estimator.

B. Motivation

1) Impact of spatial entropy on throughput estimation: Existing perception strategies lack domain-semantic awareness

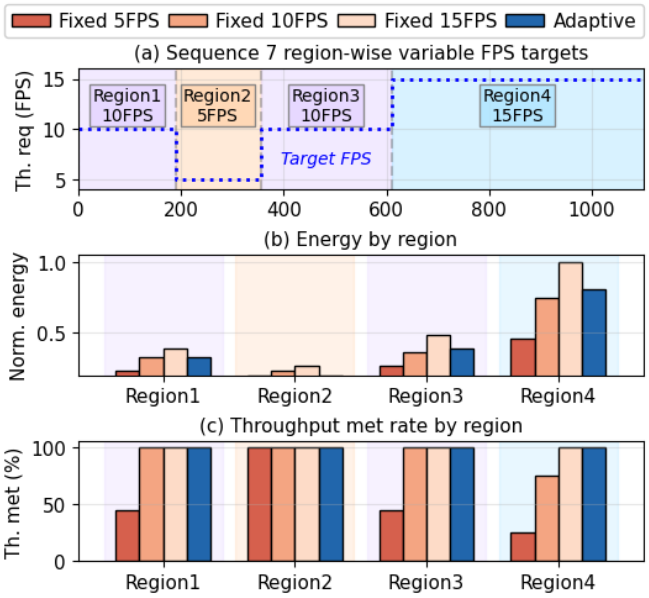

Figure 5. (a). Region-wise FPS targets in KITTI Sequence 7, (b). Normalized energy, and (c). Throughput target met rate with fixed and adaptive perception

[9], [20], [21], limiting them from considering environment-driven, run-time variable FPS targets. Spatial entropy is widely used to quantify the dynamism in a sequence of images [22]. We propose using *spatial entropy* (section III-C1) to estimate variable FPS targets based on scene complexity, thereby enabling throughput adaptivity. We demonstrate the correlation between spatial entropy and FPS targets using test sequences from the *KITTI* dataset. Typical perception module in AS operates at an average of 10 Hz (10 FPS) [4]. For simplicity of demonstration, we mapped spatial entropy values of the KITTI sequences into 3 FPS levels (5, 10, and 15), with 10 FPS as the baseline requirement. Figure 4(a) shows the distribution of estimated FPS target levels for different sequences. It should be noted that spatial entropy varies significantly across the test sequences, as reflected in the FPS requirements. Figure 4(b) shows the mean spatial entropy of test sequences 2-7. For instance, sequence 7 has the highest spatial entropy with a 12 FPS target (average over the entire sequence), whereas sequence 3 has the lowest spatial entropy with 7.5 FPS target (on average). This variation in FPS targets (1.6x) motivates the need for throughput-adaptive resource allocation, driven by scene complexity.

2) Significance of throughput-adaptive perception: We present the utility of throughput-adaptive perception over an exemplar dynamic test sequence 7 from KITTI dataset. For perception, we consider SOTA perception pipeline with FCN, Inception, YOLOv3-tiny, and TSFormerVO models. We use Jetson Orin NX (hexa-core ARM Cortex-A78 CPU, Ampere GPU with 1024 CUDA cores, and NVDLA) as the target edge AI platform. Figure 5(a) shows KITTI sequence 7, which is composed of four regions with variable FPS targets (5, 10, and 15 FPS), estimated using spatial entropy. We demonstrate the disparity between fixed and adaptive perception across all the regions. In Figure 5(a), variable FPS targets are shown in a blue dotted line. Fixed 5 FPS under-provisions for Regions

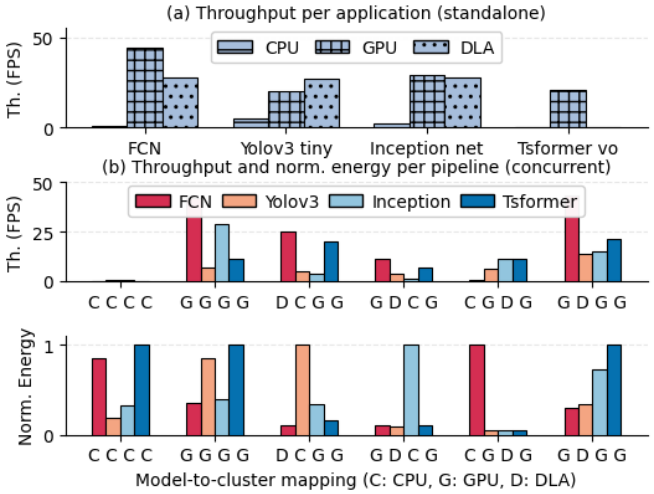

Figure 6. Throughput of perception module with different model-to-cluster mapping configurations.

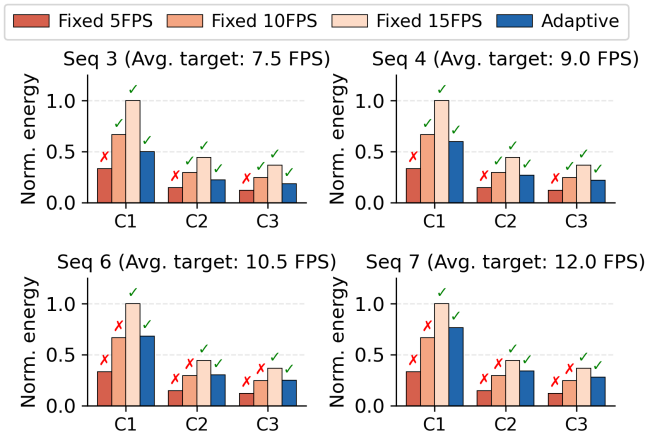

Figure 7. Energy comparison under fixed (5, 10, and 15 FPS) and variable throughput requirements. Mapping configs: C1: CPU, C2: CPU+GPU, C3: CPU+GPU+DLA

1, 3, and 4; Fixed 10 FPS target under-provisions for region 4, while over-provisioning for region 2, and Fixed 15 FPS meets all FPS targets but over-provisions for low-complexity scenes (region 1-3). This highlights the fundamental gap between fixed and scene-aware dynamic FPS targets. In contrast, adaptive perception uses a variable FPS target, estimated by quantifying the scene complexity.

Figure 5 (b) and (c) present energy and throughput met rate with fixed (5/10/15) and adaptive FPS strategies on the Orin NX platform. Fixed-FPS strategies use a static model-to-cluster mapping across all regions. With fixed strategies, energy consumption increases monotonically with FPS. An adaptive strategy uses different FPS targets per region based on scene complexity, resulting in lower energy consumption while meeting throughput targets. The adaptive strategy's region-specific run-time model-to-cluster selection, with configurations: (e.g., Regions 1 and 3: 10 FPS mapped to (G: Yolov3 tiny, G: TSformerVO, D: FCN, D: inceptionet v3), Regions 2 and 3: 5 FPS mapped to (C: Yolov3 tiny, G: TSformerVO, D: FCN, G: inceptionet v3) and higher-load regions such as Region 4 requiring more heterogeneous mappings such as (G: Yolov3 tiny, G: TSformerVO, D: FCN, G: inceptionet v3)). It should be noted that the lower energy consumption of fixed (5/10 FPS) strategies in Regions 1, 3 and 4 is due to resource under-provisioning, without meeting appropriate throughput targets. In Region 2, fixed-15 FPS over-provisions by 3× relative to scene requirements, while fixed-5 FPS under-provisions in Regions 1, 3, and 4, where higher FPS is demanded, missing 18-75% of throughput targets. Fixed-10 FPS partially balances this trade-off, matching Regions 1 and 3 but still missing 25–35% in Region 4 and over-provisioning in Region 2. Although fixed-15 FPS achieves near-100% throughput across all regions, it incurs 1.5-3x energy overhead in simpler regions (1–3). In contrast, the adaptive strategy leverages awareness of scene complexity to jointly adjust FPS and run-time model-to-cluster mapping, operating at low energy in simple regions such as Region 2, at moderate energy in Regions 1 and 3, and scaling only when necessary in Region 4. This results in 23–69% energy savings over fixed-15 FPS while achieving a 100% throughput-met rate, demonstrating that static configurations cannot co-optimize energy and performance under varying scene complexity. The actual per-region perception module orchestration challenges for an adaptive perception pipeline on HMPs are presented in Figure 6, highlighting the feasibility of model deployment across clusters for varying FPS targets.

3) Orchestrating adaptive perception on HMPs: We present challenges of orchestrating adaptive perception pipeline on HMPs from both the application and hardware perspectives under variable FPS requirements.

For the perception pipeline, we consider FCN, YoloV3Tiny, InceptionNet, and TransformerVO models, and Jetson Orin NX as the hardware platform. We execute the perception pipeline for 20 iterations on Jetson Orin NX and report the throughput and energy across runs. Figure 6(a) shows achievable throughput for each model on CPU, GPU, and DLA clusters when run in standalone mode. DNN-based models (FCN, YOLOv3-tiny, InceptionNet) achieve higher throughput on GPU while also achieving comparable throughput on DLA. In contrast, transformer-based TSFormerVO lacks operator support on DLA and is confined to the GPU. Lightweight YOLOv3-tiny shows balanced performance across GPU (20 FPS), DLA (27 FPS), and CPU (5 FPS), reflecting its flexibility in deployment. Figure 6(b) illustrates the impact of model-to-cluster mapping on throughput and energy, when multiple models are run concurrently. In this example, we map perception models to the CPU (C), GPU (G), and various combinations of CPU, GPU, and DLA (D). Mapping all workloads on CPU leads to severe throughput degradation (<1 FPS), and significantly higher energy consumption. GPU-only execution improves aggregate throughput, while yielding relatively lower normalized energy consumption for specific models. It should be noted that GPU-only execution is subject to memory constraints; running four models concurrently on a GPU can cause one or more processes to be killed due to insufficient memory and require rescheduling. DNN-based models benefit from being mapped onto DLA, achieving moderate-to-high throughput at lower energy consumption by exploiting

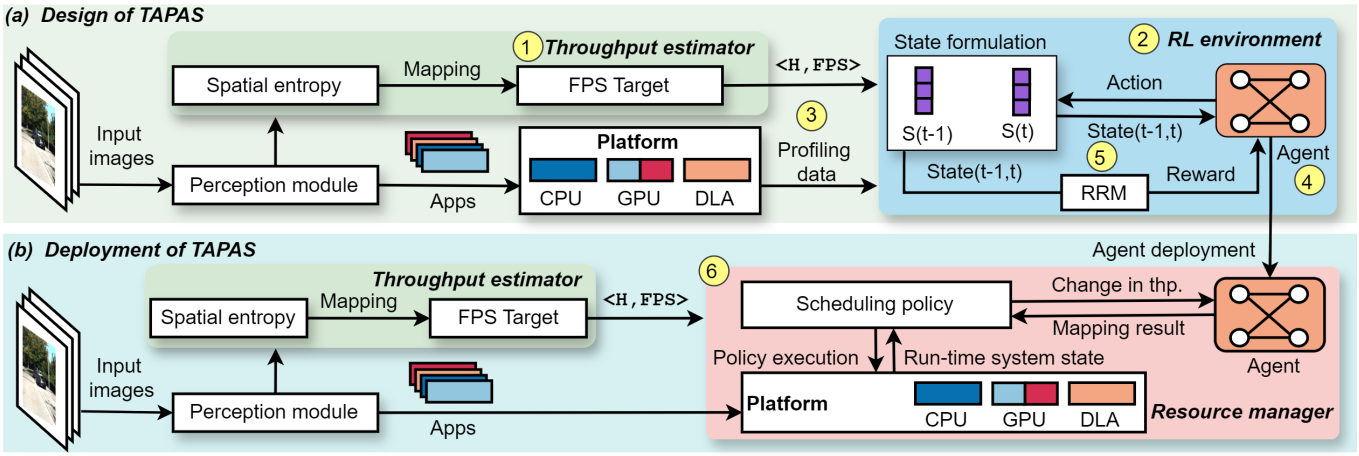

Figure 8. Training the GRU-based RL agent with offline profiled data generated using *KITTI* trajectories.

accelerator hardware. With throughput-energy diversity across clusters and models, adapting to varying throughput targets while minimizing energy consumption poses a complex optimization challenge. This necessitates a dynamic model-to-cluster mapping that considers model-cluster affinity, operator support variability, and throughput-energy trade-offs.

Platform- and scene-specific impact on adaptive perception. While adaptive perception provides throughput and energy gains, the extent of such gains largely depends on variation in scene complexity and platform-specific hardware. To illustrate this, we use KITTI sequences 3, 4, 6, and 7 with varying average FPS targets, and the Jetson Orin NX platform with CPU, GPU, and DLA clusters. We consider a lightweight perception pipeline (e.g., YOLOv11n, ResNet-50, FCN-ResNet50) for different configurations.

Figure 7 shows normalized energy with fixed (5, 10, 15 FPS) and adaptive perception strategies across C1 (CPU-only), C2 (CPU+GPU), and C3 (CPU+GPU+DLA) mapping configurations. A green tick indicates the throughput target is being met, and a red cross indicates it is not met. In seq 3, the average throughput target is 7.5 FPS, which is met with fixed 10 and 15 FPS, and adaptive perception strategies. Among these, the adaptive perception strategy has the lowest energy consumption across C1, C2, and C3 configurations. However, the extent of energy savings varies across clusters. Seq 4, with an average throughput target of 9 FPS, follows a similar trend to that of seq 3. It should be noted that the energy savings of the adaptive perception strategy are relatively lower as the average throughput target increases from 7.5 to 9 FPS between seq 3 and seq 4. Effectively, the resource wastage of the fixed 10 FPS strategy has lowered between seq 3 and seq 4, reflecting in lower energy gains with adaptive perception in this specific case. In seq 6 and seq 7, the throughput targets are 10.5 and 12 FPS, and only the fixed 15 FPS and adaptive perception strategies meet them. In this case, adaptive perception has lower overall energy consumption, while the magnitude of energy savings is similar to that in the previous sequences.

In summary, adaptive perception on HMPs faces three core challenges: (i) developing a mechanism to capture scene complexity and estimate variable FPS targets, (ii) a lightweight run-time system to configure model-to-cluster mapping based on variable FPS targets, and (iii) balancing the trade-off between throughput and energy, considering application and hardware diversity. Based on these motivations, we design *TAPAS* framework for adaptive perception on HMPs. *TAPAS* incorporates a scene-aware mechanism to estimate variable FPS targets, a lightweight agent for dynamic mapping to meet variable FPS targets, and a structured reward design for stable and grounded decision-making. The next section details the *TAPAS* framework.

III. TAPAS Framework

A. Framework Overview

We design *TAPAS* as a generic framework for orchestrating a throughput-adaptive perception pipeline on mobile HMPs. Figure 8 shows an overview of the *TAPAS* framework, split into design and deployment phases.

Design phase. In the design phase, we train an RL agent to determine model-to-cluster mapping for variable FPS requirements. Initially, we profile the perception pipeline on an HMP over different training sequences to collect execution traces of per-cluster latency, system-wide energy consumption, and an overall application profile. The ❶ *throughput estimator* module determines spatial entropy of images from the training data, and maps the entropy values to variable FPS targets. We create an ❷ *RL environment* with ❸ *profiling data*, including the execution traces and spatial entropy-FPS values. We train a Gated Recurrent Unit (GRU)-based scheduling agent within a PPO RL framework, where PPO ensures stable policy updates under dynamic throughput variations. We enhance the efficiency of the RL agent through a ❺ *Reward Reasoning Model (RRM)*, which provides structured, context-aware reward signals based on throughput-met rate, energy efficiency, and entropy-driven constraints. The recurrent ❹ *GRU* architecture learns to map temporal state sequences to optimal cluster configurations that minimize energy consumption while satisfying throughput targets. The hidden state encodes patterns of entropy and workload history across sliding windows.

Deployment phase. We monitor changes in throughput requirements at run-time while ❻ *deploying*, and enforce model-to-cluster mapping decisions determined by the RL agent. At

each frame t, we compute spatial entropy from object detection outputs (within the perception pipeline), assign a variable FPS target based on scene-complexity, and construct a state vector via n-frame temporal stacking. The GRU agent processes this state using its gating mechanisms, updates its hidden state, and makes a run-time model-to-cluster mapping decision.

B. Problem formulation

We formulate throughput-adaptive perception for AS on mobile HMPs as a resource allocation problem. The perception pipeline consists of a set of heterogeneous workloads $\mathcal{W} = w_1, w_2, \ldots, w_N$, representing deep neural networks and transformer-based models for tasks such as detection, segmentation, and tracking. These workloads are executed on an HMP platform with multiple compute clusters $\mathcal{C} = C_1, C_2, \ldots, C_M$, each exhibiting distinct performance–energy trade-offs. At run-time, the system is subject to variable throughput requirements expressed as target frame rates $T_v \in T_{v1}, T_{v2}, \ldots, T_{vn}$, which fluctuate with environmental dynamics. The goal is to learn an optimal scheduling policy $\pi(s_t) = a_t$ that maps the system state at time t to an action a_t, where the action assigns each workload w_i to a cluster $\mathcal{C}$ such that throughput target is met within lowest energy consumption.

The objective of the TAPAS policy is to jointly minimize the throughput deficit and the system-wide energy consumption. This is formalized as the following optimization problem:

$$\min_{\{h_{i,t}\}_{i=1}^{N}} O_t = \left(T_v(t) - \sum_{i=1}^{N} R_i(h_{i,t}) \right) + \left(\sum_{c \in \mathcal{C}} E_{sys}(u_{c,t}) \right), \tag{1}$$

where $\sum_{i=1}^{N} R_i(h_{i,t})$ denotes the aggregate achieved throughput, and $\sum_{c \in \mathcal{C}} E_{sys}(u_{c,t})$ captures the total energy consumed by all clusters under utilization state $u_{c,t}$. In Equation 1, the first term penalizes deviations between the target and achieved FPS, while the second term penalizes excessive energy usage. By minimizing this combined objective, the scheduler delivers the target throughput with lower energy consumption.

C. Design of TAPAS

*1) **Throughput estimator**:* The target FPS (throughput) is determined for a given frame based on the scene complexity. We consider the perception pipeline to implicitly include an object detection model, and use its output to determine spatial entropy, which quantifies scene complexity based on the number of detected objects. Since perception must complete within a fixed time window, different frame complexities translate into variable throughput requirements. TAPAS exploits this relationship by mapping spatial entropy ranges to FPS targets at run-time. For each frame I_t, the object detection model $\mathcal{O}$ obtains a semantic class map C_t, which encodes the spatial distribution of objects in the scene. A normalized histogram $\mathbf{p}_t$ is computed over C_t, and spatial entropy is derived via Shannon's formulation:

$$H_t = -\sum_{c} p_t(c) \log p_t(c) \tag{2}$$

The number of levels N_h and N_t directly control the number of levels for spatial entropy and FPS target, while the class-gap parameters CG_t and CG_h govern the granularity of both spatial entropy and FPS target, respectively. Both the class gap and the number of levels are fixed design-time parameters, with $(N_h, N_t, CG_t, CG_h) \in \{1, 2, \ldots, N\}$ configured before deployment based on the target platform's FPS operating range and expected scene complexity distribution. These parameters are necessary because a single fixed FPS target cannot capture the continuous variation in scene complexity, yet exhaustive per-frame FPS optimization is computationally infeasible at runtime; the class-gap and level count together provide a lightweight discretizations that balances adaptation granularity against scheduling overhead. Given a baseline entropy H_{base}, the entropy thresholds for N_h levels are parameterized as:

$$H_k = H_{base} + (k - \lceil N_h/2 \rceil) \cdot CG_h, \quad k = 1, 2, \ldots, N_h \tag{3}$$

where CG_h defines the bandwidth between consecutive entropy levels. Once the N_h entropy bands are established, the throughput estimator maps each frame's entropy H_t to a variable FPS target T_v across N_t throughput levels, spaced by CG_t relative to baseline T_{base}:

$$T_{vi} = T_{base} + (j - \lceil N_t/2 \rceil) \cdot CG_t, \quad j = 1, 2, \ldots, N_t \tag{4}$$

such that $H_t \in [H_k, H_{k+1})$ maps to the j-th throughput level T_{vi}. Increasing N_h and N_t enables finer-grained runtime adaptation, while larger CG_h and CG_t widen the entropy bands and the FPS target steps, respectively, providing independent control over sensitivity to scene complexity and responsiveness to throughput.

*2) **Design of RL environment**:* Training the TAPAS RL agent requires a well-defined environment comprising three interdependent components: a state space, an action space, and a reward model. This section details the RL environment design, GRU-based agent architecture, and RRM, and describes how these components are jointly trained to enable adaptive cluster mapping. We process *KITTI* trajectory image sequences over perception workloads $\mathcal{W}$ on a representative HMP with clusters C_n to calculate spatial entropy and corresponding variable throughput requirements. Overview of the training process is shown in Figure 8. As mentioned in Section III-C1, spatial entropy quantifies dynamicity within image sequences [22] using Shannon's formula (Eq. 2). We measure throughput metrics for each workload configuration on all the HMP clusters $\mathcal{C}$.

In-context state space formulation. Overview of the RL environment with GRU agent and RRM is shown in Figure 9. At decision step t, the RL environment processes perception workloads $\mathcal{W} = \{w_1, w_2, ..., w_N\}$ where each workload w_i is characterized by:

$$\phi_i = \begin{matrix}[\text{GFLOPs}_i, \text{ConvBlocks}_i, \text{TransformerEnc}_i, \\ \text{MultiHeadAttn}_i, \text{TaskType}_i, T_{vi}, H(S_t)]\end{matrix} \in \mathbb{R}^d \tag{5}$$

This captures computational intensity, architectural complexity (convolution/transformer blocks, multi-head attention), one-hot encoded perception tasks, variable throughput requirements T_{vi}, and spatial entropy $H(S_t)$.The global state aggregates all workloads as $s_t = [s_1^t, s_2^t, \ldots, s_N^t] \in \mathbb{R}^{dN}$, where each s_i^t captures the application characteristics, current

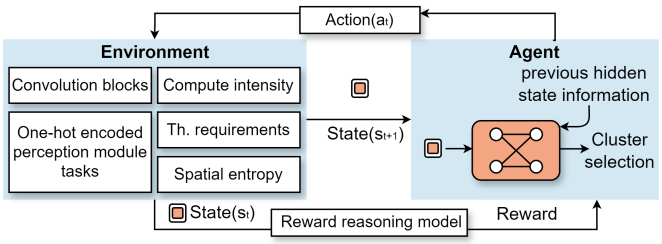

Figure 9. RL environment with GRU agent and reward reasoning model.

variable FPS target T_t and $H(S_t)$ for each perception workload i in the pipeline. To enable in-context state formulation, the current and previous state vectors are concatenated as $\tilde{s}_t = [s_{t-n} \| s_t] \in \mathbb{R}^{2dN}$, providing the GRU agent with temporal context to capture workload transition patterns and entropy evolution across consecutive frames.

Action Space: The RL agent's action space includes the selection of a cluster for perception workloads:

$$a_t = \{c_1^t, c_2^t, \ldots, c_N^t\}, \quad c_i^t \in \{\text{Compute cluster}\} \tag{6}$$

*3) **Agent architecture**:* The GRU agent maintains an in-context state representation by concatenating s_{t-1} and s_t, capturing temporal dependencies across consecutive scheduling decisions. At each timestep t, the state vector $[H_t, T_v, \phi_t]$ encoding spatial entropy, target FPS, and workload characteristics is fed into the GRU hidden state, enabling the agent to track entropy evolution, throughput stability, and workload transition patterns across frames. This recurrent formulation allows the agent to anticipate shifts in scene complexity (e.g., from sparse highway to dense intersection) and adapt cluster assignments accordingly, directly addressing the fundamental limitation of static scheduling methods that lack historical context.

The GRU [23] employs reset and update gates that selectively retain relevant historical information while filtering out obsolete scheduling decisions, enabling effective learning from past workload allocation experiences. The reset gate $r_t = \sigma(W_r \cdot [h_{t-1}, s_t])$ determines which previous scheduling patterns to forget, while the update gate $z_t = \sigma(W_z \cdot [h_{t-1}, s_t])$ controls the integration of new state information with historical context. This temporal memory mechanism enables the agent to recognize recurring throughput patterns, adapt to cyclical variations in throughput, and make informed scheduling decisions based on both the current system state and the hidden state of past allocation outcomes.

*4) **Reward Reasoning Model**:* This subsection details the RRM employed within the TAPAS framework to compute structured, multi-objective rewards for RL agent training. RL agents suffer from reward shaping problems [24]. Figure 10 compares three reward shaping approaches for RL-based scheduling using system features (variables throughput requirement P_v, workload specific states ϕ_i, S, actions a_i, metrics P_a, E_a). Heuristic-based rewards [14], [18] achieve the highest confidence (0.98) without explanation; they rely on simplistic linear combinations of throughput and energy metrics, leading to suboptimal RL policy behavior. Large Language Model (LLM)-based rewards [25] justify moderate

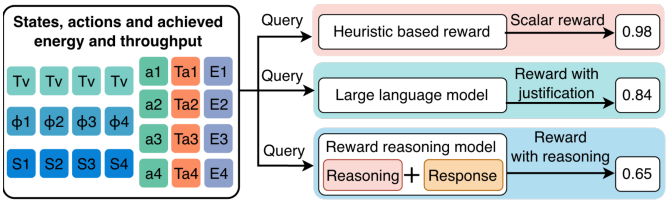

Figure 10. RL-based reward shaping approaches: heuristic-based reward, LLM-based reward, and Reward reasoning model.

scores (0.84), but lack domain-specific reasoning for HMPs and variable FPS targets. RRM [19] represents a novel approach that leverages the Qwen2 [26] Transformer-decoder architecture, offering detailed explanations with verifiable rewards (0.65). Unlike traditional reward functions, RRM formulates reward modeling as a text completion problem, auto-regressively generating comprehensive output consisting of structured thinking processes followed by a final response. This enables chain-of-thought reasoning and provides structured, interpretable feedback that dynamically evaluates scheduling decisions based on contextual factors (e.g., spatial entropy, throughput requirements, and hardware constraints) rather than on fixed weight assignments. This approach helps the GRU agent understand *why* certain allocations are beneficial beyond mere reward, enabling stabler learning and faster convergence. Further, it provides better generalization to variable FPS targets while maintaining energy efficiency across HMPs compared to traditional heuristic approaches. A potential risk of RRM is hallucination, where generated reasoning may diverge from system reality [27]. To address this, we use grounded reward with RRM to ensure that only factually consistent reasoning is propagated to the GRU agent.

Standard LLM-based reward models compute a scalar score from textual descriptions of states and actions. These approaches rely mainly on human preferences or supervised labels, without grounding predictions in real system behavior. In contrast, the RRM conditions reward prediction on both structured reasoning traces and physical measurements (throughput, energy, entropy), and is trained with a grounding objective to align outputs with measured metrics. A conventional LLM reward model computes a reward as:

$$R_{\text{LLM}}(s_t, a_t) = g_\phi\big(\text{TextEncode}(s_t, a_t)\big), \tag{7}$$

emphasize linguistic priors but lack explicit grounding in real-world outcomes.

We define a *grounded reward* as a physically constrained and measurement-consistent supervision signal for reinforcement learning. The *grounded reward* is explicitly derived from real-world system outcomes rather than from heuristic approximations or unconstrained language-model outputs. Formally, at decision step t, given the system state s_t, the selected scheduling action a_t, and the vector of offline measured platform outcomes $\mathcal{M}_t$ (including achieved throughput, latency, and energy), the grounded reward $\hat{R}_t$ is generated as

$$\hat{R}_t = f_\theta(s_t, a_t, \mathcal{M}_t), \tag{8}$$

where $f_\theta(\cdot)$ denotes the RRM parameterized by θ. Unlike conventional LLM-based reward models that score plausibility without grounding, RRM conditions rewards on measured

outcomes $\mathcal{M}_t$, penalizes predictions that contradict real observations via a grounding loss, and applies adaptive test-time reasoning to verify candidate reward estimates where uncertainty is highest, replacing free-form scoring with a measurement-anchored reward reshaping.

Algorithm 1 Agent training

Require: Offline image sequences $\mathcal{I}$, perception workloads $\mathcal{W}$, clusters $\mathcal{C}$, GRU policy π_θ, RRM reward model $\mathcal{R}$, throughput estimator $\Psi(\cdot)$
Ensure: Trained GRU scheduling policy $\pi_{\theta*}$
1: Initialize GRU parameters θ, replay buffer $\mathcal{B}$, baseline FPS T_{base}
2: **for** each trajectory τ in dataset **do**
3: **for** each frame I_t in trajectory **do**
4: $(T_v) \leftarrow \Psi(I_t, \mathcal{O}, T_{base}, N_h, N_t, CG_t, CG_h)$ {Call throughput estimator}
5: Extract workload features
$$\phi_i \leftarrow [\text{GFLOPs}_i, \text{ConvBlocks}_i, \text{TransformerEnc}_i, \text{MHA}_i, \text{TaskType}_i, T_v, H(S_t)]$$
6: Construct global state $s_t = [\phi_1, \phi_2, ..., \phi_N]$
7: Sample action $a_t \sim \pi_\theta(a|s_t)$ {Assign each workload to a compute cluster}
8: Execute scheduling action a_t on clusters $\mathcal{C}$, measure throughput P_a and energy E_a
9: Compute reward $R_t \leftarrow \mathcal{R}(s_t, a_t, P_a, E_a, T_t)$ {Reward reasoning model}
10: Store transition (s_t, a_t, R_t, s_{t+1}) into $\mathcal{B}$
11: Update GRU parameters $\theta \leftarrow \theta - \alpha \nabla_\theta L_{\text{PPO}}(\theta, \mathcal{B})$ {Policy optimization with PPO}
12: **return** Trained GRU policy $\pi_{\theta*}$

*5) **RL agent training**:* GRU-based agent learns to dynamically map perception models to heterogeneous compute clusters within a closed training loop, which comprises of throughput estimator, offline profiled data, RL environment, and RRM. Algorithm 2 summarizes the end-to-end offline training procedure for the GRU-based RL agent.

Agent training algorithm: The GRU scheduling agent is trained offline using trajectory data from KITTI and throughput labels derived from the throughput estimator. The TAPAS agent training algorithm takes as inputs: offline image sequences $\mathcal{I}$, perception workloads $\mathcal{W}$, clusters $\mathcal{C}$, GRU policy π_θ, RRM reward model $\mathcal{R}$, throughput estimator $\Psi(\cdot)$. The training loop begins by initializing GRU parameters, and the baseline FPS setting T_{base} (Line 1). For each trajectory τ in the dataset (Line 2), and each frame I_t within that trajectory (Line 3), throughput estimator $\Psi(\cdot)$ is invoked to compute variable FPS target T_v with arguments $(T_{base}, N_h, N_t, CG_t, CG_h)$ *(Line 4)*. These serve as ground-truth supervisory signals for variable throughput scheduling. The agent then extracts workload descriptors ϕ_i that capture compute intensity, architectural composition, task type, throughput target T_v, and the spatial entropy $H(S_t)$ (Line 5). These descriptors are aggregated into a global state vector s_t (Line 6) representing the complete scheduling context at time t. Using this state representation, the GRU policy samples an action a_t (Line 7), which corresponds to assigning optimal model-to-cluster mapping. The action is executed on the HMP (Line 8), allowing measurement of achieved throughput P_a and energy consumption E_a. The reward reasoning model evaluates the scheduling action (Line 9) and generates a structured, context-aware reward R_t that accounts for throughput demands, energy efficiency, entropy-driven constraints, and architecture-specific performance variations. Each transition tuple is stored in a replay buffer for subsequent optimization (Line 10). After completing a trajectory, the GRU parameters are updated using PPO's clipped surrogate loss (Line 11), enabling stable policy improvement under variable-throughput conditions. This process iterates over all trajectories (Line 2), resulting in a trained GRU scheduling policy $\pi_{\theta*}$ (Line 12).

D. Agent deployment and integration in scheduling policy

As presented in figure 8, The resource manager consists of two tightly coupled components: (i) the scheduling policy and (ii) the RL-based GRU agent. The scheduling policy continuously monitors run-time system status, including cluster availability, workload characteristics, and current throughput demand, and determines when re-allocation is required. The GRU agent is invoked by the scheduling policy to compute optimal model-to-cluster mappings under the observed system state. As shown in Figure 8, this section details the deployment stage of TAPAS, wherein the trained GRU agent is integrated into the run-time scheduling policy as a core component of the resource manager, which enforces adaptive model-to-cluster mapping decisions directly onto the targeted platform. The scheduling policy triggers agent queries whenever variations in spatial entropy indicate shifts in FPS requirements, thereby enabling run-time workload-cluster mapping decisions. The lightweight GRU architecture minimizes computational overhead while providing optimal resource allocation for HMP based on current perception-workload characteristics and variable throughput demands.

Algorithm 2 TAPAS Deployment Policy

Require: Perception workloads $\mathcal{W}$, clusters $\mathcal{C}$, throughput estimator $\Psi(\cdot)$, trained GRU policy π_{GRU}, entropy threshold τ
Ensure: Adaptive cluster mapping $M : \mathcal{W} \rightarrow \mathcal{C}$
1: Initialize $T_v^{curr} \leftarrow T_{base}$ FPS
2: **while** system active **do**
3: Observe run-time system status $z_t \leftarrow \{Ca_t\}$
4: Compute spatial entropy $H(S_t)$ from current frame I_t
5: $(T_v) \leftarrow \Psi(I_t, \mathcal{O}, I_t, T_{base}, N_h, N_t, CG_t, CG_h)$ {Throughput estimator}
6: Determine new throughput requirement $T_v^{new} \leftarrow f(H(S_t))$
7: **if** $|T_v^{new} - T_v^{curr}| > \Delta$ **then**
8: Extract workload features
$$\phi_i \leftarrow [\text{GFLOPs}_i, \text{ConvBlocks}_i, \text{TransformerEnc}_i, \text{MHA}_i, \text{TaskType}_i, T_t, H(S_t)]$$
9: Construct state $s_t \leftarrow [\phi_1, \phi_2, \ldots, \phi_N, T_v^{new}, H(S_t)]$
10: Query GRU agent: $a_t \leftarrow \pi_{GRU}(s_t)$
11: Execute cluster mapping: $\Pi_{SP} \leftarrow a_t$
12: Update $T_v^{curr} \leftarrow T_v^{new}$
13: Execute perception workloads $\mathcal{W}$ on assigned clusters through Π_{SP}

Algorithm 2 presents the TAPAS run-time scheduling policy deployed on the target HMP.The system initializes T_v^{curr} with a baseline throughput target of T_{base} FPS (Line 1) and enters continuous monitoring (Line 2). At each time step, the scheduling policy monitors run-time system status (cluster availability) (Line 3). At each frame, spatial entropy $H(S_t)$ is computed (Line 3) and passed to the throughput estimator $\Psi(\cdot)$ take arguments $(T_{base}, N_h, N_t, CG_t, CG_h)$, producing variable throughput (Line 5). The new FPS requirement T_v^{new}

is derived from $H(S_t)$ (Line 6), and a remapping decision is triggered only when a change in throughput target is detected Δ (Line 7), minimizing scheduling overhead during steady-state operation. Upon a shift, the policy extracts per-workload features with task type alongside current entropy and throughput ϕ_i (Line 8) and assembles the global state vector s_t (Line 9). The trained GRU agent is then queried to produce the optimal cluster assignment a_t (Line 10), which is executed as the updated scheduling policy mapping Π_{SP} (Lines 11-12). Perception workloads are subsequently dispatched to their assigned clusters through scheduling policy Π_{SP} (Line 13), and the loop continues monitoring for subsequent entropy-driven throughput changes.

IV. Experimental Evaluation

A. Experimental setup

Platform. We evaluate TAPAS on a representative mobile HMP platform of Nvidia Jetson Orin NX [28], featuring an Ampere GPU with 1024 CUDA cores (765 MHz max), a hexa-core ARM Cortex-A78 CPU (1.9 GHz max), and 8 GB shared memory with asymmetric CPU clusters (4+2 ARM cores). The TAPAS framework is cross-platform, portable middleware that generalizes across scenes and perception pipelines, although the achievable gains depend on the target platform.

Workloads. For the perception module, we use SOTA models spanning object detection (YOLOv3-tiny, YOLOv11n, Faster R-CNN), semantic segmentation (FCN, DeepLab V3), visual odometry (TSFormerVO, DeepVO), and obstacle avoidance (ViT, Swin Transformer, InceptionNet V3), covering diverse compute patterns such as convolution, attention, and encoder–decoder architectures. For evaluation, we set throughput estimator parameters as $N_h = N_t = 3$, $H_{base} = 1.5$, $CG_h = 1.0$, $T_{base} = 10$ FPS, and $CG_t = 5$ FPS.

Workloads Mixes. For experimental evaluation, we use five application mixes with varying compute intensity, based on eleven perception models: M0 (Faster R-CNN), M1 (Deeplab v3), M2 (Swin transformer), M3 (YOLOv11-n), M4 (FCN), M5 (TSformer VO), M6 (ViT), M7 (YOLOv3 tiny), M8 (InceptionNet), M9 (ResNet-152), and M10 (DeepVO). Application mixes contain moderate to high complexity perception modules – Mix 1: heavier detection and segmentation (M0, M1, M2), Mix 2: lightweight detection and transformer-based obstacle detection (M3, M4, M5), Mix 3: detection, segmentation, and visual odometry (M0, M4, M5), Mix 4: lightweight detection, and obstacle avoidance and heavier segmentation, and visual odometry (M7, M4, M8, M5), and Mix 5: heavier detection, obstacle avoidance, segmentation, and visual odometry. (M0, M4, M5, M8)

Dataset. For training, we use *KITTI* dataset sequences (0, 1, 9, 8,10) [29], which comprises diverse autonomous driving scenarios with varying traffic densities and lighting conditions. For evaluation and testing, we use KITTI dataset sequences (2, 3, 4, 5, 6, 7) and all the sequences of nuScenes dataset [30]. Our evaluation on test sequences from both KITTI and nuScenes datasets that are unseen by the RL agent validates the generalizability of the TAPAS framework for providing adaptive perception across diverse real-world scenarios.

Comparison w.r.t. SOTA. For evaluation, we consider three SOTA perception and multi-DNN execution strategies with varying degrees of hardware heterogeneity exploration: *EE* [12] (CPU), *Omniboost* [14] (CPU+GPU), and *Band* [13] (CPU+GPU+NPU). For EE [12], we implement power, energy, and performance models by profiling applications and incorporating them into heuristic decision policies. For *OmniBoost*, we use Gymnasium for implementing the Monte Carlo Tree Search algorithm and PyTorch for training ResNet-based agents on our profiled dataset. *Band* partitions the perception module's models into subgraphs at design time, and maps supported DNN operations onto HMP. We implemented the *Band*'s model analyzer for subgraph generation. We measure the throughput of each vision task in the perception module, including run-time variable throughput requirements and energy across diverse workload mixes.

B. Ablation studies of RL agent and RRM

(a). RL training. Our GRU-based agent utilizes PPO [17] as the RL training algorithm, complemented by a novel RRM [19] for reward shaping. We train our agent on two experimental setups: the first containing 3 models and the second with 4 models on *KITTI* dataset sequences (0, 1, 9, 8,10) with remaining sequences (2, 3, 4, 5, 6, 7) reserved for testing, demonstrating robust scheduling behavior and adaptability in unseen test environments. We train on a continuous stream of input frames from the KITTI dataset. In our evaluation, we use three FPS levels (5, 10, and 15 FPS) based on empirically observed spatial entropy levels across the KITTI dataset (Figure 4). However, our framework is designed to support multiple FPS levels (Section III), offering fine-grained control over adaptive perception. We emulate variable throughput by skipping an appropriate number of frames (as proposed in [31]) to achieve specific FPS targets (5, 10, 15).

(b). Agent and reward model design space evaluation. Figure 11(a) presents an ablation study across agent architectures (ANN [18], ResNet [14], LSTM [32], GRU) and reward models (Heuristic, LLM, RRM), showing the importance of temporal modeling and grounded rewards for *TAPAS*. Among these agents, GRU consistently outperforms others, achieving the highest performance (0.72) and stability (0.78). ANN-based agents lack memory, while ResNet agent only captures spatial features. An LSTM agent improves temporal learning but incurs higher overhead. GRU achieves the best trade-off through a lightweight two-gate design that efficiently encodes workload history and adapts rapidly to variable FPS. Across reward models, heuristic rewards under-perform due to their handcrafted nature, which yields sparse learning signals. LLM-based rewards improve semantic reasoning, but suffer from stochasticity and inconsistency across similar state-action pairs. RRM achieves the highest performance and stability (0.94/0.86) by producing grounded and deterministic rewards via test-time reasoning over spatial entropy, throughput, energy, and adaptivity. This structured reasoning reduces exploration variance by 62% and enables reliable convergence, making GRU+RRM combination particularly effective for adaptive perception scheduling across HMPs.

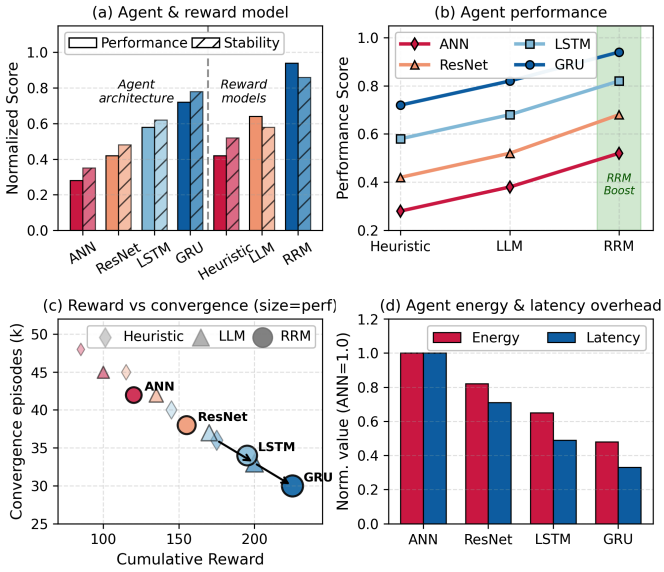

Figure 11. Ablation study of RRM and GRU-based agent in terms of performance, stability, convergence, and computational cost.

(c). Model ablation across agent architectures. Figure 11(b) shows consistent performance improvement from Heuristic→LLM→RRM across all agents: (i) ANN (+86%), (ii) Resnet (+62%), (iii) LSTM (+41%) and (iv) GRU (+31%). Heuristic→LLM transition provides moderate gains by incorporating semantic context, though improvements plateau due to stochasticity. The LLM→RRM transition yields greater improvement through adaptive test-time reasoning, eliminating variance through deterministic evaluation. The diminishing improvement percentage (86%→31%) indicates that simple agents (such as ANN) benefit more from improved rewards (RRM), while advanced agents (such as LSTM and GRU) already capture temporal dynamics effectively. Notably, ANN+RRM (0.52) approaches ResNet+Heuristic (0.42), indicating that reward quality partially compensates for architecture. Whereas, GRU+Heuristic (0.72) exceeds ANN+RRM (0.52), confirming that optimal performance requires both strong architectures and high-quality reward modeling.

(d). Convergence and cumulative reward analysis for agent architecture and reward model. Figure 11(c) illustrates progressive improvement across reward models; marker size encodes performance and arrows trace learning trajectories. GRU improves from 175 reward/36k episodes (Heuristic) → 225 reward/30k episodes (RRM), achieving 29% higher reward with 17% faster convergence. Heuristic reward stuck in a suboptimal region (36-48k episodes, 85-175 reward) due to sparse binary rewards, leading to inefficient exploration. LLM rewards provide denser semantic signals and modest speedup, but suffer from stochastic variance that introduces noisy gradients. RRM achieves near-Pareto-optimal performance through deterministic, structured reasoning over entropy, throughput-met rate, and energy metrics, enabling faster convergence and higher reward per episode. The GRU+RRM combination is optimal, as GRU's temporal modeling fully leverages RRM's grounded feedback to anticipate scene dynamics and proactively select energy-efficient configurations.

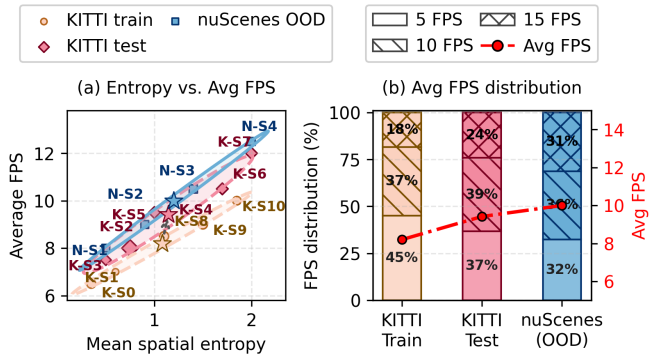

Figure 12. Comparison between KITTI and nuScenes datasets.

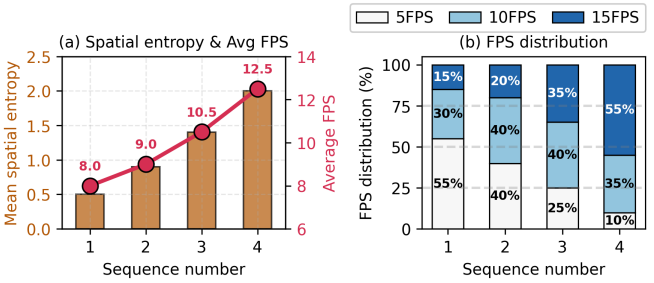

Figure 13. (a) Impact of mean spatial entropy on throughput (FPS) and (b) throughput (FPS) distribution across the NuScenes dataset's sequences.

(e). Computational efficiency and deployment analysis. The GRU-based policy achieves the lowest runtime overhead, requiring only 2.3 ms latency and 45 mJ energy, yielding 82-85% lower latency and 84-89% lower energy consumption than ANN, ResNet, and LSTM baselines. This efficiency arises from GRU's lightweight two-gate design, shared temporal weights, and reduced memory transfers through a single hidden state. Furthermore, model and data remapping contribute less than 0.2% of the total overhead, confirming lightweight execution for adaptive perception on mobile/edge HMPs.

C. Generalization analysis for throughput estimator

(a). Zero-shot generalization on Unseen Dataset. Figure 12 (a) illustrates both intra and inter-dataset variability in spatial entropy and FPS across *KITTI* (train/test) and nuScenes (out-of-distribution (OOD)) sequences. Within KITTI, the train and test ellipses exhibit partial overlap yet a measurable intra-dataset shift, with test sequences spanning higher entropy (up to 2.0) and achieving a higher average FPS (9.4 vs. 8.2), reflecting greater scene complexity in held-out sequences. Across datasets, the inter-dataset gap is more pronounced: nuScenes sequences cluster at consistently higher entropy and average FPS (10.0), with a 31% share of 15-FPS frames compared to 24% for KITTI test and only 18% for KITTI train, confirming that nuScenes poses a strictly harder OOD scheduling challenge. Figure 12 (b) shows the average FPS distribution across KITTI (train/test) and nuScenes (OOD) sequences. KITTI train is dominated by 5-FPS frames (45%), reflecting low-complexity scenes where energy savings are readily exploited, whereas nuScenes shifts the balance toward higher FPS, requiring the *TAPAS* to adapt to more frequent, high-complexity frames without prior exposure to this distri-

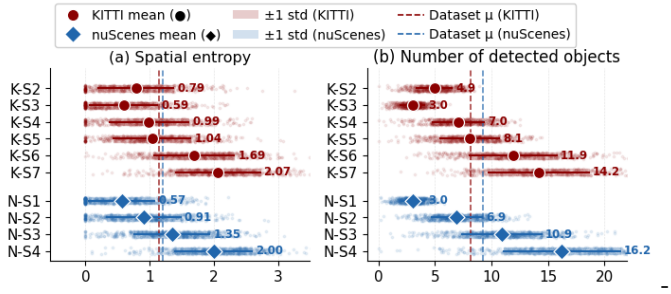

Figure 14. Sequence-wise spatial entropy and average detected objects distribution for KITTI test sequences and nuScenes OOD sequences.

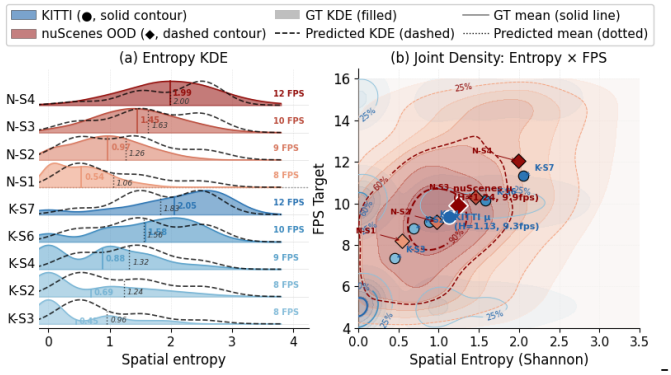

Figure 15. GT vs. Predicted entropy KDE and cross-dataset joint density.

bution. TAPAS, trained exclusively on KITTI, must generalize to both intra-dataset entropy drift and the broader inter-dataset domain gap, thereby validating its robustness.

(b). Spatial entropy: In-Distribution and OOD analysis. Figure 14(a) and (b) present the per-frame spatial entropy distribution and corresponding detected objects across KITTI test sequences (K-S2-S7) and unseen nuScenes sequences (N-S1-S4), validating the spatial entropy–detected objects mapping. The results show that the entropy preserves a consistent monotonic mapping from detected objects to spatial entropy across both in-dist. and OOD datasets. Figure 14(a) shows per-frame entropy scatter, ±1-std bands, and mean values, with dataset-level means indicated by dashed lines (μ_{KITTI}=1.13, μ_{nuScenes}=1.22). KITTI entropy increases monotonically from μ_H=0.59 (K-S3) to μ_H=2.07 (K-S7), reflecting increasing scene complexity. Low-entropy sequences (K-S2, K-S3) exhibit tight variance, while high-entropy sequences (K-S6, K-S7) show larger intra-sequence variability. nuScenes spans 0.55–2.0, covering and extending the KITTI range: N-S1 and N-S2 lie below the KITTI mean, while N-S3 and N-S4 exceed it, confirming a harder yet non-disjoint OOD distribution. Figure 14(b) shows K-S3 exhibits both the lowest entropy (0.59) and the fewest detected objects (3.0), while K-S7 attains the highest entropy (2.07) and the highest detected object count (14.2). A consistent monotonic trend is also observed in nuScenes, where N-S1 (H=0.57, 3.0 objects) progresses to N-S4 (H=2.00, 16.2 objects), with N-S4 exceeding even the densest KITTI sequence in object count, aligning with its higher OOD entropy. Furthermore, it shows that the average number of detected objects across KITTI and nuScenes sequences strongly correlates with spatial entropy, reinforcing its role as a practical proxy for scene-level complexity estimation.

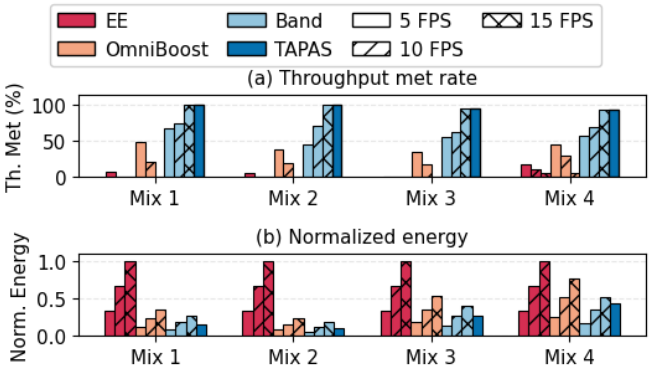

Figure 16. Throughput met (%) and energy of mix 1-4 with 5, 10, and 15 FPS on nuScenes.

(c). Robustness analysis of TAPAS on unseen dataset. Figure 13 (a) and (b) present the robustness of the throughput estimator on unseen nuScenes dataset. Figure (a) shows the relationship between scene complexity and the average FPS selected by TAPAS across four nuScenes sequences. As spatial entropy increases from 0.5 (Seq 1) to 2.0 (Seq 4), the average FPS correspondingly rises from approximately 8 FPS to 12 FPS, confirming its ability to match processing rates to the complexity of unseen scenes. Figure 13(b) reveals the FPS distribution shift: Seq 1 (low entropy, simple scenes) operates predominantly at 5 FPS (55%), whereas Seq 4 (high entropy, complex environments) requires 15 FPS for 55% of frames. This distribution shift validates our throughput estimator design with a class gap of 5 FPS, which discretizes throughput targets into 5, 10, 15 FPS. This behavior shows that the estimator reliably maps low-complexity frames to low-FPS execution while allocating higher throughput to complex scenes, demonstrating generalization across diverse environments.

(d). Entropy distribution and joint entropy–FPS density: Figure 15(a) and (b) analyzes the throughput estimator using per-sequence entropy KDEs and joint entropy–FPS density distributions across KITTI and unseen nuScenes sequences. Figure 15(a) shows predicted entropy KDEs that closely align with the ground-truth entropy distributions for both KITTI and nuScenes. KDEs are used to capture the full entropy distribution, which is not reflected by simple mean and variance. Across both KITTI and unseen nuScenes sequences, the predicted KDEs closely align with the ground-truth distributions, with only minor discrepancies caused by the discrete FPS levels. Figure 15(b) presents joint entropy-FPS density contours reveal a strong correlation between scene complexity and assigned FPS targets. While nuScenes is shifted toward higher entropy and FPS values, its distribution partially overlaps with KITTI, confirming a harder yet non-disjoint OOD domain. The consistent entropy–FPS ordering across both datasets validates the estimator's ability to generalize the learned mapping.

(e). Robustness analysis on unseen dataset: Energy efficiency and throughput met rate

For robustness evaluation, we use M1, M2, M3, and M5 mixes with increasing compute intensity using a diverse model pool (see Sec. IV-A). The mixes progressively increase system complexity. All experiments are conducted on the target em-

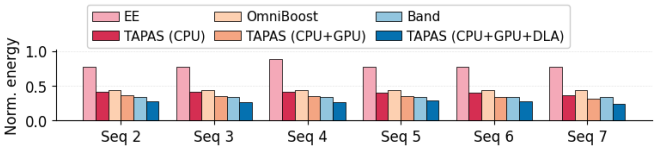

Figure 17. Impact of cluster availability on normalized energy gain

bedded platform (see Sec. IV-A), and we additionally evaluate on the nuScenes dataset to assess the generalization of the throughput estimator on unseen data. We compare TAPAS against SOTA methods (EE [12], OmniBoost [14], Band [13]) across different FPS settings (5, 10, and 15).

Figure 16 shows that *TAPAS* achieves 0.10–0.51 normalized energy while maintaining 93–100% throughput, significantly outperforming static baselines. *EE* incurs the highest energy (1.0 at 15 FPS) and poor throughput ($<$ 52% at 5 FPS, near 0% at 15 FPS) due to CPU-only processing. *OmniBoost* reduces energy (0.24–0.77) and improves throughput (68–72% at 5 FPS), but degrades at higher FPS (13–29%). *Band* performs best among baselines (0.18–0.77 energy; 62–82% at 5 FPS), yet fails under dynamic scene variations due to static FPS allocation. In contrast, *TAPAS* leverages throughput adaptive scheduling, assigning 5 FPS to simple scenes and 15 FPS to complex ones, thereby eliminating unnecessary computation. Lower energy in Mix 1–2 (0.10–0.15) arises from efficient GPU usage, while higher energy in Mix 3–4 (0.27–0.51) reflects heavier models. Overall, by predicting throughput requirements and dynamic configurations, *TAPAS* effectively resolves the energy-throughput trade-off. The slight degradation in Mix 3 (95%) and Mix 4 (93%) is due to these configurations deploying heavier models (e.g., Swin Transformer, DeepLab v3), where even optimal device selection can occasionally fail to meet tight deadlines during FPS variation.

D. Robustness analysis for hardware unavailability

(a). cluster availability. Figure 17 presents the energy consumption of *TAPAS* across three cluster availability configurations: (i) CPU-only (C1), (ii) CPU+GPU (C2), and (iii) CPU+GPU+DLA (C3), compared against EE, OmniBoost, and Band baselines over KITTI test sequences 2-7. Across all sequences and workload mixes, *TAPAS* consistently achieves the lowest energy regardless of cluster availability: *TAPAS* (C1-C3) reduces energy by up to 76%, 55%, and 35% over EE, OmniBoost, and Band respectively. This result demonstrates the robustness of *TAPAS* under varying hardware availability. The adaptivity and GRU-based policy adapt effectively even when only CPU resources are available, while larger gains are expected on heterogeneous platforms with GPUs and DLAs due to the increased performance–energy diversity available for task mapping. The scheduling policy explicitly accounts for cluster unavailability during runtime by adapting model-to-cluster mappings based on the currently available CPU, GPU, and DLA resources, ensuring robust performance even under degraded hardware conditions.

(b). Percentage cluster unavailability. Figure 18 evaluates the throughput met rate under progressive cluster unavailability (0-25%) for seq 7's four region-specific FPS targets (Region 1:

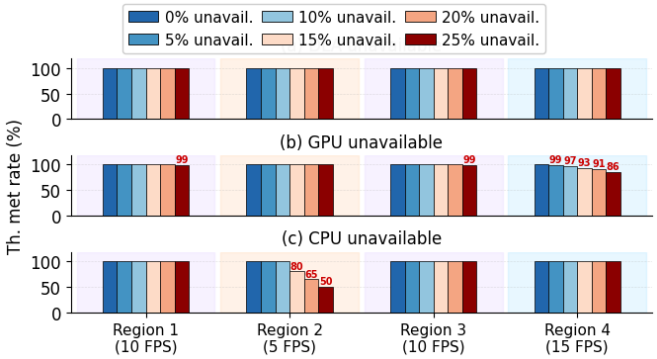

Figure 18. Impact of hardware unavailability on latency compliance of TAPAS. (a). DLA unavailable, (b). GPU unavailable, (c). CPU unavailable. Hardware unavailability is progressively varied from 0-25% per-cluster. Latency compliance is shown per region in Seq 7

10 FPS, Region 2: 5 FPS, Region 3: 10 FPS, and Region 4: 15 FPS). We achieve conditional cluster preemption by introducing controlled cluster freezing intervals of 50, 100, 150, 200, and 250 ms. This time-multiplexed execution translates into temporal unavailability of clusters, enabling systematic evaluation of robustness under varying cluster unavailability. Across all percentage cluster-unavailable scenarios, *TAPAS* demonstrates strong robustness under progressive cluster unavailability (0-25%). DLA unavailability has a negligible impact on throughput met rate, while CPU failures affect only specific regions at higher unavailability levels (15–25%). Throughput met rate remains at 100% for Regions 1-3 across all GPU unavailability settings, with degradation observed only in the highest-demand case (Region 4, 15 FPS), where performance decreases gracefully from 100% to 86% as GPU unavailability increases from 0% to 25%. Intermediate GPU unavailability levels (5%, 10%, 15%, and 20%) result in mild degradation of 1%, 3%, 7%, and 9%, respectively. In contrast, existing SOTA methods typically assume fixed hardware availability and do not explicitly account for cluster unavailability, making them more prone to significant performance degradation under such conditions. The training traces are collected from real-world execution profiles under varying resource availability conditions, enabling the learned policy to handle 5-15% cluster unavailability with negligible degradation.

E. Static vs adaptivity comparison for SOTA.

Figure 19 presents normalized energy consumption of six *KITTI* sequences (Seq 2-7) and five different workload mixes (Mix 1-5). For comparison against *TAPAS*, we use three SOTA strategies – *EE*, *OmniBoost*, *Band* with fixed FPS targets (5, 10, 15 FPS indicated by hatching). *EE* consistently exhibits the highest energy consumption across all sequences and workload mixes. *EE* strategy scales linearly with FPS targets (0.33 at 5 FPS, 0.67 at 10 FPS, 1.0 at 15 FPS), indicating a lack of adaptivity to scene complexity. All SOTA methods process frames at a fixed FPS, regardless of scene complexity, resulting in significantly higher energy consumption. *EE* incurs 79–87% higher energy than *TAPAS*, while *OmniBoost* achieves moderate savings but still consumes 45–65% more energy. *Band* performs best among baselines due to sub-

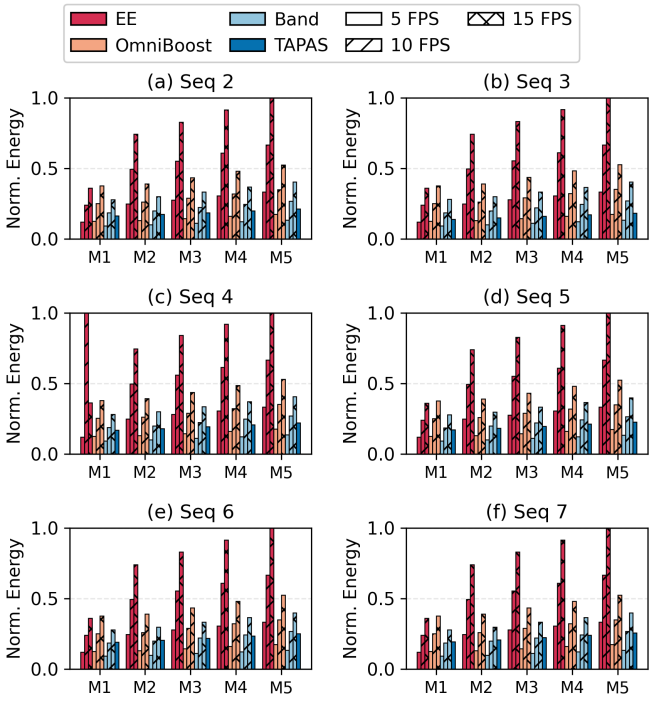

Figure 19. Normalized energy per sequence over different strategies.

graph scheduling, yet fails to adapt to throughput variation across scenes, leaving 35–55% energy savings unrealized. In contrast, model-level mapping is more effective for transformer execution in our deployment setting, providing better system-level efficiency than subgraph-level scheduling. *TAPAS* achieves the lowest energy by coupling scene awareness with dynamic model-to-cluster mapping, guided by a GRU-based agent with throughput estimation. Its energy scales with scene complexity, delivering up to 87%, 72%, and 58% energy reduction over *EE*, *OmniBoost*, and *Band*, respectively. This shows the advantage of adaptive throughput over fixed FPS with static mapping.

Figure 20 (a) and (b) jointly illustrate normalized energy consumption and throughput met rate for fixed 5/10/15 FPS targets with EE, OmniBoost, and Band, alongside the adaptive TAPAS strategy, across different mix combinations. *EE* exhibits linear energy scaling (2× at 10 FPS, 3× at 15 FPS) due to fixed FPS processing and static mapping, resulting in 69–90% higher energy than *TAPAS* and poor throughput met rate ($<$ 45% at 5 FPS and near 0% at 15 FPS). *OmniBoost* improves energy (24–77% vs EE) via CPU–GPU pipelining but achieves only moderate throughput met rate at 5 FPS (22–48%) and degrades significantly at 15 FPS (5–18%) due to lack of runtime adaptivity. *Band* leverages accelerators to reduce energy (18–77% vs EE-OmniBoost) and achieves comparable throughput met rate result to *TAPAS* at 15 FPS (93–100%) but at a significantly higher energy cost. *Band* fails to adapt to varying demands, resulting in a low throughput-met rate at 5 FPS (22–45%) and a moderate rate at 10 FPS (63–78%), thus either missing targets or over-consuming energy. In contrast, *TAPAS* addresses this limitation by dynamically aligning computation with scene-driven FPS

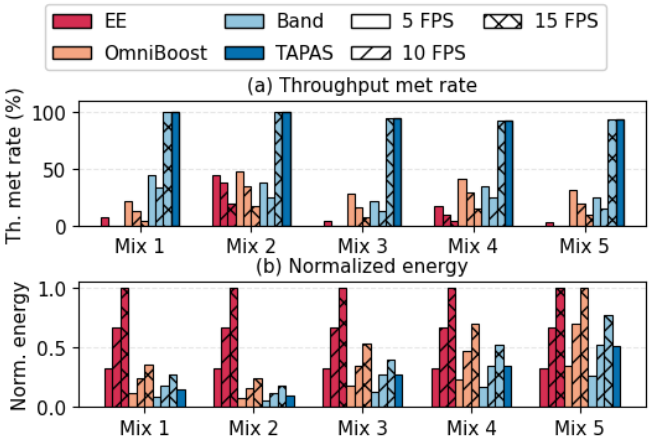

Figure 20. Throughput met (%) and energy with *EE*, *Omniboost*, *Band*, and *TAPAS* at 5, 10 and 15 FPS with mix-1-5.

targets rather than relying on a fixed FPS with a static mapping. It assigns lower FPS to simple scenes and scales to higher FPS only when the scene demands it, achieving 33–44% energy savings over *Band* (15 FPS) while maintaining 93–100% throughput met rate. This is enabled by coupling scene awareness with GRU-based temporal prediction and dynamic model-to-cluster mapping, RRM-based multi-objective decision-making, effectively utilizing available hardware only as required, and resolving the energy–throughput trade-off.

V. Related Work

Modern perception pipelines include concurrent tasks of object detection, semantic segmentation, obstacle avoidance, and visual odometry [2], [33]. This creates significant computational and energy demands, making adaptive perception crucial for mobile autonomous systems [5], [7], [34]. We classify existing perception strategies that adapt to scene-awareness and hardware heterogeneity as follows.

Scene-aware adaptivity. Existing strategies learn and leverage scene-awareness for adaptive perception. Roborun [8] develops spatial-awareness for adaptive perception by prioritizing regions of interest within frames. ADAVP [20], Marlin [35], and SHIFT [9] use rate of change in context for scene-awareness, and dynamically select appropriate DNN model settings for adaptivity. In contrast, other context-driven adaptive approaches [36] are primarily simulation-based and lack validation on real heterogeneous edge AI platforms for scene-aware throughput adaptation in AS. However, aforementioned approaches address only a single vision task and do not consider a comprehensive perception pipeline. Context-based adaptation in these approaches degrades perception quality through model approximation, substituting lightweight or compressed DNN variants when conditions permit, rather than dynamically adjusting throughput targets for existing perception modules. Existing approaches lack mechanisms for estimating throughput, which can adjust FPS based on scene demands

Hardware-aware adaptivity. Existing works [11], [12] allocate the entire perception module to CPU, overlooking HMP's low-cost accelerators such as GPU and NVIDIA Deep Learning Accelerator (NVDLA), and consume significantly

more energy. In contrast, the coordinated approaches [14], [18] extend workload allocation to both CPU and GPU but neglect energy considerations entirely. As design-time solutions, these strategies use fixed allocations that do not adapt to run-time variations in throughput, leading to high energy consumption. Recent works [13], [15] consider Neural Processing Unit (NPU) and NVDLA for resource allocation with basic DNN-specific operator support, while overlooking the inefficiency of transformer inference on NVDLA. Although BAND [13] enables finer-grained subgraph-level mapping, such granularity is not effective for transformer workloads in our setting, where system-level efficiency is better achieved through model-level assignment. Existing methods exhibit critical limitations: (1) inability to handle run-time throughput variations [12]–[14]; (2) low energy efficiency for AS; (3) improper allocation of perception modules on HMP containing low-cost accelerators; [11], [14], [18] and (4) failure to address multi-DNN and transformer scheduling challenges on low-cost accelerators [12], [13], [15], [18]. *TAPAS* overcomes these gaps through run-time adaptive resource scheduling for AS's perception module, while handling energy-aware run-time variable throughput requirements.

VI. Conclusion

We present *TAPAS*, a throughput-adaptive perception framework for AS on mobile/edge HMPs. TAPAS employs spatial entropy for runtime FPS estimation and a GRU-based RL agent with RRM for optimal task-to-cluster mapping, thereby jointly optimizing throughput met rate and energy consumption. Deployed and evaluated on Jetson Orin NX, TAPAS achieves 93–100% throughput met rate with up to 76% energy savings on KITTI sequences, and generalizes to unseen nuScenes with 97% throughput met rate and 64% lower energy. Future work will explore model approximation and DVFS for fine-grained energy optimization.

References

[1] G. Malczyk, M. Kulkarni, and K. Alexis, "Semantically-driven deep reinforcement learning for inspection path planning," *IEEE Robotics and Automation Letters*, vol. 10, no. 7, pp. 7206–7213, 2025.

[2] A. O. Françani and M. R. O. A. Maximo, "Transformer-based model for monocular visual odometry: A video understanding approach," *IEEE Access*, vol. 13, p. 13959–13971, 2025.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[4] S. M. Neuman, R. Ghosal, T. Bourgeat, B. Plancher, and V. J. Reddi, "Roboshape: Using topology patterns to scalably and flexibly deploy accelerators across robots," in *Proc. of Int. Symp. on Computer Architecture*, ser. ISCA '23, 2023.

[5] Y.-S. Hsiao, S. K. S. Hari, M. Filipiuk, T. Tsai, M. B. Sullivan, V. J. Reddi, V. Singh, and S. W. Keckler, "Zhuyi: perception processing rate estimation for safety in autonomous vehicles," in *Proc. of ACM/IEEE Design Automation Conf. (DAC)*, 2022, pp. 289–294.

[6] D. Falanga, S. Kim, and D. Scaramuzza, "How fast is too fast? the role of perception latency in high-speed sense and avoid," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1884–1891, 2019.

[7] H. Zhao, Y. Zhang, P. Meng, H. Shi, L. E. Li, T. Lou *et al.*, "Driving scenario perception-aware computing system design in autonomous vehicles," in *Proc. of IEEE Int. Conf. on Computer Design (ICCD)*, 2020, pp. 88–95.

[8] B. Boroujerdian, R. Ghosal, J. Cruz, B. Plancher, and V. J. Reddi, "Roborun: A robot runtime to exploit spatial heterogeneity," in *Proc. of ACM/IEEE Design Automation Conf. (DAC)*, 2022.

[9] J. Davis and M. E. Belviranli, "Context-aware multi-model object detection for diversely heterogeneous compute systems," in *Proc. of IEEE Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2024, pp. 1–6.

[10] N. Corporation, "Working with dla," 2025. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/work-with-dla.html

[11] S. A. Mohamed, M.-H. Haghbayan, A. Miele, O. Mutlu, and J. Plosila, "Energy-efficient mobile robot control via run-time monitoring of environmental complexity and computing workload," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2021, pp. 7587–7593.

[12] S. Shahsavari, H. Haghbayan, A. Miele, E. Immonen, and J. Plosila, "A coordinated approach to control mechanical and computing resources in mobile robots," *IEEE Trans. on Robotics*, vol. 41, pp. 347–363, 2025.

[13] J. S. Jeong, J. Lee, D. Kim, C. Jeon, C. Jeong, Y. Lee *et al.*, "Band: coordinated multi-dnn inference on heterogeneous mobile processors," in *Proc. of Int. Conf. on Mobile Systems, Applications and Services (MobiSys)*, 2022, p. 235–247.

[14] A. Karatzas and I. Anagnostopoulos, "Omniboost: Boosting throughput of heterogeneous embedded devices under multi-dnn workload," in *Proc. of ACM/IEEE Design Automation Conf. (DAC)*, 2023, pp. 1–6.

[15] I. Dagli, A. Cieslewicz, J. McClurg, and M. E. Belviranli, "Axonn: Energy-aware execution of neural network inference on multi-accelerator heterogeneous socs," in *Proc. of ACM/IEEE Design Automation Conference*, 2022, pp. 1069–1074.

[16] W. Lian, W. Lian, and Z. Luo, "Equipping diffusion models with differentiable spatial entropy for low-light image enhancement," in *Proc. of IEEE/CVF Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2024, pp. 6671–6681.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[18] Z. Taufique, A. Vyas, A. Miele, P. Liljeberg, and A. Kanduri, "Tango: Low latency multi-dnn inference on heterogeneous edge platforms," in *Proc. of Int. Conf. on Computer Design (ICCD)*, 2024, pp. 300–307.

[19] J. Guo, Z. Chi, L. Dong, Q. Dong, X. Wu, S. Huang, and F. Wei, "Reward reasoning model," 2025. [Online]. Available: https://arxiv.org/abs/2505.14674

[20] M. Liu, X. Ding, and W. Du, "Continuous, real-time object detection on mobile devices without offloading," in *Proc. of IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, 2020, pp. 976–986.

[21] M. Dharmadhikari, N. Khedekar, P. D. Petris, M. Kulkarni, M. Nissov, and K. Alexis, "Maritime vessel tank inspection using aerial robots: Experience from the field and dataset release," 2024. [Online]. Available: https://arxiv.org/abs/2404.19045

[22] L. Altieri, D. Cocchi, and G. Roli, "The use of spatial information in entropy measures," 2017. [Online]. Available: https://arxiv.org/abs/1703.06001

[23] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. [Online]. Available: https://arxiv.org/abs/1412.3555

[24] A. Gupta, A. Pacchiano, Y. Zhai, S. M. Kakade, and S. Levine, "Unpacking reward shaping: understanding the benefits of reward engineering on sample complexity," in *Proc. of Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 2022.

[25] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," 2023. [Online]. Available: https://arxiv.org/abs/2303.00001

[26] A. Yang *et al.*, "Qwen2 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2407.10671

[27] Z. Ji *et al.*, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, p. 1–38, Mar. 2023.

[28] NVIDIA, "Nvidia jetson Orin," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, 2024.

[29] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[30] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," 2020.

[31] A. Behroozi, Y. Chen, V. Fruchter, L. Subramanian, S. Srikanth, and S. Mahlke, "Slimslam: An adaptive runtime for visual-inertial simultaneous localization and mapping," in *Proc. of ACM Int. Conf.*

on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 3, 2024, p. 900–915.

[32] F. G. Blanco *et al.*, "A deep reinforcement learning based online scheduling policy for deep neural network multi-tenant multi-accelerator systems," in *Proc. of ACM/IEEE Design Automation Conf. (DAC)*, 2024.

[33] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017, p. 2043–2050.

[34] H. Zhao and et al, "Suraksha: A framework to analyze the safety implications of perception design choices in avs," in *Proc. of IEEE Int. Symp. on Software Reliability Engineering (ISSRE)*, 2021, pp. 434–445.

[35] K. Apicharttrisorn, X. Ran, J. Chen, S. V. Krishnamurthy, and A. K. Roy-Chowdhury, "Frugal following: power thrifty object detection and tracking for mobile augmented reality," in *Proc. of ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2019, p. 96–109.

[36] S. Jambotkar, L. Guo, and Y. Jia, "Adaptive optimization of autonomous vehicle computational resources for performance and energy improvement," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 7594–7600.