2

Towards Smart and Adaptive Agents for Active Sensing on Edge Devices

Devendra Vyas[1], Nikola Pižurica[2,3] Nikola Milović[3], Igor Jovančević[2,3], Miguel de Prado[1], and Tim Verbelen[1]

[1] VERSES, USA
[2] Computer Science Center, University of Montenegro, Montenegro
[3] Fain Tech, Montenegro

Abstract

TinyML has made deploying *deep learning* models on low-power edge devices feasible, creating new opportunities for real-time perception in constrained environments. However, the adaptability of such deep learning methods remains limited to data drift adaptation, lacking broader capabilities that account for the environment's underlying dynamics and inherent uncertainty. Deep learning's scaling laws, which counterbalance this limitation by massively up-scaling data and model size, cannot be applied when deploying on the *Edge*, where deep learning limitations are further amplified as models are scaled down for deployment on resource-constrained devices.

This paper presents an innovative agentic system capable of performing on-device perception and planning, enabling active sensing on the edge. By incorporating *active inference* into our solution, our approach extends beyond the capabilities of deep learning, allowing the system to plan in dynamic environments while operating in real-time with a compact memory footprint of as little as 300 MB. We showcase our proposed system by creating and deploying a Saccade agent connected to an IoT camera with pan and tilt capabilities on an NVIDIA Jetson embedded device. The Saccade agent controls the camera's field of view following optimal policies derived from

the active inference principles, simulating human-like saccadic motion for surveillance and robotics applications.

Keywords: smart agents, edgeAI, dynamic planning, active inference.

2.1 Introduction

The human visual system has a unique ability to focus on key details within complex surroundings, a process known as saccading [1]. This quick and dynamic scanning allows us to gather essential information. Saccading is part of a larger concept known as active (visual) sensing [2, 3], an innate capability that enables organisms to forage for information and dynamically adapt to an evolving environment [4].

Active sensing is critical in various applications, particularly when the information is unavailable or too vast to process. For instance, in remote sensing for Earth observation [5], or aerial search-and-rescue operations [6], the system must parse vast, detailed scenes, focusing only on critical features, such as sea ice or missing persons. Similarly, in sports events, tracking dynamic scenes requires a system to zoom in on players, capturing players' faces or gestures while not missing the play. Other areas, such as smart cities and surveillance systems [7], demand robust monitoring solutions for crowded areas to track movement, anticipate potential issues, and enhance safety. Active sensing becomes even more apparent in robotics, where the agent's actions determine the next observations for the system, driving exploration [8].

Recent advances in machine learning (ML), particularly in deep learning, have substantially improved sensing accuracy and complexity. However, state-of-the-art deep learning models exhibit limitations in their adaptability [9], specifically in the ongoing accumulation and refinement of knowledge over time. These limitations are further amplified when these models are scaled down for deployment on resource-constrained devices, where memory, computational power, and energy efficiency are limited. As a result, true active sensing—requiring both perception and planning—remains challenging to implement on embedded systems.

Active inference, an approach rooted in the first principles of physics, offers a promising alternative to address these limitations [10]. Emerging as a viable paradigm, active inference grounds learning within probabilistic principles, enabling smart systems, or agents, to model the uncertainty and variability inherent in dynamic environments, making it well-suited for

continual learning and adaptive decision-making [11]. This shift represents a move beyond perception-focused AI toward adaptive systems capable of adjusting their actions based on environmental feedback. Thus, agents on the edge provide a powerful framework for real-time perception and planning, eliminating dependence on cloud resources and ensuring low-latency responses and enhanced data privacy.

This work presents an integrated system that combines deep learning and active inference to realize an adaptive, memory-efficient, real-time Saccade agent for edge devices. Our system leverages a deep learning-based object detection module for initial perception and an active inference planning module to actively sense and adapt to the environment. The Saccade agent can observe, plan, and control a camera for strategic information gathering, demonstrating adaptive decision-making and exploration. Our deployment on an Nvidia Jetson platform showcases the potential for responsive applications in robotics and smart city environments, highlighting the feasibility of edge-based adaptive systems for complex, real-world tasks.

2.2 Related work

We categorize the related work in three main areas as described below.

2.2.1 Active Sensing

Active sensing is an essential building block across various fields where efficient scanning is necessary to locate and focus on critical details. Historically, the roots of the active sensing concept can be traced back several decades, particularly in the field of computer vision. R. Bajcsy [30] defined active sensing as an intelligent data acquisition process. In this context, a passive sensor, such as a camera, is actively used by purposefully controlling its state parameters (e.g., pose, zoom, and focus) in accordance with task goals. This view emphasizes the closed-loop feedback between perception and action, and distinguishes between local models of sensor/algorithm properties and global models that coordinate sensing strategies over time. Furthermore, [31] and [32] formalized active and "animate" vision, demonstrating that an observer who can control their viewpoint and gaze (e.g., via saccading and pan–tilt camera motions) can convert ill-posed vision problems into well-posed ones and drastically reduce computational demands by aligning sensing with task constraints.

Building upon these foundational ideas, subsequent work in robotics has developed active vision systems that plan where and how to look to support manipulation, navigation, and scene understanding. Surveys on active vision in robotic systems and view-planning highlight methods that select camera poses that constrain scene interpretation and trade off coverage, accuracy, and computational cost [33]. More recent approaches explicitly formulate active sensing as an information-gain or mutual-information maximization problem, for instance, in next-best-view selection for volumetric 3D reconstruction or multi-camera surveillance [34] for public safety in smart cities. This, combined with other deep learning methods for people tracking and counting [12], enables security, crowd management, and urban analytics applications.

Other fields, such as Earth observation, employ active sensing sea ice detection by maximizing area coverage and detection through adaptive zooming, which is indispensable for safe navigation and has prompted the AutoICE Challenge benchmark [5]. Similarly, active search strategies are also explored in rescue operations [6], emphasizing techniques such as saccading to enhance search efficiency over large areas.

Most of the methods share similarities with our work, but they optimize handcrafted information-theoretic criteria rather than deriving sensing policies from a unified active inference generative model.

2.2.2 TinyML

TinyML has made deploying ML models on low-power edge devices feasible, bringing opportunities for real-time perception in constrained embedded devices. The edge deployment pipeline is summarized in [13], streamlining the end-to-end model deployment on embedded platforms to enhance the accessibility of edgeAI applications. A popular example of this process is YOLO [14], an efficient single-stage detector that enables object detection and tracking at the edge, thereby improving the responsiveness of embedded applications. Recent works have made progress in enabling on-device domain adaptation, adjusting deployed applications to account for data distribution shifts between training and target environments [15, 16]. However, these adaptations remain limited to addressing data drifts and lack broader capabilities for behavioral changes.

Our approach overcomes this limitation by integrating active inference on top of a deep learning module, allowing the system to plan and adapt to the environment.

2.2.3 Probabilistic computing

Probabilistic computing has shown promise for active sensing by optimizing information acquisition in dynamic environments. Probabilistic principles can be utilized to maximize information gain through camera adjustments [17], which is particularly valuable in applications such as surveillance, sports analysis, and patient monitoring. This is extended to maximize mutual information gain in multi-camera setups, combining objectives like exploration and tracking to enable adaptive scene monitoring [18]. Unlike these methods, our probabilistic agent is based on active inference, grounding our approach in the Free Energy Principle.

2.3 Methodology

To develop an effective active sensing solution, it is essential to consider the unpredictable and dynamic nature of real-world environments. Smart sensors must be able to handle uncertainty and adapt to constant changes. Therefore, any change in the observed environment must influence the policy selection for the following action. For a system to operate autonomously and intelligently, it must be able to adjust its perception and actions in real time without relying on cloud processing. This requirement for on-device adaptation supports faster decision-making and enhances data privacy.

In this work, we propose an efficient active sensing agent composed of two modules: i) a deep learning-based perception module and ii) an active inference module that enables planning and control. This architecture combines deep learning's feature extraction performance with active inference's Bayes-optimal control, presenting an adaptable and scalable solution for various resource-constrained edge applications.

2.3.1 Perception

Deep learning techniques have achieved remarkable success in detecting features of interest in images, audio, or textual data [19]. Convolutional neural networks (CNNs) have demonstrated exceptional performance in visual tasks like object detection and segmentation [20]. By employing deep learning for perception, active sensing agents can rapidly process high-dimensional data and identify patterns that provide relevant spatial information.

Recent advances in transformer architectures and large language models (LLMs) have further expanded the scope of deep learning. Transformers excel

at capturing complex dependencies and long-range relationships in data, making them highly effective for tasks that require a deeper contextual understanding. The self-attention mechanism, a core component of transformers, enables models to focus selectively on the most relevant aspects of the data, thereby enhancing the agent's ability to perform active sensing by prioritizing critical visual cues.

However, while deep learning and LLMs offer powerful feature extraction, their static nature limits adaptability when deployed in uncertain or changing environments, which can only be counteracted by massively scaling data or model size. Thus, to address this challenge, our approach integrates deep learning for perception but relies on active inference as a much lighter, sample- and parameter-efficient higher-level module, enabling the agent to plan and dynamically adapt based on ongoing observations in real-time.

2.3.2 Planning

Active inference builds on the Free Energy principle, a theoretical framework that states intelligent agents minimize the discrepancy between their internal (generative) model of the environment and incoming sensory data. By reducing this discrepancy, or "free energy", the agent maintains an updated and accurate representation of its surroundings, enabling it to make predictions about its environment and actively take actions to reduce uncertainty.

Concretely, the agent's generative model is a joint probability distribution over states s and observations o. As inferring hidden states s given some observations o is typically intractable, an approximate posterior $Q(s|o)$ is introduced and optimized by minimizing the free energy F [10]:

$$\min_{Q(s|o)} F = \underbrace{D_{KL}[Q(s \mid o) \| P(s)]}_{\text{complexity}} - \underbrace{\mathbb{E}_{Q(s|o)}[\log P(o \mid s)]}_{\text{accuracy}} \tag{2.1}$$

Hence, the agent strives to provide the most accurate predictions $P(o|\ s)$ while minimizing the complexity of the model with respect to the prior $P(s)$. To select actions or policies π for the future τ, an active inference agent will evaluate and minimize the expected free energy G [10]:

$$G(\pi) = \underbrace{\mathbb{E}_{Q(o_\tau|\pi)}\left[D_{KL}\left[Q\left(s_\tau \mid o_\tau, \pi\right) \| Q\left(s_\tau \mid \pi\right)\right]\right]}_{\text{(negative) information gain}} - \underbrace{\mathbb{E}_{Q(o_\tau|\pi)}\left[\log P\left(o_\tau \mid C\right)\right]}_{\text{expected utility}} \tag{2.2}$$

The agent now averages across expected future outcomes $o\tau$ and balances the expected information gain (i.e., exploration) with the expected utility (i.e., reward) encoded in prior preferences C. Both the observations o and hidden states s can be modeled as discrete variables with Categorical distributions, whereas optimizing F and G can be done using tractable update rules [11]. Therefore, we convert the outputs of the deep learning perception module into a discrete observation space and utilize active inference for action selection in our agent.

2.4 Smart edge agent

Our smart edge agent, the Saccade agent, comprises two modules, as introduced in the previous section, which combine deep learning perception capabilities with active inference planning, as shown in Figure 2.1. Specifically, we employ i) a deep-learning object detection model, offering efficient object and human detection capabilities directly at the edge, and ii) an active inference module that enables adaptive motion control (pan and tilt), allowing the agentic system to dynamically adjust its field of view or track detected entities autonomously. This adaptive behavior enhances the camera's utility as an intelligent surveillance IoT tool or for scene exploration and information foraging in robotic applications.

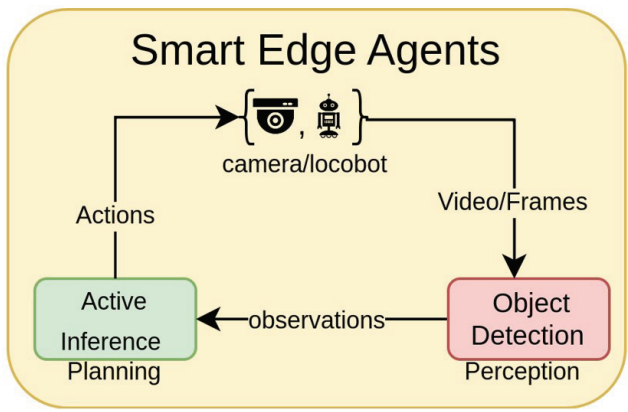

Figure 2.1 Conceptual Framework for Smart Edge Agents, composed of a deep-learning perception module and an active inference planning module for active (visual) sensing. The camera frames are processed by the object detector, which forwards the detected results to the active inference module. Our agent plans its next action, minimizing free energy, and dynamically adapting to the environment.

2.4.1 Object detection using YOLOv10

We chose *YOLOv10* [14] from the YOLO family for our perception module due to its strong balance of detection accuracy and computational efficiency. YOLO models are single-pass object detectors that predict object categories and locations, making them ideal for real-time applications. YOLOv10 consistently demonstrates state-of-the-art performance and reduced latency across various model scales (N/S/M/L/X). We employ the Nano (N) variant, with 2.3 million parameters, as it offers an efficient accuracy-speed-memory trade-off for edge deployment.

To deploy the YOLOv10n as efficiently as possible, we export it to *ONNX* [21]. ONNX has become a standard for neural network representation and exchange, and is widely supported by the software stacks of hardware vendors, including inference engines. This positions ONNX as a strong candidate for deployment in space exploration on a range of embedded devices. Thus, we create an *edge-deployment workflow* to find the most suitable deployment for YOLOv10n.

The edge-deployment workflow comprises several edge-oriented inference engines, including ONNX-runtime [22], TensorFlow Lite [23], and TensorRT [24], which can take an ONNX model as input and generate an optimized implementation for a specific target hardware platform. These frameworks apply several optimizations across the network's graph, e.g., operator fusion, quantization, on the software stack, e.g., algorithm optimization, and leverage parallel hardware acceleration, vectorization, and optimized memory scheduling. The process yields a bespoke network description and a runtime that is ready for deployment on the specified hardware platform.

2.4.2 Planning using Active Inference

To enable an efficient Saccade agent with active inference planning, we define a discrete action, observation, and hidden state space. To this end, we divide the full area the camera can pan and tilt into a discrete grid of $K \; x \; L$ blocks, as shown in Figure 2.2. Given a particular fixation point, the camera's field of view will only span $W \; x \; H$ blocks, highlighted in blue. For each block, at each timestep, an observation $o_{w,\,h}$ is a Categorical distribution with three bins, i.e., the block can have no object detected (0 - blue), an object detected (1 - red), or not visible (2 - gray). The confidence of the bounding box outputs of the object detection neural network provides the probability of an object being detected. As state space, we similarly have a state variable $s_{k,\,l}$ per block, which is Bernoulli, i.e., object not present (0) or present

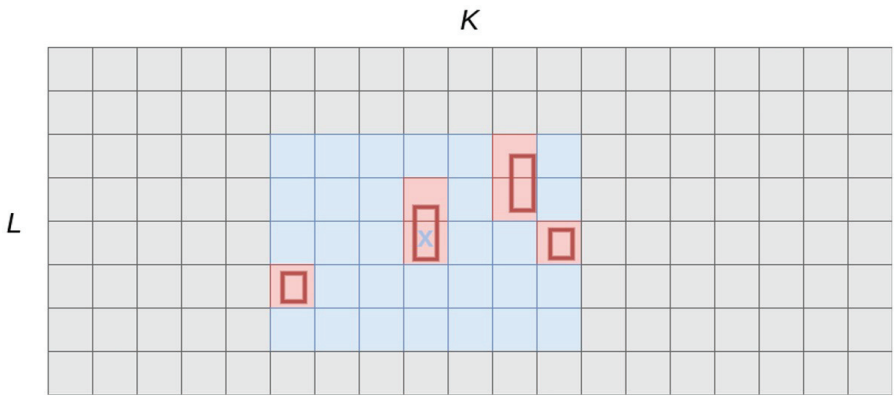

Figure 2.2 **Action space:** We discretize the action space into K x L fixation points. Given a fixation point, the field of view of the camera spans W x H blocks (in blue), and object detections are translated into discrete bins (in red).

(2.1). In addition, we also equip the agent with a proprioceptive state s_p and observation, i.e., it observes the fixation point it is currently looking at. We specify the likelihood mapping $\mathbf{A}$, which predicts observation $o_{w,\,h}$ given state $s_{k,\,l}$ and fixation point s_p:

$$A_{w,h,k,l,p} = \begin{cases} 0 & \text{if } s_{k,l} = 0, k \rightarrow_p w, l \rightarrow_p h \\ 1 & \text{if } s_{k,l} = 1, k \rightarrow_p w, l \rightarrow_p h \\ 2 & \text{otherwise} \end{cases} \tag{2.3}$$

Where $k \rightarrow_p w$ means "block k maps to observation w given the agent looks at the fixation point p". We currently also assume that objects do not move considerably between timesteps and use the previous timestep posterior as the current prior, i.e.,

$$P(s_t) = Q(s_{t-1}|o_{t-1})$$

We leave the expansion of this modeling assumption for the dynamics of the objects as future work.

The Saccade agent uses the object detections received to perform inference, updating its beliefs about the hidden states. For example, detecting a "person" with high confidence would increase the probability assigned to the hidden state "person present" corresponding to the particular block. The absence of detection, on the other hand, would decrease the probability of that object being present.

After each observation, the agent evaluates possible actions, i.e., the next fixation points, based on their predicted outcomes. Actions are chosen to minimize expected free energy, which combines two key factors:

1. **Expected observations matching prior preferences:** This essentially means selecting the most likely actions that are expected to lead to desired outcomes. For example, if the agent's goal is to locate a specific object, we set a preference $\boldsymbol{C}_{w,h} = 1$, which favors actions that increase the probability of detecting that object in the field of view. Similarly, if we set $\boldsymbol{C}\mathrm{c}_{w,}\ \mathrm{c}_h = 1$, with $(\mathrm{c}_w, \mathrm{c}_h)$ the center coordinates of the camera, this yields a "tracking" agent that pans/tilts to keep the object of interest centered.
2. **Epistemic value:** The agent also aims to reduce uncertainty about the hidden states. Actions expected to provide more informative observations about the environment have higher epistemic value (i.e., information gain, as defined in Eq. (2.2)). In our model, moving the field of view to previously unobserved blocks will result in a high amount of information gain. Hence, in the absence of objects of interest, the camera will pan and tilt to cover the whole area with as few moves as possible.

We provide a qualitative demonstration of our Saccade agent here [29].

2.5 Experimental results

2.5.1 Experimental Setup

Our experimental setup comprises the following components (see Figure 2.3):

1. **Tapo Camera:** We employ an IoT Tapo camera equipped with pan and tilt capabilities, which serves as both the input for the observations, i.e., images/frames, forwarded to the object detector, and the actuator for our Saccade agent. The agent commands the camera to perform dynamic adjustments in its field of view based on optimal policies derived by minimizing free energy, simulating a human-like saccadic motion to maintain focus on areas of interest.
2. **Locobot WX250:** We also deploy our agent on the Locobot WX250, a mobile robot platform for autonomous mobility. Equipped with a 6-DOF manipulator and pan/tilt camera, the Locobot enables active exploration of the environment. This capability complements the Tapo camera's pan and tilt actions with navigation. The robot's mobility enhances the agent's capacity for active sensing in new environments, allowing it to

Figure 2.3 Applications: Our agentic system enables active sensing solutions for edge robotics and surveillance IoT cameras. On the left, the Tapo IoT camera [27] is used for surveillance applications. On the right, the Locobot robot WX250 [28] for information gathering and scene discovery.

reposition itself and collect additional perspectives to refine perception in complex settings.

3. **Nvidia Jetson Orin NX:** Our setup leverages the Nvidia Jetson Orin NX. Equipped with an 8-core ARM Cortex-A78 CPU and a 1024-core Ampere GPU, the Jetson Orin NX represents a powerful edge AI platform designed for accelerated machine learning tasks, offering up to 100 TOPs of processing capability and 16 GB of memory, all while operating at 25W.

A pre-trained YOLOv10n model, trained on the COCO dataset [25], is deployed on the Nvidia Jetson Orin NX for real-time detection. The active inference module receives bounding boxes as observations and returns the optimal pan and tilt actions for the IoT or robotic camera. Both the perception and planning modules utilize the edge-deployment workflow to identify the most suitable deployment engine, resulting in the highest performance.

2.5.2 Results

It is worth noting that this work does not aim to provide a quantitative benchmarking of behavioral performance against alternative baselines (e.g., algorithms based on other probabilistic paradigms, or simple heuristics). While such comparative evaluations are important, the primary objective of this study is to validate the feasibility of deploying an active inference agent on resource-constrained edge hardware. For this reason, our experimental

focus was on system-level optimizations for latency and memory footprint, which directly determine real-world deployability. These metrics represent hard constraints in embedded and IoT settings and constitute a necessary prerequisite for any future large-scale behavioral benchmarking. A systematic comparison of task-level scores (e.g., accuracy or exploration efficiency) across alternative methods is therefore left as an important direction for future work.

Next, we summarize our deployment experimental results:

2.5.2.1 Perception

We optimized the YOLOv10n model, as introduced in Section 1.4, by exporting the original Torch model from the original *Ultralytics*, a training-oriented framework, to ONNX. Then, the model gets compiled with one of the various deployment inference engines. Figure 2.4 illustrates the average results of 1,000 inference runs, with one warm-up sample, when deploying each implementation on the Nvidia Jetson's CPU, single-core, and multi-core, as well as the GPU.

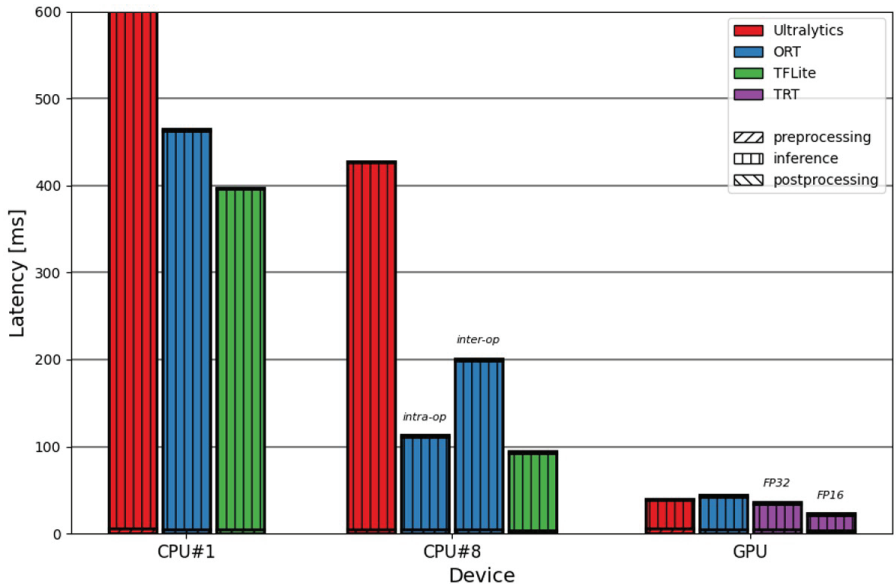

Figure 2.4 Perception module performance: Optimization achieved by exporting the YOLOv10n Torch model from Ultralytics to ONNX and compiling it with different deployment inference engines on the Nvidia Jetson Orin NX's CPU and GPU.

When evaluating the various inference engines using single-floating-point precision (FP32) operations on a single-core CPU, Ultralitics emerges as the slowest, with a latency of over 600 ms. When the model is exported to ONNX and compiled with ONNX-runtime (ORT) and TensorFlow Lite (TFLite), the latency decreases considerably by 23% and 35%, respectively. We achieve higher gains when parallelizing across all available 8 CPU cores, resulting in a 427 ms reduction when using Ultralytics. The performance can be notably increased if we deploy with ORT (intra-operator parallelization) and TFLite, with a 3.75x and 4.5x speed-up over Ultralytics, respectively.

Finally, when offloading inference to the GPU, we observe a performance boost for all GPU-available frameworks by leveraging the device's native parallel compute units, achieving inference times under 45 ms. When using TensorRT (TRT), which is specifically designed for Nvidia Jetson devices, we achieve optimized deployment through reduced numerical precision (FP16), with as little as 22 ms, resulting in 45 FPS.

Figure 2.5 shows the memory profile of the YOLOv10n deployments with the various inference engines. It can be observed that while TFLite provides the fastest deployment on the single-CPU setup, it also consumes the most

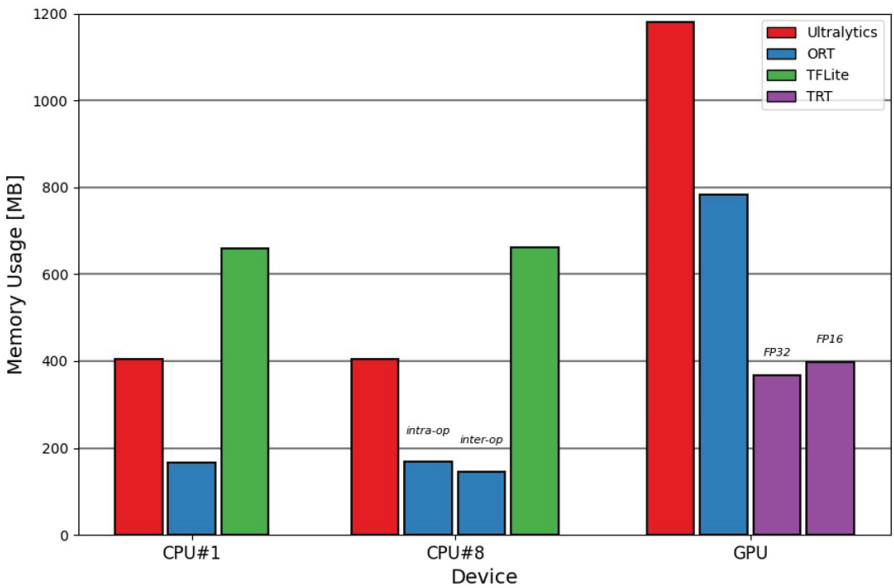

Figure 2.5 Perception module memory profile of the YoloV10n when executed with different deployment inference engines on the Nvidia Jetson Orin NX's CPU and GPU.

memory, with over 50% more than Ultralytics. On the other hand, ORT provides a good latency-memory balance, being slightly slower than TFLite, but achieves inference with as little as 168 MB of memory. This pattern is also repeated in the multi-CPU deployment setup. On the GPU side. Ultralytics, being training-oriented, consumes the most memory, with up to 1.2 GB. As we discussed earlier, TRT is designed explicitly for Nvidia Jetson devices. As such, it accomplishes a very performant deployment with an optimized memory usage under 400 MB.

Overall, the results demonstrate that the neural network inference stage makes the predominant contribution to overall latency. In contrast, the pre-processing phase exhibits minimal computational overhead, and the post-processing step remains negligible — a characteristic intrinsic to the YOLOv10 architecture. Moreover, the comparative analysis across inference engines and hardware configurations reveals distinct performance and memory usage behaviors. For instance, TFLite achieves favorable latency on single-core execution, although with the highest memory usage, while ORT offers an excellent performance-memory tradeoff. On the GPU side, TRT is the clear winner, offering excellent latency and memory performance. These observations highlight the importance of a flexible edge-deployment workflow that systematically identifies the most suitable software–hardware pairing for a given operational context.

2.5.2.2 Planning

Next, we deploy the active inference planning module, which only contains 1.34K parameters, on the Nvidia Jetson NX. The original active inference model was designed with the *Pymdp* library [26], which contains a JAX backend. We then follow a similar workflow to that of YOLOv10n, exporting the active inference model to ONNX and deploying it with several inference engines.

Figure 2.6 depicts the latency of the active inference model when planning the following position to look at. The pre-processing step accounts for decoding and mapping the object detection bounding boxes to the generative model. The inference part involves the *infer_states* method, which updates the agent's beliefs about the hidden states given the observations, and the *infer_policies* method, which defines the set of policies from which the agent will pick the best action.

When deployed on a single CPU, the planning process executes fast, achieving 6 ms using JAX and 4 ms (250 FPS) when compiled with TFLite. The Saccade agent does not benefit from parallel processing, showing similar

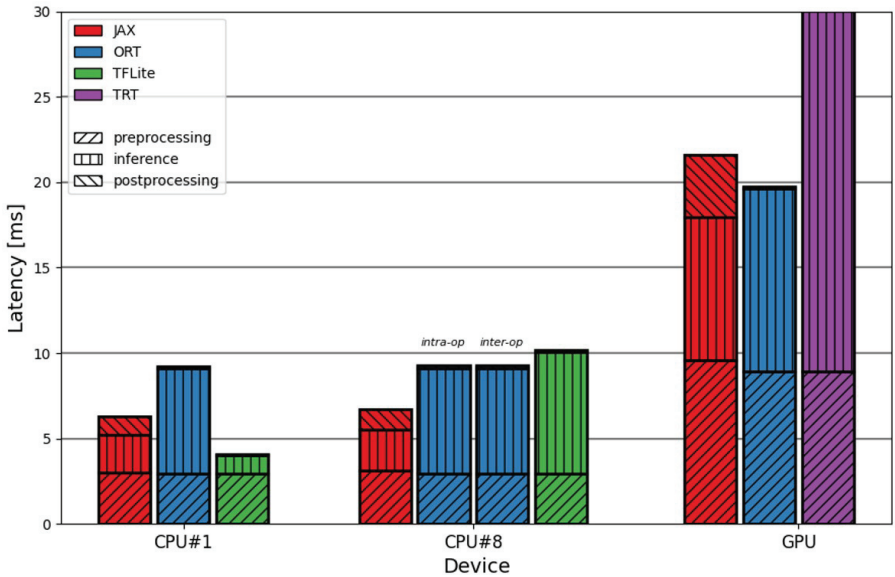

Figure 2.6 Planning module performance: Performance optimization by exporting the active inference model (saccade agent) from JAX to ONNX and compiling it with different deployment inference engines on the Nvidia Jetson Orin NX's CPU and GPU.

or worse performance when deployed on multiple CPUs and considerably worse when offloaded to the GPU. We argue that this is due to the small size of the agent, which incurs a high penalty for thread synchronization, thereby outweighing the benefits of parallel computation.

Figure 2.7 illustrates a trend consistent with that observed for the YOLOv10n model, revealing a trade-off between inference performance and memory consumption. For example, while TFLite delivers the fastest execution, it may consume up to four times more memory than ORT. This behavior may be related to internal optimization mechanisms in TFLite, potentially involving intermediate data caching or buffer reuse strategies to reduce computation time.

Overall, these results highlight the lightweight deployment of our active inference model, achieving real-time planning and adaptation to the environment for IoT and robotics saccading applications.

2.5.2.3 System

Integrating the perception and planning modules produces a highly efficient Saccade agent capable of real-time operation with only 2.3 million

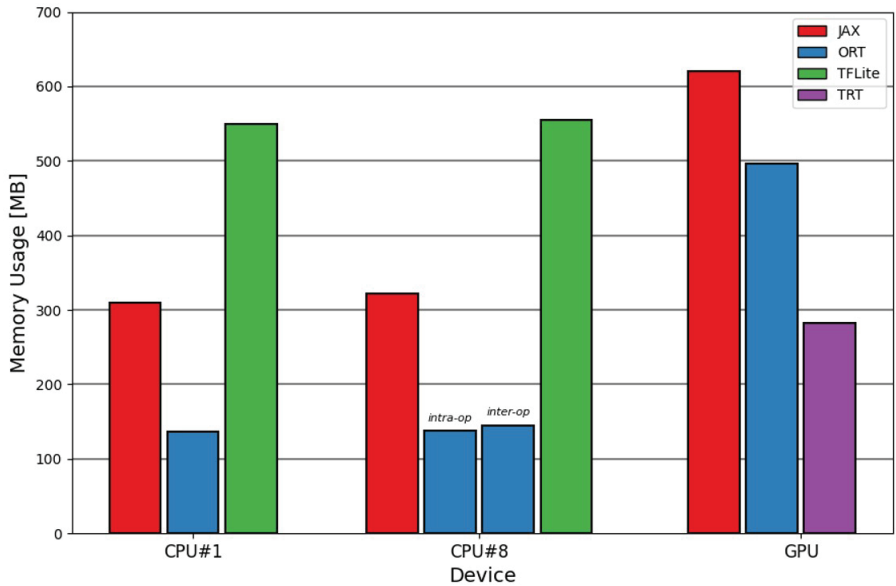

Figure 2.7 Planning module memory profile of the active inference model when executed with various inference engines on the Nvidia Jetson Orin NX CPU and GPU.

Table 2.1 System parameters: Saccade agent deployed on the Nvidia Jetson Orin NX.

Config	# Params (M)	Memory (MB)	Perception Rate (Hz)	Decision-making Rate (Hz)
A	2.3	947	45	**250**
B	2.3	533	**45**	108
C	2,3	**304**	9	108

parameters. Table 2.1 summarizes key configurations, illustrating the trade-offs between latency and memory consumption. For example, in configuration *A*, the perception module optimized with TensorRT-FP16 achieves a throughput of 45 FPS, enabling rapid feature extraction. In parallel, the active inference module using TFLite can perform planning and decision-making at up to 250 FPS, with a total memory footprint of under 1 GB. The decision-making rate can be reduced to 108 FPS using ORT, lowering memory usage to 533 MB, as shown in configuration *B*. Finally, configuration *C* utilizes the ORT (intra-operator parallelization) for perception, reducing the perception rate to 9 FPS while maintaining a compact memory footprint of 304 MB, which includes all required libraries and runtimes. This configuration effectively balances the latency and memory requirements of perception

and decision-making, enabling dynamic adaptation and efficient operation on resource-constrained edge devices.

2.6 Discussion

These results highlight the effectiveness of combining deep learning for perception with active inference for planning in edge-based intelligent agents. With only 1.34K parameters, the active inference module achieves real-time planning while maintaining minimal computational and memory demands. In contrast to large-scale foundation models that rely on billions of parameters to achieve general-purpose intelligence, our domain-oriented approach demonstrates how compact, task-specific active inference models can complement deep learning perception. The adaptability of our edge-deployment workflow illustrates deployment interoperability, facilitating the exploration of diverse configurations to achieve a suitable balance between latency and resource efficiency.

This validation establishes a solid foundation for a second stage of analysis focused on quantitative behavioural benchmarking. A systematic evaluation of task-level indicators such as decision accuracy, exploration efficiency, or robustness across environments remains an important avenue for future research.

2.7 Conclusions and Future Work

This work has introduced a smart edge agent composed of a deep learning-based perception module and an active inference planning module for active sensing. The system demonstrates the feasibility of adaptable, on-device surveillance and robotic solutions, as well as their potential to effectively handle real-world challenges. Our study highlights the potential of active inference in edge-based systems. Active inference offers a robust framework that accounts for the environment's underlying dynamics and inherent uncertainty. It allows the agent to actively reduce ambiguity or explore the environment for new information. This approach establishes a foundation for efficient edge agents for adaptive, resource-efficient decision-making in IoT and edge computing applications.

In future work, we aim to compare the proposed system against other state-of-the-art works, showing a quantitative functional comparison on popular challenges, such as the Habitat Navigation Challenge. Finally, we envision extending this concept to other applications, which could unlock a new

generation of adaptive edge devices capable of context-driven interactions in complex environments.

Acknowledgements

This work was partly supported by Horizon Europe dAIEdge under grant No. 101120726.

References

[1] S. Grossberg, K. Roberts, M. Aguilar, and D. Bullock, "A neural model of multimodal adaptive saccadic eye movement control by superior colliculus," J. Neurosci., vol. 17, no. 24, pp. 9706-9725, Dec. 1997.

[2] M. Ferro, D. Ognibene, G. Pezzulo, V. Pirrelli, "Reading as active sensing: a computational model of gaze planning during word recognition," Front. Neurorobot., vol. 4, p. 1262, 2010.

[3] T. Parr, N. Sajid, L. Da Costa, M. B. Mirza, K. J. Friston, "Generative models for active vision," Front. Neurorobot., vol. 15, p. 651432, 2021.

[4] M. B. Mirza, R. A. Adams, C. D. Mathys, K. J. Friston, "Scene construction, visual foraging, and active inference," Front. Comput. Neurosci., vol. 10, p. 56, 2016.

[5] A. Stokholm et al., "The autoICE challenge," Cryosphere, vol. 18, no. 8, pp. 3471-3494, 2024.

[6] S. McClanahan, "Object recognition and detection: Potential implications from vision science for wilderness searching," J. Search Rescue, vol. 5, no. 1, pp. 1-17, 2021.

[7] M. Kashef, A. Visvizi, and O. Troisi, "Smart city as a smart service system: Human-computer interaction and smart city surveillance systems," Comput. Hum. Behav., vol. 124, p. 106923, 2021.

[8] L. Da Costa, P. Lanillos, N. Sajid, K. J. Friston, S. Khan, "How active inference could help revolutionise robotics," Entropy, vol. 24, no. 3, p. 361, 2022.

[9] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, S. Wermter, "Continual lifelong learning with neural networks: A review," Neural Netw., vol. 113, pp. 54-71, 2019.

[10] T. Parr, G. Pezzulo, and K. J. Friston, Active Inference: The Free Energy Principle in Mind, Brain, and Behavior. Cambridge, MA, USA: MIT Press, 2022.

[11] K. J. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, J. O. Doherty, G. Pezzulo, "Active inference and learning," Neurosci. Biobehav. Rev., vol. 68, pp. 862-879, 2016.
[12] N. Krishnachaithanya et al., "People counting in public spaces using deep learning-based object detection and tracking techniques," in Proc. Int. Conf. Comput. Intell. Sustain. Eng. Solut. (CISES), 2023.
[13] M. De Prado et al., "Bonseyes ai pipeline—bringing ai to you: End-to-end integration of data, algorithms, and deployment tools," ACM Trans. Internet Things, vol. 1, no. 4, pp. 1-25, 2020.
[14] A. Wang et al., "Yolov10: Real-time end-to-end object detection," in Proc. Adv. Neural Inf. Process. Syst., vol. 37, 2024, pp. 107984-108011.
[15] Q. Zhang, R. Han, C. H. Liu, G. Wang, L. Y. Chen, "ElasticDNN: On-device neural network remodeling for adapting evolving vision domains at edge," IEEE Trans. Comput., vol. 73, no. 6, pp. 1616-1630, Jun. 2024.
[16] C. Cioflan, L. Cavigelli, M. Rusci, M. De Prado, L. Benini, "On-device domain learning for keyword spotting on low-power extreme edge embedded systems," in Proc. IEEE 6th Int. Conf. AI Circuits Syst. (AICAS), 2024.
[17] E. Sommerlade and I. Reid, "Information-theoretic active scene exploration," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2008.
[18] E. Sommerlade and I. Reid, "Probabilistic surveillance with multiple active cameras," in Proc. IEEE Int. Conf. Robot. Autom., 2010.
[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436-444, 2015.
[20] Z.-Q. Zhao, P. Zheng, S. Xu, X. Wu, "Object detection with deep learning: A review," IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 11, pp. 3212-3232, Nov. 2019.
[21] "ONNX: Open Neural Network Exchange," 2019. [Online]. Available: https://github.com/onnx/onnx
[22] "ONNX Runtime," 2021. [Online]. Available: https://onnxruntime.ai
[23] R. David et al., "Tensorflow lite micro: Embedded machine learning for tinyml systems," in Proc. Mach. Learn. Syst., vol. 3, 2021, pp. 800-811.
[24] E. Jeong, J. Kim, and S. Ha, "Tensorrt-based framework and optimization methodology for deep learning inference on jetson boards," ACM Trans. Embedded Comput. Syst., vol. 21, no. 5, pp. 1-26, 2022.
[25] T.-Y. Lin et al., "Microsoft coco: Common objects in context," in Proc. Eur. Conf. Comput. Vis., 2014, pp. 740-755.
[26] C. Heins et al., "pymdp: A Python library for active inference in discrete state spaces," arXiv preprint arXiv:2201.03904, 2022.

[27] "Tapo surveillance camera," Accessed: Nov. 2024. [Online]. Available: https://www.tapo.com/us/product/smart-camera/tapo-c210i
[28] "Locobot robot," Accessed: Nov. 2024. [Online]. Available: http://www.locobot.org
[29] "Saccade demonstration." [Online]. Available: https://drive.google.com/drive/folders/1VCN8MAMnsdbj-xY5qudjn7vLFlB5mSfi?usp=sharing
[30] R. Bajcsy, "Active perception," Proceedings of the IEEE, vol. 76, no. 8, pp. 996–1005, Aug. 1988.
[31] J. Aloimonos, I. Weiss, and A. Bandyopadhyay, "Active vision," International Journal of Computer Vision, vol. 1, no. 4, pp. 333–356, 1988.
[32] D. H. Ballard, "Animate vision," Artificial Intelligence, vol. 48, no. 1, pp. 57–86, 1991.
[33] S. Chen, Y. Li, and N. M. Kwok, "Active vision in robotic systems: A survey of recent developments," The International Journal of Robotics Research, vol. 30, no. 11, pp. 1343–1377, Sept. 2011.
[34] S. Isler, R. Sabzevari, J. Delmerico, and D. Scaramuzza, "An information gain formulation for active volumetric 3D reconstruction," in Proc. IEEE Int. Conf. Robot. Automat. (ICRA), Stockholm, Sweden, 2016, pp. 3477–3484.