University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Computer Science

Improving Zero-Shot Multi-Object Navigation by Enhancing Object Detection in LLM-Based Embodied AI Frameworks

Supervisor
Prof. Lamberto Ballan
University of Padova

Co-supervisor
Filippo Ziliotto
University of Padova

Master Candidate
MOHAMMADKAZEM RAJABI

Student ID
2070860

Academic Year
2025-2026

"I suppose therefore that all things I see are illusions; I believe that nothing has ever existed of everything my lying memory tells me. I think I have no senses. I believe that body, shape, extension, motion, location are functions. What is there then that can be taken as true? Perhaps only this one thing, that nothing at all is certain."
— Rene Descartes

Abstract

Embodied AI aims to build agents capable of understanding and interacting with 3D environments through perception, reasoning, and action. TANGO is a modular framework that leverages large language models (LLMs) to control embodied agents in a zero-shot manner, without task-specific retraining. While TANGO demonstrates impressive generalization, its performance on the Multi-Object Navigation (MultiON) benchmark is limited by the accuracy of its object detection module—particularly in detecting colored cylinders using the baseline OWL-ViT2 detector.

This thesis proposes a targeted enhancement of TANGO's detection pipeline by creating a synthetic dataset tailored to MultiON. Using the Habitat simulator and HM3D scenes, we generated 6,000 photorealistic images with 13,000 annotated instances across eight cylinder classes. Manual annotation was performed via Roboflow to ensure high precision. We fine-tuned YOLOv8 and DETR variants on this dataset and evaluated their integration within the TANGO framework.

Our best-performing model, YOLOv8x, achieved a 28% improvement in success rate on MultiON 3-ON over the baseline and outperformed the prior state-of-the-art MOPA[1] framework by 14.6% and Compared to the best end-to-end model, ProjNeuralMap[2], our modular approach achieves a significant improvement of 26% in success rate and 24% in progress on the MultiON 3-ON task, demonstrating the advantages of integrating fine-tuned object detectors within a generalizable, LLM-driven framework. We also conducted ablation studies on TANGO's memory mechanism, showing consistent gains on both 3-ON and 2-ON tasks. The results confirm that upgrading the detection module significantly boosts the capabilities of modular embodied AI systems.

Contents

Abstract v

List of figures x

List of tables xiii

Listing of acronyms xv

1 Introduction 1

2 **Overview of Embodied Artificial Intelligence** 5
- 2.1 Embodied Artificial Intelligence: Definition, Scope, and Challenges . . . 5
- 2.2 Navigation Challenges . . . 7
  - 2.2.1 PointGoal Navigation (PointNav) . . . 7
  - 2.2.2 ObjectNav . . . 7
  - 2.2.3 Instance-Image Goal Navigation . . . 8
  - 2.2.4 MultiON . . . 8
  - 2.2.5 Vision-and-Language Navigation(VLN) . . . 9
  - 2.2.6 Multi-Modal Lifelong Navigation . . . 10
  - 2.2.7 Embodied Question Answering . . . 11
  - 2.2.8 Open-Vocabulary Object Goal Navigation . . . 11
- 2.3 Evaluation Measures . . . 11
  - 2.3.1 Success Rate (SR) . . . 11
  - 2.3.2 Distance to Goal (DTG) . . . 12
  - 2.3.3 Success weighted by Path Length (SPL) . . . 12
  - 2.3.4 MultiON Metrics . . . 12
  - 2.3.5 Metrics in Embodied Question Answering (EQA) . . . 13
- 2.4 Habitat Simulator: A Platform for Embodied AI Research . . . 14
  - 2.4.1 Introduction . . . 14
  - 2.4.2 System Architecture . . . 15
  - 2.4.3 Executing Tasks in Habitat-AI . . . 17

3 **Introduction to TANGO** 19
- 3.1 Overview of TANGO: Training-free Embodied AI Agents for Open-world Tasks . . . 19
- 3.2 Modularity trough Large Language Models . . . 20
  - 3.2.1 VisProg: Modular Program Composition for Visual Reasoning . . . 20
  - 3.2.2 ViperGPT:Visual Inference via Python Execution for Reasoning . . . 23
- 3.3 Large Language Models in TANGO . . . 25
  - 3.3.1 Program Composition and Task Planning with LLMs in TANGO . . . 26
  - 3.3.2 TANGO Program Interpreter . . . 27
- 3.4 Exploration Policy using Memory targets . . . 28
  - 3.4.1 Overview of VLFM and Memory-Augmented Exploration . . . 28

3.4.2 Exploration and Value Mapping in VLFM . . . 28
3.4.3 Memory-Augmented Exploration in TANGO . . . 30
3.5 Tasks: Open-World Navigation and Embodied AI Challenges . . . 31
3.5.1 GOAT-Bench . . . 31
3.5.2 Open Vocabulary Object Goal Navigation . . . 33
3.5.3 Embodied Question Answering (OpenEQA) . . . 36

4 Object Detection Module in TANGO 41
4.1 Object Detection Module in TANGO . . . 41
4.1.1 Introduction . . . 41
4.2 Model Evaluation . . . 41
4.2.1 Mean Average Precision . . . 42
4.2.2 Object Detection Workflow in TANGO . . . 43
4.2.3 Model Selection and Loading . . . 44
4.2.4 Preprocessing the Input Image . . . 44
4.2.5 Performing Object Detection (Inference Step) . . . 44
4.2.6 Post-processing the Detection Results . . . 45
4.2.7 Classification and False Positive Filtering . . . 45
4.3 CLIP-Based Classification in TANGO . . . 45
4.3.1 Introduction . . . 45
4.3.2 Overview of CLIP's Architecture . . . 46
4.4 Vision Transformer . . . 51
4.4.1 Model Architecture . . . 51
4.4.2 Self-Attention Mechanism . . . 53
4.4.3 Multi-Head Self-Attention . . . 54
4.4.4 Positional Embeddings . . . 55
4.5 OWL-VIT . . . 55
4.5.1 Introduction . . . 55
4.5.2 Model Architecture . . . 56
4.5.3 Training Strategy . . . 57
4.6 DETER . . . 58
4.6.1 DETR Architecture . . . 58
4.6.2 Transformer Encoder . . . 59
4.6.3 Transformer Decoder . . . 59
4.6.4 Prediction Feed-Forward Networks (FFNs) . . . 60
4.6.5 Bipartite Matching with Hungarian Loss . . . 61
4.6.6 Fine-Tuning DETR for Cylinder Detection . . . 62
4.7 YOLO . . . 63
4.7.1 YOLO Architecture . . . 63
4.7.2 Fine-Tuning YOLO Using Ultralytics . . . 67

5 Dataset 69
5.1 Dataset Creation . . . 69
5.1.1 Overview of the Dataset Creation Process . . . 69
5.1.2 Use of Habitat-Sim for Synthetic Data Generation . . . 70
5.1.3 Loading HM3D Scenes into the Simulator . . . 70
5.1.4 Object Placement in Scenes using Rigid Object Management API . . . 71
5.1.5 Simulation and Image Capture . . . 71
5.2 Dataset Statistics and Output Format . . . 72

5.2.1 Data Augmentation for Dataset Expansion . . . 74
5.2.2 Dataset Annotation Using Roboflow . . . 76
5.3 Dataset Annotation Formats . . . 77
5.3.1 YOLO Annotation Files . . . 77
5.3.2 DETR Annotation Files . . . 77

6 Experimental Results and Evaluation 79
6.1 Fine-Tuning Object Detection Models . . . 79
6.1.1 Fine-Tuning YOLO Models on Custom Dataset . . . 79
6.1.2 Fine-Tuning DETR Models on Custom Dataset . . . 81
6.2 Memory Mechanism Ablation Study . . . 82
6.3 Object Detection Verification Using CLIP-Based Classification . . . 83

7 Conclusion 87

References 89

Acknowledgments 95

Listing of figures

2.1 Traditional AI vs. Embodied AI . . . 6
2.2 ObjectNav Success Zone . . . 8
2.3 2-On example . . . 9
2.4 Vision-and-Language Navigation (VLN) . . . 10
2.5 Example rendered . . . 15
2.6 Software stack . . . 17

3.1 Visprog example . . . 21
3.2 Modules currently supported in VISPROG. . . . 22
3.3 Program generation in VISPROG . . . 23
3.4 ViperGPT example . . . 24
3.5 ViperGPT . . . 25
3.6 Modular components in Tango . . . 26
3.7 VLFM . . . 29
3.8 Memory mechanisms in TANGO . . . 31
3.9 GOAT-Bench . . . 32
3.10 OVON policy . . . 35
3.11 Different task category questions in EQA . . . 36
3.12 LLM-Match evaluation . . . 38
3.13 (EQA) exampl in Tango . . . 39

4.1 IoU . . . 43
4.2 CLIP . . . 47
4.3 ConVIRT framework . . . 48
4.4 Vision Transformer . . . 52
4.5 Self Attention mechanism . . . 54
4.6 Positional embeddings . . . 55
4.7 OWL-vit . . . 57
4.8 Reducing the cosine similarity . . . 58
4.9 DETR Architecture . . . 61
4.10 YOLO Design . . . 65

5.1 Dataset Examples . . . 75
5.2 Image augmentations . . . 76

6.1 YOLO Detection Examples . . . 81
6.2 DETR Detection Examples . . . 82

Listing of tables

5.1 Comparison of HM3D with other 3D scene datasets. 70
5.2 Summary of image capture settings and dataset-level characteristics used in this project. . . . 73
5.3 Dataset Distribution 74

6.1 Performance Comparison of Fine-Tuned YOLO Models on Custom Dataset 80
6.2 Training summary 80
6.3 Performance Comparison of Fine-Tuned DETR Models on Custom Dataset 81
6.4 Ablation Study of Memory Mechanism in TANGO on MultiON 3-ON Task 83
6.5 Effect of CLIP-Based Classification on Object Detection Performance 84
6.6 Performance Comparison on MultiON 3-ON 84
6.7 Performance comparison 85

Listing of acronyms

ViT . . . . . Vision Transformer

CNN . . . . . Convolutional Neural Network

MSA . . . . . Multi-Head Self-Attention

MLP . . . . . Multi Layer Perceptron

MultiON . . . . . Multi Object Goal Navigation

1 Introduction

Embodied Artificial Intelligence (AI) focuses on developing agents that can perceive, navigate, and interact with their environments through sensory inputs and physical actions. Unlike traditional AI systems that operate on static data, embodied AI emphasizes real-time learning and decision-making within dynamic settings. This approach is essential for tasks such as autonomous navigation, object manipulation, and embodied question answering, where agents must adapt to changing conditions and incomplete information. Embodied AI integrates various AI disciplines, including computer vision, natural language processing, and robotics, to develop adaptable systems capable of handling complex real-world scenarios**[3]**.

To support large-scale research, simulators and frameworks like Habitat[4] and AI2-THOR[5] provide photorealistic 3D environments where agents can safely train, refine their behaviors, and benchmark performance on tasks such as navigation and interaction. These platforms allow researchers to measure progress in embodied AI by providing reproducible and controlled virtual environments for experimentation.

Modular zero-shot approachs have proven succesfull in a large variety of computer vision tasks [6], [7]. The extensions of these concepts in Embodied AI has remained largely unexplored, but in the last years significat progress has been made [8], [9]. The core of this work is to adapt TANGO to MultiON[2] task. which introduces a modular, zero-shot approach to task execution. TANGO leverages Large Language Models (LLMs) to dynamically compose solutions without requiring task-specific training. By integrating pre-trained vision, language, and navigation modules, TANGO supports various tasks.

The MultiON[2] task requires agents to navigate within a 3D environment and sequentially locate multiple target objects, such as colored cylinders. In each episode, the agent is provided with an ordered list of objects and must efficiently plan its exploration strategy to complete the task. The challenge lies in the agent's ability to integrate perception, memory, and path optimization. It must remember previously visited locations, minimize redundant movement, and manage long-horizon navigation objectives. MultiON serves as a benchmark for evaluating how well agents can generalize their navigation and object recognition capabilities across different scenarios.

Within the TANGO framework, the object detection module is central to task execution, particularly for identifying and locating targets in complex environments. The current implementation of the detection module employs OWL-ViT2 for general object detection and DETR for detecting objects within the COCO-class categories. However, in synthetic 3D scenes, these detectors often produce a high number of false positives. To mitigate this issue, TANGO forwards the detected bounding boxes to a classify module for further refinement. This module uses an open-set classifier based on CLIP [10], which maps both image features and textual descriptions into a shared feature space. By differentiating between similar subcategories the classify module improves detection accuracy and reduces false positives.

This research builds upon the TANGO framework by enhancing its detection pipeline for the MultiON task through the integration of advanced object detectors and refinement of the object classification process Specifically, I have implemented and fine-tuned advanced models such as YOLO[11] and DETR[12] to optimize performance.The main reason for selecting YOLO is its computational efficiency and speed, which makes it suitable for real-time embodied AI tasks. Additionally, I have created a custom dataset using dynamically placed objects (colored cylinders) with the Habitat simulator [4], leveraging the HM3D dataset [13].

The thesis is organized as follows:

• Chapter 2 provides an overview of Embodied Artificial Intelligence (Embodied AI), highlighting its principles, differences from traditional AI, and key applications. It covers major navigation challenges along with evaluation metrics . The chapter also explores the Habitat simulator, tracing its evolution from Habitat 1.0 to 3.0 and its significance in training embodied agents.

•Chapter 3 introduces TANGO, a training-free framework that leverages Large Language Models (LLMs) to enable embodied AI agents to perform open-world tasks without additional training. It presents key tasks evaluated within the TANGO framework, including GOAT-Bench for multi-modal lifelong navigation, Open-Vocabulary Object Navigation (OVON), and Embodied Question Answering (OpenEQA). The chapter also discusses how LLMs facilitate modular reasoning and program composition, integrating components like VisProg and ViperGPT. Additionally, it explores memory-augmented exploration policies inspired by Vision-Language Frontier Maps (VLFM) to enhance navigation efficiency.

• Chapter 4 presents the object detection module in TANGO, which is central to recognizing, localizing, and classifying objects in the environment. It details the integration of multiple detection models—including YOLO, DETR, and OWL-ViT—to balance speed, accuracy, and open-vocabulary capabilities. The chapter also discusses the fine-tuning process applied to YOLO and DETR for improved performance on a task-specific dataset, the evaluation metrics (e.g., mAP) used for model assessment, and the incorporation of vision-language models like CLIP for false-positive filtering. This comprehensive review of the object detection pipeline underlines its critical role in enabling robust scene understanding, navigation, and interaction within the TANGO framework.

• Chapter 5 presents the dataset used for training object detection models within the TANGO framework. It details the creation of a synthetic dataset using the Habitat-Sim simulator, specifically tailored for the MultiON challenge. The chapter describes the data generation pipeline, from loading HM3D[13] scenes to placing target objects and capturing images from multiple viewpoints. It also discusses dataset statistics, including object distributions and image resolution, and explores data augmentation strategies to enhance generalization. Additionally, it covers annotation formats, ensuring compatibility with YOLO and DETR-based models. This structured dataset serves as a foundational component for training robust object detectors in TANGO.

• Chapter 6 presents the experimental results and evaluation of the TANGO framework, focusing on Mul-

tiON task performance. It details the fine-tuning of YOLO and DETR models, an ablation study on memory mechanisms, and CLIP-based classification for reducing false positives. The chapter also compares TANGO with the MoPA framework, highlighting its superior zero-shot navigation capabilities.

• Chapter 7 reports the concluding remarks of the work, summarizing the contributions and suggesting future research directions for further advancements in embodied AI.

2

Overview of Embodied Artificial Intelligence

2.1 Embodied Artificial Intelligence: Definition, Scope, and Challenges

Embodied Artificial Intelligence (Embodied AI) [3] is an interdisciplinary field that integrates artificial intelligence into physical entities, enabling them to interact dynamically with their environment. Unlike traditional AI systems, which primarily rely on symbolic reasoning and abstract data processing, Embodied AI emphasizes the necessity of physical presence, sensorimotor integration, and real-time interaction. Intelligence in this paradigm emerges from an agent's continuous engagement with its environment, where learning occurs through action, perception, and adaptation rather than static data analysis.

A key distinction between Embodied AI and traditional AI lies in how cognition is approached. Traditional AI models process data in abstract computational spaces, relying heavily on pre-defined datasets and symbolic reasoning. These systems are effective in structured environments where inputs and outputs are well-defined, such as image recognition and natural language processing. However, they struggle with adaptability when confronted with real-world variations. Embodied AI, in contrast, operates on the principle that intelligence is inherently linked to an agent's physical experiences. Learning is achieved through direct interaction with the environment, where agents refine their behaviors through continuous feedback. This enables embodied systems to generalize beyond their initial training and respond dynamically to novel situations. As shown in Figure 2.1, Embodied AI differs from traditional AI by enabling agents to actively perceive and interact with their environment, rather than relying solely on precomputed data and static models.

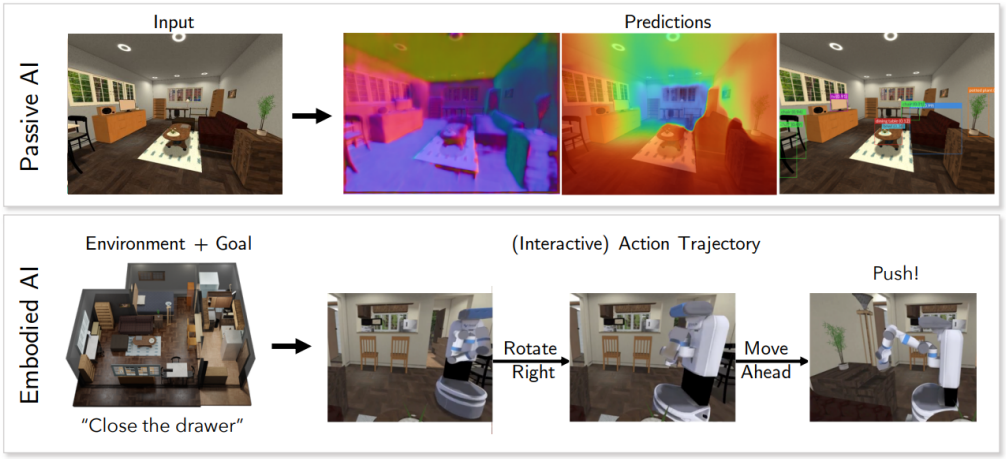

Figure 2.1: Traditional AI vs. Embodied AI. Traditional AI processes information in a static and abstract manner, while Embodied AI integrates perception, decision-making, and action, allowing agents to interact dynamically with their surroundings.The illustrative picture is taken from[3]

While Embodied AI overlaps with fields like robotics, computer vision, and machine learning, its focus is distinct. All robots are embodied, but not all embodied systems are robots—augmented reality (AR) glasses are one example. Robotics often emphasizes hardware, low-level control, and sensor processing, whereas Embodied AI explores intelligence in realistic environments, sometimes abstracting away those physical details.

For instance, the ALFRED[14] benchmark removes low-level manipulation and allows agents to interact using high-level commands (e.g., Open, Pickup) to complete tasks like "rinse the egg and put it in the microwave." Other work investigates cognitive functions such as navigation without requiring mapping or detailed control, highlighting how Embodied AI prioritizes intelligent behavior over physical realism.

The development of Embodied AI faces significant challenges. One major hurdle is the complexity of physical embodiment itself. Unlike software-based AI, embodied agents require sophisticated hardware capable of interacting with the real world. Designing and controlling robotic bodies that exhibit flexibility, dexterity, and adaptability remains an ongoing engineering challenge. Furthermore, real-time processing presents computational difficulties, as embodied agents must interpret vast amounts of sensory data and execute precise motor actions with minimal latency. Achieving the necessary speed and efficiency for real-time decision-making requires specialized hardware and optimized algorithms.

Another challenge is learning in dynamic and unpredictable environments. Unlike traditional AI models that rely on pre-collected datasets, embodied agents must continuously learn from ever-changing surroundings. The problem of lifelong learning remains an open research area, as embodied systems must retain previously acquired knowledge while simultaneously adapting to new experiences. Additionally, ethical and societal concerns arise with the increased deployment of Embodied AI in everyday life.

A critical aspect of Embodied AI research is the challenge of Sim-to-Real Transfer (Sim2Real)[15], which refers to the gap between behaviors learned in simulation and their real-world applicability. Simulation environments provide a scalable and cost-effective means for training embodied agents, but they often fail to capture the full complexity of the physical world. This discrepancy results in performance differences when deploying trained models outside of controlled simulations. Researchers have explored techniques such as domain adaptation, do-

main randomization, and policy fine-tuning to bridge this gap. The Habitat-PyRobot Bridge (HaPy) [15] is an example of an effort to enable direct execution of simulation-trained policies on physical robots. Evaluations using the Sim-vs-Real Correlation Coefficient (SRCC) have shown that standard simulation methods often yield low predictivity, as agents exploit simulator artifacts that do not translate into real-world conditions. However, advancements in simulation fidelity and adaptive learning strategies have significantly improved transferability, increasing the reliability of simulation-based training.

The evolution of Embodied AI continues to shape the future of intelligent systems, highlighting the shift from passive perception to active engagement. While substantial progress has been made in areas such as autonomous robotics, assistive technology, and multi-modal human-AI interaction, future research will focus on refining real-time learning mechanisms, improving simulation realism, and developing ethical frameworks for responsible deployment. The potential of Embodied AI extends beyond robotics and automation, influencing cognitive science and neuroscience by providing deeper insights into the fundamental nature of intelligence. As researchers work to overcome existing limitations, Embodied AI is expected to play an increasingly transformative role in various fields, driving innovations that enable artificial agents to operate more autonomously and interact seamlessly with the physical world.

2.2 Navigation Challenges

The tasks consist of an agent operating in a simulated 3D environment (e.g. a household), where its goal is to move to some target. For each task, the agent has access to an egocentric camera and observes the environment from a first-person's perspective. The agent must learn to navigate the environment from its visual observations.The challenges primarily differ based on how the target is encoded (e.g. ObjectGoal, PointGoal, AudioGoal), how the agent is expected to interact with the environment (e.g. static navigation, interactive navigation, social navigation).

2.2.1 PointGoal Navigation (PointNav)

In PointNav[16], the agent's goal is to navigate to target coordinates in a novel environment that are relative to its starting location (e.g. navigate 5m north, 3m west relative to its starting pose), without access to a prebuilt map of the environment. The agent has access to egocentric sensory inputs (RGB images, depth images, or both), and an egomotion sensor (sometimes referred to as GPS+Compass sensor) for localization. The action space for the robot consists of: Move Forward 0.25m, Rotate Right 30° , Rotate Left 30° , and Done. An episode is considered successful if the agent issues the Done command within 0.2 meters of the goal and within 500 maximum steps.

2.2.2 ObjectNav

In ObjectNav[17], the agent is tasked with navigating to one of a set of target object types (e.g. navigate to the bed) given ego-centric sensory inputs. The sensory input can be an RGB image, a depth image, or combination of both. At each time step the agent must issue one of the following actions: Move Forward, Rotate Right, Rotate Left, Look Up, Look Down, and Done. The Move Forward action moves the agent by 0.25m and the rotate and look actions are performed in 30° increments. Episodes are considered successful if the object is visible in

the camera's frame the distance between the agent and the target object is within 1 meter and the agent issues the Done action. The starting location of the agent is a random location in the scene. The concept of the success zone in ObjectNav is illustrated in Figure Figure 2.2 , emphasizing the importance of proximity, visibility, and navigability for successful episode termination.

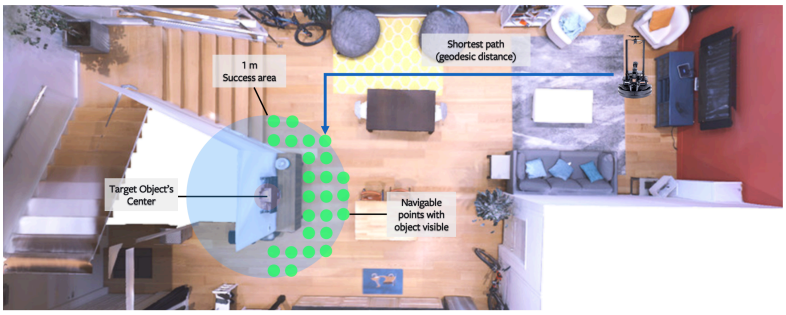

Figure 2.2: The goal object (a monitor, shown in pink) is surrounded by a 1m radius area (blue circle) defining potential success positions. Within this area, only navigable locations with a clear line of sight to the object—highlighted in green—constitute the success zone. If the agent stops within this zone, the episode is deemed successful. The blue arrows indicate the shortest path from the agent's initial random position to the nearest point in the success zone.

2.2.3 Instance-Image Goal Navigation

The InstanceImageNav [18]task involves guiding an agent to locate a specific object instance within an unfamiliar environment, using an RGB image that prominently displays the target object. The agent begins at a random position and must rely on egocentric vision and the provided image goal to identify and navigate to the exact instance of the object.This task differs from ObjectNav not only in how the goal is defined—using an RGB image rather than an object category label—but also in its requirement for precise instance matching. While ObjectNav allows the agent to stop near any object of the specified category (e.g., any bed in the environment), InstanceImageNav requires the agent to reach the exact object shown in the reference image. Stopping at any other instance of the same category would be considered a failure.

2.2.4 MultiON

It extends traditional ObjectGoal Navigation by requiring an agent to locate multiple objects in a predefined sequence, emphasizing the ability to store, retrieve, and utilize spatial and semantic information. Unlike simpler navigation tasks where an agent receives direct goal coordinates, MultiON[2] requires agents to explore their surroundings, recognize target objects, and recall their locations for sequential navigation. This introduces additional challenges, particularly in handling long-horizon dependencies and multi-step reasoning, making it an effective testbed for evaluating embodied AI agents.

The effectiveness of map-based navigation strategies in MultiON can be analyzed by comparing different types of memory representations. Agents may rely solely on implicit memory mechanisms, such as recurrent neural networks, or explicitly construct map-based representations of the environment. Some agents operate without

explicit maps, while others utilize oracle maps with ground-truth knowledge, storing either occupancy data or object location information. Learned map agents dynamically construct their representations by integrating egocentric sensory data. The core objective of MultiON is to determine what kind of information should be stored in maps to best assist navigation.

The MultiON framework is structured to allow controlled complexity in navigation tasks. The number of objects that an agent must locate is adjustable (e.g., 1-ON, 2-ON, 3-ON), making it possible to evaluate how well different agents handle increasing cognitive loads. Each episode consists of a unique ordered sequence of goal objects placed within a realistic 3D environment with varied layouts. Figure 2.3 illustrates an example of the 2-ON task, where the agent must navigate to two target objects in a predefined sequence The agent operates with a constrained action space, including movement and a FOUND action, which is used to confirm when an object has been reached. A key requirement of the task is that the agent must remember past observations and retrieve them later when revisiting previously encountered objects, distinguishing MultiON from simpler navigation problems.

Figure 2.3: An example of the 2-ON task involves navigating to two target objects in a specific order. The objects in the scene consist of eight distinct cylinders, each with a different color, placed within the environment

To perform effectively in MultiON, different navigation models are employed, ranging from implicit memory agents that use RNNs or memory-based transformers to store past observations, to egocentric mapping agents that dynamically construct maps from sensory inputs such as RGB and depth. Some approaches involve projected neural feature maps that store learned visual features in spatial memory for later retrieval, while others use oracle maps that provide full environmental knowledge to set theoretical performance limits. The structured evaluation of these models within MultiON demonstrates that map-based representations significantly enhance performance over agents relying solely on recurrent memory. Semantic object memory, in particular, proves to be more valuable than occupancy information alone, as it enables agents to efficiently recall and navigate towards previously identified goals.

MultiON serves as a robust benchmark for evaluating embodied AI agents in multi-object navigation. It provides valuable insights into the role of memory-based spatial reasoning and long-horizon planning in navigation tasks. The framework enables a deeper understanding of how agents can construct, utilize, and benefit from semantic maps, making it a critical tool for advancing research in embodied AI and cognitive navigation systems..

2.2.5 Vision-and-Language Navigation(VLN)

The Vision-and-Language Navigation (VLN) task[19] involves an autonomous agent navigating a real-world environment by following natural language instructions while relying solely on visual perception. The agent receives step-by-step textual guidance, such as "Walk past the sofa and turn right at the painting," and must interpret the

language, perceive its surroundings, and take appropriate navigation actions without prior knowledge of the environment's layout. The Vision-and-Language Navigation (VLN) challenge requires an autonomous agent to interpret natural language navigation instructions in a previously unseen real-world environment. Unlike traditional navigation tasks, VLN reflects the open-ended nature of real-world scenarios where both instructions and environments are not predefined.

The Room-to-Room (R2R)[19] task, introduced within requires an embodied agent to follow natural language instructions to navigate from a starting position to a target location. At the beginning of each episode, the agent receives an instruction sequence $\bar{x} = (x_1, x_2, ..., x_L)$, where L is the length of the instruction and each x_i represents a word token. The agent also perceives an initial RGB image observation o_0, determined by its starting pose $s_0 = (v_0, \psi_0, \theta_0)$, where v_0 is the 3D position, ψ_0 is the heading, and θ_0 is the elevation. The agent must then execute a sequence of actions $(s_0, a_0, s_1, a_1, ..., s_T, a_T)$, with each action leading to a new pose $s_{t+1} = (v_{t+1}, \psi_{t+1}, \theta_{t+1})$ and generating a new image observation o_{t+1}. The episode ends when the agent selects the special stop action, indicating that it has reached the goal location. As shown in Figure 2.4, VLN differs from VQA in that it requires generating a sequence of actions to control the camera viewpoint, rather than producing a textual response.

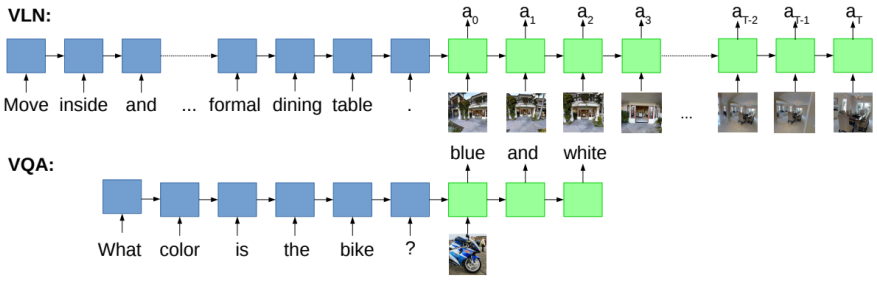

Figure 2.4: Differences between Vision-and-Language Navigation (VLN) and Visual Question Answering (VQA)[20]. While both tasks can be framed as visually grounded sequence-to-sequence problems, VLN involves significantly longer sequences. Unlike other vision-and-language benchmark tasks that use real images, VLN requires the model to generate a sequence of actions $(a_0, a_1, ..., a_T)$, which actively control and adjust the camera viewpoint, rather than simply producing a textual response.THe figure is taken form [19]

The task is considered successful if the agent's final position v_T is within 3 meters of the target location v^*. The navigation error is defined as the shortest path distance between v_T and v^* in the navigation graph G, ensuring a measurable and precise evaluation.

2.2.6 Multi-Modal Lifelong Navigation

Goat-Bench[21] is a task where an embodied agent must navigate within an environment across multiple episodes. Each episode presents the agent with different goal specifications that are provided in various input modalities.

The task emphasizes handling multi-modal goals, requiring the agent to navigate to targets defined by diverse inputs, including category names (e.g., "recliner chair"), natural language descriptions (e.g., "the white book on the coffee table"), and reference images (e.g., a picture of an oven).

Additionally, the task stresses lifelong learning capabilities. The agent is expected to improve its performance by leveraging experiences from previous tasks. It must explore the environment effectively and use memory to plan and optimize its navigation strategy over time.

2.2.7 Embodied Question Answering

The OpenEQA[22] task is an Embodied Question Answering (EQA) problem where an AI agent must understand an environment well enough to answer open-ended natural language questions. This understanding can be achieved through episodic memory (using past observations) or active exploration (navigating and gathering information). The task evaluates an agent's ability to recognize objects, spatial relationships, functional reasoning, and world knowledge. Agents process multi-modal sensory inputs like images, depth, and trajectories to generate accurate responses. The task is designed to test situated perception, memory, and reasoning in real-world environments. Evaluation is done through open-ended answer generation, scored based on similarity to human-annotated responses. OpenEQA serves as a benchmark for developing and assessing AI models in embodied AI, robotics, and interactive assistants.

2.2.8 Open-Vocabulary Object Goal Navigation

In Open-Vocabulary Object Navigation (OVON)[23], an agent is instructed to navigate to an object described in free- form language, rather than being restricted to a predefined set of object categories. This task enables agents to interpret language-based object descriptions, requiring the integration of visual and textual modalities to locate target objects within an environment. OVON benchmarks test an agent's generalization ability to navigate toward unseen object categories without explicit retraining.

2.3 Evaluation Measures

2.3.1 Success Rate (SR)

This metric [16]measures the proportion of episodes in which the agent successfully reaches the specified goal. It provides a straightforward indication of the agent's effectiveness in completing navigation tasks:

$$\text{SR} = \frac{1}{N}\sum_{i=1}^{N} S_i \tag{2.1}$$

where S_i is a binary indicator (1 if the agent reaches the goal, 0 otherwise).

2.3.2 Distance to Goal (DTG)

This measures how close the agent is to the goal at the end of an episode:

$$\text{DTG} = \frac{1}{N}\sum_{i=1}^{N} d_i \tag{2.2}$$

where d_i is the geodesic distance from the agent's final position to the goal.

2.3.3 Success weighted by Path Length (SPL)

To address the limitations of SR and PL individually, SPL[16] combines both metrics. It evaluates the efficiency of the agent's path by weighting the success of reaching the goal against the length of the path taken, normalized by the shortest possible path length. This metric is particularly valuable as it balances both success and efficiency, providing a more comprehensive assessment of navigation performance.

$$\frac{1}{N}\sum_{i=1}^{N} S_i \frac{\ell_i}{\max(p_i, \ell_i)}.$$

2.3.4 MultiON Metrics

Success (Binary Indicator)

The **SUCCESS** metric determines whether an episode is **completed successfully**. An episode is considered successful if the agent **visits all designated goals in the correct sequence** and correctly executes the **FOUND action** at each goal **within the allowed step limit**. If the agent **incorrectly calls FOUND**, the episode is **immediately terminated**.

$$\text{SUCCESS} = \begin{cases} 1, & \text{if agent reaches all goals in order and correctly calls FOUND at each} \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

Progress (PROGRESS)

The **PROGRESS** metric quantifies the **proportion of goals successfully located** within an episode. It is computed as the **ratio of found goals to the total number of goals**. For single-object navigation tasks (**1-ON**), this metric is **equivalent to SUCCESS**.

$$\text{PROGRESS} = \frac{\text{Number of correctly found goals}}{\text{Total goals in the episode}} \tag{2.4}$$

Success weighted by Path Length (SPL)

is extended in **MultiON** to evaluate performance in **multi-goal navigation**. It considers both **task success** and **path efficiency**, rewarding agents that reach their goals while traveling the shortest possible path.

$$SPL = s \cdot \frac{d}{\max(p, d)} \tag{2.5}$$

where:

- s is the **binary success indicator** (1 for success, 0 otherwise),
- p is the **total path length** traveled by the agent,
- $d = \sum_{i=1}^{N} d_{i-1,i}$ is the **total geodesic shortest path distance** from the starting position through all goal locations,
- $d_{i-1,i}$ represents the **geodesic distance** between goal $i-1$ and goal i, with goal 0 **being the starting point**.

Progress-weighted Path Length (PPL)

The **PPL** metric is an adaptation of **SPL**[16]that incorporates **partial credit** for agents that locate only a subset of the total goals. Instead of relying on binary success, it weights the **path length based on the fraction of goals found**.

$$PPL = \bar{s} \cdot \frac{\overline{d}}{\max(p, \overline{d})} \tag{2.6}$$

where:

- $\bar{s}$ is the **PROGRESS value** (fraction of goals found),
- $\overline{d} = \sum_{i=1}^{l} d_{i-1,i}$, where l is the **number of successfully found goals**,
- p and $d_{i-1,i}$ are defined as in **SPL**.

Unlike **SPL**, which assigns equal importance to every goal, **PPL ensures that shorter goal-to-goal distances do not disproportionately affect the final score**. For single-object navigation tasks (**1-ON**), **PPL is identical to SPL**.

2.3.5 Metrics in Embodied Question Answering (EQA)

In Embodied Question Answering (EQA), we evaluate both navigation and QA performance using the following metrics:

Navigation Metrics

- d_T **(Final Distance)**: Distance from the agent's final position to the target object. Lower is better.
- T_{10}, T_{30}, T_{50}: The agent is spawned at 10, 30, or 50 actions away from the target, and its final navigation distance d_T is measured.
- **Random**: The agent is spawned at a random distance from the target.

Question Answering (QA) Metrics

- **Top-1 Accuracy** : Percentage of correctly answered questions.
- T_{10}, T_{30}, T_{50}: QA accuracy when the agent starts from 10, 30, or 50 actions away from the target.
- **Random**: QA accuracy when the agent starts from a random location.

These metrics assess how well the agent navigates and answers questions when starting from different distances, including both predefined distances and a random spawn.

2.4 Habitat Simulator: A Platform for Embodied AI Research

2.4.1 Introduction

The development of Habitat-AI[24] is a long-term initiative aimed at establishing a unified framework for research in embodied AI. The platform is designed to facilitate systematic progress in developing and evaluating embodied agents—AI systems that perceive and interact with their environments through sensors and actuators. By providing a high-performance simulation environment, modular APIs, and a scalable architecture, Habitat-AI enables researchers to train, test, and benchmark AI models in realistic 3D environments.

Design Principles and Requirements

To support a broad range of embodied AI research tasks, Habitat-AI is built around the following design requirements:

High-Performance Rendering Engine: The platform includes an optimized rendering pipeline capable of generating multiple visual outputs (e.g., RGB, depth, semantic segmentation, surface normals, and optical flow) efficiently, even when multiple agents operate concurrently. Flexible Dataset Integration: A standardized API allows seamless integration of various 3D scene datasets,AI2-THOR[5], Gibson[25], and Matterport3D[26] making the platform dataset-agnostic and enabling researchers to use custom environments.

Modular Agent API

The framework supports parameterized agents, allowing users to define their geometry, physics, and actuation properties to simulate diverse embodied AI scenarios. Comprehensive Sensor Suite API: Researchers can configure an arbitrary number of sensors, including RGB cameras, depth sensors, GPS, compass, and contact sensors, which are attached to agents for multimodal perception. A standardized mechanism for defining tasks and their evaluation protocols ensures portability and comparability of research results across different studies. The platform is developed with a C++ core for computational efficiency while offering an accessible Python API for ease of use and compatibility with popular deep learning frameworks. Habitat-AI supports distributed training on computing clusters, enabling large-scale experiments and efficient deployment of models. The framework includes mechanisms for human-in-the-loop simulations, allowing researchers to study human behavior in virtual environments and analyze human-agent interactions. Users can programmatically modify the environment by changing object placement and scene configurations, making it adaptable for various experimental setups. Figure 2.5 illustrates sample outputs from RGB, depth, and semantic sensors used in simulation.

Figure 2.5: Example rendered sensor observations for three sensors (color camera, depth sensor, semantic instance mask). The figure is taken from [24]

2.4.2 System Architecture

Habitat-AI is designed with a multi-layer architecture to provide flexibility and efficiency in simulation, data management, and agent interaction. The platform consists of two primary components: **Habitat-Sim** – A high-performance 3D simulator responsible for loading 3D environments, configuring agents and their sensors, simulating agent motion, and providing sensory data. **Habitat-API** – A modular high-level library that facilitates task definition, agent interaction, and dataset handling, making it easier to develop and evaluate embodied AI models.

Habitat-Sim

The Simulation Core At the lowest level, Habitat-Sim serves as the core simulation engine. It is responsible for:

- **Efficient Scene Representation:** Uses a hierarchical scene graph to standardize 3D environment representation, making it compatible with various synthetic and real-world datasets. This abstraction enables procedural scene generation, scene editing, and real-time programmatic modifications.
- **Optimized Rendering Engine:** The simulator leverages Magnum, a high-performance graphics middleware, to ensure efficient frame rendering across different hardware setups. A key feature of its rendering pipeline is the use of a multi-attachment "uber-shader", allowing multiple sensor outputs (e.g., RGB, depth, semantic masks) to be generated in a single render pass. This optimization significantly reduces computational overhead.
- **Scalability and Performance:** Habitat-Sim operates at thousands of frames per second per simulator thread, making it orders of magnitude faster than previous embodied AI simulators.
- **GPU Optimization:** Frames are exposed as Python tensors via shared memory, with ongoing improvements focused on eliminating GPU-to-CPU memory transfers through CUDA-GL interoperation, which is expected to double rendering speed.

Habitat-API

A High-Level Interface for Embodied AI On top of the simulation core, Habitat-API provides a flexible, task-oriented interface that simplifies interaction with the environment. It consists of the following key components:

- **Task Module:** Extends the simulator's observation and action space, defining task-specific evaluation criteria.
- **Episode Module:** Defines individual experimental instances, including the agent's initial position, goal position, scene ID, and optional shortest path information.
- **Environment Module:** Abstracts all necessary components for working on embodied AI tasks, integrating the simulator, task definitions, and dataset handling.

These modules enable end-to-end embodied AI development by bridging the gap between simulation-based perception and decision-making algorithms. The overall task setup and sensory configuration used in Habitat-AI simulations are illustrated in Figure 2.6.

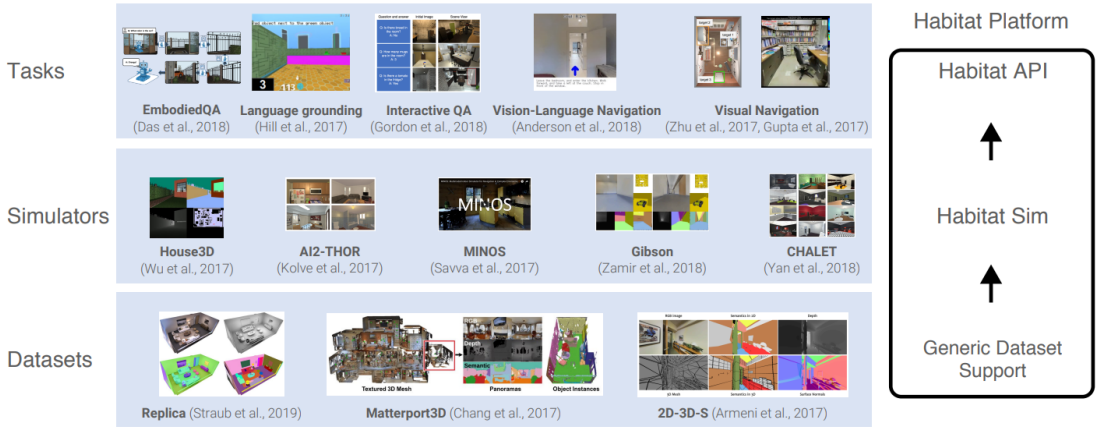

Figure 2.6: The 'software stack' for training embodied agents. Image is taken from [24]

2.4.3 Executing Tasks in Habitat-AI

The Habitat-AI framework offers a structured methodology for defining and executing embodied AI tasks. These tasks are specifically designed to evaluate an agent's capability to perceive, plan, and navigate within complex 3D environments using only its onboard sensory inputs.

In a standard Habitat-AI task, the process begins by placing an agent at a random position and orientation within a scene. The agent is then given a goal state, which it must reach by relying solely on its sensors—without any prior map of the environment. The action space available to the agent is continuous, enabling it to move in a realistic manner and interact with obstacles along the way. This setup allows for lifelike navigation and interaction, rather than simplified or discrete control schemes.

The framework is modular by design. This ensures that tasks can be easily reproduced and compared across different datasets, simulation settings, or research implementations. It also supports a wide range of task-specific configurations to reflect realistic embodied intelligence scenarios.

To accurately simulate physical interaction with the environment, agents are represented as physical bodies with defined shapes and movement constraints. Unlike in some simulators that allow "teleportation-based" navigation, Habitat-AI employs a realistic collision model. This means agents may partially move or slide along surfaces when encountering obstacles, making the simulation more true-to-life.

Habitat-AI also allows for different types of goal specification. In static goal configurations, the target location remains unchanged throughout the episode, closely mimicking typical real-world navigation tasks. For localization, the platform supports the use of idealized sensors like GPS and compass modules to provide orientation and positional feedback when needed.

Regarding perception, agents are equipped with flexible, configurable sensor suites. A standard setup typically includes RGB cameras to provide visual input and depth sensors to capture spatial structure. In many cases, multimodal combinations such as RGB-D are used to enrich the agent's understanding of its surroundings. GPS and compass sensors can also be integrated for tasks requiring precise localization.

3

Introduction to TANGO

This chapter provides a detailed description of the tasks and datasets used to evaluate embodied AI agents within the TANGO framework.

3.1 Overview of TANGO: Training-free Embodied AI Agents for Open-world Tasks

TANGO[8] is a novel framework that leverages Large Language Models (LLMs) to enable embodied AI agents to perform a diverse range of tasks without requiring additional training. Unlike traditional approaches that rely on extensive task-specific fine-tuning, TANGO takes advantage of the compositional reasoning capabilities of LLMs to dynamically structure and execute tasks in an open-world setting. The foundation of TANGO lies in its ability to integrate different functional modules using LLMs, allowing the agent to reason about its environment and adapt to various scenarios. The key components of TANGO include:

- **LLM-driven Program Composition:** Instead of training an agent for each specific task, TANGO constructs executable task programs in real-time using in-context examples, enabling flexible and generalizable behavior.
- **Navigation and Exploration Strategies:** TANGO employs a combination of a PointGoal Navigation model[27] and a memory-based exploration policy. This enables the agent to efficiently explore new environments and locate objects of interest, even in unfamiliar settings.

What sets TANGO apart is its ability to achieve state-of-the-art results in these tasks without requiring additional training or fine-tuning. This paradigm shift suggests that LLMs, when appropriately structured, can serve as powerful tools for orchestrating task execution in embodied AI. By allowing agents to interpret instructions,

explore environments, and provide meaningful responses, TANGO significantly enhances the feasibility of AI agents operating in real-world scenarios without the burden of extensive pretraining.

TANGO demonstrates how training-free approaches can enable embodied agents to generalize across tasks, marking an important step toward more adaptable and scalable AI-driven navigation, exploration, and interaction in open-world environments.

3.2 Modularity trough Large Language Models

This section presents a detailed account of the role Large Language Models (LLMs) play within the TANGO framework, emphasizing how they facilitate structured and modular reasoning in embodied AI tasks. It begins with an overview of VisProg—a modular visual reasoning system that has influenced TANGO's architecture—and then delves into how LLMs are employed for task composition, navigation, and execution within the system.

3.2.1 VisProg: Modular Program Composition for Visual Reasoning

Recent advances in **neuro-symbolic AI** have introduced methods for solving complex visual reasoning tasks by dynamically composing **modular programs**. **VisProg**[6] is a prominent example of this approach, demonstrating how **large language models (LLMs)** can generate **Python-like programs** that execute sequences of vision-based operations without requiring task-specific training.

VisProg addresses **multi-step visual tasks** by leveraging the **in-context learning ability** of **GPT-3**[28], where instead of directly solving a task through a single model prediction, it generates a structured program composed of **modular steps**, each invoking **pre-trained vision models, symbolic functions, or logical subroutines**.

Program Synthesis Through In-Context Learning

A **key innovation** of VisProg is its ability to **dynamically generate structured programs** using in-context learning. Unlike traditional vision models that require end-to-end training for specific tasks, VisProg allows for **flexible and adaptable task execution** without additional fine-tuning. The process works as follows:

- The **LLM (GPT-3)** is provided with a **few-shot prompt**, consisting of **example inputs (natural language task descriptions) and their corresponding high-level programs**.
- Without needing to **see** the input image, VisProg generates a **Python-like pseudo-program**, where each step **calls a specific vision module, processing intermediate outputs**.
- The generated program is **executed sequentially**, where each module's output serves as input for subsequent operations.

An example of VISPROG's compositional visual reasoning pipeline is shown in Figure 3.1, highlighting its modular execution and interpretable visual outputs.

Figure 3.1: VISPROG is a modular neuro-symbolic system for compositional visual reasoning. It uses in-context learning with GPT-3 to generate high-level programs from natural language instructions, which are then executed on input images to produce predictions. The system also provides interpretable visual rationales by summarizing intermediate outputs and supports tasks involving image understanding, manipulation, knowledge retrieval, and logical operations.The figure is taken from [6]

This approach enables VisProg to **generalize to new tasks** without requiring explicit re-training. By leveraging **pre-existing models and symbolic reasoning**, VisProg demonstrates a **scalable and reusable** approach to vision-based problem solving.The modules currently supported in VISPROG are shown in Figure 3.2

Modular Execution of Vision Tasks

VisProg operates by **sequentially executing modular components**, where each line in the generated program represents a **specific operation**. Unlike monolithic models that must be trained end-to-end, VisProg provides a **structured approach to task execution**, allowing individual vision modules to work together dynamically.

Each step in a VisProg-generated program can invoke:

- **Pre-trained neural models** for standard vision tasks (e.g., object detection, segmentation, classification).
- **Symbolic and logical operations** such as counting, spatial reasoning, and comparative analysis.
- **Knowledge retrieval** functions to integrate external information for enhanced reasoning.

Each module is implemented as a **Python class**, responsible for:

- **Parsing the execution step**, extracting the required inputs and outputs.

- **Performing computations**, potentially invoking **pre-trained models**.
- **Updating the program state**, ensuring results are passed correctly to later steps.

VisProg maintains **full transparency** in decision-making, making it possible to inspect the reasoning process behind each step of execution. This ensures **interpretability**, a crucial advantage over black-box deep learning models. The set of functional modules used by VISPROG is illustrated in Figure 3.2.

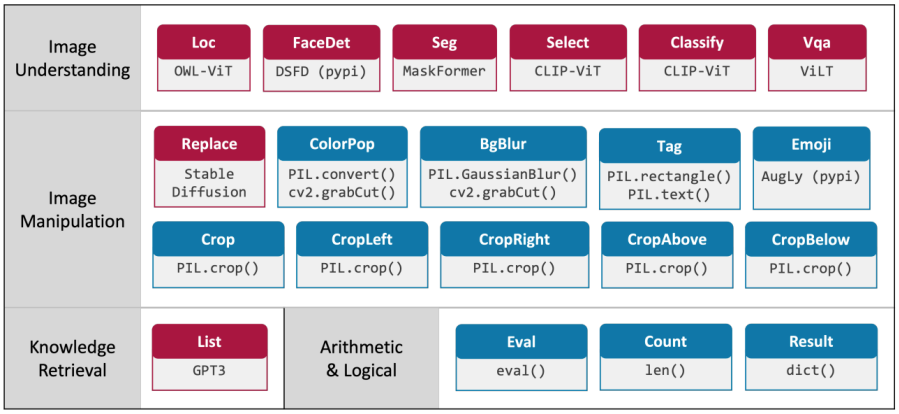

Figure 3.2: Modules currently supported in VISPROG.

Advantages of VisProg Over End-to-End Models

VisProg offers several advantages compared to conventional **end-to-end trained models**, particularly in scenarios requiring **multi-step reasoning and compositionality**. These advantages include:

- **High-level composability**: Instead of relying on a single network, VisProg dynamically composes **task-specific pipelines**, making it inherently modular and reusable.
- **Integration of multiple models**: The framework can **invoke various pre-trained models** instead of relying on a single learned representation, allowing for task flexibility.
- **Interpretable reasoning**: Each generated program is **human-readable**, allowing researchers and developers to **inspect and debug** decision-making processes.
- **Zero-shot generalization**: Since VisProg is guided by **few-shot in-context learning**, it can execute **unseen tasks** without requiring additional training.

This modular approach makes VisProg highly suitable for **tasks requiring diverse capabilities**, such as visual question answering, spatial reasoning, and image manipulation. Figure 3.3 illustrates the program generation process in VISPROG.

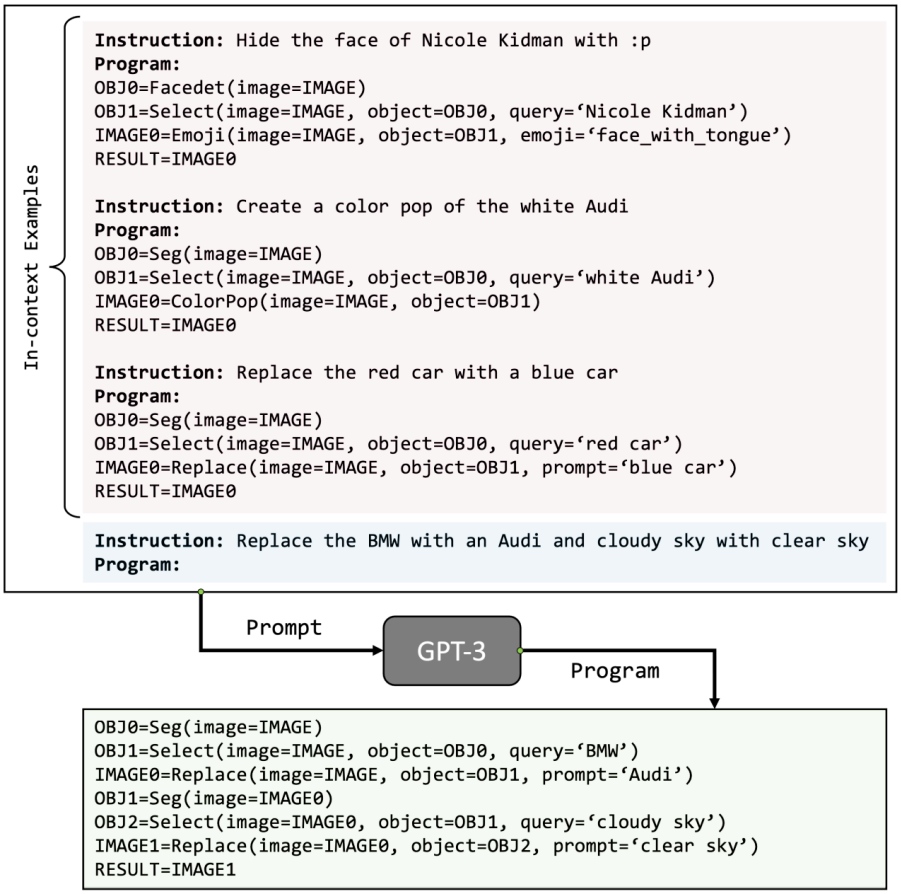

Figure 3.3: Program generation in VISPROG.The figure is taken from[6]

3.2.2 ViperGPT:Visual Inference via Python Execution for Reasoning

Solving complex visual tasks requires more than just high-performance neural networks; it necessitates compositional reasoning, where models can dynamically sequence different processing steps. ViperGPT[7] introduces a framework that enables modular task execution through code generation, allowing vision and language models to work together in a structured way. Unlike traditional deep learning architectures that rely on end-to-end optimization, ViperGPT achieves state-of-the-art results by generating Python programs that call upon pre-trained vision models, reasoning functions, and mathematical operations in a flexible and interpretable manner.

The motivation behind ViperGPT is that many real-world vision-and-language problems involve multiple steps of reasoning, similar to how humans break down complex problems. For instance, answering a query about an image may require object detection, counting, and mathematical computations. Instead of learning all these steps within a single monolithic neural network, ViperGPT dynamically composes the solution by leveraging pre-existing models and Python-based logic. Figure 3.4 illustrates how ViperGPT breaks down a visual query into

logical, interpretable steps.

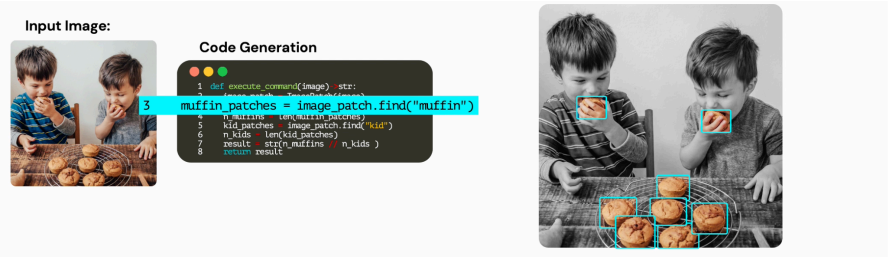

Figure 3.4: ViperGPT decomposes visual queries into interpretable steps for the query ' how many muffins can each kid have for it to be fair? .'

Compositional Reasoning Through Program Synthesis

ViperGPT differs fundamentally from traditional deep learning methods by relying on programmatic reasoning instead of direct input-to-output mappings. When provided with an input image and a textual query, it generates a Python program that outlines the sequence of operations needed to answer the query. This program includes function calls to specialized modules for vision processing, depth estimation, logical inference, and mathematical computation. The generated code is then **executed**, resulting in a final prediction that is both transparent and interpretable. Unlike static neural architectures that hard-code reasoning pathways, ViperGPT dynamically adapts its reasoning process based on the specific query, enabling more flexible and context-aware inference. ViperGPT employs a language model specialized in code generation, such as Codex[29], to construct task-specific programs. Rather than relying on supervised learning or reinforcement learning to define task structures, the model infers the required composition of functions through in-context learning. The process of generating programs begins by analyzing the input query to identify the necessary functions and models. Using an API of available modules, the system then constructs structured Python code that invokes relevant vision models and reasoning functions. This program is executed step by step, where intermediate outputs are used as inputs for subsequent operations, allowing the system to build complex reasoning pipelines dynamically This approach eliminates the need for extensive dataset-specific training, allowing ViperGPT to generalize across a wide variety of tasks.

ViperGPT API and Modular Execution

A central feature of ViperGPT is its API, which provides a structured way to interact with vision and reasoning models. The API defines a set of functions that serve as modular building blocks for answering complex visual queries. Instead of directly training a model to perform an entire task, ViperGPT composes task-specific programs by calling these predefined functions. The API consists of three main categories of functions:

- **Image Processing Modules:** These handle tasks such as object detection, segmentation, and depth estimation.
- **Mathematical and Logical Operators:** These enable operations like counting, averaging, and performing set-based reasoning.

- **Vision-Language Integration Functions:** These allow the system to interpret and connect visual outputs with natural language understanding.

By abstracting these functions within an API, ViperGPT ensures flexibility and adaptability, allowing the model to generate programs without being constrained by a fixed task formulation.

Figure 3.5 shows how ViperGPT solves complex visual queries through programmatic execution.

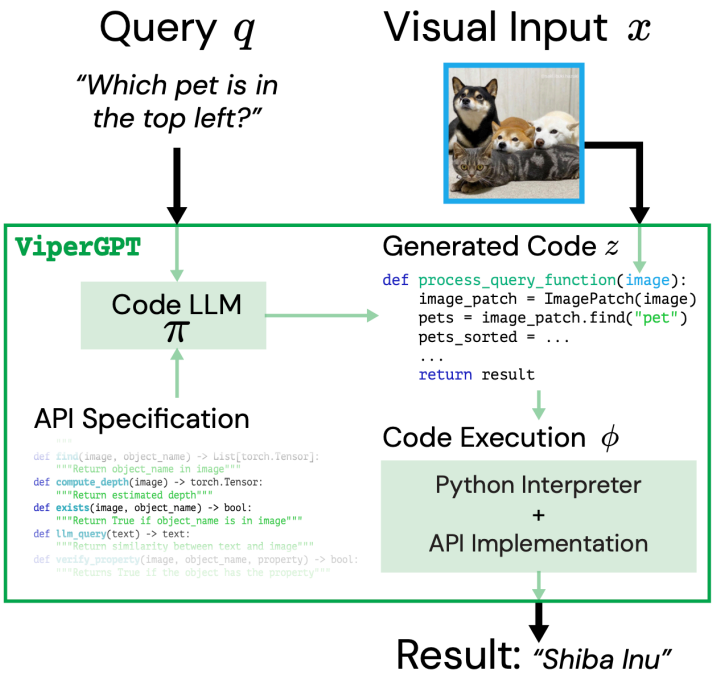

Figure 3.5: ViperGPT is a framework for solving complex visual queries programmatically.The figure is taken from [7]

3.3 Large Language Models in TANGO

Recent advancements in Large Language Models (LLMs) have enabled powerful programmatic reasoning, allowing structured task execution by composing modular functions. While these capabilities have been widely utilized in vision-and-language tasks, their application to embodied AI and real-world navigation has remained largely unexplored.

TANGO extends these capabilities by leveraging LLMs as high-level planners, allowing agents to dynamically sequence modular primitives for solving embodied tasks without requiring additional training. Instead of relying on end-to-end learning, which requires extensive fine-tuning, TANGO uses in-context learning, where an LLM generates structured programs based on a small set of examples. This enables the system to generalize across multiple tasks, making it highly adaptable to different environments. TANGO integrates modular program execution in a dynamic, interactive environment, addressing the challenges of navigation and real-world perception

that were not considered in VisProg. In the next section, we explore how TANGO extends these concepts to embodied tasks, incorporating LLMs for structured decision-making in complex 3D environments.

3.3.1 Program Composition and Task Planning with LLMs in TANGO

A fundamental innovation in TANGO is its modular program composition approach. Instead of relying on monolithic, end-to-end trained policies, TANGO decomposes complex tasks into smaller, interpretable submodules that interact with vision, language, and action components.

This method draws inspiration from VisProg[6] and ViperGPT[7], which demonstrated how in-context learning can generate compositional programs that sequence multiple operations. While these earlier systems focused on static image tasks, TANGO extends this paradigm to embodied agents that must act, explore, and interact with the environment dynamically.

TANGO incorporates a diverse set of modular components designed to address vision, language, and navigation tasks. These modules are categorized based on their underlying mechanisms:

- Python-based subroutines handle logical operations, simple transformations, and planning mechanisms.
- Pre-trained vision models process environmental observations, aiding object detection and scene understanding.
- Navigation modules implement foundational PointGoal Navigation policies[27], with *explore scene* extending this capability through a memory-integrated approach.

The modular design enables the system to efficiently compose different functionalities, making it possible to address a wide range of tasks while ensuring interpretability and adaptability.

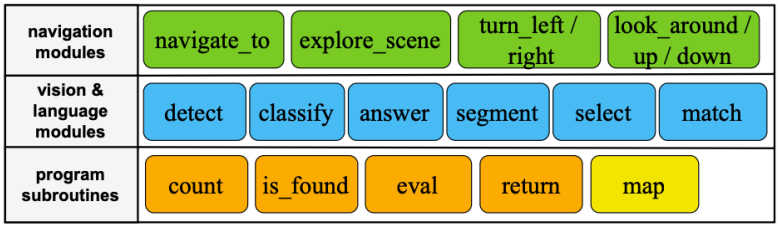

Figure 3.6: Modules span a variety of inputs and outputs. Orange modules use Python subrou- tines, while blue modules use pre-trained computer vision models . The navigate to and explore scene modules, in green, both implement our foundational PointNav module; how- ever, only explore scene integrates the memory mechanism.The figure is taken from [8]

At the core of TANGO's reasoning framework is its LLM-powered task planner, responsible for translating natural language instructions into structured execution plans. Instead of relying on reinforcement learning or behavior cloning, TANGO provides the LLM with 15 in-context examples, enabling it to generalize across different tasks without requiring additional training.

The task planning process consists of three key steps:

- **Task Understanding**: The LLM processes the user query and determines the necessary sub-tasks.

- **Program Generation**: A structured program is composed, containing modular commands such as *navigate to location, detect object, retrieve memory, and answer query*.
- **Execution and Feedback**: The generated program is executed sequentially, and memory updates ensure improved decision-making in subsequent steps.

This structured reasoning approach ensures that task execution remains interpretable, modular, and adaptable, allowing the agent to flexibly respond to a wide range of embodied challenges.

3.3.2 TANGO Program Interpreter

A critical component of the TANGO framework is its **Program Interpreter**, which serves as a bridge between LLM-generated programs and real-world execution. While the LLM generates high-level plans, the interpreter is responsible for:

- **Parsing the Pseudo-Program**: Extracting structured execution steps from the LLM-generated task plan.
- **Executing Modular Primitives**: Calling pre-trained vision models, navigation modules, and interaction functions to complete the required steps.
- **Dynamic Adaptation**: Adjusting program execution in real-time, based on scene perception and memory retrieval.

At runtime, the interpreter processes each line of LLM-generated pseudo-code and executes corresponding primitives by interacting directly with Habitat-Sim. For instance, the `explore_scene()` command triggers a 360° camera sweep, initializes frontier-based exploration, and uses the `Target` module to identify promising regions in the environment. Under the hood, this maps to low-level Habitat-Sim actions such as `turn_left` and `move_forward`, executed via `self.habitat_env.execute_action()`.

Another example is `navigate_to(object_name)`, which uses detections from pretrained object detectors to compute target coordinates. The interpreter calls the `Target` module to convert bounding boxes and depth information into polar coordinates, which are then passed to Habitat-Sim for step-by-step movement toward the goal. During this process, the interpreter also updates the semantic map and stores observations, enabling memory-based navigation and recovery if the object is lost.

This design allows each high-level primitive to act as a wrapper around precise, environment-specific operations, while maintaining full interpretability and control throughout execution.

TANGO introduces a key innovation in its ability to **generalize** across diverse embodied AI tasks without requiring extensive fine-tuning. This is achieved through in-context learning, where the language model adapts to new tasks using example-based prompting, and zero-shot execution, which enables the system to handle novel navigation and interaction tasks without prior training. These capabilities make TANGO highly scalable and adaptable across varying task settings.

By combining these learning mechanisms, TANGO efficiently transfers knowledge across different tasks, making it highly scalable and adaptable.

3.4 Exploration Policy using Memory targets

TANGO incorporates a memory mechanism in the form of a stored feature map, allowing the agent to retain information about previously explored regions for more efficient navigation. The details of this memory system and its role in lifelong navigation are discussed in the following sections.

3.4.1 Overview of VLFM and Memory-Augmented Exploration

Developing autonomous agents that can navigate unfamiliar environments using semantic understanding is a key step toward achieving human-like search behavior. Unlike traditional methods that depend on predefined maps or task-specific training, zero-shot navigation approaches seek to generalize across environments without prior fine-tuning. One such method is **Vision-Language Frontier Maps (VLFM [30])**, which draws inspiration from human reasoning to enable semantic exploration in a zero-shot setting. VLFM enables an agent to navigate towards unseen semantic objects in unfamiliar environments by combining depth-based occupancy mapping with vision-language modeling. Unlike prior approaches that depend on object detectors or text-based reasoning, VLFM directly extracts semantic cues from RGB images and integrates them into a language-grounded value map. This allows for more efficient and interpretable navigation in novel settings. VLFM constructs occupancy maps from depth data to identify unexplored frontiers in the environment. To determine which frontier is most likely to lead to the target object, it uses a vision-language model that compares RGB observations from the agent's current viewpoint with a text prompt describing the object. This comparison is done via cosine similarity, producing semantic scores that are projected onto a spatial grid. The resulting language-grounded value map highlights the most promising regions to explore, enabling the agent to prioritize semantically relevant areas and improving overall navigation performance.

Building upon VLFM, the TANGO framework introduces a memory-augmented exploration mechanism that allows for sequential goal navigation by leveraging a stored feature vector map. By integrating this memory-driven approach, TANGO improves efficiency in long-term navigation by recalling previously seen target locations, thereby reducing redundant exploration.

3.4.2 Exploration and Value Mapping in VLFM

The VLFM framework operates in three sequential phases: **initialization, exploration, and goal navigation.** During the initialization phase, the agent performs a full 360-degree rotation in place to construct the initial frontier and value maps. These maps provide a spatial-semantic understanding of the environment, forming the basis for targeted exploration. In the exploration phase, the agent continually updates its maps with new observations, generating frontier waypoints and selecting the most semantically promising direction based on vision-language similarity scores. Once a valid instance of the target object category is detected, the system transitions to the goal navigation phase. In this final stage, the agent navigates toward the closest point on the detected object and issues a STOP action once it is within an acceptable distance. The VLFM consists of :

1. **Frontier Waypoint Generation:** VLFM constructs a *frontier map* by processing depth and odometry data, identifying unexplored regions that may contain the target object. The robot iteratively updates this map as

it moves through the environment, selecting new frontiers to explore based on their proximity and likelihood of leading to the object.

2. Value Map Construction: A key component of VLFM is the *value map*, which assigns a semantic relevance score to different regions in the environment. This is achieved using a pre-trained vision-language model (BLIP-2)[31], which computes a similarity score between the current RGB observation and a text prompt representing the target object. The resulting semantic scores are projected onto a top-down 2D grid, which informs the selection of the next frontier to explore.

To ensure robustness, VLFM employs a confidence-based update mechanism when revisiting previously seen areas. Confidence scores are assigned to each pixel based on its position relative to the robot's field of view. When overlapping regions are encountered again, the semantic scores are updated using a weighted average approach, incorporating both past and current observations.

3. Object Detection and Navigation: Once a target object instance is detected, VLFM transitions from exploration to goal navigation. Object detection is performed, and Mobile-SAM[32] is then used to extract the object contour, enabling precise localization.

For movement, VLFM employs a *Point Goal Navigation* policy[16], which guides the robot to the selected frontier or target object. The policy is trained with reinforcement learning and does not require explicit geometric maps, making it highly adaptable. Unlike prior approaches, VLFM does not rely on remote servers for computationally expensive text-based reasoning, making it deployable on standard consumer hardware.

Figure 3.7 illustrates how VLFM uses vision-language models to generate semantic value maps for target-driven navigation.

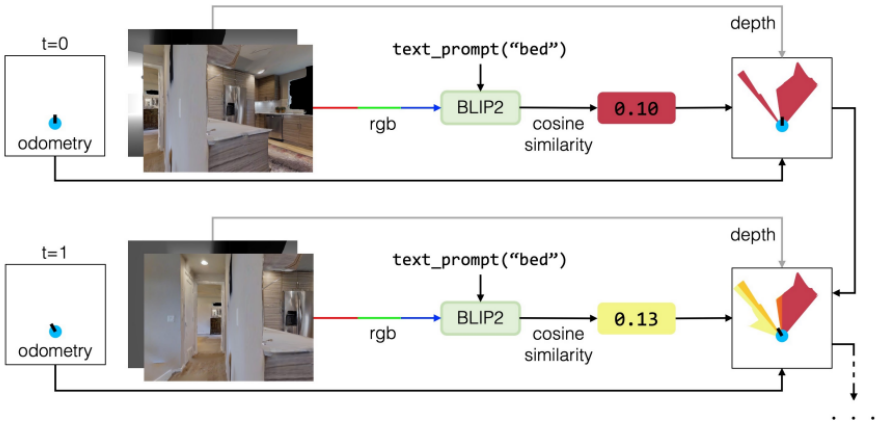

Figure 3.7: VLFM incrementally generates value maps for target-driven navigation by leveraging BLIP-2 to calculate the cosine similarity between an RGB image from the robot's current viewpoint and a text prompt describing the target object. The resulting semantic value scores are mapped onto a top-down 2D grid corresponding to the camera's field of view, with occluded areas excluded based on obstacle information derived from the depth image.Figure from[30]

3.4.3 Memory-Augmented Exploration in TANGO

TANGO extends the exploration policy of [30] by integrating a memory mechanism based on a stored feature vector map. At each step, the current RGB observation is processed using a vision-language model (BLIP-2[31]), which updates a triangular-shaped region in the map with feature vectors for each pixel. This feature map stores semantic information about previously seen areas and is used to refine navigation when encountering sequential goals.

When the navigation target changes, the value map is updated using the stored feature map. The new value map is computed by applying cosine similarity between the vector representation of each pixel and the embedding of the new target, obtained from a text description or image:

$$\cos\left(map^{val}_{(i,j)}\right) = \frac{\vec{map}^{feat}_{(i,j)} \cdot \vec{E}^{target}}{\left\|\vec{map}^{feat}_{(i,j)}\right\| \cdot \left\|\vec{E}^{target}\right\|} \quad \forall (i,j) \in map^{val} \tag{3.1}$$

where: - $\vec{map}^{feat}_{(i,j)}$ represents the feature vector stored at pixel (i,j), - $\vec{E}^{target}$ is the embedding vector of the specified target, - $map^{val}_{(i,j)}$ represents the updated semantic value map used in navigation. Frontiers are retrieved from an obstacle map calculated on the fly [30].

Once the new value map is obtained, the system evaluates whether the agent has previously encountered the target. This is determined by sampling the highest value in the updated value map. If this value exceeds a predefined threshold τ, it indicates that the agent "remembers" the target's location, prompting direct navigation to that position:

$$\max_{(i,j)} map^{val}_{(i,j)} > \tau \quad \Rightarrow \quad \text{Navigate to stored target location.} \tag{3.2}$$

If the target object is not found at the expected location, the agent resumes standard frontier-based exploration.

This memory-augmented mechanism improves efficiency in multi-goal scenarios by leveraging past observations, reducing redundant exploration, and enabling the agent to recall previously seen targets more effectively.

Figure 3.8 illustrates the memory mechanism used in TANGO during sequential target navigation.

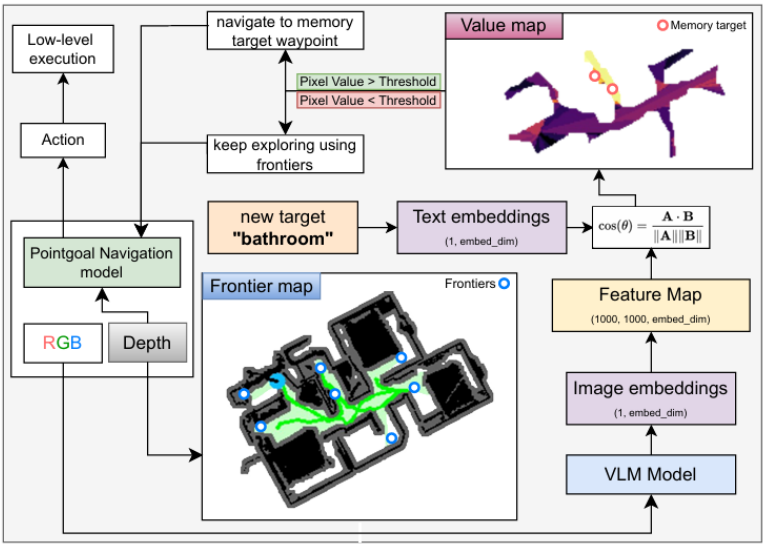

Figure 3.8: Exploration Policy. Illustration of the implemented memory mechanisms in TANGO, when a new sequential target is found. The figure is taken from [8]

3.5 Tasks: Open-World Navigation and Embodied AI Challenges

To showcase the versatility of the proposed framework, TANGO has been evaluated on three well-known benchmarks: (i) Open-Vocabulary ObjectGoal Navigation [23], (ii) Multi-Modal Lifelong Navigation [21], and (iii) Embodied Question Answering[22] . The results are comparable to or exceed those of previous methods, achieving this performance without the need for specialized training.Here, we briefly introduce each task.

3.5.1 GOAT-Bench

It stands for GO to AnyThing Benchmark[21], is a comprehensive evaluation framework designed to test embodied AI agents in multi-modal lifelong navigation tasks. These tasks require agents to complete sequences of navigation goals, where each goal is defined by a different input modality, such as category names, natural language descriptions, or reference images. The benchmark emphasizes critical capabilities such as generalization across modalities, multi-modal reasoning, and lifelong learning. This makes GOAT-Bench a crucial benchmark for advanced embodied AI systems like those developed in the TANGO framework. In GOAT tasks, agents are placed in simulated environments and tasked with navigating to multiple targets in a sequence. These targets may be specified through different modalities. For example, a task might instruct the agent to navigate to ”a chair” (category name), ”the red chair next to the dining table” (language description), or an image of a target object (reference image). The agent must correctly interpret each input, explore its environment, and locate the target

object while minimizing travel time and resource usage. An example of a multi-modal navigation task is shown in Figure 3.9

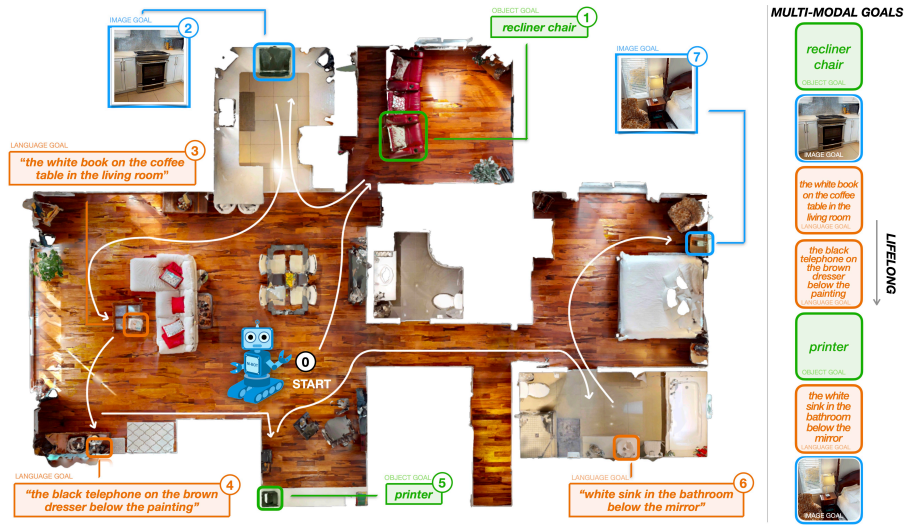

Figure 3.9: In this example, the agent is tasked with sequentially navigating to a series of targets: (1) a recliner chair selected from a fixed set of k categories, (2) a specific oven depicted in a reference image, (3) an object described in natural language as "the white book on the coffee table in the living room," along with other objects present in the environment. The goal of this benchmark is to drive progress toward developing universal, multi-modal agents capable of lifelong learning and generalization.The figure is taken from [21]

The benchmark is designed to evaluate several core aspects of agent performance. Multi-modal navigation assesses the agent's ability to handle tasks where goal specifications change modalities between episodes. For example, one episode might present goals through natural language instructions, while another uses reference images. Lifelong learning and adaptation are also critical. Agents are expected to leverage knowledge and experience from previous episodes to improve their performance over time, using memory to retain information about explored areas, detected objects, and environmental layouts. Additionally, GOAT-Bench emphasizes open-vocabulary navigation, where agents encounter novel object categories beyond the predefined training classes. This challenges the agent's ability to generalize to unseen scenarios without extensive retraining. In GOAT-Bench tasks, agents must perform sequential navigation to multiple targets within a single episode. Since agents are not provided with a map, they must explore and create internal representations of the environment. This requires advanced capabilities in sensory perception, object recognition, language comprehension, and memory management. An effective agent must interpret sensory inputs such as RGB and depth images, understand complex natural language descriptions, and recognize objects based on visual features. GOAT-Bench evaluates two main types of navigation models: monolithic reinforcement learning (RL) models and modular approaches. Monolithic models are trained end-to-end, mapping sensory inputs directly to actions. While these models can achieve high success rates, they often struggle with path optimization and generalization across unseen modalities. Modular approaches, on the other hand, decompose the task into components such as perception, exploration, and planning. These systems frequently leverage explicit memory representations, making them better suited for tasks that require adaptability

and reasoning. However, monolithic models tend to be more robust when handling variations in goal specifications, such as synonyms or paraphrased instructions. Despite its comprehensive design, GOAT-Bench highlights several challenges for embodied AI research. Multi-modal fusion, where agents integrate information from multiple sensory and input modalities, remains a significant challenge. Agents must efficiently combine data from visual, textual, and spatial inputs to make accurate decisions. Lifelong learning also presents difficulties, as agents must develop strategies to retain and apply knowledge across different episodes and environments. Furthermore, generalizing to open-world tasks, where agents encounter unfamiliar objects and instructions, requires advances in open-set recognition and representation learning. In this research, GOAT-Bench is a key tool for evaluating the TANGO framework within the Habitat Simulator.

3.5.2 Open Vocabulary Object Goal Navigation

Object Goal Navigation (ObjectNav) has traditionally relied on a **closed-set vocabulary**, meaning that agents are trained and evaluated on a predefined set of object categories. However, real-world applications demand a more flexible approach where agents can navigate to objects **described in free-form language**, even if they belong to **unseen categories**. This capability is known as **Open-Vocabulary Object Navigation (OVON) [23]**

Traditional approach for Open-Vocabulary Object Goal Navigation

Encoding the Goal Object Category and Observations

In Open-Vocabulary Object Navigation (OVON), the agent must navigate toward an object category specified using free-form language rather than a fixed set of predefined labels. This requires the system to encode both visual and textual inputs into a shared embedding space to enable effective decision-making.

To achieve this, the system utilizes SigLIP[33], a vision-language model that encodes both RGB observations and goal text descriptions into comparable feature representations. The encoding process consists of the following steps:

Visual Encoding: The agent's current visual observation, denoted as I_t, is passed through SigLIPRGB:

$$i_t = \text{SigLIPRGB}(I_t)$$

This produces a 768-dimensional feature vector, representing the agent's current first-person view.

Textual Encoding of the Goal Category: The goal object category G is encoded using SigLIPtext:

$$g = \text{SigLIPtext}(G)$$

This also produces a 768-dimensional feature vector, ensuring that the textual description is mapped into the same embedding space as the visual observations.

Previous Action Encoding: The agent's previous action a_{t-1} (e.g., moving forward, turning, stopping) is also considered as input. This action is transformed into a 32-dimensional embedding via a learnable embedding layer:

$$p_t = \varphi_a(a_{t-1})$$

Concatenation into a Unified Observation Vector: The three embeddings (visual, textual, and action) are concatenated to form the final observation embedding:

$$o_t = [i_t, g_t, p_t]$$

This consolidated representation captures the agent's current perception, goal information, and past movement, making it a rich feature vector for decision-making.

Processing Observations Using a Transformer-Based Policy

Once the observation embedding o_t is generated, it is processed by a decoder-only transformer network π_θ, which serves as the policy model that selects the agent's next action.

The transformer maintains a context window of the last 100 observations:

$$[o_{t-99}, \ldots, o_t]$$

This allows the agent to consider long-term dependencies and not just react to the most recent frame.

The transformer has 4 layers, 8 attention heads, and a hidden size of 512.

Each observation embedding is processed through self-attention layers, allowing the model to learn relationships between past visual experiences, goal descriptions, and previous actions. The transformer then outputs a feature vector representing the current state of the agent.

Action Selection via a Linear Action Head: The transformer's output is passed through a linear action head, which predicts a categorical distribution over possible actions:

$$a_t \sim \pi_\theta(\cdot | o_{t-99}, \ldots, o_t)$$

The agent samples an action from this distribution, selecting its next movement. During RL, an additional linear layer (critic head) is used to project the feature vector into a value estimate for the current state.

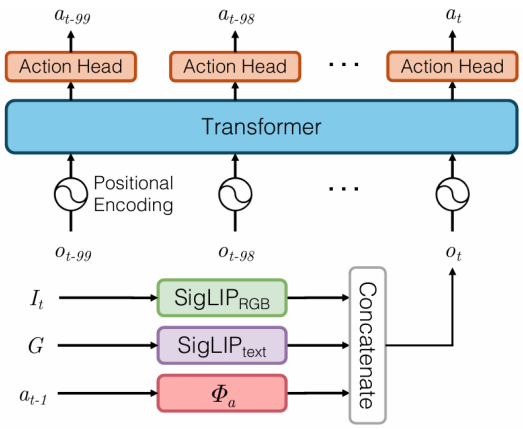

Figure 3.10: OVON policy encodes the current visual observation It, the goal object category G, and the previous action at−1 to form observation embedding ot. At each step, the embedding sequence for the past 100 time steps is fed into a transformer, which uses an action head to sample an action at.The figure is taken from [23]

Training Methods

Training Methods The HM3D-OVON benchmark employs multiple learning methodologies, including imitation learning, rein- forcement learning, and modular approaches. These methods are evaluated to determine their effectiveness in open-vocabulary object navigation.

Open Vocabulary Object Navigation in TANGO

TANGO performs Open Vocabulary Object Navigation (OVON) by leveraging a modular pipeline that integrates object detection, waypoint-based navigation, and real-time visual matching without requiring task-specific training. Unlike traditional ObjectGoal Navigation (ObjectNav), which trains agents to associate goal categories with navigation policies, TANGO remains task-agnostic and operates using a pre-trained PointGoal Navigation model[27] that solely moves the agent from one location to another.

To achieve object navigation without explicit category encoding, TANGO utilizes a zero-shot object detector, such as Owlv2[34] or DETR[12] (for COCO-class objects), to identify the target object within the environment. Instead of learning spatial priors or object-location associations, TANGO processes each detection dynamically and extracts the center of the detected bounding box as a waypoint.

Since the agent does not inherently differentiate between object categories or possess a learned object-search strategy, TANGO integrates a real-time matching module to refine navigation decisions in **image-based scenarios**, resulting in accurate navigation towards an instance image. This module, based on SuperGlue[35], optimizes the agent's ability to compare ongoing visual observations with the detected object, ensuring accurate localization and reducing ambiguity in open-set environments.

A critical distinction between TANGO and conventional ObjectNav approaches is that TANGO does not require reinforcement learning or supervised training on object categories. While standard ObjectNav methods encode goal categories and optimize movement policies for object-driven navigation, TANGO simply converts de-

tected objects into navigational waypoints, allowing it to generalize across diverse target categories without retraining. This modular and flexible framework enables TANGO to process arbitrary navigation targets and function effectively in open-world scenarios where object categories and spatial distributions are unknown.

3.5.3 Embodied Question Answering (OpenEQA)

Task Definition

Embodied Question Answering (EQA)[22] is a challenging problem in Embodied AI, where an agent is placed in a 3D environment and must navigate to gather information to answer a question posed in natural language. The OpenEQA dataset[22] enhances this task by providing over 1,600 high-quality, human-generated question-answer pairs sourced from more than 180 real-world environments. Example question types from the OpenEQA dataset are shown in Figure 3.11

Figure 3.11: Example questions and dataset statistics of OpenEQA,Image taken from [22]

Task Categories

The OpenEQA benchmark consists of two primary task categories:

- **Episodic Memory-EQA (EM-EQA)**: The agent must answer questions using episodic memory, which consists of stored sequences of past sensory observations.
- **Active-EQA (A-EQA)**: The agent must autonomously explore the environment to gather the necessary information and answer questions within a time constraint of 500 steps.

Active Embodied Question Answering (A-EQA)

Active Embodied Question Answering (A-EQA) is a task where an autonomous agent can explore its environment to gather the necessary information before answering a question. Unlike passive question-answering systems, an A-EQA agent actively navigates the environment to search for relevant information. For example, when asked, *"Do we have canned tomatoes at home?"*, the agent must move through the environment and verify their presence before responding, *"Yes, I found canned tomatoes in the pantry."*

For simplicity, the OpenEQA benchmark focuses on navigation-based exploration, where agents rely only on movement to locate objects or areas relevant to the question. However, in future applications, this approach can be extended to mobile robotic systems capable of both navigation and manipulation, allowing them to perform additional actions like opening doors or cabinets to retrieve information.

Formally, an instance of A-EQA is represented as a three-element tuple (Q, S, A^*):

- Q represents the question posed to the agent.
- A^* is the human-annotated correct answer.
- S refers to the simulator environment, initialized at a specific state, containing all the necessary assets to recreate the scene.

Once the agent is placed in the environment at state S, it must perform exploratory navigation to find relevant objects or locations before providing an answer A.

In Tango our focus is on Active-EQA (A-EQA), which requires the agent to balance exploration and efficiency while answering questions related to spatial reasoning, object attributes, and functional relationships.

LLM-Match: Evaluating Correctness of Answers

Since OpenEQA involves open-ended, free-form answers, traditional evaluation methods (e.g., exact-match accuracy) are insufficient. To address this, OpenEQA introduces LLM-Match, a Large Language Model (LLM)[36]-based evaluation protocol that assesses answer correctness in a more flexible and human-like manner.

LLM-Match works as follows:

- An LLM compares the agent's predicted answer with the human-annotated ground truth.
- The LLM then assigns a similarity score based on semantic relevance, ranging from 1 to 5, with 5 indicating a fully correct response.
- The LLM considers synonyms, rewordings, and contextual accuracy, making it more adaptable than exact-matching approaches.

The LLM-Match evaluation process is illustrated in Figure 3.12

Formally, the correctness score for an answer A is defined as: where **score** $\sigma_i \in \{1, \ldots, 5\}$.

$$E = \frac{1}{N} \sum_{i}^{N} \frac{(\sigma_i - 1)}{4} \times \frac{l_i}{\max(p_i, l_i)} \times 100\%, \tag{3.3}$$

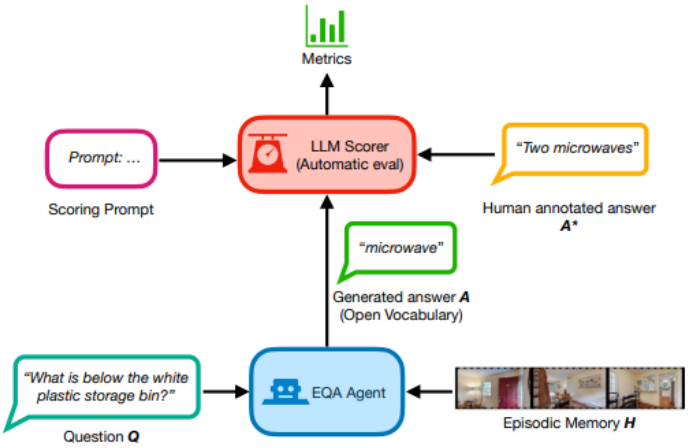

Figure 3.12: Illustration of LLM-Match evaluation and workflow. The figure is taken from [22]

Evaluating Efficiency for A-EQA

Efficiency in Active-EQA (A-EQA) evaluates how effectively an agent gathers the necessary information to answer a question while minimizing unnecessary exploration. It prioritizes agents that engage in targeted navigation, directing their movement towards areas that are most relevant to the question rather than aimlessly exploring the environment.

To measure efficiency, we weight the correctness score by the normalized path length, ensuring that agents taking the shortest sufficient route are rewarded. The normalization accounts for the timesteps taken by the agent relative to the optimal ground truth path length, which represents the minimum required steps needed to obtain the necessary information.

This metric ensures that agents are evaluated not only based on correctness but also on how strategically and efficiently they navigate to answer questions within the given time constraints.

Formally, our efficiency metric is defined as:

$$E = \frac{1}{N}\sum_{i}^{N}\frac{(\sigma_i - 1)}{4} \times \frac{l_i}{\max(p_i, l_i)} \times 100\% \tag{3.4}$$

where p_i is the timesteps taken by the agent and l_i is the timesteps taken in a ground truth path and it can be interpreted as modified version of SPL.

MP3D-EQA dataset

The MP3D-EQA[37] dataset extends the Embodied Question Answering (EQA) task into photorealistic environments using the Matterport3D dataset[26]. Unlike earlier EQA datasets that relied on synthetic environments, MP3D-EQA introduces real-world complexity by using high-resolution 3D reconstructions of homes. This dataset consists of 1,136 question-answer pairs grounded in 83 different environments, requiring an agent to navigate through an unfamiliar space and answer questions based on what it perceives. MP3D-EQA focuses

on three types of questions: location-based, which require the agent to determine the room in which an object is located; color-based, which involve recognizing the color of an object; and color-location questions, where the agent must identify an object's color within a specified room. These questions are generated programmatically from the semantic annotations available in Matterport3D[26]. Since MP3D lacks explicit color annotations, human workers were employed via Amazon Mechanical Turk to label object colors. The dataset is designed to minimize answer biases and ensure that an agent must actively explore the environment to derive an accurate response. qsectionTANGO for Open Embodied Question Answering TANGO addresses the Embodied Question Answering (EQA) task by leveraging a Large Language Model (LLM) as a planner. The LLM decomposes natural language queries into structured navigation subtasks, using its reasoning abilities and prior knowledge. This allows the agent to systematically explore its environment and gather information necessary to answer the query. To this end, TANGO integrates a specific **answer module**, relying on BLIP2 [31] as its core foundation. TANGO was evaluated using the MP3D-EQA dataset and OpenEQA , where predefined category-based answers were required. Since TANGO generates natural language responses, a mismatch can occur if the model provides synonymous answers (e.g., *"white"* instead of *"off-white"*). To address this, an *answer constraint* mechanism was introduced, grouping similar answers into the same category. This adjustment improved performance, demonstrating TANGO's ability to navigate towards the target even when it is not initially visible. An example of the Embodied Question Answering (EQA) process is illustrated in Fig3.13

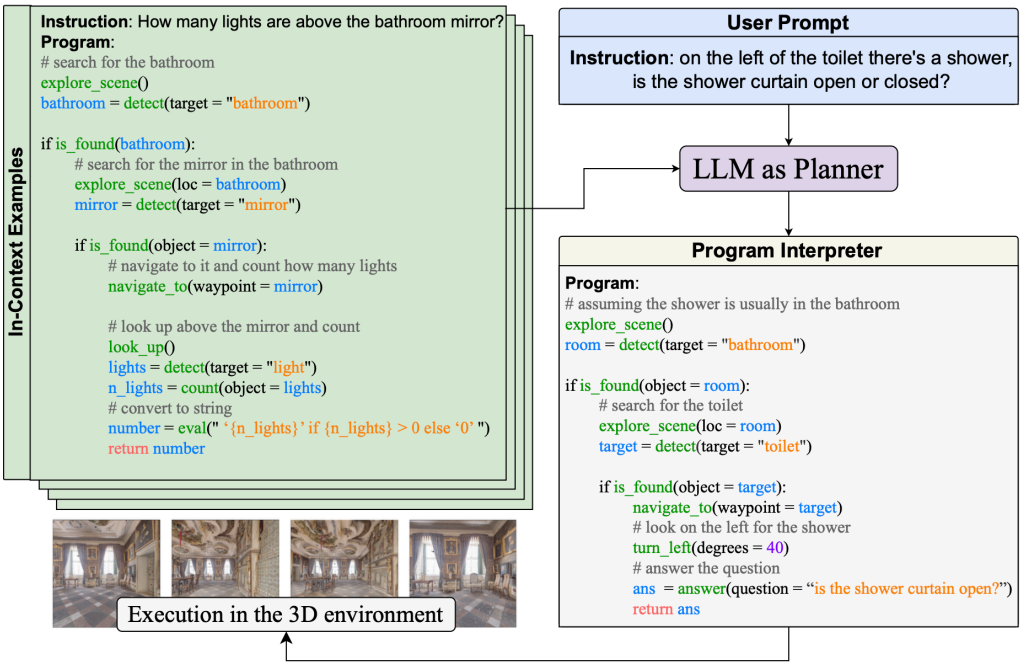

Figure 3.13: (EQA) example for the user's question: "on the left of the toilet there's a shower, is the shower curtain open or closed?". The LLM, which acts as a planner, transforms the user's initial prompt into navigation subtasks by leveraging its reasoning abilities and powerful priors. This decomposition enables the model to break down complex queries into more manageable steps, guiding the agent's navigation process effectively.The figure is taken form [8]

4

Object Detection Module in TANGO

4.1 Object Detection Module in TANGO

4.1.1 Introduction

Object detection plays a crucial role in the **TANGO framework**, enabling the system to recognize, localize, and categorize objects in its environment. This module is essential for tasks such as **scene understanding, navigation, and interaction with objects**. TANGO integrates multiple object detection models, each selected for specific strengths and applications. The supported models include **YOLO[11], DETR[12], Owlv [34],owl-vit2[34], allowing for adaptability based on the detection task at hand.**

A key improvement in this work is the **fine-tuning of YOLO and DETR**, optimizing them for specific tasks and datasets used in TANGO. These enhancements improve the **detection accuracy and reliability of the system**.

This chapter provides a **detailed step-by-step explanation of how object detection is implemented in TANGO**, covering the entire pipeline from model selection to final integration with other components.

Before introducing object detection within the TANGO framework, we briefly review the fundamentals of object detection and common evaluation metrics.

4.2 Model Evaluation

Evaluating model performance in computer vision tasks requires specific metrics to quantify accuracy. The most commonly used evaluation metric for object detection is *mean average precision* (mAP). This metric is calculated for both bounding boxes (mAP_{box}) and segmentation masks (mAP_{mask}) in the case of instance segmentation mod-

els. Leading object detection frameworks such as YOLO and DETR rely on mAP to assess model accuracy and performance in research publications.

The effectiveness of mAP lies in its suitability for detecting multiple objects within an image, ensuring that all objects of interest are considered. Additionally, it incorporates both precision and recall, balancing false positives (FP) and false negatives (FN) to provide a more comprehensive measure of detection accuracy. This section provides an in-depth explanation of mAP computation, detailing its application in evaluating the model used in this research.

4.2.1 Mean Average Precision

The **mean average precision (mAP)** is computed as the mean of individual class-wise average precision (AP) values, which are derived from recall and precision scores. These values range between 0 and 1. The methodology for computing mAP is outlined below, focusing on object detection (bounding boxes). The same principles apply to segmentation tasks.

- **Intersection over Union (IoU)**: The first step in evaluating prediction accuracy is computing the IoU score, which measures the overlap between the predicted and actual object bounding boxes. A prediction is considered correct (true positive) if its IoU with the ground truth exceeds a predefined threshold (e.g., 50%, 75%, 90%). This threshold determines the strictness of the mAP calculation. Predicted bounding boxes are sorted in descending order based on confidence scores, and those below a certain threshold are discarded from evaluation. Figure 4.1 illustrates the IoU computation.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{4.1}$$

- **Precision and Recall**: These two key metrics are computed for each object class as follows:

$$\begin{cases} \text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \\ \text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \end{cases} \tag{4.2}$$

  where:

  - True Positives (TP) refer to correctly detected objects (IoU above the threshold).
  - False Positives (FP) represent incorrect detections that do not match any ground truth.
  - False Negatives (FN) are ground truth objects that the model fails to detect.

- **Precision-Recall Curve**: Precision-recall curves are generated for different threshold values. The area under these curves corresponds to the average precision (AP) for each class. The calculation uses the trapezoidal rule:

$$\text{AP}^{IoU} = \int_0^1 p(r)dr \tag{4.3}$$

  where $p(r)$ represents the interpolated precision at recall r.

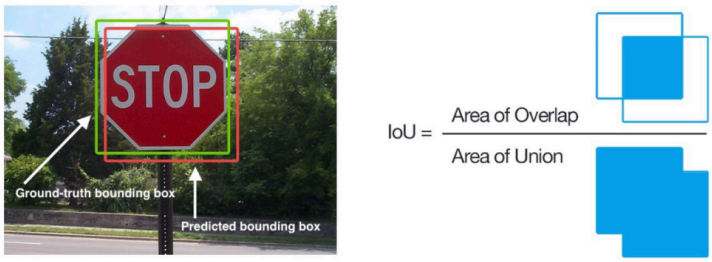

Figure 4.1: Illustration of the Intersection over Union (IoU) metric, which quantifies bounding box prediction accuracy.

- **Average Precision (AP)**: The mean of AP values is computed across all N classes to obtain a final score for each IoU threshold. The IoU thresholds range from 0.1 to 0.95, with increments of 0.05, resulting in 17 equally spaced intervals. The formula is given by:

$$\text{mAP}^{IoU} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i^{IoU}, \qquad \text{where} \quad IoU \in [0.1, 0.95] \tag{4.4}$$

- **Final mAP Computation**: The overall model performance is given by the mean of all mAP^{IoU} values.

To account for different levels of strictness in object detection, mAP is often computed at multiple IoU thresholds, leading to variations such as AP^{50} and AP^{75}. Specifically: - AP^{50}represents the AP value when the IoU threshold is at least 0.5. - AP^{75} is a stricter metric, requiring at least 75% overlap for a true positive detection.

Since evaluating mAP on large datasets at every step of training can be computationally expensive, it is typically calculated at specific checkpoints (e.g., every n training epochs) to track model progress.

The Intersection over Union (IoU) metric used for evaluating bounding box accuracy is illustrated in Figure 4.1

4.2.2 Object Detection Workflow in TANGO

The object detection module follows a structured workflow that ensures efficient detection and processing. The pipeline consists of several key stages:

- **Selecting and Loading the Model:** Choosing the appropriate object detection model and loading its parameters.
- **Preprocessing Input Images:** Preparing images to match the required format of the selected model.
- **Running Inference:** Performing object detection on the image to generate predictions.
- **Post-processing the Results:** Refining detections by applying filtering and non-maximum suppression.
- **Classification and False Positive Filtering:** Using a classifier to validate and refine detections.

Each of these steps is crucial for achieving accurate and efficient object detection.

4.2.3 Model Selection and Loading

The first step in object detection is **selecting the appropriate model** based on the requirements of the task. TANGO supports a variety of object detection architectures, each suited for different scenarios.

- **YOLO (You Only Look Once):** Known for its speed and efficiency, making it ideal for real-time detection tasks.
- **DETR :** Uses a transformer-based approach for robust object detection in structured environments.
- **OWL-ViT2:** Incorporates vision and language models, allowing for open-vocabulary object detection.

The **detect** module utilizes Owl-V2[34] for detecting general object categories, while DETR[12] is used specifically for objects belonging to the COCO[38] dataset classes.

For the MultiON[2] task, we use fine-tuned models such as YOLO and DETR to improve performance on the specific dataset used in TANGO.

4.2.4 Preprocessing the Input Image

After loading the model, the input image must be processed to ensure compatibility with the chosen model. The preprocessing steps vary depending on the detection architecture.

Some models, like **YOLO**, do not require extensive preprocessing. The image can be directly passed to the model. Others, such as **DETR, OWL-ViT**, require **image encoding and transformation** before inference.

Common preprocessing steps include:

- **Adjusting the color format** to match the model's requirements.
- **Resizing the image** to fit the input dimensions expected by the model.
- **Normalizing pixel values** to ensure consistent processing.
- **Encoding the image** into tensors or other representations suitable for deep learning models.

Once preprocessed, the image is passed to the detection model for inference.

4.2.5 Performing Object Detection (Inference Step)

With the image prepared, the next step is **running the object detection model** to generate predictions. During inference, the model analyzes the image and outputs:

- **Bounding boxes** that define the detected objects' locations.
- **Confidence scores** that indicate the certainty of each detection.
- **Class labels** that specify the object categories.

The detection process involves:

- **Extracting feature representations** from the image.
- **Processing the image through the model's detection layers.**
- **Computing object scores and bounding box coordinates.**

The model's output consists of raw predictions that require further **post-processing** to refine the detections.

4.2.6 Post-processing the Detection Results

Once raw detections are obtained, they undergo **post-processing** to improve accuracy and remove false positives.

Filtering Low-Confidence Detections: Any detection with a confidence score below a predefined **threshold** is discarded.

Non-Maximum Suppression (NMS): In cases where multiple detections overlap, **NMS is applied** to remove redundant bounding boxes. The goal is to keep only the **most relevant** detection for each object.

The NMS algorithm:

- **Sorts detections** by confidence score.
- **Compares overlapping boxes** and removes those with lower scores.
- **Retains the most confident detection** for each object.

By applying **NMS**, the system ensures that final detections are **precise and non-redundant**.

4.2.7 Classification and False Positive Filtering

To improve detection reliability, an additional **classification module** is used to filter false positives. This classifier, based on **CLIP**[10], verifies whether the detected object matches expected categories.

The classification module works by:

- Extracting features from detected objects.
- Comparing object embeddings against learned category representations.
- Filtering out detections classified as incorrect or uncertain.

The integration of this classification step significantly reduces false detections, improving the overall accuracy and robustness of the object detection pipeline.

4.3 CLIP-Based Classification in TANGO

4.3.1 Introduction

Object classification within the TANGO framework requires a robust mechanism to distinguish between true and false detections while maintaining flexibility for open-set classification. To address this, a verification step is introduced using an open-set classifier such as CLIP to assess whether the content of the detected bounding box semantically corresponds to the predicted category. This is done by applying a confidence score threshold to distinguish between the target class and a generic "other" category.

This section details the role of CLIP[10] in TANGO's object detection module, its architecture, the mechanism behind its classification process.

4.3.2 Overview of CLIP's Architecture

Modifications from ConVIRT in CLIP

While CLIP builds upon the contrastive learning approach established in ConVIRT[39], several key modifications were introduced to improve training efficiency and simplify implementation:

- **Training from Scratch:** Unlike ConVIRT[39], which utilizes pre-trained weights for initialization, CLIP is trained entirely from scratch, avoiding dependencies on ImageNet or pre-trained text encoders.
- **Linear Projection Instead of Non-Linear:** CLIP simplifies the projection process by using a linear mapping from encoder representations to the shared embedding space, removing the non-linear projection introduced by Bachman et al. (2019)[40].
- **Simplified Text Preprocessing:** The text transformation function from ConVIRT, which sampled a single sentence from longer text, was removed in CLIP since most image-text pairs in its dataset consist of single-sentence descriptions.
- **Minimal Image Transformations:** CLIP reduces data augmentation complexity by applying only a random square crop from resized images, in contrast to ConVIRT's more extensive image transformations.

A new dataset comprising 400 million image–text pairs was constructed to train a simplified variant of ConVIRT from scratch. This approach, named CLIP (Contrastive Language-Image Pre-training), demonstrates the effectiveness of learning visual representations directly from natural language supervision.

CLIP consists of two main components:

1. An Image Encoder: Processes visual input and extracts high-dimensional representations.
2. A Text Encoder: Converts textual descriptions into a comparable embedding space.

Both encoders are trained jointly to align their representations in a shared multi-modal space, allowing the model to determine the similarity between an image and a textual description.

Image Encoder

CLIP's image encoder is based on either:

- A ResNet[41] architecture with modifications for enhanced feature extraction.
- A Vision Transformer (ViT)[**?**], which captures long-range dependencies within an image, improving generalization across diverse object categories.

Regardless of the backbone, the image encoder outputs a feature vector, mapping the visual input into a shared representation space.

Text Encoder

The text encoder in CLIP is a Transformer[42]-based model. It processes textual prompts (e.g., "a photo of a car") and generates semantic embeddings that align with corresponding visual representations.

The key advantage of this approach is that it does not require manually defined categories; instead, it learns relationships between text and images in a flexible, open-ended manner.

Image-Text Alignment and Embedding Space

CLIP operates by aligning images and text in a common feature space. During training, it learns to associate an image with its correct textual description while pushing apart mismatched pairs. The result is a multi-modal embedding space, where visually similar objects are closer to their corresponding textual descriptions.

How CLIP Matches Images to Text:

1. The image and text encoders produce feature vectors for both modalities.
2. These feature vectors are normalized and projected into the same dimensional space.
3. The model computes the cosine similarity between the image and multiple textual embeddings.
4. The textual description with the highest similarity score is considered the best match.

This process allows CLIP to perform classification tasks without explicitly training on specific object categories, making it highly effective for zero-shot learning.

An overview of the CLIP training and inference pipeline is shown in Figure 4.2.

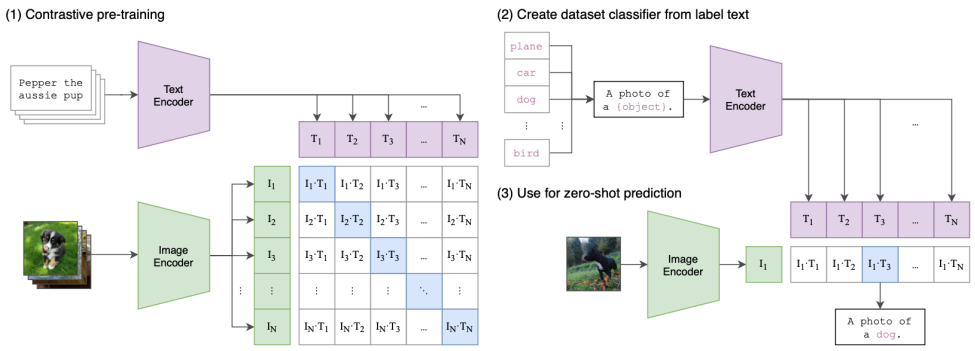

Figure 4.2: Unlike conventional image models that train an image encoder alongside a linear classifier to predict predefined labels, CLIP trains both an image encoder and a text encoder simultaneously to learn associations between images and their corresponding textual descriptions. During inference, the text encoder generates a zero-shot classifier by embedding the class names or descriptions from the target dataset.The figure is taken from [10]

Contrastive Learning for Visual Representation: The ConVIRT Framework

Introduction to ConVIRT

Contrastive Visual Representation Learning from Text (ConVIRT)[39] is a framework designed to enhance visual representation learning by leveraging the natural pairing of images and textual descriptions. It builds on the principle that text provides rich contextual information, which can serve as a powerful supervisory signal for training visual models. By maximizing the similarity between true image-text pairs and minimizing the similarity between incorrect pairs, ConVIRT enables more effective representation learning.

ConVIRT introduces a contrastive learning approach that allows models to align images with their corresponding text while ensuring that non-matching pairs remain distinct. Unlike traditional supervised learning

that requires explicit labels, this framework takes advantage of existing textual descriptions, making it highly scalable. Since its introduction, ConVIRT has influenced several large-scale vision-language models, including CLIP, which utilizes a simplified version of its contrastive learning methodology.

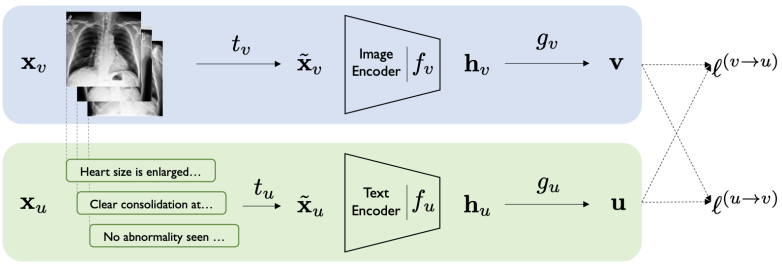

Figure 4.3: The blue and green shades represent the image and text encoding pipelines, respectively. Our method relies on maximizing the agreement between the true image-text representation pairs with bidirectional losses $\ell^{(v\to u)}$ and $\ell^{(u\to v)}$. The figure is taken from[39]

Task Definition

The core objective of ConVIRT is to learn visual representations by mapping images to a high-dimensional feature space where they maintain semantic relationships with textual descriptions. The learning process assumes paired input data, where each image is accompanied by a corresponding textual description. The goal is to develop an image encoder that transforms input images into fixed-dimensional vectors, capturing meaningful features that can be used for downstream applications such as classification and retrieval.

Each input sample consists of:

- x_v: An image or a set of images representing visual information.
- x_u: A text sequence describing the content of x_v.

The objective is to train an encoder function f_v, which maps an image x_v into a compact vector representation. This encoder is modeled as a convolutional neural network (CNN), allowing it to extract visual features efficiently. The learned image representations can then be transferred to downstream tasks, enabling flexible and efficient applications.

The use of paired image-text data is particularly advantageous, as such associations naturally occur in many real-world contexts. By learning from these image-text pairings, ConVIRT captures meaningful relationships between modalities, leading to improved generalization in vision-based tasks.

Contrastive Learning with Image-Text Pairs

The ConVIRT framework applies a contrastive learning paradigm, where images and their corresponding text descriptions are embedded into a shared vector space. The fundamental idea is to train the model such that representations of correct image-text pairs are close together, while representations of mismatched pairs are pushed apart. This learning strategy enables the model to develop a strong cross-modal understanding.

To achieve this, both images and text undergo a transformation process before being fed into their respective encoders:

- For images, a transformation function applies random augmentations such as cropping, flipping, and color jittering to introduce variability and prevent overfitting.
- For text, a span of words is randomly sampled from the full description to provide a more generalized representation.

The encoded representations are then projected into a common embedding space using learnable transformation functions. At this stage, the similarity between image and text embeddings is computed using cosine similarity, and a contrastive loss function is applied to optimize the model. The objective is to ensure that the model correctly associates images with their textual descriptions while discouraging false pairings.

Bidirectional Contrastive Learning

Unlike conventional contrastive learning methods that operate within a single modality, ConVIRT employs a bidirectional contrastive objective. This means the model is trained to:

- Align image representations with their corresponding text descriptions.
- Align text representations with their corresponding images.

This bidirectional learning strategy is crucial for improving the model's ability to retrieve relevant visual content given a text query, and vice versa. It also ensures that both modalities contribute equally to the representation space, leading to better generalization across different tasks.

The training process involves a contrastive loss function that enforces alignment between matched image-text pairs while penalizing incorrect pairings. This loss function is asymmetric for each modality, ensuring that the model learns meaningful associations across both image and text domains.

Realization of the ConVIRT Framework

ConVIRT is designed to be flexible in its implementation, allowing for various choices of encoders and transformation functions. The framework consists of the following key components:

- **Image Encoder:** A convolutional neural network (CNN), specifically a ResNet-50 [41]architecture, is used to encode images into feature representations.
- **Text Encoder:** A transformer-based model, is used to encode text into vector representations. The text encoder is initialized with pre-trained weights and fine-tuned for contrastive learning.
- **Projection Functions:** Separate neural networks with single-hidden-layer architectures are used to map image and text representations into a common embedding space.

To improve generalization, the text encoder is partially frozen during training, with only the last few transformer layers being updated. This strategy ensures that the model retains its pre-trained knowledge while adapting to the contrastive learning objective.

Additionally, data augmentation plays a crucial role in training. Image transformations include cropping, flipping, affine transformations, and color jittering, which introduce variability and improve robustness. For textual data, random sentence sampling is applied to enhance diversity while preserving semantic meaning.

By leveraging these techniques, ConVIRT creates a shared embedding space where image-text relationships are effectively learned and optimized for downstream tasks.

Optimization Strategy

The optimization process in ConVIRT is centered around a contrastive loss function that enhances alignment between image and text representations. The model learns to:

- Increase similarity scores between correctly matched image-text pairs.
- Decrease similarity scores between mismatched pairs.

To achieve this, a bidirectional loss function is employed, incorporating: where h_{v_i}, u_i represents the cosine similarity

$$h_v, u_i = \frac{v \cdot u}{\|v\| \|u\|}$$

- **Image-to-Text Contrastive Loss:** Ensures that an image representation aligns with its correct textual description.

$$\ell_i^{(u \to v)} = -\log \frac{\exp\left(\langle \mathbf{u}_i, \mathbf{v}_i \rangle / \tau\right)}{\sum_{k=1}^{N} \exp\left(\langle \mathbf{u}_i, \mathbf{v}_k \rangle / \tau\right)} \tag{4.5}$$

- **Text-to-Image Contrastive Loss:** Ensures that a text representation aligns with its correct image.

$$\ell_i^{(v \to u)} = -\log \frac{\exp\left(\langle \mathbf{v}_i, \mathbf{u}_i \rangle / \tau\right)}{\sum_{k=1}^{N} \exp\left(\langle \mathbf{v}_i, \mathbf{u}_k \rangle / \tau\right)} \tag{4.6}$$

The final optimization function is computed as a weighted combination of these loss functions, ensuring balanced learning across modalities. This approach results in a robust model that excels in multi-modal retrieval and zero-shot learning tasks.

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(\lambda \ell_i^{(v \to u)} + (1 - \lambda) \ell_i^{(u \to v)} \right) \tag{4.7}$$

To better understand the design and capabilities of OWL-ViT2 and DETR, two key models used in TANGO's detection module, it is essential to first introduce the concept of Vision Transformers (ViT), which form the architectural backbone of both models. Therefore, this chapter also includes a dedicated section on Vision Transformers, providing the necessary background before diving into adaptation of OWL-ViT2 and DETR within TANGO.

4.4 Vision Transformer

The Vision Transformer (ViT) [43] is a relatively recent innovation in computer vision that presents a compelling alternative to traditional Convolutional Neural Networks (CNNs). Unlike CNNs, which rely on hierarchical feature extraction through convolutional operations, ViT is built upon the Transformer model—an architecture that has revolutionized natural language processing (NLP) and set new benchmarks in various domains.

One of the key advantages of ViT lies in its ability to process images efficiently through self-attention mechanisms rather than convolutional layers. This approach allows for a more global understanding of an image, capturing long-range dependencies between different regions. While ViT has demonstrated superior computational efficiency compared to CNNs, it comes with a significant drawback: it demands large-scale datasets for effective training and generalization. Unlike CNNs, which benefit from strong inductive biases that help them learn efficiently from smaller datasets, ViTs require extensive pretraining on massive datasets to achieve competitive performance.

4.4.1 Model Architecture

The **Vision Transformer (ViT)** follows the standard Transformer architecture as closely as possible while adapting it for **computer vision tasks**. The key difference from traditional convolutional neural networks (CNNs) is that ViT processes images **globally**, rather than through localized feature extraction. The architecture consists of four primary stages:

- **Converting 2D Images into 1D Patch Embeddings**
  Instead of feeding an entire image into the model as a whole, ViT **splits the image into smaller, fixed-size patches**. These patches are then **flattened into 1D vectors** before being passed to the Transformer. The size and number of patches depend on the image resolution and the chosen patch size, both of which act as hyperparameters. This conversion step allows the Transformer to process image data in a structured format, much like words in an NLP task.
- **Adding a Classification Token & Positional Embeddings**
  Since Transformers **do not inherently understand spatial relationships**, additional information must be provided. To do this:
  - A **classification token** is prepended to the sequence of patches, acting as a global representation of the entire image.
  - **Positional embeddings** are incorporated to retain spatial information, ensuring that the model knows the relative positioning of each patch.
- **Processing Through the Transformer Encoder**
  The sequence of patch embeddings, along with the classification token, is passed through multiple layers of the **Transformer Encoder**. Each layer applies:
  - **Multi-Head Self-Attention (MSA)**, allowing patches to exchange information across the entire image.
  - **Feedforward Neural Networks (MLPs)** to refine features.

– **Layer Normalization** and **Residual Connections**, which stabilize training and improve performance.

- **Final Prediction Layers**
  The **classification token** output from the last Transformer layer is used to generate predictions. For **classification tasks**, a **simple MLP head** is added on top. In contrast, for **object detection and segmentation**, more complex structures (e.g., detection heads like those in DETR) are integrated into the framework.

An overview of the Vision Transformer architecture is shown in Figure 4.4.

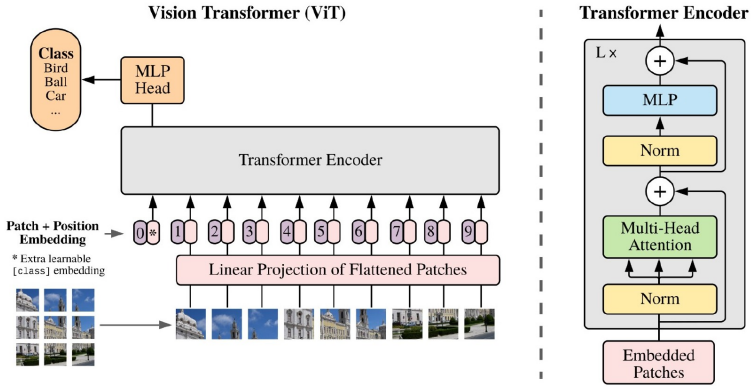

Figure 4.4: Vision Transformer model overview The illustrative picture is taken from [43]

Mathematical Representation of ViT Processing

To mathematically formalize the process, let's define an input image x with resolution $H \times W$ and C color channels (e.g., RGB images have $C = 3$):

- **Patch Extraction**
  The image is divided into N non-overlapping square patches of size $P \times P$, leading to:

$$N = \frac{HW}{P^2} \tag{4.8}$$

  Each patch is flattened and projected into a higher-dimensional embedding space using a learnable matrix E, resulting in:

$$z_{LP} = [x_p^1 E, x_p^2 E, \ldots, x_p^N E], \quad E \in \mathbb{R}^{(P^2 \cdot C) \times d_{model}} \tag{4.9}$$

  The classification token is a learnable random vector of size d_{model}

- **Adding the Classification Token & Positional Embeddings**
  A special learnable **classification token** x_{cls} is added to the beginning of the sequence:

$$z_{LP}^{'} = [x_{cls}, x_p^1 E, x_p^2 E, \ldots, x_p^N E] \tag{4.10}$$

Positional embeddings E_{pos} are then incorporated:

$$z_0 = z^{'}_{LP} + E_{pos}, \quad E_{pos} \in \mathbb{R}^{(N+1)\times d_{model}} \tag{4.11}$$

These embeddings provide the Transformer with spatial awareness, compensating for its lack of an inherent sense of locality.

- **Transformer Encoding**
  and the resulting z_0 is passed on to the Transformer Encoder. At this point, each encoder block computes the output z_l as follows::

$$\begin{cases} z'_l = \text{MSA}(\text{LN}(z_{l-1})) + z_{l-1} \\ z_l = \text{MLP}(\text{LN}(z'_l)) + z'_l \end{cases} \tag{4.12}$$

  where:

  - **MSA** refers to **Multi-Head Self-Attention**.
  - **MLP** is a feedforward neural network.
  - **LN** stands for **Layer Normalization**.

- **Final Prediction Stage**
  Once the final Transformer layer is processed, the **classification token output** is passed to an **MLP head** for the final classification:

$$\text{logits} = \text{MLP}_{\text{head}}(z^0_L) \tag{4.13}$$

  For object detection tasks, a specialized detection head replaces the classification head.

4.4.2 Self-Attention Mechanism

The self-attention mechanism is a fundamental aspect of transformer-based models. It enables each element in an input sequence to interact with every other element, allowing the model to determine which parts are most relevant for understanding the given data. This mechanism constructs three vectors—query, key, and value—for every input element, which are then used to compute attention scores. These scores help determine the significance of each element in relation to the others. The final output is obtained by computing a weighted sum of the value vectors, where the weights are derived from the relationship between the query and key vectors.

In the field of natural language processing, self-attention plays a crucial role in identifying important words and their relationships within a sentence. For example, in translation tasks, the model must understand how words relate to one another to generate an accurate translation. Self-attention allows the model to focus on relevant words while disregarding less important ones, facilitating high-quality translations.

In the Vision Transformer (ViT) model, self-attention operates using Scaled Dot-Product Attention, which enhances computational efficiency and stability. Mathematically, given an input sequence of n vectors:

$$x = [x^1, x^2, ..., x^n], \quad x \in \mathbb{R}^{n\times d_{model}} \tag{4.14}$$

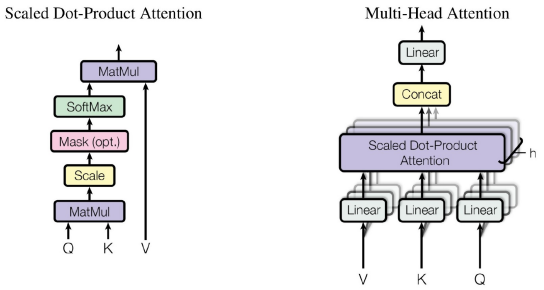

Figure 4.5: On the left, Scaled Dot-Product Attention. On the right, Multi-Head Attention consists of several attention layers running in parallel. The illustrative picture is taken from [44]

three corresponding matrices—Query (Q), Key (K), and Value (V)—are created by multiplying the input with learned parameter matrices:

$$\begin{cases} Q = x \cdot W^Q, & W^Q \in \mathbb{R}^{d_{model} \times d_k}, Q \in \mathbb{R}^{N \times d_k} \\ K = x \cdot W^K, & W^K \in \mathbb{R}^{d_{model} \times d_k}, K \in \mathbb{R}^{N \times d_k} \\ V = x \cdot W^V, & W^V \in \mathbb{R}^{d_{model} \times d_v}, V \in \mathbb{R}^{N \times d_k} \end{cases} \tag{4.15}$$

The Q and K matrices reside in the same space ($\mathbb{R}^{N \times d_k}$), while the V matrix may exist in a different space ($\mathbb{R}^{N \times d_v}$). The next step is computing the attention scores, represented as matrix S, which captures the relevance of each element relative to a given input:

$$S = Q \cdot K^T, \quad S \in \mathbb{R}^{n \times n} \tag{4.16}$$

To ensure stable gradients, the score matrix is scaled by $\sqrt{d_k}$, followed by applying a softmax function to normalize the attention weights. These weights are then multiplied by the value matrix V to produce the final output sequence:

$$z = Attention(Q, K, V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V, \quad z \in \mathbb{R}^{n \times d_k} \tag{4.17}$$

4.4.3 Multi-Head Self-Attention

The Multi-Head Self-Attention (MSA) mechanism is an extension of standard self-attention that enhances the model's ability to process diverse features across multiple perspectives. Instead of relying on a single attention function, MSA leverages multiple attention layers, referred to as attention heads. Each head processes the input independently, capturing distinct relationships between elements in the sequence. This design enables the model to represent information more effectively and learn richer dependencies across different regions of the input.

Each attention head utilizes a separate set of weight matrices, denoted as W_Q, W_K, and W_V, which correspond to queries, keys, and values. These weights are initialized randomly and are refined during training. The outputs of all attention heads are concatenated and then transformed using a learned weight matrix, U_{MSA}, producing the

final output representation. This multi-headed approach allows the model to understand multiple aspects of the input data simultaneously, contributing to the overall success of Transformer-based architectures.

$$z = [h^1; h^2; ....; h^b] \cdot U_{MSA}, \quad U_{MSA} \in \mathbb{R}^{b \cdot d_v \times d_{model}} \tag{4.18}$$

4.4.4 Positional Embeddings

Unlike convolutional neural networks (CNNs), which inherently process spatial information, Vision Transformers (ViTs) treat input images as sequences of patches. However, since Transformers lack an intrinsic understanding of spatial relationships, positional embeddings are incorporated to retain spatial structure. These embeddings encode positional information into the model, helping it recognize how image patches relate to each other. An illustration of positional embeddings in ViT is shown in Figure 4.6, highlighting the spatial relationships encoded across image patches.

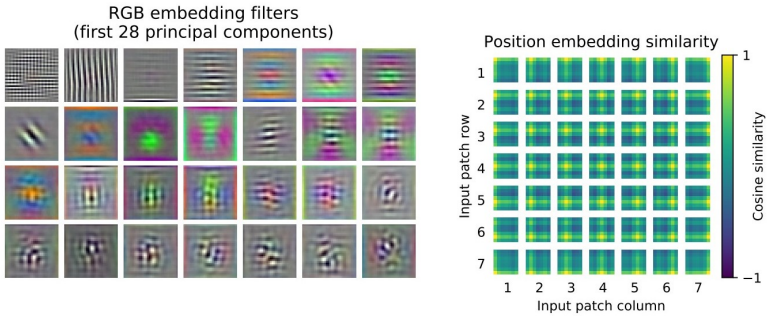

Figure 4.6: Left: Filters of the initial linear embedding of RGB values of ViT-L/32. Right: Similarity of position embeddings of ViT-L/32. Tiles show the cosine similarity between the position embedding of the patch with the indicated row and column and the position embeddings of all other patches(The image is taken from [43])

Positional embeddings can be introduced in various ways. One common method involves using sinusoidal positional embeddings, which are predefined and added to the input data. Alternatively, ViTs often employ learnable positional embeddings, where the model assigns a unique trainable vector of dimension d_{model} to each patch. These embeddings are learned alongside the model parameters and play a crucial role in maintaining spatial coherence in the representation.

4.5 OWL-VIT

4.5.1 Introduction

Conventional object detectors typically rely on closed-vocabulary classification, where models are trained on a fixed set of object categories and struggle to generalize to novel classes without further fine-tuning. Open-Vocabulary Object Detection (OVOD) addresses this limitation by enabling detection of previously unseen objects through

textual descriptions. OWL-ViT (Open-World Learning Vision Transformer) [45] is a state-of-the-art OVOD method that combines contrastive vision-language pre-training with the scalability of Vision Transformers (ViTs). It follows a two-stage pipeline: (1) pre-training an image-text encoder on large-scale image-text pairs using contrastive learning, and (2) fine-tuning lightweight detection heads on standard object detection datasets. OWL-ViT retains separate encoders for images and text, enabling efficient and flexible query processing for open-vocabulary detection.

4.5.2 Model Architecture

OWL-ViT builds on the Vision Transformer (ViT)[43] framework, originally designed for classification tasks, and extends it for open-vocabulary object detection. The key architectural modifications include:Contrastive Pre-Training The image and text encoders are trained contrastively on a large dataset of paired images and text descriptions. This process ensures that the representations of semantically similar image-text pairs are aligned in a shared embedding space, similar to models like CLIP[10], and LiT[46]. This enables the model to learn generalized visual representations without the need for exhaustive human annotations.Transformer-Based Image and Text Encoders.The image encoder is a Vision Transformer (ViT) that processes an input image into a set of feature tokens. The text encoder is a Transformer model that encodes class names or textual descriptions into vector representations.Unlike classical object detectors that use a fixed classification layer, OWL-ViT replaces learned class embeddings with text embeddings obtained from the text encoder. This substitution enables the model to classify objects based on textual queries at inference time. Adaptation for Object Detection The final token pooling layer is removed from the ViT backbone to retain per-token feature representations. A lightweight classification and localization head is attached to each transformer output token. The number of predicted objects is equal to the sequence length of the image encoder (e.g., 576 tokens for ViT-B/32 at 768×768 resolution), which is sufficient given the object instance counts in modern datasets.

Bounding Box Prediction

The model predicts object bounding boxes using a small MLP head attached to the per-token embeddings. it resembles DETR, but is simplified by removing the decoder.

Open-Vocabulary Object Detection (OVOD)

The defining feature of OWL-ViT is its ability to perform open-vocabulary detection without requiring explicit retraining on new object categories. This is achieved by replacing traditional object classification layers with text embeddings, allowing detection based on flexible text-based queries.

Text Embeddings as Queries The model classifies detected objects by comparing their embeddings to text-derived embeddings, which are obtained by encoding category names or descriptive text phrases (e.g., “a red cylinder ”.

Query-Specific Label Space Unlike fixed-label object detectors, OWL-ViT dynamically defines the label space per image based on user-provided queries. This allows the model to:

Detect objects described by arbitrary textual inputs. Support zero-shot detection, where the model generalizes to unseen categories.

OWL-ViT processes each text query separately, rather than merging all queries into a single input sequence. Each query is independently passed through the text encoder, allowing the model to handle a dynamic and flexible label space per image. This approach improves efficiency, as the image encoding does not need to be recomputed for each new query. allowing us to use thousands of queries per image.

Figure 4.7 illustrates the OWL-ViT pipeline, highlighting the contrastive pre-training and adaptation stages for open-vocabulary object detection.

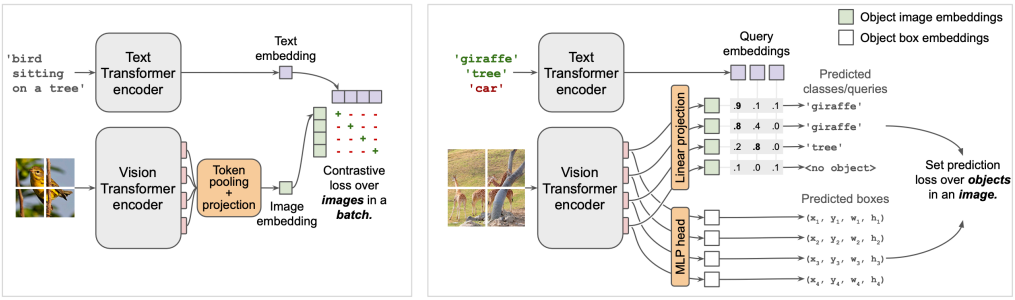

Figure 4.7: It consists of two main stages. First, we pre-train both an image and text encoder using a contrastive learning strategy with image-text pairs, following a process similar to prior models. Next, we adapt these pre-trained encoders for open-vocabulary object detection by eliminating token pooling and integrating lightweight object classification and localization heads directly into the image encoder's output tokens. Open-vocabulary detection is enabled by embedding query strings through the text encoder, which are then used for classification. The model undergoes fine-tuning on conventional object detection datasets. During inference, it can utilize text-based embeddings for open-vocabulary detection or image-based embeddings for few-shot, image-conditioned detection. The image is taken from [45]

4.5.3 Training Strategy

The training pipeline of OWL-ViT consists of two stages:

- The image and text encoders are trained contrastively using a large-scale image-text dataset. A contrastive loss function ensures that image and text embeddings for matching pairs are drawn closer in representation space, while non-matching pairs are pushed apart.

  The text representation is derived from the end-of-sequence (EOS) token of the text encoder, while the image representation uses multi-head attention pooling (MAP).

- Fine-Tuning for Open-Vocabulary Detection:Fine-tuning pre-trained models for classification is a well-researched area, particularly for large Transformer-based architectures, which require precise regularization and data augmentation techniques to perform effectively. Standard training strategies for classifiers are well-documented in the literature. In this work, we adopt a similar fine-tuning methodology for open-vocabulary object detection.

  The overall training process closely resembles that of closed-vocabulary detectors, with the key difference being that object category names are provided as text-based queries for each image. As a result, instead of using a fixed global label space, the classification head produces logits based on a dynamically defined label space that is specific to each image, determined by the user-specified queries.it uses the bipartite

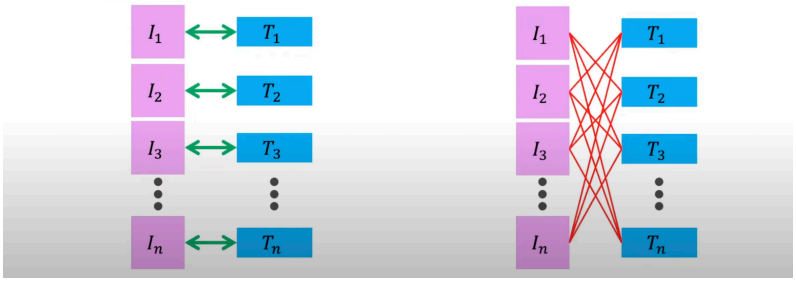

Figure 4.8: Left: Increasing the cosine similarity of correct pairs. Right: Reducing the cosine similarity of the $n^2 - n$ incorrect pairings.

matching loss from DETR[12] but modifies it to suit the requirements open-vocabulary detection. Since exhaustively annotating large detection datasets is impractical, federated annotation[47] is used. In this approach, datasets are constructed as a union of smaller datasets, where each subset is fully annotated for specific object categories. This results in a structured positive and negative set for each category, ensuring that evaluation is focused only on images where category presence is well-defined. Federated datasets help manage the long-tailed distribution of object categories while reducing annotation workload by avoiding unnecessary labeling of frequent categories. Given this dataset structure, focal sigmoid cross-entropy is used instead of softmax cross-entropy to handle class imbalance effectively. Furthermore, positive and negative annotations are used as queries during training, and additional pseudo-negative categories are sampled to ensure at least 50 negative queries per image.

4.6 DETER

DETR[12] presents a paradigm shift by treating object detection as a direct set prediction problem. Instead of relying on heuristics such as anchor boxes, DETR employs a transformer-based architecture that predicts all objects in a single pass. This end-to-end approach eliminates the need for region proposals and post-processing techniques, making the model both conceptually simpler and more generalizable.

4.6.1 DETR Architecture

The DETR model is composed of three main components:

1. **CNN Backbone:** Extracts feature maps from the input image.
2. **Transformer Encoder-Decoder:** Processes and refines features using self-attention mechanisms.
3. **Prediction feed-forward networks (FFNs):** Produces the final object classes and bounding boxes.

CNN Backbone

The CNN backbone processes the input image $x_{\text{img}} \in \mathbb{R}^{3 \times H_0 \times W_0}$, where H_0 and W_0 are the original height and width of the image. The backbone extracts feature representations and produces a lower-resolution feature map:

$$f \in \mathbb{R}^{C \times H \times W}, \tag{4.19}$$

where C is (typically 2048) and

$$H = \frac{H_0}{32}, \quad W = \frac{W_0}{32}. \tag{4.20}$$

This feature map is then passed into the transformer encoder.

4.6.2 Transformer Encoder

Before passing the feature map into the transformer encoder, a 1×1 convolution is applied to reduce the number of channels from C to a smaller dimension d. This results in a new feature map:

$$z_0 \in \mathbb{R}^{d \times H \times W}. \tag{4.21}$$

Since transformers operate on sequences rather than spatial feature maps, the spatial dimensions $H \times W$ are flattened into a single sequence of length HW, forming an input tensor of shape $d \times HW$.

Each encoder layer consists of:

- **Multi-head self-attention:** Enables each feature vector to attend to all others, capturing global dependencies.
- **Feed-forward network (FFN):** Applies transformations to enhance feature representation.
- **Positional Encoding:** Since transformers do not have an inherent notion of spatial relationships, fixed positional encodings are added to retain spatial information.

4.6.3 Transformer Decoder

The transformer decoder in DETR follows the standard architecture of transformers and is responsible for converting the encoded image features into object predictions. Unlike conventional transformer decoders, which generate sequences in an autoregressive manner (one token at a time), DETR's decoder operates in parallel, predicting all objects simultaneously.

Parallel Decoding of Object Queries

The input to the decoder consists of:

- The feature representations output by the transformer encoder.

- A set of N learned embeddings, referred to as **object queries**.

Each object query is a fixed-length vector that represents a potential object in the image. These object queries are not associated with specific regions in the image beforehand; instead, the model learns to associate them with objects during training. Since the decoder operates on all N object queries simultaneously, it can leverage global relationships between all predicted objects.

Self-Attention and Encoder-Decoder Attention

Each decoder layer consists of:

- **Multi-head self-attention:** This mechanism allows each object query to attend to all others, helping the model refine predictions by modeling interactions between objects. For instance, if one query attends to an object, another query attending to the same object can be suppressed, reducing duplicate detections.
- **Encoder-decoder attention:** This mechanism enables each object query to attend to relevant image features from the encoder output. Through this attention process, object queries extract information about object locations and categories.

The self-attention mechanism is particularly useful in crowded scenes, where multiple objects may be spatially close to each other. The ability to reason globally helps in assigning each query to a distinct object while avoiding redundant predictions.

Final Prediction

The output of the transformer decoder is a set of N learned embeddings, each of which corresponds to a potential object. These embeddings are then passed through a small prediction module consisting of:

- A feed-forward network (FFN) that predicts the class label using a softmax function.
- A separate linear layer that predicts bounding box coordinates in a normalized format.

Since DETR predicts a fixed number of objects, some object queries will not correspond to actual objects in the image. To handle this, an additional "no object" class is introduced. The model learns to assign unused queries to this category, allowing it to effectively filter out empty detections.

4.6.4 Prediction Feed-Forward Networks (FFNs)

Each object query is processed through a small FFN to generate the final object detection predictions. This module consists of:

- A three-layer perceptron with ReLU activations.
- A linear layer that predicts class probabilities using a softmax function.
- A separate linear layer that predicts bounding box coordinates.

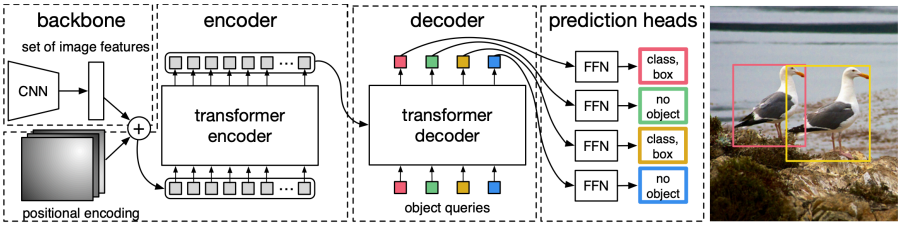

Figure 4.9: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "no object" class prediction. Image is taken from [12]

Bounding boxes are predicted in a normalized format relative to the input image dimensions. Since DETR predicts a fixed number of N bounding boxes, an additional label $\emptyset$ is introduced to represent "no object" predictions.

Figure 4.9 presents an overview of DETR's end-to-end detection pipeline, which combines a CNN and Transformer with bipartite matching during training.

4.6.5 Bipartite Matching with Hungarian Loss

Since DETR predicts a fixed number of objects, a matching strategy is required to associate predictions with ground truth objects. This is done using the Hungarian algorithm, which finds the optimal assignment $\sigma \in S_N$ that minimizes:

$$\hat{\sigma} = \arg\min_{\sigma \in S_N} \sum_{i=1}^{N} L_{\text{match}}(y_i, \hat{y}_{\sigma(i)}), \tag{4.22}$$

where:

- y_i is a ground truth object.
- $\hat{y}_{\sigma(i)}$ is the predicted object assigned to y_i.
- L_{match} is the matching cost.

The matching cost considers classification probability and bounding box similarity.
Once the optimal assignment is found, DETR computes a Hungarian loss:

$$L_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \Big[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + 1\{c_i \neq \emptyset\} L_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}(i)}) \Big]. \tag{4.23}$$

The second component of the loss function in DETR is the bounding box loss, which ensures that the predicted bounding boxes are as close as possible to the ground truth boxes. Unlike traditional object detection methods that use anchor boxes and offsets, DETR directly predicts the normalized coordinates of the bounding boxes

relative to the input image. To handle variations in object sizes and positions, the bounding box loss is formulated as a combination of two terms:

$$L_{\text{box}}(b_i, \hat{b}_{\sigma(i)}) = \lambda_{\text{iou}} L_{\text{iou}}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{L_1} ||b_i - \hat{b}_{\sigma(i)}||_1. \tag{4.24}$$

where:

- b_i represents the ground truth bounding box for object i.
- $\hat{b}_{\sigma(i)}$ is the predicted bounding box assigned to b_i based on the Hungarian matching.
- $L_{\text{iou}}(b_i, \hat{b}_{\sigma(i)})$ is the generalized Intersection over Union (IoU) loss, which measures how well the predicted and ground truth boxes overlap.
- $||b_i - \hat{b}_{\sigma(i)}||_1$ is the L_1 loss, which ensures precise regression of bounding box coordinates.
- λ_{iou} and λ_{L_1} are hyperparameters that control the relative importance of the two loss terms.

4.6.6 Fine-Tuning DETR for Cylinder Detection

Object detection is a fundamental task in computer vision, and transformer-based models such as the DEtection TRansformer (DETR) have demonstrated significant improvements over conventional convolutional architectures. DETR, originally designed for detecting objects in the COCO dataset, requires adaptation when applied to specialized domains. In this study, we fine-tuned DETR to detect cylinder-like objects by modifying its classification layer and optimizing it for our custom dataset.

Adapting DETR to a Custom Dataset

DETR is pre-trained on the COCO [38] dataset, which consists of 91 object classes. Our dataset, however, contains only 8 distinct cylinder categories. This discrepancy required modifying the classification head, which originally had 92 output neurons (91 COCO classes plus one background class). We replaced this layer with a new classifier 8 specifically designed to predict our object classes along with an additional background class, making a total of 10 output neurons.

Since the pre-trained model was optimized for COCO, directly using its classification head would introduce class mismatches and degrade performance. The new classification layer was initialized randomly, ensuring that it was learned from scratch while preserving all other pre-trained weights of the model. This approach allowed us to benefit from DETR's strong feature extraction and object detection capabilities while tailoring it to our specific application.

Training Strategy

To effectively train the modified DETR model, we used a structured fine-tuning approach. The learning rate was carefully adjusted to optimize both the newly introduced classification layer and the pre-trained backbone. A

lower learning rate was used for the backbone to prevent excessive modifications to the learned feature representations, while a slightly higher learning rate was applied to the newly initialized classification layer. This ensured stable convergence and prevented catastrophic forgetting of previously learned features.

The training process involved supervised learning with a combination of classification, bounding box regression, and generalized Intersection over Union (IoU) losses. These loss functions allowed the model to accurately classify objects and localize them within an image. Furthermore, data augmentation techniques were applied to improve generalization, including geometric transformations such as rotations, translations, and scale variations.

Evaluation and Performance Assessment

The fine-tuned DETR model was evaluated using standard object detection metrics, including Average Precision (AP) and Average Recall (AR). These metrics provided insights into the model's detection accuracy across different object sizes and confidence thresholds. Comparisons with the original COCO[38]-trained DETR model were conducted to assess the benefits of fine-tuning on a domain-specific dataset.

Through this process, DETR was successfully adapted to recognize cylinder objects with high accuracy. The modifications to the classification head, combined with an optimized training procedure, enabled effective fine-tuning while preserving the advantages of the transformer-based detection framework.

4.7 YOLO

Object detection is one of the core tasks in computer vision, involving the identification and localization of objects within an image. Traditional object detection methods such as R-CNN and Faster R-CNN [48]rely on multiple processing stages to identify and classify objects, making them computationally expensive.

YOLO (You Only Look Once) revolutionized object detection by treating it as a single regression problem. Unlike previous methods, YOLO applies a single neural network to the full image and directly predicts bounding boxes and class probabilities. This chapter provides a comprehensive explanation of YOLO's working mechanism, including its architecture, training methodology, loss function, advantages, limitations, and a comparison with other object detection models.

4.7.1 YOLO Architecture

Grid-Based Approach The input image is divided into an $S \times S$ grid. Each grid cell is responsible for detecting objects whose center falls within that cell. Each grid cell predicts:

- B bounding boxes, each defined by:

$$(x, y, w, h, c) \tag{4.25}$$

  where:

  - x, y - Center coordinates of the bounding box (relative to the grid cell).
  - w, h - Width and height of the bounding box (relative to the entire image).

– c - Confidence score, defined as:

$$c = P(\text{Object}) \times IOU_{\text{pred, truth}} \tag{4.26}$$

- C class probabilities for each object category.

Thus, the final output tensor size is:

$$S \times S \times (B \times 5 + C) \tag{4.27}$$

Network Design

The YOLO network consists of two main parts:

- **Feature Extractor:** The first part is a deep convolutional neural network (CNN) that extracts features from the input image.
- **Detector:** The second part consists of fully connected layers that predict bounding boxes and class probabilities.

Convolutional Layers YOLO uses 24 convolutional layers inspired by the GoogLeNet architecture. Instead of inception modules , it uses 1×1 convolutions followed by 3×3 convolutions for dimensionality reduction. Fully Connected Layers After feature extraction, YOLO uses two fully connected layers to predict:

- Bounding box coordinates (x, y, w, h)
- Confidence scores (c)
- Class probabilities $(P(\text{Class}|\text{Object}))$

Pretraining on ImageNet

To build an efficient object detection system, YOLO's convolutional layers are initially **pretrained** on the **ImageNet [49]**. The pretraining phase focuses on feature extraction, where the first **20 convolutional layers** are trained using an average-pooling layer and a fully connected layer[11].

The YOLO network architecture, consisting of convolutional and fully connected layers optimized for real-time detection, is illustrated in Figure 4.10.

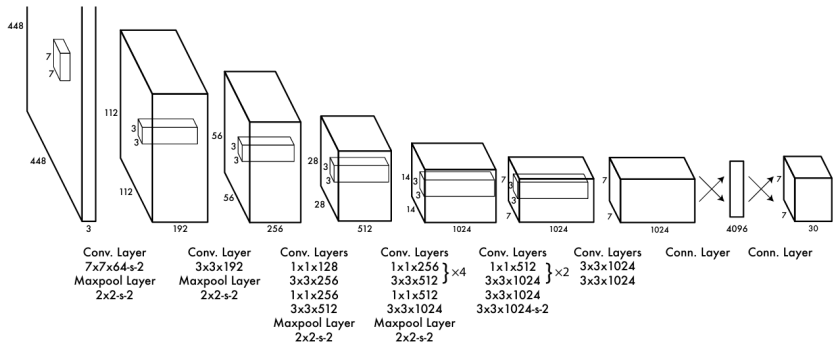

Figure 4.10: network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 * 1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224*224 input image) and then double the resolution for detection.The illustrative picture is taken from [11]

Conversion to Object Detection

After pretraining, the model is converted for object detection by adding:

- **Four additional convolutional layers** for feature refinement.
- **Two fully connected layers** to predict bounding boxes and class probabilities.

This architecture modification enhances fine-grained detection accuracy. Additionally, the input image resolution is increased from 224×224 **to** 448×448 to capture finer details.

Bounding Box Prediction Mechanism

The final layer predicts:

1. **Class probabilities** for each detected object.
2. **Bounding box coordinates** normalized as:

    - The final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell loca tion so they are also bounded between 0 and 1.

A **linear activation function** is used for the final layer, while other layers employ **leaky rectified linear activation**, defined as:

$$\varphi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \tag{4.28}$$

This prevents neuron inactivity and improves gradient flow during training.

Loss Function and Optimization Strategy

YOLO employs a **sum-squared error loss function** to optimize the network. While this approach is computationally efficient, it does not perfectly align with the goal of **maximizing average precision**. One limitation of this loss function is that it **equally weights localization, classification, and confidence errors**, which may not be ideal for object detection.

Additionally, in every image, most grid cells **do not contain any object**. This imbalance leads to strong gradients from background cells that push the confidence scores toward zero, often overpowering the gradients from cells containing objects. This can lead to model instability and cause training to diverge.

To address these issues, YOLO introduces **two key modifications**:

- Increasing the loss from bounding box coordinate predictions by a factor of $\lambda_{\text{coord}} = 5$.
- Decreasing the loss from confidence predictions for boxes that do not contain objects by using $\lambda_{\text{noobj}} = 0.5$.

The complete loss function consists of five main components, which balance localization accuracy, classification accuracy, and confidence scores.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{\text{ij}}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \tag{4.29}$$

$$+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{\text{ij}}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \tag{4.30}$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{\text{ij}}^{obj} (C_i - \hat{C}_i)^2 \tag{4.31}$$

$$+\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{\text{ij}}^{noobj} (C_i - \hat{C}_i)^2 \tag{4.32}$$

$$+ \sum_{i=0}^{S^2} 1_{\text{i}}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \tag{4.33}$$

where:

- 1_{obj}^{i} denotes if an object appears in cell i, and 1_{obj}^{ij} denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.
- $S \times S$ is the number of grid cells in the image.
- B is the number of bounding boxes per grid cell.
- (x_i, y_i) are the center coordinates of the predicted bounding box.
- (w_i, h_i) are the width and height of the predicted bounding box.
- C_i represents the confidence score predicted for the bounding box.

- $p_i(c)$ is the class probability for object class c.

4.7.2 Fine-Tuning YOLO Using Ultralytics

Fine-tuning YOLO using the Ultralytics framework involves loading a pre-trained model and training it on our dataset. Different model sizes (e.g., YOLOv8x, YOLOv8s, YOLOv8m, YOLOv8l) can be used depending on the desired trade-off between speed and accuracy. YOLO models predict object classes in the final classification layer. When fine-tuning on a custom dataset with a different number of classes, this layer must be modified. The existing classification head is replaced with a new layer that has outputs matching the number of custom classes. This new layer is randomly initialized and learns to recognize the new object categories during training while the rest of the model retains pretrained weights.

Process:

- The pretrained model's classification head (originally trained on standard datasets) is removed.
- A new classification layer with outputs equal to the custom dataset's classes is added.
- This layer starts with random weights and is trained from scratch.
- The backbone and detection layers retain their pretrained weights, ensuring faster convergence.

5 Dataset

5.1 Dataset Creation

5.1.1 Overview of the Dataset Creation Process

In order to train robust object detection models for an embodied AI task, we generated a synthetic dataset of images within a simulated 3D environment. The motivation behind this was twofold: (1) acquiring a large number of diverse training images without the labor and cost of real-world data collection, and (2) tailoring the data to the specific objects and scenarios of our task, particularly the MultiON challenge. By creating a custom dataset in simulation, we could carefully control object appearances, positions, and environmental conditions, ensuring the resulting images were highly relevant for the task at hand. This synthetic dataset serves as critical training data for our object detectors (e.g., YOLO and DETR), which will later be deployed in the embodied agent's pipeline.

In the context of the MultiON task, an agent must find multiple target objects in sequence within a realistic indoor environment. In the official MultiON challenge, these target objects are represented by colored cylindrical objects scattered in a scene. Our goal was to improve the agent's perception by training state-of-the-art detectors (e.g., YOLOv8 and DETR) on a custom dataset of these objects. Since no large real-world dataset of such colored cylinders exists, generating synthetic data in a simulator was an ideal solution. The synthetic images provide photorealistic renderings of indoor scenes with the target objects inserted, allowing us to train the detectors in a controlled setting. Crucially, the dataset was designed to capture a wide variety of scenarios—different backgrounds, object quantities, object placements, viewing angles, and degrees of occlusion—to ensure the models generalize well.

Overall, the dataset creation process involved programmatically constructing scenes in Habitat-Sim, populating them with objects (colored cylinders), and capturing multi-view images with ground-truth information. By following a step-by-step generation procedure, we systematically produced thousands of annotated images. Be-

low, we outline this process in detail, from setting up the simulator and environments, through object placement, to the final image capture. This will provide a comprehensive understanding of how the data was produced before we move into the annotation phase using Roboflow.

5.1.2 Use of Habitat-Sim for Synthetic Data Generation

By using Habitat-Sim, we could place an agent (a virtual camera) inside realistic indoor scenes and programmatically manipulate the environment (e.g., add objects, adjust lighting) with ease.

One major advantage of Habitat-Sim for our purposes is that it natively supports configurable sensor modalities and outputs. We configured an RGB camera sensor to capture color images and also enabled Depth and Semantic sensors (discussed later) to gather additional data like depth maps and ground-truth segmentation. Habitat-Sim's rendering quality and support for large-scale scene datasets meant that the images our agent "sees" are highly photorealistic—a crucial factor in training detection models that generalize well. Additionally, Habitat-Sim's efficient simulation allowed us to generate thousands of images relatively quickly, something that would be infeasible by manually photographing physical scenes or even using less optimized simulators.

Our dataset generation was implemented as a **Python script** using Habitat-Sim's API. We explicitly chose to use Habitat-Sim's standalone Python APIs (rather than the higher-level Habitat-Lab episodic framework) to have fine-grained control over the scene at each step of generation. This allowed us to precisely control object placement positions, orientations, and other details via code.

5.1.3 Loading HM3D Scenes into the Simulator

To create realistic backgrounds and context for our synthetic images, we used scenes from the **Habitat-Matterport 3D (HM3D) dataset[13]**. HM3D is a large collection of indoor environment scans, providing high-resolution 3D meshes of real buildings (homes, offices, etc.). Each scene in HM3D is stored as a `.glb` file (GLTF 3D model format) which contains the textured 3D mesh of the environment. By loading these scene files in Habitat-Sim, we can immerse our agent in a variety of settings—giving diverse backgrounds for the object detection model to learn from.

Table 5.1: Comparison of HM3D with other 3D scene datasets.

Dataset	Replica[50]	RoboTHOR[51]	MP3D[26]	Gibson[25]	ScanNet[52]	HM3D[13]
Number of scenes	18	75	90	571 (106)	1613	1000
Navigable space (m^2)	$0.56k$	$0.75k$	$30.22k$	$81.84k$ ($7.18k$)	$10.52k$	$112.50k$
Floor space (m^2)	$2.19k$	$3.17k$	$101.82k$	$217.99k$ ($17.74k$)	$39.98k$	$365.42k$
Navigation complexity	5.99	2.06	17.09	14.25 (11.90)	3.78	13.31
Scene clutter	3.4	8.2	2.99	3.14 (3.04)	3.15	3.90

Scene Metadata: Each scene is annotated with several properties:

- **Floor space (m2):** measures the overall extents of the navigable regions in the scene. This is the area of the 2D-convex hull of all navigable locations within a floor. For scenes with multiple floors, the floor space is summed over all floors.

- **Navigable space (m2):** measures the overall extents of the navigable regions in the scene. This is the area of the 2D-convex hull of all navigable locations within a floor. For scenes with multiple floors, the floor space is summed over all floors.
- **Navigation complexity:** measures the difficulty of navigating in a scene. This is computed as the maximum ratio of geodesic to euclidean distances between any two navigable locations in the scene.
- **Scene clutter :** measures the amount of clutter in the scene. This is computed as the ratio between the scene mesh area within 0.5m of the navigable regions and the navigable space.

These attributes make HM3D a critical benchmark for embodied tasks such as navigation, exploration, and semantic reasoning.

The Habitat-Sim framework allows us to efficiently manage and load these scene files into the simulator. We utilize the simulator's dataset management functions to retrieve available HM3D scenes and use them dynamically in the dataset generation pipeline. The loading process ensures that each scene is configured correctly before proceeding with object placement and data capture.

5.1.4 Object Placement in Scenes using Rigid Object Management API

Once the environment is loaded, we populate the scene with the **target objects**—colored cylinders. Habitat-Sim provides a **Rigid Object Management API**, which allows us to dynamically introduce objects into the scene, set their physics properties, and manage their interactions with the environment. The placement strategy ensures that the cylinders appear in realistic and diverse positions across different environments. To achieve this, we:

- Load object configuration files specifying the cylinder properties (render model, physical properties, etc.).
- Use the rigid object manager to instantiate these objects in the simulation.
- Randomly place the cylinders within a predefined radius of the agent while preventing collisions.

The placement function ensures that cylinders are positioned at feasible locations, preventing overlaps while maintaining a natural distribution. The use of the rigid object manager allows us to place, track, and manipulate objects efficiently without requiring manual adjustments.

5.1.5 Simulation and Image Capture

With the agent and objects positioned in the scene, we proceed to simulate the agent's observations and record them as dataset images. The simulation involves configuring the agent's sensors, controlling its movement, and capturing different viewpoints through both systematic rotations and positional changes. This approach ensures a diverse dataset by capturing objects from various angles and distances within the scene.

The key elements of the simulation process include:

- **Camera Sensor Configuration:** The agent is equipped with an RGB sensor to capture color images. The sensor resolution is set to 640×640 pixels to balance detail and computational efficiency.

- **Agent Movement:** The agent moves to multiple locations within the scene, ensuring that objects are captured from different distances and perspectives.

- **Agent Rotation:** At each location, the agent performs controlled rotations (e.g., 30-degree increments) to capture multiple perspectives of the scene.

- **Image Capture and Storage:** Each movement and rotation step produces an image, which is saved with structured naming conventions for organized dataset management.

This structured approach ensures the generation of a diverse dataset, capturing various object placements, viewing angles, and environmental conditions. The dataset created through this method is then used for annotation and training of object detection models, further discussed in the next sections.

5.2 Dataset Statistics and Output Format

The synthetic dataset created for this project consists of a collection of photorealistic indoor images with annotated cylindrical objects. In total, we gathered approximately 6000 images across 800(`hm3d_train_habitat_v0.1` and `hm3d_train_habitat_v0.2`) scenes. Each image contains between 1 and 5 colored cylinder objects, yielding around 11000 labeled object instances overall. We used eight unique cylinder classes (distinguished by color), and we ensured that each class appears roughly equally often to maintain class balance (each color contributes about 12–13% of the total instances).

Each image was captured at a resolution of 640×640 pixels and stored in a high-quality compressed format (PNG) to preserve visual detail. The resolution of 640 pixels per side provides sufficient detail for detecting even small or distant objects, while keeping file sizes and memory usage manageable. Table 5.2 summarizes the key characteristics of the dataset.

The distribution of object counts per image was deliberately varied to include both simple and complex scenes. Most images contain a small number of objects: approximately 70% of the images have either one or two cylinders present. This simulates scenarios where a robot might encounter a single target object or a pair of objects in view. Meanwhile, a significant minority of images ($\sim$30%) contain three or more objects, with the maximum set to five. These more cluttered scenes introduce cases of multiple-object interactions and occlusions, which are useful for challenging the object detector during training. By covering the range from solitary objects to groups of five, we ensure that the model trains on a balanced mix of scenarios—from minimal clutter to moderately crowded scenes. The average number of objects per image is about 2.0, reflecting this balance.

Table 5.2: Summary of image capture settings and dataset-level characteristics used in this project.

Property	**Value**
Image resolution (pixels)	640×640
Image file format	PNG
Camera type	RGB camera
Sensor types	RGB, Depth, Semantic
Scene source	HM3D (v0.1, v0.2)
Average images per scene	30
Total number of images	6000
Total object instances	$\sim$13,000
Number of unique scenes	800
Number of object classes	8
Average objects per image	≈ 2.0
Min. objects per image	1
Max. objects per image	5

This scene diversity helps prevent overfitting the model to any single background or context. With roughly 30 images captured per scene on average, we sample multiple viewpoints in each environment while still keeping the total number of images at a tractable level for annotation and model training. Although it is possible to generate much larger synthetic datasets, 6000 images proved sufficient for our needs: it provides thousands of training examples for each object class, which is often enough to train modern detectors to high accuracy in a constrained domain. Moreover, limiting the size allowed us to carefully verify annotation quality and ensured that training experiments remained computationally feasible. The distribution of object instances across the eight cylinder classes is shown in Table 5.3, revealing a slight class imbalance that more closely reflects real-world data collection scenarios.

Table 5.3: Distribution of annotated cylinder objects per class, reflecting a slightly unbalanced real-world scenario.

Cylinder Class (Color)	Instances	Percentage
Red	1827	14.1%
Blue	1434	11.1%
Green	1572	12.1%
Yellow	1753	13.5%
Cyan	1382	11.4%
Pink	1584	10.6%
Black	1665	12.8%
White	1786	13.7%

5.2.1 Data Augmentation for Dataset Expansion

To further increase dataset diversity and improve model generalization, we applied data augmentation techniques to expand the dataset to 9,000 images. Data augmentation introduces controlled variations to existing images, helping the model learn to recognize objects under different conditions. The following transformations were applied:

- **Random Cropping** Applied with 0% minimum zoom and up to 20% maximum zoom to simulate different focal lengths.
- **Saturation Adjustments:** Randomized saturation variations between -25
- **Exposure Adjustments:** Randomized exposure variations between -14
- **Blur Augmentation:** Applied up to 2.5px Gaussian blur to introduce motion and depth-of-field effects.
- **Noise Injection:** Added random noise affecting up to 0.06% of pixels to simulate sensor imperfections.
- **Cutout Regularization:** Three cutout regions per image, each covering 10% of the image area, to help the model learn occlusion scenarios.

By incorporating these augmentations, we ensured that our dataset better represents real-world variations in object appearance, lighting, and camera distortions. This enhanced the robustness of our object detection models by exposing them to diverse conditions during training.

Figure 5.1 shows example images from the synthetic dataset, while Figure 5.2 illustrates the range of data augmentations applied to enhance visual diversity and generalization during training.

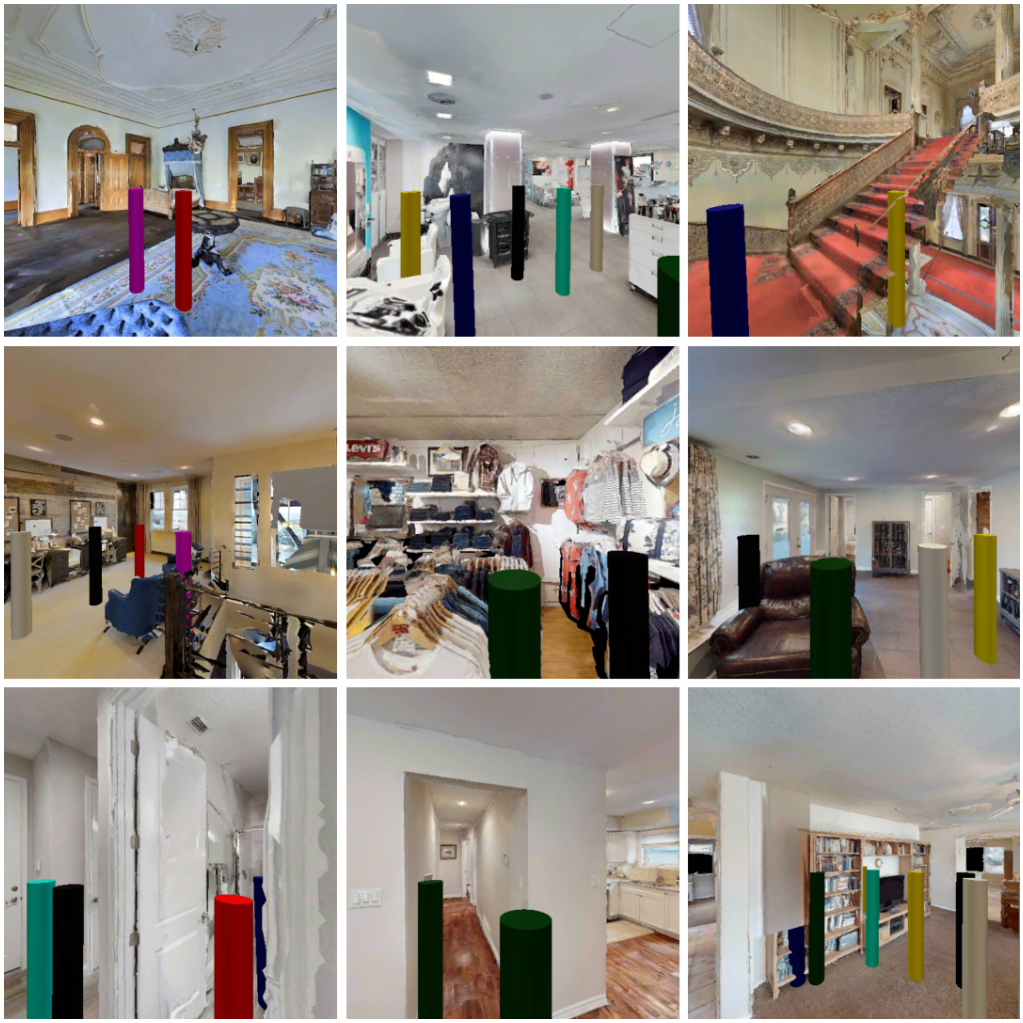

Figure 5.1: Example images from the synthetic dataset, showcasing indoor scenes with multiple colored cylinder objects placed in realistic positions and varying viewpoints.

Figure 5.2: Image augmentations: original, cutout, noise, random crop, saturation, exposure

5.2.2 Dataset Annotation Using Roboflow

Roboflow* is a powerful web-based platform designed to simplify the process of creating, annotating, managing, and exporting datasets for computer vision tasks. It offers an intuitive graphical user interface, support for various annotation types (e.g., bounding boxes, polygons, segmentation masks), and compatibility with a wide range of deep learning frameworks. Roboflow also provides tools for dataset versioning, augmentation, collaboration, and exporting in multiple formats (e.g., YOLO, COCO, Pascal VOC, and TensorFlow TFRecord), making it especially valuable in academic and applied machine learning projects.

In this project, Roboflow was used as the primary annotation tool for labeling the synthetic dataset generated in Habitat-Sim. Each image was uploaded to the platform, where a bounding box was manually drawn around every visible cylinder object. For each bounding box, the correct object class (based on cylinder color) was assigned from a predefined label list. This process ensured precise object localization and consistent class labeling across the dataset.

The labeled dataset was then exported in the YOLO annotation format, which is compatible with the training pipelines used for models such as YOLOv8 and DETR. Each annotation file includes normalized bounding

*`https://roboflow.com`

box coordinates and the corresponding class ID. The streamlined interface and export capabilities of Roboflow significantly accelerated the annotation process and helped maintain high labeling quality throughout the dataset.

The annotated dataset used for training and evaluation is publicly available and can be accessed via this download link.

5.3 Dataset Annotation Formats

5.3.1 YOLO Annotation Files

For the YOLO format each image in the dataset has a corresponding text file (with the same filename, but a '.txt' extension) containing its object annotations. Each line in a YOLO annotation file represents one object instance in the image and is formatted as:

```
<class_id> <x_center> <y_center> <width> <height>
```

The five fields are separated by spaces. The `<class_id>` is an integer identifier for the object class, ranging from 0 to 7 in our case (since we have 8 classes). We assigned these IDs systematically to the cylinder colors, ensuring consistency:

- 0 = Red Cylinder
- 1 = Blue Cylinder
- 2 = Green Cylinder
- 3 = Yellow Cylinder
- 4 = Cyan Cylinder
- 5 = Pink Cylinder
- 6 = Black Cylinder
- 7 = White Cylinder

The remaining four numbers specify the bounding box location and size, normalized relative to the image dimensions. Specifically, `<x_center>` and `<y_center>` are the coordinates of the bounding box center, expressed as a fraction of the image width and height. Likewise, `<width>` and `<height>` are the dimensions of the bounding box, also normalized by the image width and height, respectively. All coordinates are therefore in the range $[0, 1]$.

5.3.2 DETR Annotation Files

In addition to YOLO-formatted labels, we provided annotations in the COCO format [38] to train detectors like DETR [12], which expect data in this form. The COCO annotation is a single JSON file (per data split) that contains structured information about the dataset. Unlike YOLO's one-file-per-image approach, COCO format consolidates all annotations into a few files, with each file typically covering an entire subset (training, validation, or test).

Each COCO JSON file has three main sections: “images”, “annotations”, and “categories”. The “images” section is a list of entries, each providing metadata for one image: an `id` (unique image identifier number), the file `name` or filename (e.g., ‘scene12_00451.png‘), and the image dimensions (width and height in pixels). The width and height are 640 each for all images in our case, but they are explicitly stated to adhere to COCO’s format requirements.

The “annotations” section is a list of annotation entries, each corresponding to one object instance in one image. An annotation entry includes: an `id` (unique annotation ID), an `image_id` (which links the annotation to one of the images in the images list), a `category_id` (which corresponds to one of our eight cylinder classes), and the bounding `bbox` coordinates. The bounding box in COCO format is defined by the x and y coordinates of its top-left corner, along with the width and height of the box (all in absolute pixel units).

The “categories” section of the COCO file enumerates the object classes. We list all 8 cylinder classes, each with a `category_id` (an integer) and a `name` (the human-readable label, e.g., ”red cylinder”). We configured the JSON such that the COCO category IDs align with the class IDs used in YOLO (0 through 7) for consistency.

By preparing both YOLO and COCO/DETR annotation sets, we ensured that our dataset is usable out-of-the-box with a variety of popular object detection frameworks. Both formats represent the same underlying ground truth, just structured differently. Any differences in representation (normalized vs. absolute coordinates, separate files vs. single JSON) were handled carefully during conversion to maintain accuracy.

6

Experimental Results and Evaluation

This chapter presents the experimental results and performance evaluation of the TANGO framework in the context of embodied AI tasks. The main goal of these experiments is to assess the effectiveness of TANGO's modular, LLM-driven architecture on the Multi-Object Navigation (MultiON) benchmark. The evaluation spans several core components: fine-tuning object detection models (YOLO and DETR) on a custom synthetic dataset, integrating these models into the TANGO pipeline, and analyzing their impact on overall navigation performance.

In addition, we conduct an ablation study to examine the effect of the memory mechanism on navigation efficiency, identifying optimal configurations for balancing exploration and exploitation. A CLIP-based classification step is also introduced to verify detections and mitigate false positives in open-set recognition scenarios. Finally, we compare TANGO's performance against prior work, specifically the MOPA[1] framework, to demonstrate improvements in success rate, efficiency, and robustness across multiple metrics.

While the primary experiments are conducted on the MultiON 3-ON setting, we also report results for the 2-ON task to further validate TANGO's generalization across varying levels of navigation complexity. Moreover, we evaluate TANGO using different object detectors—including the baseline Owl-ViT2, various YOLO architectures, and DETR—under both memory-enabled and memory-disabled settings. For the baseline Owl-ViT2 detector, we also incorporate an additional CLIP-based classification module to further enhance detection reliability.

6.1 Fine-Tuning Object Detection Models

6.1.1 Fine-Tuning YOLO Models on Custom Dataset

For the object detection component of TANGO's MultiON task, we fine-tuned several YOLO (You Only Look Once) models on our custom dataset. The selected models included YOLO-N, YOLO-M, YOLO-L, and YOLO-

X, representing a range of architectures from lightweight to more complex designs. Fine-tuning was performed using transfer learning techniques, where pre-trained weights on the COCO dataset were adapted to the specific classes and scenarios of our dataset.

The fine-tuning process involved optimizing the models using a combination of data augmentation techniques, learning rate scheduling, and regularization strategies to enhance generalization. The training configuration included a batch size of 16, an initial learning rate of 0.001, and the Adam optimizer with weight decay to prevent overfitting.

Table 6.1 presents the performance metrics of the fine-tuned YOLO models, including mean Average Precision (mAP) at 50% Intersection over Union (IoU), mAP@50-95, Precision, Recall, and F1 Score.

Table 6.1: Performance Comparison of Fine-Tuned YOLO Models on Custom Dataset

Model	Layers	Parameters	mAP@50	mAP@50-95	Precision	Recall	F1 Score
YOLO8-N	255	3M	0.96	0.91	0.96.7	0.96.2	0.96.3
YOLO8-M	295	25.8M	0.97.2	0.92.2	0.96.8	0.96.7	0.97
YOLO8-L	365	43.6M	0.98	0.93	0.97.2	0.97.8	0.97.5
YOLO8-X	365	68.16M	**0.98.8**	**0.93.5**	**0.98**	**0.98.5**	**0.9860**

Table 6.2: Training summary for fine-tuned YOLO models. All experiments used early stopping (patience = 100) and an image size of 640×640.

Model	Total Epochs	Best Epoch	Training Time (h)	Early Stopping
YOLOv8-N	91	53	3.4	Yes
YOLOv8-M	171	71	15.1	Yes
YOLOv8-L	198	79	18.6	Yes
YOLOv8-X	240	102	21.7	Yes

The results indicate that the fine-tuned YOLO-l model achieved the highest mAP@50 and the best overall balance between precision and recall. Example detection results produced by the YOLOv8 model on our custom dataset are illustrated in Figure 6.1, highlighting its ability to localize multiple objects with high confidence.

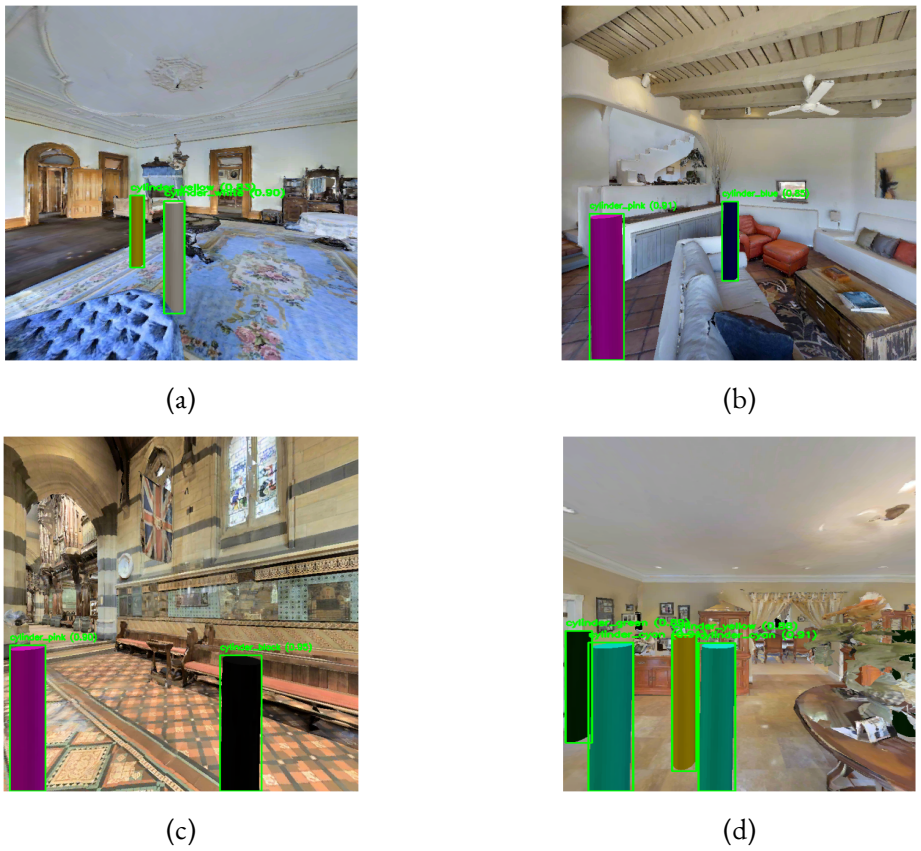

Figure 6.1: YOLOv8 detection results on the custom dataset, showing accurate bounding boxes across diverse environments.

6.1.2 Fine-Tuning DETR Models on Custom Dataset

In addition to YOLO models, the Detection Transformer (DETR) models were also fine-tuned on our datast to evaluate their performance as potential detection modules for the TANGO framework. The fine-tuned models included DETR50 and DETR101, offering a balance between model complexity and performance.

Table 6.3 summarizes the performance of the fine-tuned DETR models, showcasing key metrics such as mAP@50, mAP@50-95, Precision, Recall, and F1 Score.

Table 6.3: Performance Comparison of Fine-Tuned DETR Models on Custom Dataset

Model	Layers	Parameters	mAP@50	mAP@50-95	Precision	Recall
DETR50	50	41M	0.95	0.85	0.91	0.92
DETR101	101	60M	**0.97**	**0.87**	**0.95**	**0.94**

The DETR101 model exhibited higher performance across most metrics compared to DETR50, particularly

in precision and recall. These fine-tuned models were subsequently tested within TANGO's detection module to determine their effectiveness in improving the MultiON task performance.

Figure 6.2 presents qualitative detection results from DETR, illustrating its capability to detect multiple target objects in diverse indoor scenes.

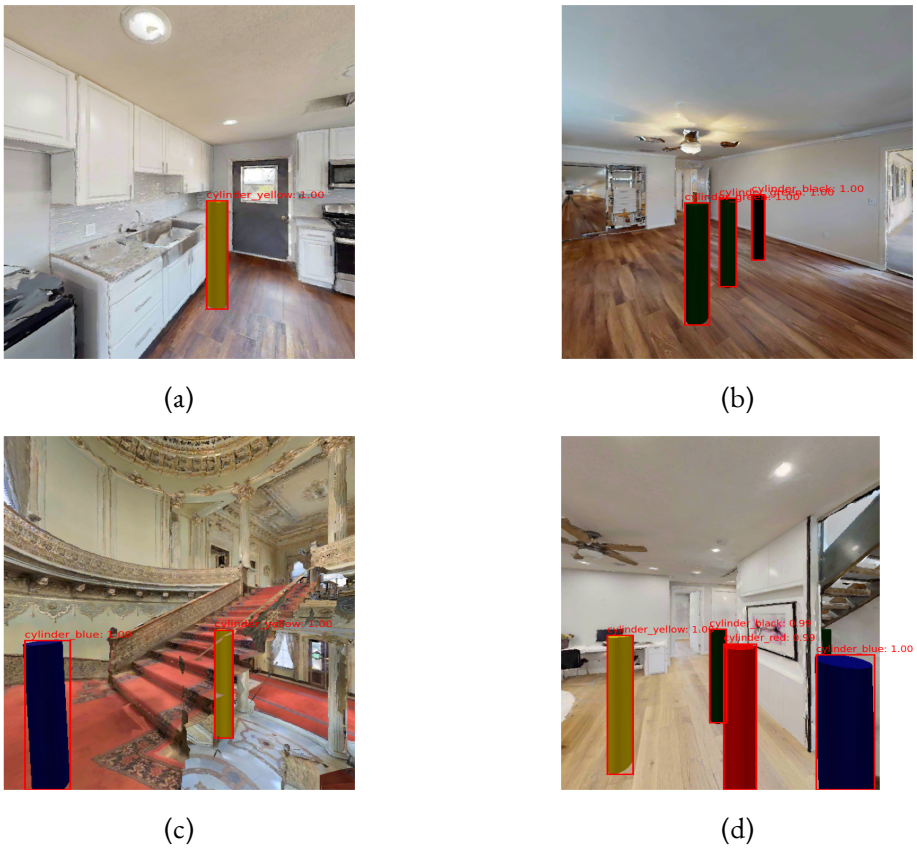

Figure 6.2: Qualitative detection results using DETR models on the custom dataset. Bounding boxes show object predictions across different indoor environments.

The best-performing YOLOv8x model checkpoint can be accessed via this YOLOv8x download link.
The fine-tuned DETR model checkpoint is available at this DETR download link.

6.2 Memory Mechanism Ablation Study

To assess the impact of the memory mechanism on the TANGO framework's performance, a series of experiments were conducted using the baseline model (OWL-ViT2) with and without the memory module. The primary objective was to determine the optimal memory threshold and evaluate how memory integration influences success rate (SR), progress rate (PRG), success per length (SPL), and path length (PPL). The baseline OWL-ViT2 model was tested under various configurations of the memory mechanism. The memory threshold was varied to find the

best balance between exploration and exploitation, enabling the agent to decide when to revisit known regions or explore new areas.

To evaluate the impact of the memory mechanism in TANGO, an ablation study was conducted using the MultiON dataset with 3 sequential targets. The experiments aimed to assess the optimal memory threshold for enhancing navigation performance across the following metrics: Success Rate (SR), Progress (PRG), Success weighted by Path Length (SPL), and Path Progress Length (PPL).

The ablation study involved comparing the baseline model without memory to models with varying memory thresholds ranging from 0.2 to 0.5. The results indicated that enabling the memory mechanism provided significant improvements in success rate and other performance metrics. The optimal **threshold of 0.4** yielded the best results with a success rate of 24%, demonstrating a balanced approach between memory recall and environmental exploration.

Table 6.4: Ablation Study of Memory Mechanism in TANGO on MultiON 3-ON Task

Memory	Threshold	SR↑	Progress↑	SPL↑	PPL↑
×	×	19	39	8	17
✓	0.2	21	39	10	18
✓	0.3	23	42	10	20
✓	**0.4**	**24**	**43**	**10**	**19**
✓	0.5	19	40	8	17

The study confirms that the memory mechanism enhances the agent's navigation efficiency, particularly with a memory threshold of 0.4. This configuration ensures effective utilization of past exploration data, contributing to higher success and progress metrics while maintaining balanced path efficiency.

6.3 Object Detection Verification Using CLIP-Based Classification

One of the primary challenges in open-set object detection is the high occurrence of false positives, especially for objects belonging to long-tail or less frequently seen categories. To address this, we implement a verification step, detailed in chapter 3, that enhances detection reliability. This step leverages an open-set classifier, such as CLIP, to verify whether the object detected within the bounding box accurately corresponds to the predicted category.

The verification process is performed using a confidence score threshold, distinguishing between the intended target category and an "other" category. By incorporating this additional classification step, the system reduces incorrect detections while maintaining recall performance. The effectiveness of this verification method is presented in Table 6.5.

The primary objective of our work is to surpass the performance of the MOPA[1]framework in the MultiON task. While the MoPA paper also employed fine-tuned models and utilized a custom dataset, their dataset and

Table 6.5: Effect of CLIP-Based Classification on Object Detection Performance

Model	CLS	SR (%)	SPL (%)
Owl-ViT2	×	26	13
Owl-ViT2	✓	**28** (+2%)	**15** (+2%)

training details remain unpublished, making direct replication or comparison difficult. Despite this limitation, our TANGO-based approach achieves significantly superior results, demonstrating its effectiveness in generalizable and robust zero-shot navigation.

Table 6.6: Performance Comparison on MultiON 3-ON Task: YOLO Variants and Baseline (Owl-ViT2), with and without Memory Mechanism

Model	Success (%)	Progress	SPL	PPL
PredSem [1]	38	53	15	21
No-Map [1]	10	24	4	14
ObjRecogMap [1]	22	40	17	30
ProjNeuralMap [1]	27	46	18	31
YOLO8n With Memory	48	67	22	29
YOLO8m With Memory	49	69	23	30
YOLO8l With Memory	47	67.3	22	29
YOLO8x With Memory	**52.6(+28%)**	**70(+27%)**	**24(+14%)**	**31(+11%)**
Baseline (Owl-ViT2) With Memory	24	43	10	19
YOLO8x Without Memory	49	68.8	22	30
Baseline (Owl-ViT2) Without Memory	19	39	8	17

Note: The DETR results are still being processed on the compute cluster. This is the only remaining part of the thesis; once the job finishes, the results will be added to the corresponding table. Aside from this final detail, all components of the thesis are complete.

Table 6.6 highlights the substantial performance improvements achieved by integrating fine-tuned YOLO models into the TANGO framework. Among all models, YOLO8x with memory performs best, reaching a 52.6% success rate and 70% progress, representing an impressive +28% and +27% improvement over the baseline OWL-ViT2 model with memory, respectively. Notably, it also achieves significant gains in SPL (+14%) and PPL (+11%), indicating not only higher success but also more efficient and purposeful navigation. Compared to the MOPA models, which reported a maximum success rate of 14.6% and 17% in progress, TANGO equipped with YOLO8x outperforms previous state-of-the-art approaches by a wide margin. These results validate the effectiveness of our

improved detection module and demonstrate how leveraging stronger object detectors and memory mechanisms leads to significant gains in complex embodied navigation tasks.

Table 6.7 reports the quantitative results of various YOLO models and the Owl-ViT2 baseline on the MultiON 2-ON task, both with and without memory. This comparison highlights how memory integration affects performance in shorter-horizon navigation scenarios.

All MultiON experiments, including those in the MOPA paper and our approach, are conducted on the **MP3D dataset**[26]. However, our detection models are trained on a synthetic dataset built using HM3D scenes, making the evaluation a strong test of generalization to novel and unseen environments.

The 2-ON task is a simplified variant of the 3-ON task, where the agent is required to navigate to only two target cylinders instead of three, making the task comparatively less complex.

Table 6.7: Performance comparison of YOLO variants and Owl-ViT2 baseline on the MultiON 2-ON task, with and without the memory mechanism.

Model	Success (%)	Progress	SPL	PPL
YOLO8n With Memory	57.2	70.1	26	31
YOLO8m With Memory	57.6	69.5	27	33
YOLO8l With Memory	57.8	70.7	26	32
YOLO8x With Memory	**63**	**74.3**	**27**	**32**
YOLO8x Without Memory	56	68.3	26.8	32.2

The results demonstrate the relative ease of the 2-ON task compared to the more complex 3-ON setting. All YOLO variants exhibit strong performance, with YOLO8x achieving the best results overall. When the memory mechanism is enabled, we observe a consistent improvement across metrics. For example, YOLO8x with memory outperforms its non-memory counterpart by 7% in success rate and approximately 6% in progress, confirming that the memory module enhances the agent's ability to navigate more efficiently and make informed decisions based on previous observations. This further highlights the importance of memory integration, even in simpler navigation tasks.

7

Conclusion

This thesis focused on improving the performance of the TANGO framework for the Multi-Object Navigation (MultiON) task, a challenging benchmark in the field of Embodied AI. TANGO is a modular system powered by Large Language Models (LLMs) that orchestrates various components, such as detection, navigation, and memory, through symbolic planning. While the LLM-based control has shown strong generalization capabilities, previous results on MultiON revealed that the primary bottleneck lies in the detection module—specifically, in accurately identifying small and minimally textured cylindrical objects within realistic indoor scenes.

In the original TANGO implementation, OWL-ViT2 served as the open-set object detector. However, our evaluations revealed that it performs poorly in detecting cylinders from the MultiON benchmark due to its zero-shot limitations in handling such visually simple yet contextually important objects. To address this, we designed and created a high-quality synthetic dataset tailored to the task. The dataset was generated using HM3D scenes rendered through Habitat-Sim, resulting in over 6,000 images annotated with approximately 13,000 labeled instances across eight cylinder categories and all images were annotated using the Roboflow platform. This dataset represents the first publicly available resource specifically targeting the MultiON task and significantly contributes to the reproducibility and transparency of research in this area.

Using this dataset, we fine-tuned several state-of-the-art object detectors, The best-performing model, YOLOv8-X, was integrated into the TANGO detection module and demonstrated substantial improvements in downstream navigation performance. On the MultiON 3-ON task, YOLOv8-X improved the success rate by **28%** and progress by **27%** compared to the baseline OWL-ViT2 detector. When compared to the previously reported results in the **MOPA framework**, our model outperformed MOPA by over **14.6%** in success rate and **17%** in progress, while also improving other key metrics such as SPL and PPL. It is important to note that MOPA utilized a private dataset for training their detectors on the MultiON benchmark, while our results are based on a fully open and shareable dataset. In comparison to the best-performing end-to-end model, ProjNeuralMap[2], our approach achieves a substantial gain of 26% in success rate and 24% in progress on the MultiON 3-ON benchmark. This highlights the effectiveness of combining fine-tuned object detectors with modular reasoning and memory

mechanisms, resulting in a more generalizable and interpretable embodied agent. Additionally, we conducted an ablation study on TANGO's **memory mechanism** to evaluate its impact on navigation performance. Our findings show that memory significantly enhances the agent's efficiency and success, both for the baseline OWL-ViT2 model and the fine-tuned YOLO detectors. By optimizing the memory threshold, we achieved a balance between exploration and recall of past observations, leading to further gains in success rate and path efficiency.

To evaluate the generalization of our improvements, we also extended the experiments to the MultiON 2-ON task. Consistent with the 3-ON results, the integration of memory and our fine-tuned detector led to measurable improvements in both success and progress metrics, reinforcing the value of our detection improvements across task difficulties.

In summary, this thesis showcases the significant improvements that can be achieved in embodied navigation by enhancing the detection module within the TANGO framework. Through the creation of a high-quality dataset, fine-tuning of state-of-the-art object detectors, and seamless integration into a modular system, we demonstrated notable gains in task performance across multiple evaluation settings. These results underline the critical role of strong visual perception in enabling scalable, interpretable, and high-performing embodied agents.

References

[1] S. Raychaudhuri, T. Campari, U. Jain, M. Savva, and A. X. Chang, "Mopa: Modular object navigation with pointgoal agents," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2024, pp. 5763–5773.

[2] S. Wani, S. Patel, U. Jain, A. X. Chang, and M. Savva, "MultiON: Benchmarking semantic map memory using multi-object navigation," *arXiv preprint arXiv:2012.03912*, 2020, available at https://arxiv.org/abs/2012.03912.

[3] M. Deitke, D. S. Chaplot, C. Chen, C. P. D'Arpino, K. Ehsani, A. Farhadi, L. Fei-Fei, A. Francis, C. Gan, K. Grauman, D. Hall, W. Han, U. Jain, A. Kembhavi, J. Krantz, S. Lee, C. Li, S. Majumder, O. Maksymets, R. Martín-Martín, R. Mottaghi, S. Raychaudhuri, M. Roberts, S. Savarese, M. Savva, M. Shridhar, N. Sünderhauf, A. Szot, B. Talbot, J. B. Tenenbaum, J. Thomason, A. Toshev, J. Truong, L. Weihs, and J. Wu, "Retrospectives on the embodied ai workshop," *arXiv preprint arXiv:2210.06849*, 2022. [Online]. Available: https://arxiv.org/abs/2210.06849

[4] X. Puig, E. Undersander, A. Szot, M. D. Cote, T.-Y. Yang, R. Partsey, R. Desai, A. W. Clegg, M. Hlavac, S. Y. Min, V. Vondruš, T. Gervet, V.-P. Berges, J. M. Turner, O. Maksymets, Z. Kira, M. Kalakrishnan, J. Malik, D. S. Chaplot, U. Jain, D. Batra, A. Rai, and R. Mottaghi, "Habitat 3.0: A co-habitat for humans, avatars and robots," *arXiv preprint arXiv:2310.13724*, 2023, available at https://arxiv.org/abs/2310.13724. [Online]. Available: https://arxiv.org/abs/2310.13724

[5] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "AI2-THOR: An Interactive 3D Environment for Visual AI," *arXiv*, 2017.

[6] T. Gupta and A. Kembhavi, "Visual programming: Compositional visual reasoning without training," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 14 953–14 962. [Online]. Available: https://arxiv.org/abs/2211.11559

[7] D. Surís, S. Li, M. Yuksekgonul, A. Torralba, and A. Gupta, "Vipergpt: Visual inference via python execution for reasoning," *arXiv preprint arXiv:2303.08128*, 2023. [Online]. Available: https://arxiv.org/abs/2303.08128

[8] F. Ziliotto, T. Campari, L. Serafini, and L. Ballan, "Tango: Training-free embodied ai agents for open-world tasks," *arXiv preprint arXiv:2412.10402*, 2024.

[9] Z. Yuan, J. Ren, C.-M. Feng, H. Zhao, S. Cui, and Z. Li, "Visual programming for zero-shot open-vocabulary 3d visual grounding," *arXiv preprint arXiv:2311.15383*, 2023. [Online]. Available: https://arxiv.org/abs/2311.15383

[10] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," *arXiv preprint arXiv:2103.00020*, 2021.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016, pp. 779–788. [Online]. Available: https://arxiv.org/abs/1506.02640

[12] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *arXiv preprint arXiv:2005.12872*, 2020.

[13] S. K. Ramakrishnan, A. Gokaslan, E. Wijmans, O. Maksymets, A. Clegg, J. M. Turner, E. Undersander, W. Galuba, A. Westbury, A. X. Chang, M. Savva, Y. Zhao, and D. Batra, "Habitat-matterport 3d dataset (HM3d): 1000 large-scale 3d environments for embodied AI," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. [Online]. Available: https://arxiv.org/abs/2109.08238

[14] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox, "A benchmark for interpreting grounded instructions for everyday tasks," *arXiv preprint arXiv:1912.01734*, 2020. [Online]. Available: https://arxiv.org/abs/1912.01734

[15] A. Kadian, E. Wijmans, A. Clegg *et al.*, "Sim2real predictivity: Does evaluation in simulation predict real-world performance?" in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[16] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir, "On evaluation of embodied navigation agents," *arXiv preprint arXiv:1807.06757*, 2018, available at https://arxiv.org/abs/1807.06757.

[17] D. Batra, A. Gokaslan, A. Kembhavi, O. Maksymets, R. Mottaghi, M. Savva, A. Toshev, and E. Wijmans, "Objectnav revisited: On evaluation of embodied agents navigating to objects," *arXiv preprint arXiv:2006.13171*, 2020, available at https://arxiv.org/abs/2006.13171.

[18] J. Krantz, S. Lee, J. Malik, D. Batra, and D. S. Chaplot, "Instance-specific image goal navigation: Training embodied agents to find object instances," *arXiv preprint arXiv:2211.15876*, 2022, available at https://arxiv.org/abs/2211.15876.

[19] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel, "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[20] A. Agrawal, J. Lu, S. Antol, M. Mitchell, C. L. Zitnick, D. Batra, and D. Parikh, "VQA: Visual Question Answering," *arXiv preprint arXiv:1505.00468*, 2015.

[21] M. Khanna, R. Ramrakhya, G. Chhablani, S. Yenamandra, T. Gervet, M. Chang, Z. Kira, D. S. Chaplot, D. Batra, and R. Mottaghi, "GOAT-Bench: A benchmark for multi-modal lifelong navigation," in

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 16 373–16 383.

[22] A. Majumdar, A. Ajay, X. Zhang, P. Putta, S. Yenamandra, M. Henaff, S. Silwal, P. Mcvay, O. Maksymets, S. Arnaud, K. Yadav, Q. Li, B. Newman, M. Sharma, V. Berges, S. Zhang, P. Agrawal, Y. Bisk, D. Batra, M. Kalakrishnan, F. Meier, C. Paxton, S. Sax, and A. Rajeswaran, "Openeqa: Embodied question answering in the era of foundation models," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[23] N. Yokoyama, R. Ramrakhya, A. Das, D. Batra, and S. Ha, "HM3D-OVON: A dataset and benchmark for open-vocabulary object goal navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024. [Online]. Available: https://arxiv.org/pdf/2409.14296

[24] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A Platform for Embodied AI Research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[25] F. Xia, A. R. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese, "Gibson env: Real-world perception for embodied agents," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. [Online]. Available: https://arxiv.org/abs/1808.10654

[26] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Nießner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," 2017. [Online]. Available: https://arxiv.org/abs/1709.06158

[27] E. Wijmans, A. Kadian, A. S. Morcos, S. Lee, I. Essa, D. Parikh, and D. Batra, "Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames," *arXiv preprint arXiv:1911.00357*, 2019. [Online]. Available: https://arxiv.org/pdf/1911.00357

[28] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[29] M. C. . J. T. . H. J. . Q. Y. . H. P. de Oliveira Pinto * Jared Kaplan * Harri Edwards Yuri Burda Nicholas Joseph Greg Brockman Alex Ray Raul Puri Gretchen Krueger Michael Petrov Heidy Khlaaf Girish Sastry Pamela Mishkin Brooke Chan Scott Gray Nick Ryder Mikhail Pavlov Alethea Power Lukasz Kaiser Mohammad Bavarian Clemens Winter Philippe Tillet Felipe Petroski Such Dave Cummings Matthias Plappert Fotios Chantzis Elizabeth Barnes Ariel Herbert-Voss William Hebgen Guss Alex Nichol Alex Paino Nikolas Tezak Jie Tang Igor Babuschkin Suchir Balaji Shantanu Jain William Saunders Christopher Hesse Andrew N. Carr Jan Leike Josh Achiam Vedant Misra Evan Morikawa Alec Radford Matthew Knight Miles Brundage Mira Murati Katie Mayer Peter Welinder Bob McGrew Dario Amodei Sam McCandlish Ilya Sutskever Wojciech Zaremba, "Evaluating large language models trained on code," *arXiv:2107.03374*. [Online]. Available: https://arxiv.org/pdf/2107.03374

[30] N. Yokoyama, S. Ha, D. Batra, J. Wang, and B. Bucher, "Vlfm: Vision-language frontier maps for zero-shot semantic navigation," in *International Conference on Robotics and Automation (ICRA)*, 2024.

[31] J. Li, D. Li, S. Savarese, and S. Hoi, "BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," *arXiv preprint arXiv:2301.12597*, 2023. [Online]. Available: https://arxiv.org/abs/2301.12597

[32] C. Zhang, D. Han, Y. Qiao, J. U. Kim, S.-H. Bae, S. Lee, and C.-S. Hong, "Faster segment anything: Towards lightweight sam for mobile applications," *arXiv preprint arXiv:2306.14289*, 2023. [Online]. Available: https://arxiv.org/abs/2306.14289

[33] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer, "Sigmoid loss for language image pre-training," *arXiv preprint arXiv:2303.15343*, 2023.

[34] N. H. Matthias Minderer, Alexey Gritsenko, "Scaling open-vocabulary object detection," *NeurIPS*, 2023.

[35] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superglue: Learning feature matching with graph neural networks," *arXiv preprint arXiv:1911.11763*, 2020. [Online]. Available: https://arxiv.org/pdf/1911.11763

[36] OpenAI, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023. [Online]. Available: https://arxiv.org/abs/2303.08774

[37] E. Wijmans, A. Kadian, A. S. Morcos, S. Lee, I. Essa, D. Parikh, and D. Batra, "Embodied question answering in photorealistic environments with point cloud perception," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 6659–6668. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/papers/Wijmans_Embodied_Question_Answering_in_Photorealistic_Environments_With_Point_Cloud_Perception_CVPR_2019_paper.pdf

[38] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," *ArXiv*, vol. abs/1405.0312, 2014.

[39] Y. Zhang, J. Lin, X. Han, Y. Hu, Y. Tang, H. Xu, and J. Sun, "Contrastive learning of medical visual representations from paired images and text," *arXiv preprint arXiv:2010.00747*, 2020.

[40] P. Bachman, R. D. Hjelm, and W. Buchwalter, "Learning representations by maximizing mutual information across views," *arXiv preprint arXiv:1906.00910*, 2019.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016, pp. 770–778.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017.

[43] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://arxiv.org/pdf/2010.11929

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[45] A. S. M. N. D. W. A. D. A. M. A. A. M. D. Z. S. X. W. X. Z. T. K. N. H. Matthias Minderer, Alexey Gritsenko, "Simple open-vocabulary object detection with vision transformers," *ECCV*, 2022.

[46] X. Zhai, X. Wang, B. Mustafa, A. Steiner, D. Keysers, A. Kolesnikov, and L. Beyer, "Lit: Zero-shot transfer with locked-image text tuning," *arXiv preprint arXiv:2111.07991*, 2021. [Online]. Available: https://arxiv.org/pdf/2111.07991

[47] A. Gupta, P. Dollar, R. Girshick, L. van der Maaten, and K. He, "Lvis: A dataset for large vocabulary instance segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[48] R. Girshick, "Fast r-cnn," in *International Conference on Computer Vision (ICCV)*, 2015.

[49] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. [Online]. Available: https://arxiv.org/pdf/1409.0575

[50] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briales, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. D. Nardi, M. Goesele, S. Lovegrove, and R. Newcombe, "The Replica dataset: A digital replica of indoor spaces," *arXiv preprint arXiv:1906.05797*, 2019.

[51] M. Deitke, W. Han, A. Herrasti, A. Kembhavi, E. Kolve, R. Mottaghi, J. Salvador, D. Schwenk, E. VanderBilt, M. Wallingford, L. Weihs, M. Yatskar, and A. Farhadi, "Robothor: An open simulation-to-real embodied ai platform," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[52] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

Acknowledgments

These years of university have allowed me to understand many things about myself. Through the highs and lows, I've been fortunate to have the support of people who truly matter. I'm deeply grateful to my family for their unwavering encouragement and belief in me throughout this journey, even during the most challenging times. A thank you goes to Professor **Lamberto Ballan** for the inspiring course he teaches, the fascinating work he carries out with the entire VIMP group, and for giving me the opportunity to contribute despite the tight deadlines. I'm also truly thankful to **Filippo** for her support and guidance during the thesis work. And finally, my deepest gratitude goes to my wife, **Bahar**. Your love, patience, and constant presence have meant everything to me. Thank you for being by my side.