ATLASv2: LLM-Guided Adaptive Landmark Acquisition and Navigation on the Edge

Mikolaj Walczak, Uttej Kallakuri, Tinoosh Mohsenin

Johns Hopkins University
mwalcza1@jh.edu, ukallak1@jh.edu, tinoosh@jhu.edu

Abstract

Autonomous systems deployed on edge devices face significant challenges, including resource constraints, real-time processing demands, and adapting to dynamic environments. This work introduces ATLASv2, a novel system that integrates a fine-tuned TinyLLM, real-time object detection, and efficient path planning to enable hierarchical, multi-task navigation and manipulation all on the edge device, Jetson Nano. ATLASv2 dynamically expands its navigable landmarks by detecting and localizing objects in the environment which are saved to its internal knowledge base to be used for future task execution. We evaluate ATLASv2 in real-world environments, including a handcrafted home and office setting constructed with diverse objects and landmarks. Results show that ATLASv2 effectively interprets natural language instructions, decomposes them into low-level actions, and executes tasks with high success rates. By leveraging generative AI in a fully on-board framework, ATLASv2 achieves optimized resource utilization with minimal prompting latency and power consumption, bridging the gap between simulated environments and real-world applications.

Introduction

The growing demand for autonomous systems capable of navigating and manipulating objects in real-world environments has driven advancements in artificial intelligence, robotics, and edge computing (Gill et al. 2024; Navardi et al. 2024; Prakash et al. 2024a; Mazumder et al. 2023). These systems must operate with minimal latency, high efficiency, and robust adaptability in dynamic, unstructured environments. This paper explores deploying generative AI on the edge (Shaharear et al. 2024), using large language models (LLMs) as dynamic planners for hierarchical, multi-task autonomous navigation.

Edge computing, the paradigm of processing data locally on devices rather than relying on cloud infrastructure, plays a pivotal role in this work. Processing on the edge offers several advantages, including enhanced privacy and security, and improved reliability in environments with limited or no internet connectivity. In addition, edge deployment minimizes the dependency on external servers, ensuring consistent performance even in challenging operational scenarios (Singh et al. 2023).

This paper specifically focuses on the deployment of generative AI on the edge to enable natural language-driven navigation and task execution. LLMs are used not only to interpret high-level natural language instructions but also to dynamically generate actionable plans for autonomous robots (Shah et al. 2023). We integrate navigation and manipulation tasks, allowing the system to adapt seamlessly to new environments. By dynamically scheduling processes like LLM execution and object detection, the proposed system optimizes onboard power consumption and latency, ensuring efficient operation on resource-constrained hardware.

To provide the Large Language Model (LLM) with information about landmarks in the environment we integrate an onboard object detector. The objects detected by the detector expand the number navigable landmarks enabling the LLM to use the new landmarks for future tasks.

This work bridges the gap between simulated environments and real-world deployment by presenting a resource-efficient, edge-based system capable of performing complex, hierarchical tasks with high-level reasoning and adaptability. The findings highlight the potential of generative AI at the edge for advancing intelligent planning with autonomous systems in diverse and challenging settings. The contributions of this paper are summarized as follows:

- A fully on-board solution and system architecture that integrates a path planning module, object detection, and a large language model for navigation and object manipulation tasks.
- A novel method for continuously expanding the set of skills and navigable landmarks by detecting objects in the environment and incorporating them into the system's knowledge base with the assistance of the LLM.
- Minimizing latency and onboard power consumption through dynamic scheduling of processes, such as LLM execution and object detection.

Related Work

The integration of LLMs into robotics is a rapidly growing area, aiming to enable embodied AI that can interpret and execute natural language way-finding instructions in dynamic environments. Many of these works target a fully simulated environment or when deployed use very large and intelligent cloud-based solutions or high power devices (Dor-

bala et al. 2024; Rajvanshi et al. 2024; Lin et al. 2024; Kallakuri et al. 2024; Zhu et al. 2024). For instance, (Song et al. 2023) introduced LLM-Planner, a system that leverages few-shot learning to allow embodied agents to interpret and execute high-level natural language instructions. This approach improves adaptability to diverse tasks with minimal prior examples, enhancing their versatility in dynamic settings. LLM-Planner demonstrates the deployment of a robotic agent equipped with a low-level planner in a virtual environment, integrating a pre-trained BERT-base-uncased model (Devlin et al. 2018) with the ChatGPT API, enabling embodied navigation based on high-level plans.

In this work we deploy an LLM for high-level planning on the edge device Jetson Nano, which presents significant challenges with identifying compact models that are able to operate within strict resource constraints. Additionally, many open-source models are trained on generalized datasets, which can result in outputs lacking the precision and focus necessary for effective high-level task decomposition. This misalignment can lead to inefficiencies, consuming resources on decoding and generating responses that do not meet task-specific requirements. To address these challenges, (Chen et al. 2024) proposed the "Octo-planner," a framework that separates planning and action components, optimized for edge devices. The Octo-planner fine-tunes LLMs to improve success rates and efficiency, tackling issues such as context length and computational costs. By utilizing techniques like multi-LoRA training, this framework supports multi-domain queries with reduced resource demands, making it suitable for mobile devices. The Octo-planner demonstrates significant potential for scalable, real-time, and privacy-preserving AI applications on resource-constrained platforms.

This work focuses on the deployment of a robotic agent within a controlled environment that integrates a high-level LLM planner (Prakash et al. 2024b), a low-level motion planner, and an object detector, on the Jetson Nano edge device. This paper provides practical insights into the deployment of integrated onboard systems, inspired by frameworks such as (Kallakuri et al. 2024), (Navardi et al. 2023) and (Song et al. 2023). In integrating the high-level LLM planner, we adopt an approach similar to the Octo-Planner framework (Chen et al. 2024), fine-tuning a compact language model to enhance system reliability, generate task-specific responses, and minimize latency. A significant challenge with the Octo-Planner model is that with 2 billion parameters, even in its open-source form and after quantization, it occupies approximately 2 GB of memory. This poses a risk of exceeding the Jetson Nano's limited 4 GB memory capacity when coupled with an object detector and motion planner. To address this limitation, we incorporate the state-of-the-art TinyLLaMAv1.1 model (Zhang et al. 2024), an open-source, compact language model with 1.1 billion parameters designed for efficiency and accessibility. TinyLLaMA achieves competitive performance while significantly reducing computational overhead, making it well-suited for resource-constrained environments. By fine-tuning and quantizing the model, we further optimize its performance for real world deployment.

Algorithm 1: The integration of ATLASv2 components is formalized in the algorithm below, which outlines the dynamic interaction between the navigation, object detection, object interaction and KB building processes.

function MAIN
    STARTKBPROCESS($initialKB$)
    while True **do**
        Wait for: $prompt$
        $subtasks \leftarrow LLM(prompt)$
        EXECTASKS($subtasks$)
    end while
end function
function EXECTASKS($subtasks$)
    for all $action, obj \in subtasks$ **do**
        if $obj \notin KB$ **then**
            return $fail$
        else if $action$ is $navigate$ **then**
            $coo \leftarrow$ KB(obj)
            $ros\_navigate(coo)$
        else
            $moveArm(obj, action)$
        end if
    end for
    $spin()$
    return $success$
end function
function STARTKBPROCESS($initialKB$)
    $KB \leftarrow initialKB$
    while True **do**
        $dets \leftarrow$ detectAndFilterObjs()
        for all $obj \in dets$ **do**
            $objPos, objOrient \leftarrow$ getCurPos()
            $KB[obj] \leftarrow objPos, objOrient$
        end for
    end while
end function

Proposed Approach

The proposed system integrates key components to enable hierarchical multi-task autonomous navigation on the Jetson Nano edge device. This architecture combines efficient path planning, real-time object detection, and high-level reasoning with an LLM, all while operating within the strict computational and energy constraints. The approach focuses on leveraging generative AI at the edge for dynamic task planning and knowledge enrichment.

At the core of the system is the interaction between the path planning module, object detection module, and LLM-based planner. These components enable the agent to interpret high-level natural language instructions, plan and execute navigation tasks, and adapt to new environments by expanding its knowledge base (KB). The high-level interaction of these components is outlined in Algorithm 1,

When using the system a natural language prompt describing the desired task is provided. The LLM planner interprets the input and generates a structured plan. This plan decomposes the task into smaller steps, which include nav-

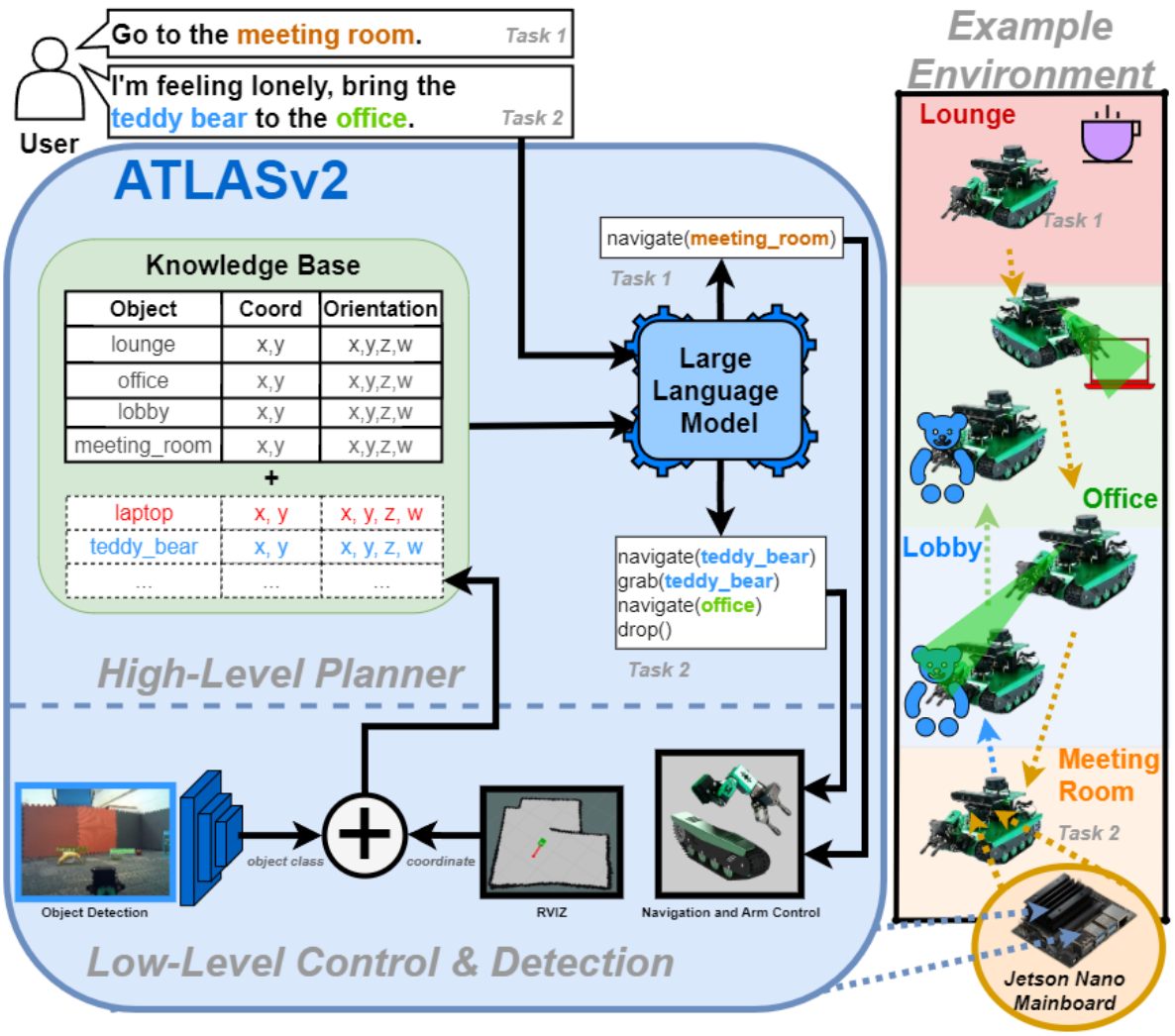
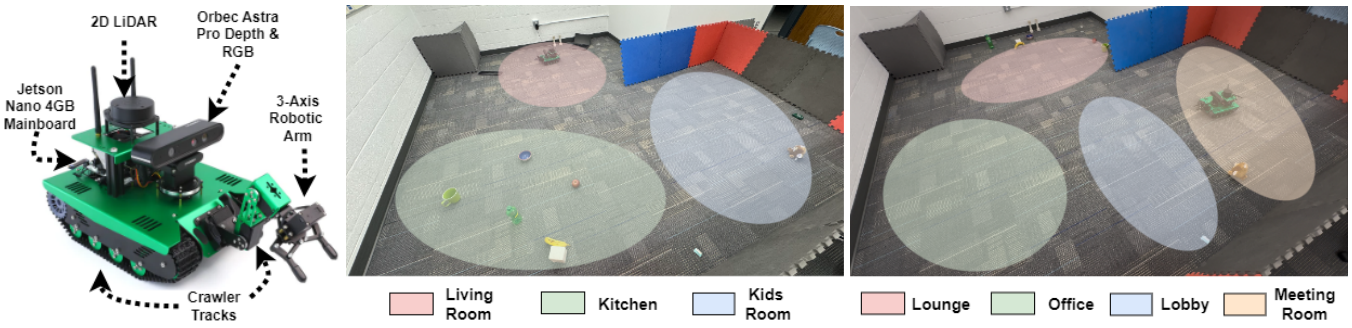


Figure 1: Block diagram showing how ATLASv2 is used given two sample prompts provided by the user. The system begins with a prompt received from the user (Task 1) for which the knowledge base is appended to provide the available landmarks. The LLM then generates a plan (if available) that is executed in sequence by the navigation and manipulation package. During execution if a new object of interest is detected (eg. laptop and teddy bear), the object and its location, visualized using RVIZ (Kam et al. 2015) , is appended to the expanding knowledge base. The user can then use the newly detected objects for future tasks (Task 2). The full system is deployed into the Jetson Nano edge device and receives prompts remotely from the user.

igating to specific landmarks, and grabbing and dropping an object. During task execution newly detected objects and their locations are added to the internal knowledge base for future tasks. This process is outlined in Figure 1 showing the interaction of the elements of ATLASv2 when provided two sample prompts in an office emulated environment. All components, including the object detector and LLM planner are deployed directly on the Jetson Nano to ensure reliable operation in areas devoid of a network connection, essential for real-time decision-making. The core components of ATLASv2 are outlined as follows:

Path Planning and Execution: The path planning and object manipulation module developed in Robot Operating System 1 (ROS) (Stanford Artificial Intelligence Laboratory et al. 2018) is responsible for executing the plans generated by the LLM. It translates waypoints or landmarks into motion commands using services such as *navigate_to_location* for path following, *spin_service* for rotational scanning, *grab_object* for picking up objects, and *drop_object* to release them. This configuration enables efficient exploration of the environment while detecting objects of interest with which can interact. Physically gripping objects falls outside of the scope of the system, meaning during manipulation-based tasks a gripping or dropping-like movement is executed near targets to simulate moving an object.

Object Detection and Knowledge Expansion: The object detector selected for this work is YOLO-v5n (Jocher et al. 2020), which is a robust, open source, compact, and easy to use object detector compatible with TensorRT (NVIDIA 2023) acceleration. TensorRT optimizations such as precision reduction and layer fusion improve inference speed and energy efficiency, making the model suitable for



(a) (b) (c)

Figure 2: The robotic agent used for deployment of ATLASv2 (a) Yahboom Transbot which contains a 2D lidar for mapping, an Orbbec Astra Pro camera for depth and RGB input, a 3-Axis robotic arm for manipulating objects, crawler tracks for moving around the environment, and a Jetson Nano. Additionally, the configurations of the real-world environment used for system evaluation including (b) The home emulated environment, which includes a living room, kitchen, and kids room, each populated with objects typical of a residential setting and (c) the office emulated environment, comprising of a lounge, office, lobby, and meeting room, with items representative of a professional workplace.

edge deployment. Detected objects are localized using depth sensors and filtered to determine their relevance as landmarks. Relevant objects are added to the KB, which evolves dynamically, enabling the agent to improve its understanding of the environment over time.

Edge LLM Deployment: Deploying an LLM on the memory-constrained Jetson Nano requires balancing computational efficiency and performance. TinyLLama was selected for its ability to deliver competitive performance within a reduced computational footprint. The deployment leverages llama.cpp (Gerganov 2023), which enables splitting model components between the GPU and CPU swap memory, which is vital when balancing the load of the object detector on the limited memory of the Jetson Nano. The TinyLLama model is fine-tuned and quantized using the Q5_K medium quantization method, provided by llama.cpp, with the Unsloth (Han et al. 2023) framework, ensuring alignment with task-specific scenarios. This approach allows the model to reliably translate natural language prompts into focused actionable plans while minimizing memory and processing requirements.

Dynamic Process Scheduling: To minimize latency and power consumption, the system employs dynamic scheduling for computationally intensive processes. To ensure objects are not missed when navigating in the environment, the object detector continuously evaluates images while the LLM planner is loaded and only initialized when a new prompt becomes available. This method ensures the system can respond to dynamic changes in the environment while preserving computational and energy resources.

The integration of these components ensures the system can handle hierarchical tasks, such as multi-step navigation and manipulation, in a resource-constrained setting.

The deployment pipeline is summarized as follows:

- YOLO asynchronously detects objects and localizes them using depth data, which are filtered and updated into the KB dynamically.
- The user provides a natural language task to the LLM to generate a structured plan with waypoints, objects or goals.
- Using generated subtasks the agent executes the plan using navigation, scanning, and object manipulation actions.

By combining TensorRT-optimized YOLO and GPU-accelerated LLMs with dynamic scheduling, the system achieves a balance between performance and resource efficiency. Comprehensive profiling of the system demonstrates the viability of deploying generative AI on edge devices, showcasing its ability to perform high-level reasoning and real-time decision-making in diverse, real-world scenarios.

Experimental Setup and Results

This section outlines the experimental setup and results used to evaluate the proposed system and its components in a real world environment when deployed on a robotic agent.

Robotic Agent and Environment

The Yahboom Transbot (Yahboom 2025) was selected as the robotic platform (the agent) for executing the desired tasks, which include both navigation and manipulation shown in Figure 2 (a). The Jetson Nano 4GB edge computing device, which, along with 8 GB of swap memory, manages the autonomous control of the agent and supports the deployment of the ATLASv2 LLM-guided navigation package.

ATLASv2 was evaluated in a real-world environment constructed within an office space. The environment included foam pads to emulate various sections of a typical home and office, as depicted in Figure 2 (b) and (c). The home setting 2 (b), featured three predefined landmarks in the KB: living room, kitchen, and kids room, while the office setting 2 (c), contained four landmarks: lounge, lobby, office, and meeting room. Each landmark contains relevant objects to the object detector, for example, the kitchen included items such as a

Metric	Input Prompt	Cohere	Quantized TinyLlama	
			Original (Pre-Trained)	Fine-Tuned
Navigation Prompt	Navigate to the garage to check if the delivery truck is still here	1/1 navigate(garage)	0/1 I am not able to perform tasks, but I can provide you with a step-by-step guide on how to navigate ...	1/1 navigate(garage)
Manipulation Prompt	Go to the vending machine grab a bottle and bring it to the office	4/4 navigate(vending – machine) grab(bottle) navigate(office) drop()	0/4 Here's a possible solution:1.Navigate to the vending machine 2. grab a bottle from the shelf 3. Bring the bottle to the office ...	4/4 navigate(vending – machine) grab(bottle) navigate(office) drop()
Our Prompt Score	**-**	**29/30**	**0/30**	**27/30**

Table 1: Evaluation of LLM performance, comparing the cloud-based Cohere LLM as a baseline to the pre-trained TinyLlama model and our fine-tuned model post quantization. Results depict the total correctly generated sub-tasks versus the expected sub-tasks across 12 high-level natural language prompts. Two representative sample prompts are included, showcasing the expected sub-tasks alongside the sub-tasks generated by each model.

cup, bowl, apple, orange, and bottle, while the living room contained a laptop and keyboard.

In this environment, the Yahboom Transbot was initialized with a pre-loaded map containing the defined landmarks. Using LiDAR-based mapping and simultaneous localization and mapping (SLAM) (Thrun and Leonard 2008) packages provided by ROS, the agent navigated between these landmarks. Additionally, through the integration YOLOv5n the robot identified objects within the environment to incrementally build a KB. This enables the agent to plan and execute future tasks effectively. In between tasks, the agent is controlled remotely via an SSH connection to provide new high-level prompts to the LLM planner for controlling the agent.

Edge LLM Fine-Tuning and Compression

To ensure optimal resource utilization when deploying the TinyLLama LLM after applying quantization, the model is reduced to a size of 745 MB, ensuring compatibility with the Jetson Nano's limited memory. However, since TinyLLaMA is pre-trained with general knowledge, it requires fine-tuning to produce reliable outputs tailored to the specific prompts and tasks for ATLASv2.

To illustrate the need for fine-tuning, table 1 presents a comparison of TinyLLaMA's original outputs and the cloud LLM Cohere's (Cohere 2025) responses for sample prompts. While TinyLLaMA generates relevant outputs, they lack the specificity required to decode low-level tasks effectively. For example, two sample prompts and comparison to the baseline cloud LLM can be observed in columns 3 and 4 of table 1 showing the TinyLLama LLM provides relevant outputs, however lack focus needed for deployment.

Prior to fine-tuning, a dataset was constructed to expose TinyLLaMA to prompts and required low-level sub-tasks similar to the tasks it will encounter when deployed.

Two prompt templates (or "skeletons") were created:

1. **Manipulation-based tasks**, involving a sequence of tasks to instruct the agent to navigate to an object and move it to another location.
2. **Navigation-based tasks**, for directing the agent to navigate to a specific destination.

An example of how these templates are used by the model is shown in column 3 of Table 1, which displays Coheres' responses to both prompt types.

Using these templates, class names detected by YOLOv5n were inserted, creating an initial dataset. To further expand the dataset and introduce variation, prompts were augmented and reworded using ChatGPT-4. The dataset included diverse system headers with varying KBs to produce context for the LLM creating the final dataset of 20,000 samples.

TinyLLaMA was fine-tuned using the generated dataset of 15,000 training and 5,000 testing samples to enable decomposition of natural language prompts into sequential low-level actions. Fine-tuning involved training the model using a batch size of 5, over three epochs with a learning rate of 5×10^{-5}. Before training, the model's loss on the dataset was 2.3; after fine-tuning, the loss decreased to 0.4, demonstrating significant improvement.

After fine-tuning, the TinyLLaMA model was quantized, reducing its size to 745 MB. The fine-tuned LLM was then evaluated using natural language prompts to verify its performance. Notably:

- The LLM accurately responded to prompts containing objects it was fine-tuned on including simple navigation and manipulation tasks.
- For prompts involving entities (e.g., garage, lounge) not exposed during fine-tuning, the LLM generalized effectively, responding with navigation or manipulation subtasks containing the target entity.

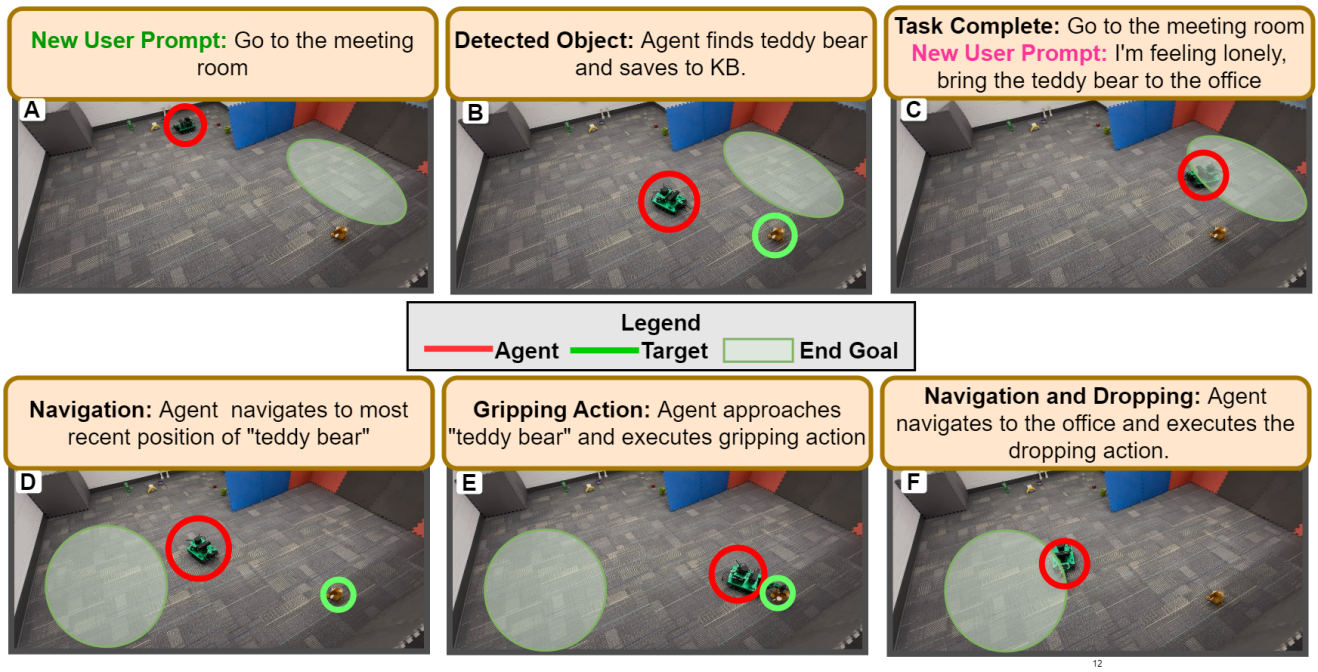


Figure 3: Samples from the office setting experiment showing the agent successfully completing high-level tasks provided to the onboard LLM. The Figure displays two of the prompts executed by the agent: "Go to the meeting room" (A, B, and C) and "I'm feeling lonely, bring the teddy bear to the office" (C, D, E, and F). For the first prompt the agent successfully navigates to the meeting room (C) and adds the "teddy bear" to the knowledge base along its path (B). Then, in the second prompt the agent successfully performs actions representing grabbing the teddy bear (E) and moving it to the office (F).

- Complex prompts containing additional context (e.g., table 1 row 2 column 5) were successfully decomposed into actionable outputs.

The responses to the original sample prompts provided to the pre-trained TinyLLaMa LLM are displayed in column 5 of table 1. Comparing to the original outputs the fine-tuned LLM provides a focused and accurate response.

These results demonstrate that the fine-tuned TinyLLaMA model is robust to input variations and capable of accurately handling tasks it was not explicitly trained on. This adaptability ensures its effective deployment on the Jetson Nano for real-world applications.

Full System Deployment and Evaluation

With a significant portion of the Jetson Nano's RAM allocated to the object detector, components of the LLM must be distributed between the GPU and CPU swap memory. To address this challenge, the optimized llama.cpp framework is deployed which supports running LLMs on resource-constrained devices like the Jetson Nano by allowing selective offloading of layers to the GPU, while the remaining layers are managed between RAM and swap memory. Although additional latency is produced from CPU processing, this overhead is marginal, as the LLM is queried only at the beginning of a sequence to generate a plan or evaluated during execution of previous plans.

To ensure efficiency and dynamic resource management, the LLM is loaded into memory as a separate process and initialized for decoding only when a new prompt is available. This approach conserves computational resources while maintaining the responsiveness required for high-level task planning.

With the object detector and LLM high-level planner integrated, the agent is deployed in the two emulated home and an office settings described previously. Using an initial map and pre-loaded landmarks, the ATLASv2 KB building package is initialized alongside the navigation system. The TensorRT-optimized YOLOv5n object detector is launched to analyze images and identify objects in the environment, contributing to the KB. Finally, the LLM is initialized to enable user interaction through natural language prompts.

The system is evaluated under two configurations:

- Cloud-based LLM (Cohere): A cloud-based LLM serves as the high-level planner, establishing a baseline for comparison.
- Fully onboard fine-tuned LLM: The fine-tuned TinyLLaMA model is deployed entirely on the Jetson Nano.

A sample demonstration showing the agent performing the desired actions outlined in two sample prompts, in real-time, using the office environment setting and onboard LLM planner is shown in Figure 3. Both configurations successfully decomposed high-level tasks into actionable low-level sub-tasks, enabling the agent to navigate the environment and perform actions representing object manipulation tasks, such as gripping and dropping actions near specified targets.

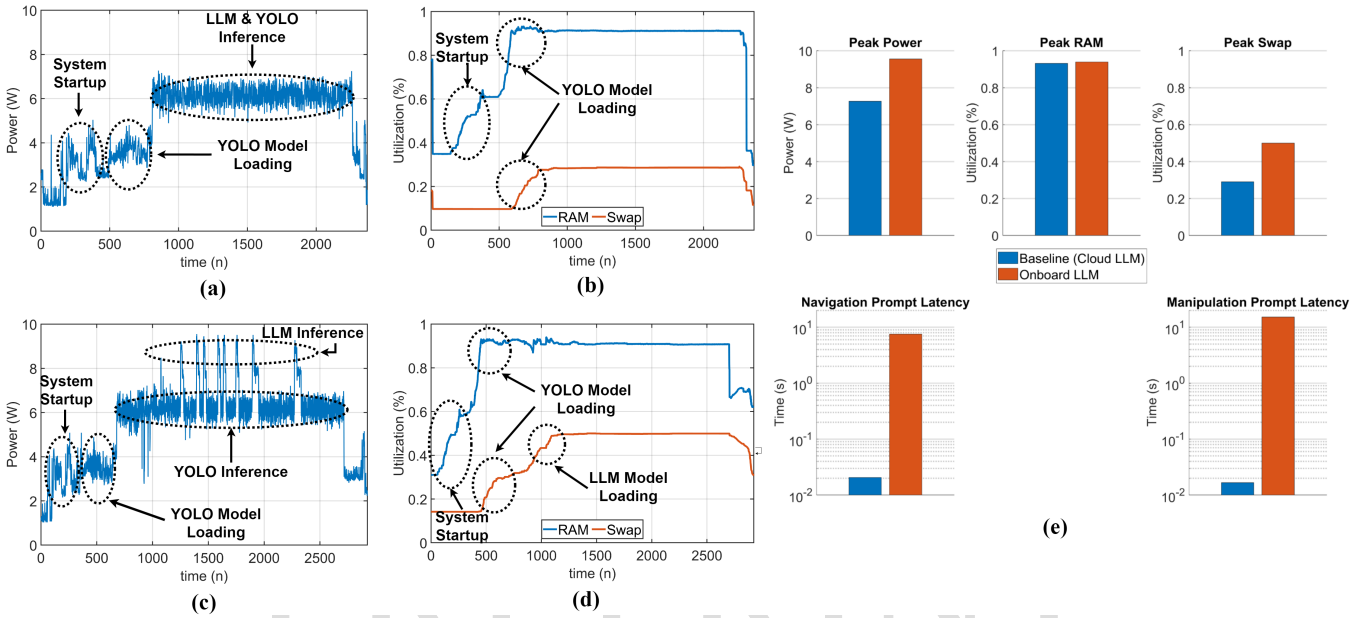


Figure 4: Total power, RAM utilization, swap utilization, and latency results for navigation and manipulation based tasks on the Jetson Nano, run in 10W mode using a 1.43 ghz CPU clock frequency, 921 mhz GPU clock frequency, 4 GB of RAM and 8 GB of swap, during the real-world office setting experiment for the cloud and onboard implementations. The Figure shows results over time for (a) power consumption when deploying the system using the cloud LLM Cohere, (b) memory utilization for the system using the cloud LLM Cohere, (c) power for the fully onboard system (d) memory utilization for the fully onboard system and (e) bar plots for peak power and memory utilization along with prompt processing latency results for simple navigation prompts and more complex manipulation prompts.

For both configurations, the agent receives initial prompts to navigate to pre-loaded landmarks, building the KB. Once the KB is established, the agent is tasked with more complex manipulation prompts, such as transporting an object to a new location. Key metrics including LLM prompt decomposition, successful execution, power consumption, RAM usage, and swap utilization are recorded for analysis. The prompts used to evaluate the system in both environments are displayed in column 2 of Table 2.

The system utilization results in Figure 4, reveal resource utilization during the experiment in the second office environment for cloud-based and on-board configurations:

- Power Consumption: In both configurations, loading YOLOv5n causes power consumption to stabilize at approximately 6 W. In the cloud-based LLM approach, power consumption remains constant, with no significant change during LLM prompting. In contrast, the fully onboard configuration shows notable spikes to 9.2 W during LLM processing.
- Memory and Swap Utilization: For both configurations due to the load of the object detector, KB, and navigation packages the RAM utilization remains at a steady 92% to manage both swap and GPU memory. In contrast, the swap utilization reaches approximately 25% for the offboard approach, while the LLM configuration on-board doubles the swap to 50%.
- Latency: in the cloud-based approach prompt response latency averaged to 20 milliseconds for all tasks while the onboard approach produced latencies of 8 seconds, given simple navigation-based tasks, and up to 10 seconds, given more complex manipulation-based tasks.

These results highlight the computational load and full deployment of ATLASv2 on the Jetson Nano in a real-world scenario. While the cloud-based configuration offers advantages in terms of prompt response latency and consistent power consumption, it relies on robust network connectivity, which may not always be feasible in dynamic or remote environments. On the other hand, the fully onboard configuration demonstrates a clear trade-off, with increased power and memory demands leading to greater latency.

To emphasize the benefit of the developing KB, Table 2 presents the prompts provided to the LLM planner for both experimental settings alongside the number of successful tasks executed by the agent. The agent is evaluated in a scenario where it cannot grow the KB versus the ATLASv2 growing KB approach. During these experiments, the agent is first given tasks to navigate to the initial pre-loaded landmarks so the agent begins to explore the environment detecting objects along the way. Then, the agent is provided with more complex manipulation based tasks, using objects which were not available when the system was initialized, to determine the effectiveness of the growing KB. When the system is unable to grow the KB, the agent fails to complete the manipulation prompts while, leveraging the ATLASv2 approach to save landmark positions for use in future tasks enables successful task completion.

Setting	Input Prompt	Sub-Tasks (# Success/ Total)	
		Fixed	Growing
Home	*Go to the kitchen*	1/1	1/1
	Go to the kids room	1/1	1/1
	Go to the living room	1/1	1/1
	I'm hungry bring the banana to the laptop	**0/4**	**4/4**
	My son forgot his teddy bear, take the teddy bear to the kids room	**1/4**	**4/4**
Office	*Go to the lounge*	1/1	1/1
	Go to the lobby	1/1	1/1
	Go to the office	1/1	1/1
	Go to the meeting room	1/1	1/1
	I'm feeling lonely, bring the teddy bear to the office	**1/4**	**4/4**
	Guests are here and they are thirsty bring the bottle to the lobby	**1/4**	**4/4**

Table 2: Onboard LLM comparison between a fixed KB and our dynamic growing KB for the real world home and office setting experiments. Prompts are provided in sequence for each setting and the number of correctly executed sub-tasks is recorded for the fixed and growing KB. Initial prompts use locations in the initial KB followed by tasks with objects not loaded into the initial KB shown in bold.

Conclusion and Future Work

This work demonstrates deployment of ATLASv2 a system architecture integrating generative AI and edge computing for autonomous navigation and task execution in indoor, controlled real-world environments. By integrating a fine-tuned TinyLLM with real-time object detection and efficient path planning, ATLASv2 achieves dynamic, hierarchical task planning and execution on the resource-constrained Jetson Nano. The approach not only bridges the gap between simulated and real-world deployments but also establishes a framework for integration of a cohesive fully on-board solution capable of executing complex multi-stage tasks.

Future work will focus on evaluating ATLASv2 in more complex outdoor environments and enhancing the system's capabilities through the integration of more sophisticated open-vocabulary object detection, advanced object manipulation techniques, and improved environment exploration. These advancements aim to increase the versatility of ATLASv2 in dynamically changing outdoor environments.

Acknowledgments

We thank Dr. Xiaomin Lin for his review and valuable comments that helped improve this paper. This project was sponsored by the U.S. Army Research Laboratory.

References

Chen, Y.; et al. 2024. Octo-planner: On-device Language Model for Planner-Action Agents. *arXiv preprint arXiv:2406.18082*.

Cohere. 2025. Cohere - Natural Language Processing Platform. Accessed: 2025-01-10.

Devlin, J.; et al. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805.

Dorbala, V. S.; et al. 2024. Can LLMs Generate Human-Like Wayfinding Instructions? Towards Platform-Agnostic Embodied Instruction Synthesis. arXiv:2403.11487.

Gerganov, G. 2023. llama.cpp. Accessed: 2025-01-06.

Gill, S. S.; et al. 2024. Edge AI: A Taxonomy, Systematic Review and Future Directions. *Cluster Computing*, 28(1).

Han, D.; et al. 2023. Unsloth. Accessed: 2025-01-06.

Jocher, G.; et al. 2020. YOLOv5: A state-of-the-art object detection model. Accessed: 2025-01-06.

Kallakuri, U.; et al. 2024. ATLAS: Adaptive Landmark Acquisition using LLM-Guided Navigation. In *First Vision and Language for Autonomous Driving and Robotics Workshop*.

Kam, H. R.; et al. 2015. RViz: a toolkit for real domain data visualization. *Telecommun. Syst.*, 60(2): 337–345.

Lin, B.; et al. 2024. Correctable Landmark Discovery via Large Models for Vision-Language Navigation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12): 8534–8548.

Mazumder, A. N.; et al. 2023. Reg-TuneV2: Hardware-Aware and Multi-Objective Regression-Based Fine-Tuning Approach for DNNs on Embedded Platforms. *IEEE Micro*.

Navardi, M.; et al. 2023. Metae2rl: Toward metareasoning for energy-efficient multi-goal reinforcement learning with squeezed edge yolo. *IEEE Micro*.

Navardi, M.; et al. 2024. MetaTinyML: End-to-End Metareasoning Framework for TinyML Platforms. *IEEE Embed. Syst. Lett.*, 16(4).

NVIDIA. 2023. TensorRT: A high-performance deep learning inference library. Accessed: 2025-01-06.

Prakash, B.; et al. 2024a. Using LLMs for Augmenting Hierarchical Agents with Common Sense Priors. In *The International FLAIRS Conference Proceedings*, volume 37.

Prakash, B.; et al. 2024b. Using LLMs for Augmenting Hierarchical Agents with Common Sense Priors. In *The International FLAIRS Conference Proceedings*, volume 37.

Rajvanshi, A.; et al. 2024. SayNav: Grounding Large Language Models for Dynamic Planning to Navigation in New Environments. arXiv:2309.04077.

Shah, D.; et al. 2023. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on robot learning*, 492–504. PMLR.

Shaharear, M. R.; et al. 2024. ViT-Reg: Regression-Focused Hardware-Aware Fine-Tuning for ViT on tinyML Platforms. *IEEE Design & Test*.

Singh, R.; et al. 2023. Edge AI: A survey. *Internet of Things and Cyber-Physical Systems*, 3: 71–92.

Song, C. H.; et al. 2023. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. arXiv:2212.04088.

Stanford Artificial Intelligence Laboratory et al. 2018. Robotic Operating System.

Thrun, S.; and Leonard, J. J. 2008. *Simultaneous Localization and Mapping*, 871–889. Springer Berlin Heidelberg. ISBN 978-3-540-30301-5.

Yahboom. 2025. Transbot for Jetson Nano. Accessed: 2025-01-06.

Zhang, P.; et al. 2024. TinyLlama: An Open-Source Small Language Model. Technical Report, arXiv:2401.02385.

Zhu, Y.; et al. 2024. ChatNav: Leveraging LLM to Zero-shot Semantic Reasoning in Object Navigation. *IEEE Transactions on Circuits and Systems for Video Technology*, 1–1.