B.Comp. Dissertation

Improving and Extending Vision-based Navigation with Language-based Assistance

by

Liu Zechu

Department of Computer Science

School of Computing

National University of Singapore

2021/2022

B.Comp. Dissertation

Improving and Extending Vision-based Navigation with Language-based Assistance

by

Liu Zechu

Department of Computer Science

School of Computing

National University of Singapore

2021/2022

Project ID: H021450

Advisor: Dr. Terence Sim

Deliverable:

Report: 1 Volume

Abstract

This project is based on the original paper titled *Vision-based Navigation with Language-based Assistance via Imitation Learning with Indirect Intervention* (abbreviated as VNLA) by Nguyen et al. (2019). In VNLA, an agent with egocentric visual perception is guided via language to find objects in photorealistic indoor environments. A human requester gives the agent a goal with the natural-language format of either (1) "find [object] in [room]" or (2) "find [object]", starting from a random location in an environment. Afterwards, the agent needs to navigate inside the environment to locate the object. During navigation, the agent may request help from an advisor, who will provide natural-language answers to help the agent achieve the goal. Metrics such as success rate, room success rate, navigation error and the number of steps are computed to evaluate the performance of the agent. Our project comprises two lines of inquiry: (1) using information about object co-occurrence in general household environments to improve the performance of a VNLA agent, and (2) extending VNLA to handle tasks with multiple priorities, where the priority levels either are explicitly specified by the human requester or have to be implicitly inferred by the agent.

Original paper available at: https://arxiv.org/abs/1812.04155
Code available at: https://github.com/LiuZechu/vnla

Subject Descriptors:
I.2.6 Learning
I.2.7 Natural Language Processing
I.2.8 Problem Solving, Control Methods, and Search
I.2.10 Vision and Scene Understanding

Keywords:
Machine Learning, Artificial Intelligence, Embodied AI, Computer Vision, Natural Language Processing

Implementation Software and Hardware:
Ubuntu 16.04 LTS, Python 3.7.6, PyTorch 1.0.2, NVIDIA GTX 1080TI with CUDA 10.0

Acknowledgement

I would like to thank my project advisor Professor Terence Sim for guiding me through this project with his generous support and helpful advice. I would also like to thank Mr Pranavan Theivendiram for helping me with the project amidst his PhD works, and my group-mate Yehezkiel Raymundo Theodoroes for his support.

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

1.1 Background

As the field of artificial intelligence advances, we are seeing many machine learning models and techniques that achieve increasingly better results in specialised tasks such as navigation, natural language processing and computer vision. However, many of these techniques focus on one single "sense" or "capability" unlike a human being who can master the tasks of seeing, thinking, moving, and communicating all at the same time. In order to create agents that can see, move, speak, understand language and make intelligent decisions, the concept of "Embodied AI" is gaining popularity in the research community. An example Embodied AI problem mimics the following scenario: a human asks a robot: "Please get me the water bottle in my study room," and the robot needs to navigate to the correct object without having a map of the environment. In addition, the robot may ask questions (within a fixed budget) and receive natural-language help from a human advisor during the navigation process. There are existing works that try to formulate this problem and provide solutions, such as vision-based navigation with language-based assistance (VNLA) proposed by Nguyen et al. (2019).

1.2 Problem Statement

In the original VNLA paper, the goals given to an agent come in two formats: (1) "Find [object] in [room]" and (2) "Find [object]", where we substitute [object] with a noun representing a type of object and [room] with a room label in the environment. The guidance given by the advisor to the agent is in the form of fine-grained instructions, such as "turn 60 degrees right, go forward, turn left". Even with such detailed and fine-grained instructions, the task success rates[1] are only 52.26% in seen environments and 34.95% in unseen environments. Here, "seen environments" refer to those that the agent has been exposed to during training, albeit with different end-goals from the training phase. In contrast, "unseen environments" are totally new to the agent during testing.

The relatively low success rates and the limited formats of goals motivate our two lines of inquiry in this project. Firstly, an agent that succeeds only roughly half the time in seen environments, and significantly less than half the time in unseen environments, has limited usability in a real-world household environment. Thus, how can we devise techniques to improve the performance of the agent? How can we make the agent learn better during training so that it can generalise well to unseen environments? We notice that in a household, some objects tend to co-occur next to each other. For example, a bathtub is likely near a toilet, and a microwave oven is likely found near a refrigerator. In addition, some objects tend to appear more in a particular room type. For example, a sofa is likely inside a living room instead of a bedroom. Can we make use of such knowledge to inform the agent to make better decisions while completing its tasks? These questions motivate our **first line of inquiry**.

Secondly, the original VNLA agent can only handle a single target at a time. This constraint limits the possible set of tasks that the agent can perform. For example, the agent would be unable to execute the task "first find a lamp in the bedroom, then find an oven in the kitchen". This task involves understanding the relative priority between

[1]The task success rate is defined as the fraction of the test set on which the agent stops at a point within d metres from any of the goals, where d is the success radius.

the two targets/subtasks in the task, and navigating to the two targets sequentially. These skills are outside the capabilities of the original VNLA. But in a real-world scenario, the agent is likely required to perform a greater variety of tasks, such as tasks with multiple subtasks where different subtasks may have different priority levels assigned to them. How can we generalise and extend the agent to perform such composite tasks with multi-priority targets? Additionally, how can we enable the agent to assign priorities to subtasks when priorities are not explicitly given by the requester? In such scenarios, can the agent order the subtasks in such a way that it can complete them in the fewest steps possible? These questions motivate our **second line of inquiry**.

1.3 Our Solutions

1.3.1 Improving Performance with Object Co-occurrence

The **first line of inquiry** of this project aims to improve the performance of the VNLA agent in completing its tasks. These tasks are navigational in nature, but the agent does not have a map. Thus, navigation is based on a general understanding of the internal structure of a house, the common composition of a room and additional help from the advisor. One possible way to help the agent locate the goal better is through learning the co-occurrence of household objects. Hypothetically, this information can help in the following two ways.

Firstly, for a goal in the form of "find [object] in [room]", the agent may realise it is in the wrong room by looking at the objects in the room. It may then go out and explore other rooms. While exploring the house, the agent may get a partially obscured view of a room by glancing through a door. If the agent sees an object that tends to occur inside the same room as the target room, the agent can enter that room to locate the goal.

Secondly, for a goal in the form of "find [object]", the object co-occurrence information is even more useful. The agent can figure out in which room the object is more likely

to occur, so that the agent can move towards that room. In addition, while moving along the hallway, if the agent sees a partially obscured view of a room with objects that tend to co-occur with the target object, there is a higher likelihood that the target object is nearby. Thus, the agent can navigate to that room so that the target object may be more easily located.

With these rationales, we decide to explore how to represent object co-occurrence information and use this information to improve the performance of the VNLA agent. This line of inquiry will be elaborated in Chapter 3.

1.3.2 Tasks with Multi-priority Subtasks

The **second line of inquiry** of this project aims to create a more generalised version of VNLA and extend the types of tasks that the VNLA agent can perform. Specifically, we would like the agent to be able to perform a task with multiple subtasks where each subtask is assigned a priority. At the current stage, we have implemented an extension of VNLA where the agent can complete a task with two subtasks. One subtask has a higher priority than the other. The agent is required to navigate to a goal of the higher-priority subtask, before navigating to a goal of the lower-priority subtask. There are two types of multi-priority tasks that the agent can perform: explicit and implicit multi-priority tasks.

In an **explicit** multi-priority task, the priority levels of the subtasks are specified by the human requester (i.e., already given in the task instruction). These tasks come in the format of "First find [object1] in [room1], then find [object2] in [room2]." The subtask represented by the sub-instruction following "first" has a higher priority whereas the subtask represented by the sub-instruction following "then" has a lower priority.

In an **implicit** multi-priority task, the priority levels of the subtasks are **not** specified by the human requester (i.e., not explicitly given in the task instruction).These tasks come in the format of "Find [object1] in [room1]. Find [object2] in [room2]." Notably, the ordering of subtasks in the instruction does not indicate their relative priority levels. The

agent has to infer their priority levels based on their perceived distances to the starting location. The subtask with a nearer goal has a higher-priority whereas the subtask with a farther goal has a lower priority. To enforce the distance difference and to provide the agent with information on subtask priority, the goal of the nearer subtask is located in the same region/room as the agent's starting location, while the goal of the farther subtask is located in a different region/room.

After implementing this extension, we design new performance metrics and conduct experiments to evaluate how well the multi-priority agent can handle the different types of tasks. This line of inquiry will be elaborated in Chapter 4.

1.4 Contributions

The main contributions of this project are:

- We explore the effectiveness of using object co-occurrence information in improving the performance of VNLA.
- We propose a new generalised variant of VNLA and implement an extension of VNLA that can handle tasks comprising two subtasks with varying priorities, where the priority levels either are explicitly specified by the requester or have to be implicitly inferred by the agent. This extension takes VNLA a step closer towards real-world applications of more versatile mobile agents that can assist humans in locating multiple objects in a house.
- We demonstrate that the agent is able to reduce the number of steps taken in successful execution of implicit multi-priority tasks by ordering the subtasks based on their relative distances. This ability potentially enables mobile agents under resource constraints (e.g. battery level) to complete a series of tasks faster and more efficiently.

Chapter 2

Related Work

2.1 Embodied AI

There has been increasing popularity in the research space for "Embodied AI", which is the field for solving AI problems for virtual robots that can move, see, speak, and interact in the virtual world (Bermudez, 2021). The virtual world is usually a simulated environment which attempts to simulate the ambiguous, uncertain, disordered, and multi-modal nature of our real world. Facebook AI Research (FAIR), which has been spearheading several research projects in Embodied AI, describes the aim of advances in this field as seeking to "teach machines to understand and interact with the complexities of the physical world as people do" (Grauman & Batra, 2020).

Embodied AI may trace its inspiration from the fields of psychology and cognitive science. In 2005, Linda Smith and Michael Gasser put forth the embodiment hypothesis which states that "intelligence emerges in the interaction of an agent with an environment and as a result of sensorimotor activity" (Smith & Gasser, 2005). When applied to AI, this idea means that an Embodied AI agent needs to process sensory information in multiple modalities (such as vision, language, audio, and movement) and actively interact with the environment to gather information.

The paradigm of Embodied AI contrasts with that of "Internet AI", which has long

been the main focus of AI research. Internet AI learns from static Internet datasets such as ImageNet (Deng et al., 2009), COCO (Lin et al., 2014), and VQA (Agrawal et al., 2015) (Bermudez, 2021). Thanks to the advancement of deep learning techniques, increasing amounts of data and greater computing power, many Internet AI tasks have seen tremendous success, including image classification (Wortsman et al., 2022), image segmentation (Sultana et al., 2020), and machine translation (Edunov et al., 2018). However, Internet AI models passively take in static datasets and make predictions without the ability to interact in the real world like humans do. Embodied AI thus seeks to bring together fields such as computer vision, natural language processing, reinforcement learning, audio processing and robotics, so as to create agents that can solve complex problems within realistic simulated environments.

2.2 Embodied Question Answering

There are various tasks in the field of Embodied AI that are under active research. One such task is Embodied Question Answering (EmbodiedQA) (Das et al., 2018). In EmbodiedQA, an agent starts at a random location in a simulated 3D environment. It is asked a question such as "What colour is the car?". The agent then must intelligently navigate and explore the environment, gather information through egocentric vision and then give an answer such as "orange" (Fig. 2.2.1). Such an agent requires a range of AI skills including active perception, language understanding, goal-driven navigation, common-sense reasoning, and grounding of language into actions. The agent is trained via imitation learning and fine-tuned using reinforcement learning for the goal of answering questions. An important point to note is that EmbodiedQA uses House3D datasets for virtual interactive environments based on 3D indoor scenes from the SUNCG dataset (Song et al., 2017). SUNCG consists of *synthetic* 3D scenes with room and furniture layouts. However, these scenes are not real-world images and are usually uncluttered, which may be a poor representation of real-world environments.

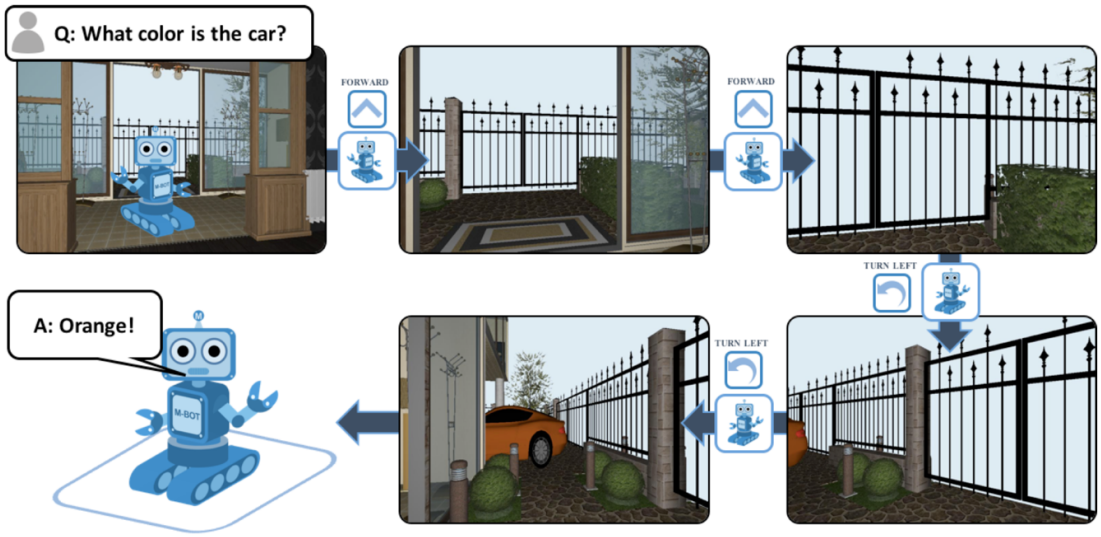


Figure 2.2.1: EmbodiedQA tasks agents with navigating rich 3D environments in order to answer questions. These embodied agents must jointly learn language understanding, visual reasoning, and navigation to succeed. (Das et al., 2018).

2.3 Vision-based Navigation with Language-based Assistance (VNLA)

As our work is mainly based on VNLA, we will describe it in greater detail in this section as compared to other related works. Nevertheless, this section only provides a summary sufficient for understanding this report. For full details, please refer to the original paper.

2.3.1 Overall Setup

Nguyen et al. (2019) present VNLA as a "grounded vision-language task where an agent with visual perception is guided via language to find objects in photo-realistic indoor environments". In this task formulation, there are three actors: requester, agent, and advisor. The requester gives a high-level end-goal in natural language, such as "find a sofa in the living room" or simply "find a sofa". The agent starts at a random location in

an indoor environment and needs to navigate to the goal. At any point during navigation, the agent may request help from the advisor. The advisor is assumed to have perfect knowledge of the environment and the agent's current state. Upon request, the advisor will provide the agent with accurate and fine-grained instructions on how to achieve the goal (Fig. 2.3.1). These instructions will then be prepended to the end-goal. Throughout the entire navigation process, the agent only knows the end-goal in the form of natural language, and the current egocentric vision through its top-mounted camera in the form of an image. These inputs are the only external information available to the agent. The agent does not have any extra information (for example, the agent has neither a map of the house nor its own location).

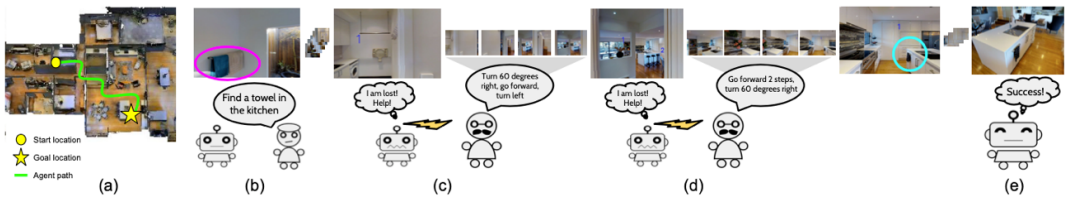


Figure 2.3.1: An example run of VNLA in an unseen environment. A video demo of the original VNLA is available at https://youtu.be/Vp6C29qTKQ0.

2.3.2 Agent Model

The entire architecture is an encoder-decoder model with a multiplicative attention mechanism and coverage modeling, which encodes an end-goal (a sequence of words) and decodes a sequence of actions. Both the encoder and decoder are LSTM-based recurrent neural networks.

At every time step while navigating to perform a task, the agent runs on two policies: the navigation policy π_{nav} and the help-requesting policy π_{ask}. π_{nav} generates the next navigational action to perform, while π_{ask} determines whether to ask for help. These two policies are computed by the neural architecture shown in Fig. 2.3.2. The model has two decoding passes at each time step: (a) the first decoding pass computes the tentative

navigation distribution, which is used as a feature for computing the help-requesting distribution. The help-requesting distribution outputs a binary decision of whether to request for help; (b) The second decoding pass computes the final navigation distribution. At the end of the second decoding pass, the agent chooses the argmax of the final navigation distribution as the next action to take.

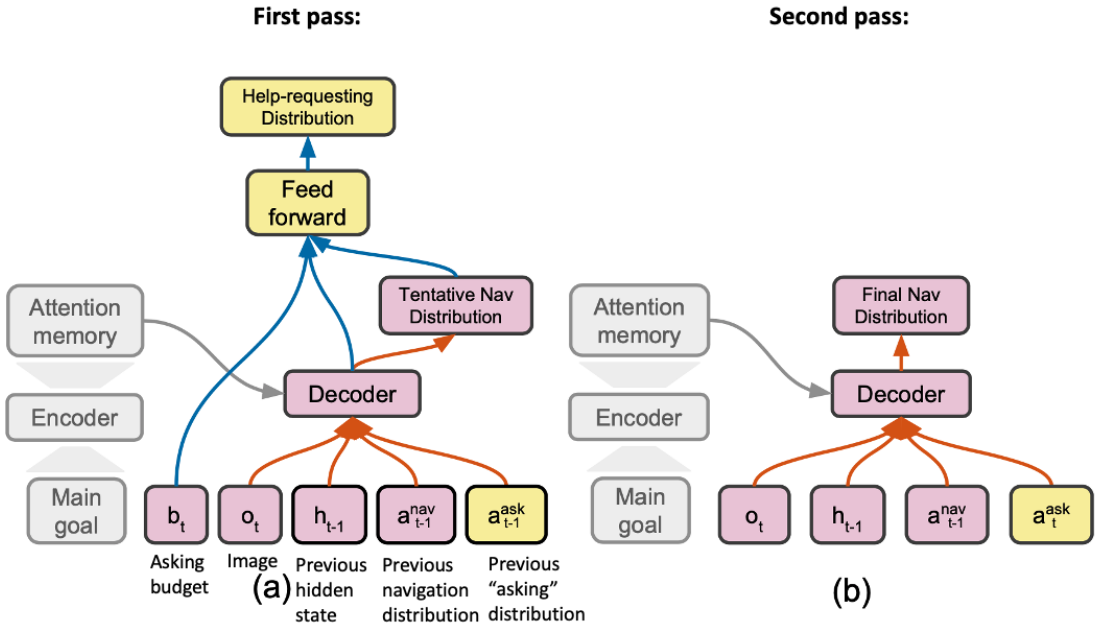


Figure 2.3.2: Original VNLA neural architecture from (Nguyen et al., 2019).

More specifically:

(1) At time step t, the encoded attention memory of the end-goal instruction, image feature of the current visual observation of the agent o_t, hidden state from the previous time-step h_{t-1}, previous time-step's navigation distribution a^{nav}_{t-1} and asking distribution a^{ask}_{t-1}, are all concatenated and fed into the decoder to predict a tentative navigation distribution.

(2) The output from the decoder, the embedding of the asking budge b_t and the tentative navigation distribution are concatenated and fed into a feedforward network to generate the help-requesting distribution, from which a new asking action will be sampled.

(3) If the agent chooses to ask a question at this point, the advisor will prepend the answer to the end-goal.

(4) Afterwards, the attention memory of the modified main goal is again concatenated with the image feature o_t, hidden state from the previous time-step h_{t-1}, previous time-step's navigation distribution a_{t-1}^{nav} and *current* time-step's asking distribution a_t^{ask}, and then fed into the decoder to predict the final navigation distribution.
(5) Finally, the agent will sample an action from this distribution and act in the environment.

2.3.3 Oracles

Oracles have full knowledge of the environment as well as the agent's current state. In the VNLA programme, the navigation teacher and the advisor are implemented as a **navigation oracle**, while the help-requesting teacher is implemented as an **asking oracle**. During training, the learned policies (π_{nav} and π_{ask}) are updated according to the actions outputted by the navigation teacher (π_{nav}^*) and the help-requesting teacher (π_{ask}^*). During testing, the navigation and help-requesting teachers are no longer present. The agent outputs its navigation and help-requesting actions based on the learned policies π_{nav} and π_{ask}. In addition, the agent can request for help from an advisor during both training and testing. However, it should be noted that, although the oracles' actions are treated as optimal in this work, they may not be the true optimal in reality since the oracles are heuristics-driven.

Navigation oracle. The navigation oracle serves as both the navigation teacher (during training) and the advisor (during training and testing). This oracle generates the next k consecutive actions by choosing actions along the shortest path from the agent's current viewpoint to the nearest goal viewpoint (Note that there can be multiple possible goal viewpoints; reaching any one of them is considered a success). It issues a `stop` action when the agent reaches one of the goal viewpoints. When the oracle serves as the advisor, it maps the actions {`left, right, up, down, forward, stop`} to natural-language phrases { "turn left", "turn right", "look up", "look down", "go forward", "stop" } respec-

tively. Repeated actions are aggregated and finally, action phrases are joined by commas to form subgoals to be prepended to the agent's end-goal instruction.

Asking oracle. The asking oracle serves as the help-requesting teacher during training. It uses a set of heuristics to determine whether the agent should request for help:
(a) The agent deviates from the shortest path by more than δ metres.
(b) The agent is "confused", defined as when the difference between the entropy of the uniform distribution and the entropy of the agent's tentative navigation distribution is smaller than a threshold ϵ.
(c) The agent has remained at the same viewpoint for the last μ steps.
(d) The help-requesting budget is greater than or equal to the number of remaining steps.
(e) The agent is at a goal viewpoint, but the highest-probability action of the tentative navigation distribution is `forward`.

2.3.4 Imitation Learning with Indirect Intervention

VNLA uses Imitation Learning with Indirect Intervention (I3L). The agent is trained via imitation learning, where the agent imitates the optimal actions of the navigation and help-requesting teachers. The agent's learned policies are updated according to the loss functions below, at every time step t:

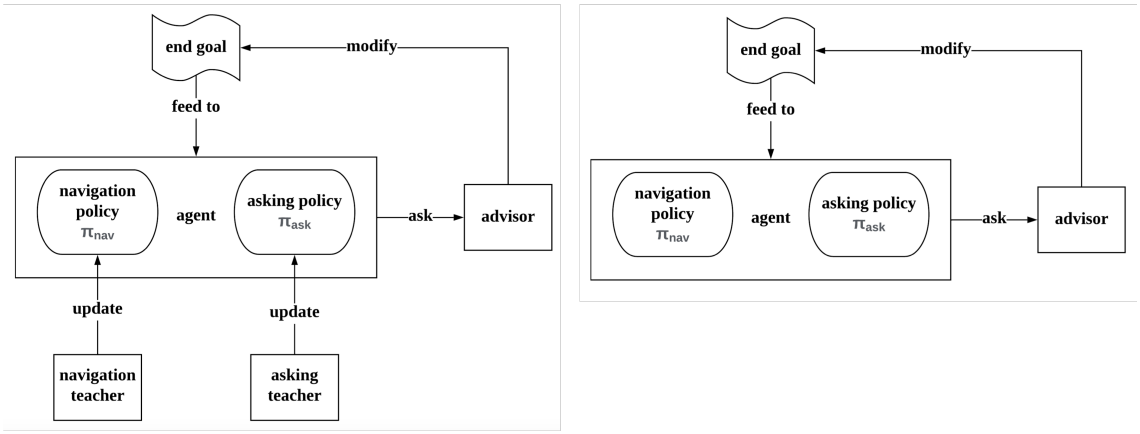
$$L_t^{nav} = \text{cross\_entropy}(p_{t,final}^{nav}, a_{nav,t}^*) \tag{2.1}$$

$$L_t^{ask} = \text{cross\_entropy}(p_t^{ask}, a_{ask,t}^*) \tag{2.2}$$

where $p_{t,final}^{nav}$ is the final navigation distribution at the end of the second pass, p_t^{ask} is the asking distribution, $a_{nav,t}^*$ is the optimal navigation action, and $a_{ask,t}^*$ is the optimal help-requesting action.

However, I3L differs from conventional imitation learning in two ways. Firstly, an advisor with complete knowledge of the environment can assist the agent both during

training and testing. Secondly, I3L uses indirection intervention to assist the agent during testing. Indirect intervention means that, instead of overriding the agent's actions, the advisor changes the environment to influence the agent's behaviour by prepending help instructions to the end-goal of the agent. Thus, the agent must learn how to interpret this intervention in order to assist its navigation. It should be noted that the intervention is indirect during testing; however, during training, the agent's decisions are overridden by the optimal actions from the navigation teacher if the agent asks for help in the previous k steps. This teacher-forcing only takes place during training, and its purpose is so that the agent never deviates from the trajectory suggested by the intervention and thus never encounters conflicts between the teacher and the advisor. In short, the navigation oracle intervenes both directly (by overriding actions) and indirectly (by modifying the environment) during training, but intervenes only indirectly during testing. Fig. 2.3.3 illustrates the interactions between the various modules during training and testing.



Training Testing

Figure 2.3.3: Interactions among VNLA model components during training and testing. Figure courtesy of (Han, 2020).

2.3.5 Differences from EmbodiedQA

VNLA differs from EmbodiedQA (Section 2.2) in three significant ways. Firstly, VNLA uses the Matterport3D dataset, which contains 10,800 panoramic views inside 90 real building-scale scenes constructed from 194,400 RGB-D images (Chang et al., 2017). Each scene is a residential building consisting of multiple rooms and floor levels, and the images are photographs taken in the real world (Fig. 2.3.4). Compared to the synthetic scenes in EmbodiedQA, Matterport3D is more realistic and thus more challenging for the task, since the scenes have more clutter, more diverse objects, and more complex environments. Secondly, a VNLA agent can ask for and receive help during navigation, whereas an EmbodiedQA agent cannot do so. Thirdly, VNLA frames the goal as a statement like "Find [object] in [room]", while EmbodiedQA frames the goal as a question like "What colour is [object]?".

Figure 2.3.4: An example view in an environment from the Matterport3D dataset.

2.4 Vision-Language Navigation with QA Interactions

As an extension of the original VNLA paper by Nguyen et al. (2019), Han (2020) expands the types of questions that the agent can ask the advisor during navigation. He also considers the advisor's help in the original work (e.g. "turn 60 degrees right, go forward 2 steps, turn left") as "too direct and over-detailed". Han's work modifies this interaction into Q&A dialogues. The agent can ask one of the pre-defined questions, and the advisor only gives simple and short answers (Fig. 2.4.1). This modification makes the tasks more difficult than the original VNLA, as it is much harder for the agent to utilize the short binary answers compared to detailed instructions. In addition, the agent not only needs to decide *whether* to ask given the limited "asking budget", but also *which* question to ask.

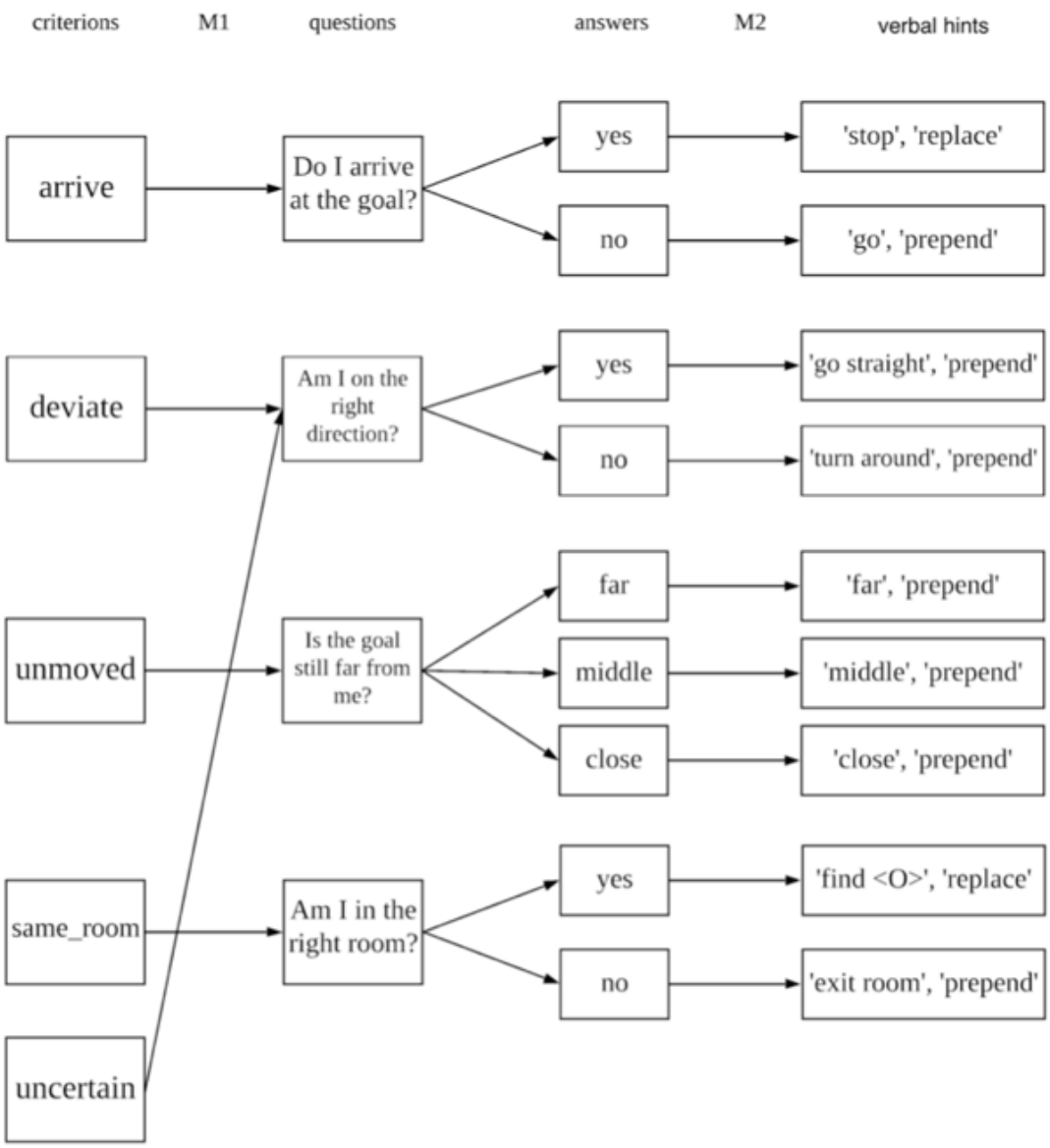


Figure 2.4.1: Mapping from possible questions that the agent can ask to the corresponding answers from the advisor in (Han, 2020).

Chapter 3

VNLA with Object Co-occurrence

3.1 Overview

This chapter details the first line of inquiry in this project: using information about object co-occurrence in general household environments to improve the performance of a VNLA agent. The subsequent sections in this chapter will cover related work on object co-occurrence in other computer vision applications, our methodology of inquiry, experiments and their results.

3.2 Related Work on Object Co-occurrence

Object co-occurrence information has been used to solve computer vision tasks. Rabinovich et al. (2007) incorporate semantic object context as a post-processing step to reduce ambiguity in visual object categorisation. Semantic object context can be learned from training data. This information is represented as context matrices, which are "symmetric, non-negative matrices that contain the co-occurrence frequency among object labels in the training set of the database" (Fig. 3.2.1). In this matrix, an entry ij represents the number of times that an object with label i appears in the same training image with an object with label j. A diagonal entry corresponds to the frequency of each object in the

training set. This context matrix is then processed mathematically and used in a conditional random field (CRF) formulation to maximize contextual agreement of the objects' labels.

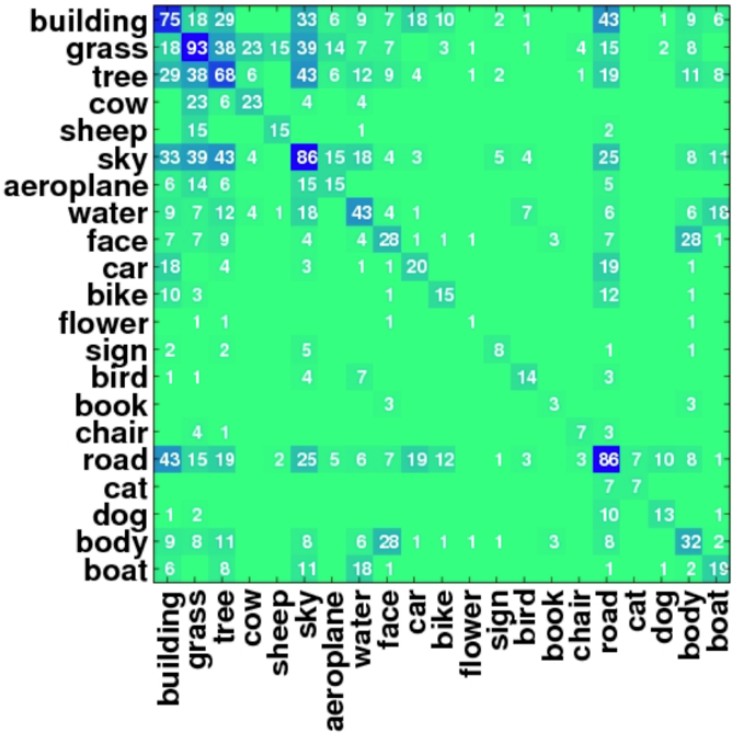


Figure 3.2.1: Context matrix constructed from object co-occurrence counts (Rabinovich et al., 2007).

In a similar vein, Bengio et al. (2013) explore the incorporation of contextual information to improve object recognition and localization. However, unlike Rabinovich et al. (2007) who aim to learn object co-occurrence information from image training datasets, Bengio et al. (2013) use co-occurrence statistics from web documents. By merely counting the number of times nouns co-occur in web documents, they obtain a good estimate of expected co-occurrences in visual data. They then cast the problem of combining textual co-occurrence statistics with the predictions of image-based classifiers as an optimization problem. As a result, there are significant improvements in recognition and localization rates for both ImageNet Detection 2012 and Sun 2012 datasets.

Object co-occurrence information in (Bengio et al., 2013) is represented as a symmetric matrix where each entry represents the number of times that two object labels co-occur. Afterwards, estimates for the point-wise mutual information are computed using

$$s_{i,j} = log(\frac{p\,(i,\,j)}{p\,(i)\,p\,(j)}) \tag{3.1}$$

where $p(i, j)$ and $p(i)$ are the counts normalised as probabilities. Finally, the scores are normalised using the logit function:

$$S_{i,j} = \begin{cases} \frac{1}{1+exp(-s_{i,j})}, \text{if} s_{i,j} > 0 \\ 0, \text{otherwise} \end{cases} \tag{3.2}$$

This matrix (Fig. 3.2.2) is then used in the downstream task of object recognition and localisation with an image classifier.

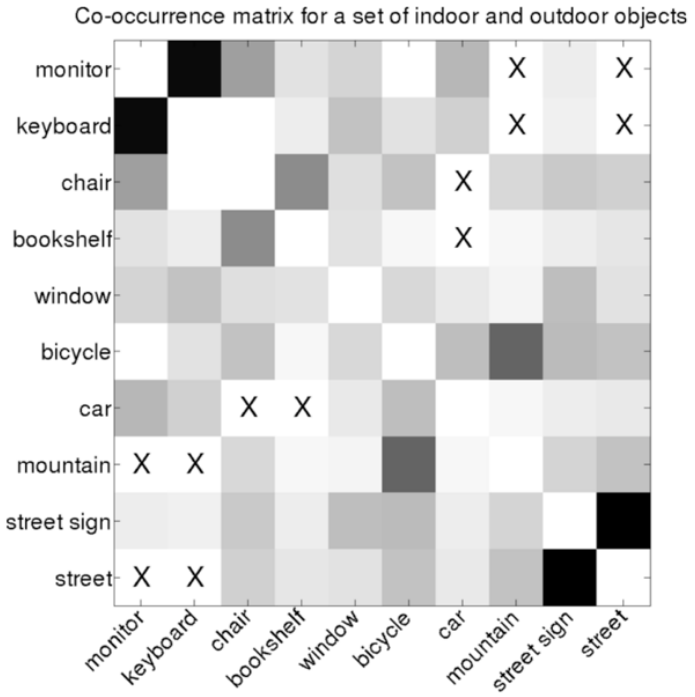


Figure 3.2.2: Graphical illustration of the co-occurrence scores for 10 object classes in (Bengio et al., 2013). Dark values show high correlation; crosses mean negative correlation; the diagonal is not used.

3.3 Methodology

3.3.1 Object Co-occurrence Counts

To incorporate information about co-occurrence of household objects in the decision-making process of the VNLA agent, we first construct a co-occurrence matrix similar to that in (Rabinovich et al., 2007) and (Bengio et al., 2013) (Section 3.2), in which each entry ij counts the number of times an object with label i co-occurs with an object with label j. We ignore the diagonal entries as we are more interested in how objects co-occur with different objects. There may be several ways to construct this matrix, so long as the information can generalise well about an unseen indoor environment (e.g. we should expect the co-occurrence count of toilet-sink to be greater than that of toilet-oven). In

our implementation, we use co-occurrence statistics generated from scenes in our seen dataset, which will be elaborated next.

VNLA splits the 90 environments in Matterport3D into 61 training, 11 validation and 18 test environments. Among the 61 training environments, each environment/house has a meta-data file named `xxx.house` (where `xxx` is the house ID). Each `xxx.house` file contains a manually specified decomposition of a house into levels, rooms, and objects with semantic labels. The file also contains information about the objects and object types found inside every room. We parse through each of these 61 meta-data files in the training set and consider two object types to co-occur once if they are found within the same room in an environment. Note that we consider the object types rather than specific object instances. Even if there are 10 pictures and 5 tables in the same room in the same environment, we only add 1 to the co-occurrence count of picture-table (and symmetrically, table-picture). In addition, we only consider 289 object types which correspond to the set of object labels used by VNLA to generate its goals.

3.3.2 Conditional Occurrence Matrix (COM)

After generating the co-occurrence matrix, we try two methods to normalise the matrix to produce a conditional occurrence matrix (COM). The first method involves normalising each row so that entries on the same row (excluding the diagonal entry) add up to one. For the object type that each row represents, the entries indicate the conditional probabilities of locating other object types near the said object type. The second method uses the point-wise mutual information normalised with logit function as described in (Bengio et al., 2013) (Section 3.2), producing a symmetric matrix where entries range from 0 to 1 (Fig. 3.3.1). In both methods, diagonal entries default to one. Since the second method is shown to perform better than the first in the experiments, we use the second method when presenting our experiment results.

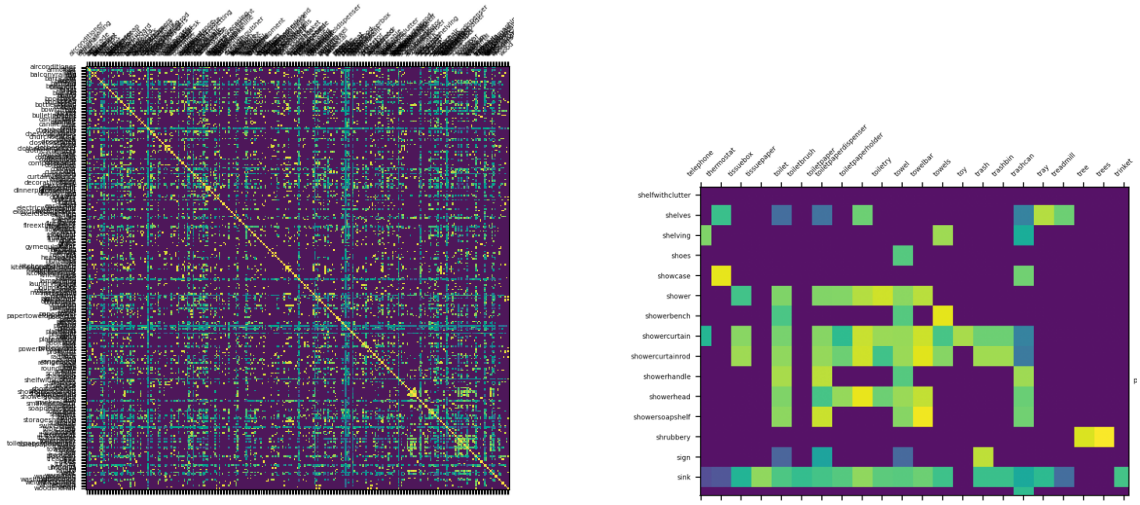


Figure 3.3.1: The left illustration shows the whole COM normalised with the second method, where yellow indicates a higher number (greater degree of co-occurrence). The right illustration shows a zoomed-in version on a small part of the matrix.

3.3.3 Variants of COM

In addition to the above COM, we also generate two other variants: (1) COM with room-object co-occurrence (COM-RO), and (2) COM with word similarity score (COM-WS).

COM-RO. COM-RO treats a room as just an ordinary object. A room and an object are considered to co-occur once if the object appears in the room type in a house. A room does not co-occur with another room of a different type.

COM-WS. COM-WS is inspired by (Bengio et al., 2013) to use external information to inform the agent of general object co-occurrence. Bengio et al. (2013) use word co-occurrence statistics from web text to construct the co-occurrence matrix. However, it is challenging to find word co-occurrence statistics for the specific words in VNLA. Thus, we use the cosine similarity score of word embeddings as an approximation. In this implementation, we use GloVe (Pennington et al., 2014), whose training is performed on aggregated global word-word co-occurrence statistics from a corpus. Other types of word embeddings are possible too. The assumption is that word embeddings with a high similarity score likely mean that the words appear in similar contexts. Thus, the objects

that these words represent may tend to co-occur in a general household environment. In our experiments, COM-WS uses `GloVe-wiki-gigaword-50` word embeddings. When constructing COM-WS, the co-occurrence counts of object pairs are replaced with the word similarity score between their object labels.

Both COM-RO and COM-WS are normalised by the point-wise mutual information method described in the previous paragraph.

3.3.4 Integrating COM into VNLA Model

To utilise the COM to help the agent in navigational decision-making, we modify the model used in the original VNLA agent as shown in Fig. 3.3.2. The green regions are modules added by us whereas the non-green regions are the original modules. The model of the original VNLA has been described in Subsection 2.3.2.

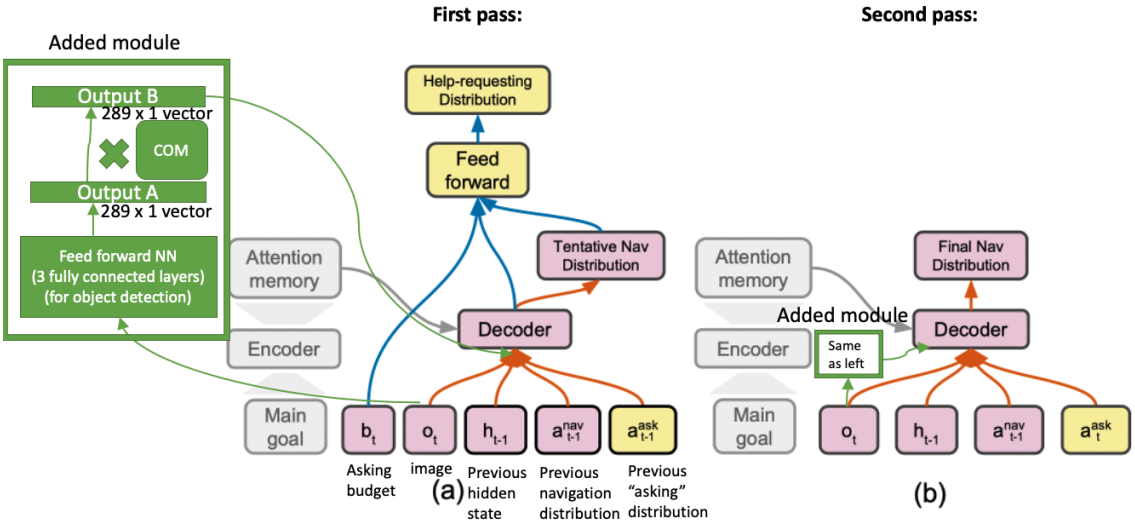


Figure 3.3.2: Our modification on top of the original VNLA agent model from (Nguyen et al., 2019).

To incorporate the COM, we first pass the image features into a feedforward neural network with three fully connected layers for object detection. The output (Output A) of this network has 289 dimensions, representing the objects that the agent currently sees.

When multiplied with the COM, it produces Output B which represents the probabilities of objects that are currently nearby. This vector is then fed into the decoder to be used in subsequent decision-making. The same module is added for both the first and second decoding passes (simplified as the green box marked "Same as left" in Fig. 3.3.2). Note that the entries in COM are fixed values learned from the training environments (as described earlier), which are not tuned by training.

3.4 Experiments

3.4.1 Setup

We train the modified model with the COM (and its variants) on both the original VNLA (Nguyen et al., 2019) and Han's version (Han, 2020), and then evaluate the performance metrics against the unmodified models (without incorporating the COM). All hyperparameters used are the same as the original VNLA.

All experiments are conducted on a single NVIDIA GTX 1080Ti with CUDA 10.0 in Ubuntu 16.04 LTS. Code is written in Python 3.7.6 and PyTorch 1.0.2.

To evaluate the navigational performance of the agent, the following metrics in Table 3.4.1 are used. The main metric that we focus on is the success rate. A task is considered successful if the agent stops at a location within d metres from any of the possible goal viewpoints. Here, d is the success radius, a task-specific hyperparameter. In both the original work and this project, d is set at 2 metres. Also, the agent has to complete a task within the assigned time budget. All tasks are designed such that they can be completed within their respective time budgets.

Metric Name	Description
Success rate	Fraction of test set on which the agent stops at a point within d metres from any of the goals
Oracle success rate	Fraction of test set on which any point on the agent's trajectory falls within d metres from any of the goals
Room success rate	Fraction of test set on which the agent stops at any room containing a goal
Navigation error	Average distance from the point where the agent stops to its closest goal
Length	Average distance between where the agent starts and stops
Steps	Average number of steps the agent takes on each trajectory

Table 3.4.1: Navigational metrics to evaluate performance, reproduced from (Han, 2020).

3.4.2 Results and Discussion

The results from our experiments are displayed in the tables below. We evaluate the agent with five different random seeds and report the mean of each metric. Tables 3.4.2 and 3.4.3 are for the first task type "find [object] in [room]", comparing COM with the baselines (original VNLA and Han's modified version). Tables 3.4.4 and 3.4.5 are for the second task type "find [object]", comparing COM with the baselines as well. Tables 3.4.6 and 3.4.7 present the results comparing the COM variants (COM-RO and COM-WS) with the baselines for the first task type "find [object] in [room]". Note that experiments involving the first task type are run on the AskNav dataset (tasks *with* room labels) whereas experiments involving the second task type are run on the NoRoom dataset (tasks *without* room labels).

Model	**Success Rate** (%)	Oracle Success Rate (%)	Room Success Rate (%)	Navigation Error (m)	Length (m)	Steps
Seen						
Original VNLA	**51.45**	59.94	64.87	3.48	11.43	14.13
With COM	**51.03**	59.70	64.57	3.54	11.65	14.19
Unseen						
Original VNLA	**34.23**	42.30	43.88	5.58	9.67	14.65
With COM	**34.85**	41.09	45.62	5.55	9.50	14.01

Table 3.4.2: Results for the original VNLA versus our modified model, for tasks in the form of "find [object] in [room]".

Model	**Success Rate** (%)	Oracle Success Rate (%)	Room Success Rate (%)	Navigation Error (m)	Length (m)	Steps
Seen						
Han's	**42.27**	51.87	58.76	4.40	11.32	15.31
With COM	**43.43**	52.06	61.34	4.35	11.42	15.14
Unseen						
Han's	**21.02**	27.60	29.87	7.44	8.01	15.79
With COM	**21.67**	28.48	30.32	7.34	7.82	15.79

Table 3.4.3: Results for Han's version versus our modified model, for tasks in the form of "find [object] in [room]".

COM for first task type. We want to investigate whether using object co-occurrence information in the form of the COM can improve the agent's performance for the task type of "find [object] in [room]". As seen from Table 3.4.2, our modification on the original VNLA leads to a slight drop in performance across all the metrics in seen environments, but it performs slightly better in unseen environments. From Table 3.4.3, our modification

on Han's version results in a slight increase in performance across all the metrics in both seen and unseen environments. The most significant improvement is shown in the room success rate in seen environments, where it increases from 58.76% to 61.34%. However, the amounts of improvement in other metrics in both seen and unseen environments are not significant.

Model	**Success Rate (%)**	Oracle Success Rate (%)	Navigation Error (m)	Length (m)	Steps
Seen					
Original VNLA	**53.15**	62.11	3.58	11.51	14.72
With COM	**53.37**	60.78	3.56	11.34	14.74
Unseen					
Original VNLA	**44.80**	52.82	4.44	9.76	14.73
With COM	**44.42**	52.52	4.58	9.82	15.09

Table 3.4.4: Results for the original VNLA versus our modified model, for tasks in the form of "find [object]".

Model	**Success Rate** (%)	Oracle Success Rate (%)	Navigation Error (m)	Length (m)	Steps
Seen					
Han's	**40.03**	50.46	4.87	11.05	16.81
With COM	**38.60**	49.47	4.89	10.92	16.89
Unseen					
Han's	**25.93**	35.10	6.71	8.76	17.42
With COM	**26.56**	35.24	6.39	8.18	17.35

Table 3.4.5: Results for Han's version versus our modified model, for tasks in the form of "find [object]".

COM for second task type. We want to investigate whether using object co-occurrence information in the form of the COM can improve the agent's performance for the task type of "find [object]". Results show that our modification generally has similar levels of performance as the unmodified original VNLA in both seen and unseen environments as seen from Table 3.4.4. From Table 3.4.5, our modified model leads to worse performance metrics in seen environments but marginal improvements in unseen environments.

Model	**Success Rate** (%)	Oracle Success Rate (%)	Room Success Rate (%)	Navigation Error (m)	Length (m)	Steps
Seen						
Original VNLA	**51.45**	59.94	64.87	3.48	11.43	14.13
With COM-RO	**53.22**	60.10	66.00	3.35	11.35	14.19
With COM-WS	**52.82**	59.50	64.54	3.52	11.48	14.18
Unseen						
Original VNLA	**34.23**	42.30	43.88	5.58	9.67	14.65
With COM-RO	**33.75**	39.88	43.82	5.61	9.17	14.33
With COM-WS	**34.29**	41.72	42.89	5.67	9.72	14.56

Table 3.4.6: Results for the original VNLA versus our modified model with COM-RO and COM-WS, for tasks in the form of "find [object] in [room]".

Model	**Success Rate (%)**	Oracle Success Rate (%)	Navigation Error (m)	Length (m)	Steps
Seen					
Original VNLA	**53.15**	62.11	3.58	11.51	14.72
With COM-RO	**54.28**	62.29	3.53	11.47	15.00
With COM-WS	**52.81**	59.89	3.66	11.22	14.91
Unseen					
Original VNLA	**44.80**	52.82	4.44	9.76	14.73
With COM-RO	**42.61**	49.57	4.70	9.18	15.56
With COM-WS	**42.70**	51.09	4.73	9.89	14.77

Table 3.4.7: Results for the original VNLA versus our modified model with COM-RO and COM-WS, for tasks in the form of "find [object]".

COM variants. To further explore different methods of utilising object co-occurrence information, we also conduct experiments on the two COM variants we design: COM-RO (added with room-object co-occurrence) and COM-WS (with word similarity between object labels). These experiments aim to find out whether using these COM variants can improve the agent's performance for both types of task descriptions (Tables 3.4.6 and 3.4.7). We hypothesise that COM-RO can tell the agent in which room the target object is likely to appear, thereby helping the agent with the second task type ("find [object]") where the room label is unavailable. To the contrary, the agent fitted with COM-RO shows similar trends across both types of tasks, where the agent performs better in seen environments but worse in unseen environments. For COM-WS, we expect that using external sources of information can help the agent achieve better results than relying on information that the agent might have already learned. Although COM-RO marginally

improves the agent's performance for the first task type, it results in worse performance for the second task type where room labels are absent. Overall, the experiments with the variants COM-RO and COM-WS do not show any significant improvement in the performance metrics, for both types of questions (with or without room labels).

Possible reasons for ineffectiveness. There are several reasons for the general lack of significant improvement or even decrease in performance of our modified model. Firstly, the entire model is trained end-to-end, so there is no guarantee that the semantic meanings of "Output A" and "Output B" in Fig. 3.3.2 conform to our intention. The placement and design of the newly added module may not work well in utilising the information in the COM. Secondly, information about object co-occurrence may not help with the navigational task. Even if the agent knows that "the oven tends to appear near the fridge", it cannot utilise this knowledge when it is in a different room and trying to find a path to the kitchen. It can only utilise this knowledge when it gets sufficiently close to the kitchen and sees an oven. The extent of help provided by the COM is limited. Thirdly, even if object co-occurrence information can be of some help, the agent might have already learned this knowledge implicitly during training. The information in the COM is directly derived from the training environments to which the agent has been exposed, so it may not give the agent any new insights to complete its tasks better. Therefore, the COM and the COM-RO do not provide much additional help since the agent may be already using this knowledge. Although COM-WS aims to solve this issue by bringing in external knowledge, using similarity scores of word embeddings may not be suitable. Similarity scores may reflect synonyms but not contextual proximity.

3.5 Conclusion

From our discussion above, we conclude that our current methods of incorporating object co-occurrence information does not result in significant improvement in the agent's performance metrics across both types of tasks in seen and unseen environments. Never-

theless, our experiment results offer insights into how object co-occurrence information can affect the agent's performance. Several instances of performance improvement in the experiment results can serve as further points of inquiry. For future work, we can explore more methods of designing the COM and incorporating it effectively into the network so as to improve the performance of VNLA.

Chapter 4

Multi-priority VNLA

4.1 Overview

This chapter details the second line of inquiry in this project: extending VNLA to handle tasks with multiple priorities, where the priority levels either are explicitly specified by the human requester or have to be inferred by the agent. The subsequent sections in this chapter will cover related work on multi-target navigation, our methodology of inquiry, experiments and their results.

4.2 Related Work on Multi-target Navigation

There is little existing literature dealing with egocentric navigation problems or embodied question answering problems where the tasks have multiple priority levels. Since multi-priority tasks have more than one target/goal, a closely related work is Multi-Target Embodied Question Answering (MT-EQA). Yu et al. (2019) present MT-EQA as a generalisation of EmbodiedQA (Das et al., 2018) (Note that EmbodiedQA is introduced in Section 2.2 of this report). Unlike EmbodiedQA where the agent is limited to answering a query about only **one** target, MT-EQA handles questions that have **multiple** targets in them, such as "Is the dresser in the bedroom bigger than the oven in the kitchen?", where

the agent has to navigate to multiple locations ("dresser in bedroom", "oven in kitchen") and perform comparative reasoning ("dresser" bigger than "oven") before it can answer a question (Fig. 4.2.1).

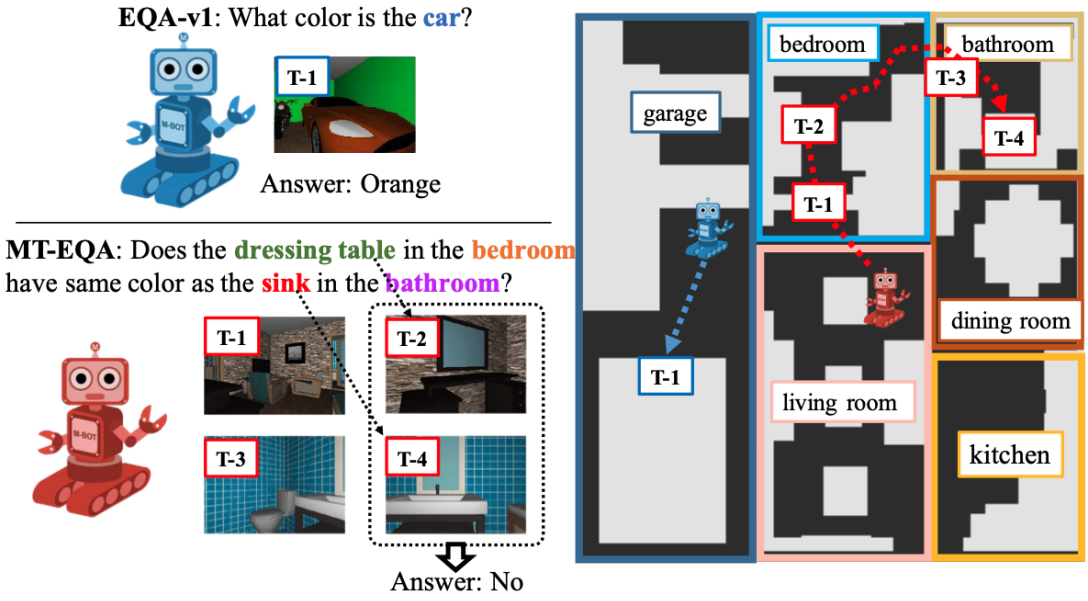


Figure 4.2.1: An illustration of the differences between EmbodiedQA and MT-EQA. While an EmbodiedQA agent only needs to navigate to a single target "car" since its question only involves one target, MT-EQA's question requires the agent to navigate to multiple targets (e.g., bedroom, dressing table, bathroom, sink). In addition, the MT-EQA agent needs to perform attribute comparison between multiple targets (e.g., colour difference between dressing table and sink). (Yu et al., 2019).

To facilitate the more complex nature of the tasks, Yu et al. (2019) propose a new modular architecture composed of a program generator, a controller, a navigator, and a VQA module. The program generator converts the given question into sequential executable sub-programs; the navigator guides the agent to multiple locations pertinent to the navigation-related sub-programs; and the controller learns to select relevant observations along its path. These observations are then fed to the VQA module to predict the answer. Also, like the original EmbodiedQA, MT-EQA uses House3D datasets for virtual interactive environments based on 3D indoor scenes from the SUNCG dataset (Song et al., 2017). SUNCG consists of *synthetic* 3D scenes which may be a poor representation

of real-world environments.

Importantly, MT-EQA differs from our work on multi-priority VNLA. In MT-EQA's tasks, the targets do not have any inherent priority attached to them. They can be visited in any order. In contrast, tasks in multi-priority VNLA consist of multiple subtasks which have either explicit or implicit priority levels. When performing each task, the agent is required to first visit the higher-priority subtask before reaching the lower-priority subtask.

4.3 Methodology

4.3.1 Types of Multi-priority Tasks

Currently, our implementation supports a multi-priority task with two subtasks. There are two formats of multi-priority VNLA tasks: (1) **explicit** multi-priority tasks, and (2) **implicit** multi-priority tasks.

An **explicit** multi-priority task consists of two subtasks. Each subtask has a priority level **already specified** by the requester in the instruction. For such a task, the instruction is in the format "First find [object1] in [room1], then find [object2] in [room2]". The higher-priority subtask requires the agent to navigate to [object1] located in [room1]. After completing the higher-priority subtask, the agent is required to complete the lower-priority subtask. The lower-priority subtask involves navigating to [object2] in [room2]. Fig. 4.3.1 depicts an example of an explicit multi-priority task.

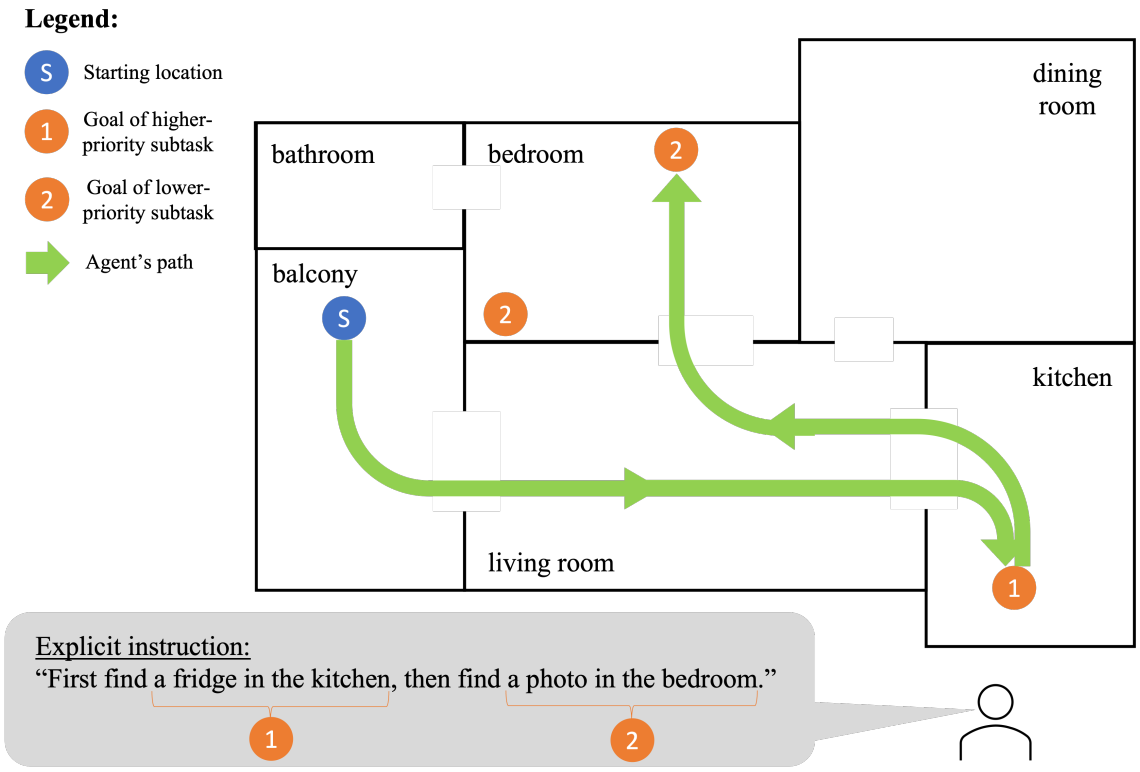


Figure 4.3.1: Illustration of an explicit multi-priority task, whose instruction clearly specifies the priority levels of the subtasks. Note that a subtask may have more than one goal viewpoints. In this example, there are two photos in the bedroom. Reaching any one of them is considered a success.

An **implicit** multi-priority task consists of two subtasks. Each subtask has a priority level based on the distance of its goal to the agent's starting location. However, unlike an explicit task, the instruction of an implicit multi-priority task **does not** contain explicitly-specified priority levels. For such a task, the instruction is in the format "Find [object1] in [room1]. Find [object2] in [room2]". There is a 0.5 probability that "Find [object1] in [room1]" is the higher-priority subtask and "Find [object2] in [room2]" is the lower-priority subtask, and a 0.5 probability otherwise. The higher-priority subtask is nearer to the agent's starting location whereas the lower-priority one is farther. The agent first has to infer the priority levels of the subtasks. Then, the agent is required to navigate to the goal of the higher-priority subtask before navigating to the goal of the lower-priority

subtask. In addition, the higher-priority subtask is located in the same room as the agent's starting location so as to ensure a shorter distance compared to the lower-priority subtask. Fig. 4.3.2 depicts an example of an implicit multi-priority task.

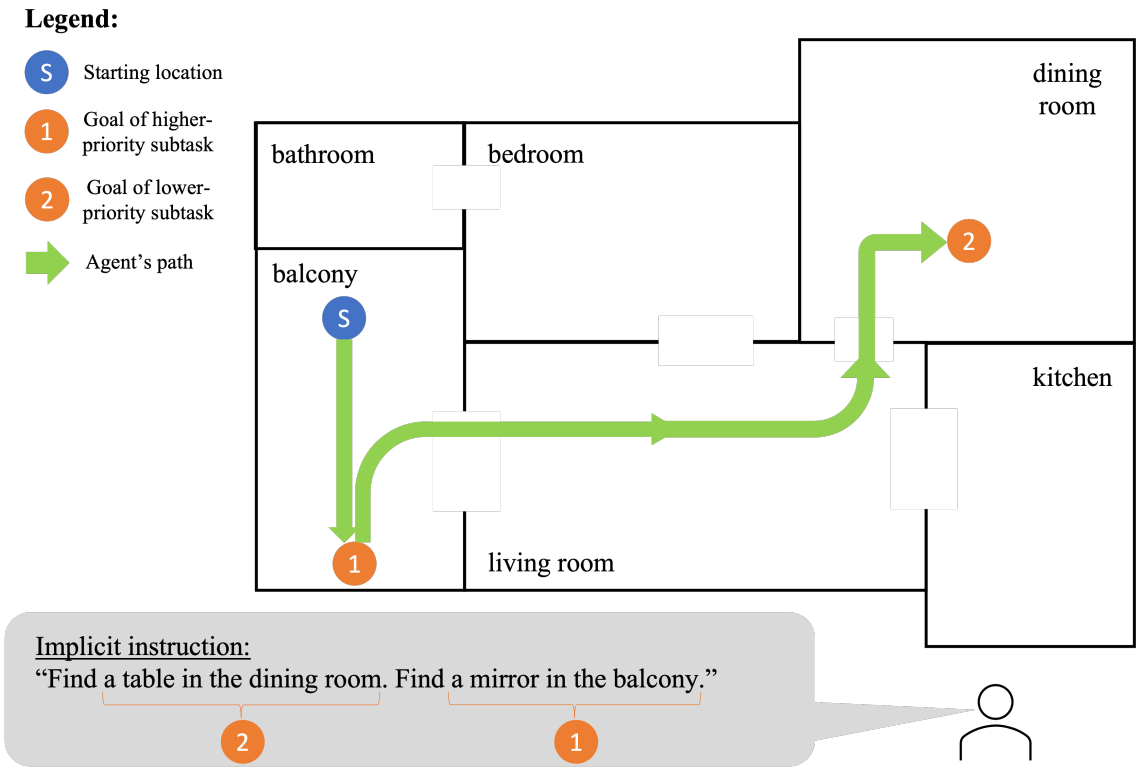


Figure 4.3.2: Illustration of an implicit multi-priority task. Note that the ordering of the subtasks in the task instruction is random, and thus does not indicate their priority levels.

4.3.2 Key Terms

For clarify of discussion in this chapter, the following key terms will be defined:

Task: A task is a data point in the datasets. Each task is a tuple (*house*, *start pose*, *first goal viewpoints*, *second goal viewpoints*, *instruction*). In our current implementation, a multi-priority task consists of two subtasks.

House: A house (also called "scan" or "environment" in the Matterport3D dataset, the original work on VNLA, as well as the code base) is a virtual house environment in which the agent will navigate to complete the given task. Each house consists of viewpoints that

form a connected graph.

Instruction: Each task comes with an instruction written in natural language (English). Each instruction contains two subtasks. An instruction can take the format of "First find [object1] in [room1], then find [object2] in [room2]" for an explicit multi-priority task, or "Find [object1] in [room1]. Find [object2] in [room2]" for an implicit multi-priority task.

Subtask: Each task consists of two subtasks. For example, if the task's intruction is "*First find a chair in the dining room, then find a coffee table in the familyroom*", then the first subtask requires the agent to navigate to a chair in the dining room. After reaching the goal in the first subtask, the agent needs to complete the second subtask. The second subtask requires the agent to navigate to a coffee table in the family room.

Priority: Priority indicates the relative order in which the subtasks should be completed. Each subtask is associated with a priority level, whether explicit or implicit. The agent needs to complete the subtask with a higher priority before the subtask with a lower priority.

Viewpoint: A viewpoint is a discrete location in a house. A house consists of a connected graph of many viewpoints. The agent can move from one viewpoint to another.

Goal: A goal is a viewpoint that the agent can navigate to in order to fulfil a subtask. Each subtask can have multiple goals since there may be multiple objects that fulfil the same subtask. For example, for the subtask "*Find a photo in the bedroom*", there are multiple goals if there are multiple photos in the bedroom. Navigating to any one of these goals is considered to fulfil the subtask.

Start pose: A start pose specifies the agent's starting location (a viewpoint in the house) and its initial orientation (the direction in which the agent's egocentric camera points).

First goal viewpoints: First goal viewpoints are a set of goals (or viewpoints) for the higher-priority subtask. Passing any of these viewpoints is considered to have fulfilled the higher-priority subtask.

Second goal viewpoints: Second goal viewpoints are a set of goals (or viewpoints) for the lower-priority subtask. Reaching any of these viewpoints is considered to have fulfilled

the lower-priority subtask.

4.3.3 Implementation

To generalise VNLA to handle a multi-priority task with two successive subtasks, we make the following modifications to the original VNLA code base. Note that the bulk of the modifications takes place in the **oracle** module, which serves as the navigation teacher during training and as the advisor during training and testing. All other unmentioned components remain the same as the original VNLA (described in detail in Section 2.3).

Instruction input. The instruction for each task is in the form of natural language English sentences. Each instruction takes the form of "First find [object1] in [room1], then find [object2] in [room2]" for explicit tasks and "Find [object1] in [room1]. Find [object2] in [room2]" for implicit tasks. Since the original VNLA can already take in textual input of varying lengths, there is no need to modify the component that takes in the textual instruction input. Like the original VNLA, each multi-priority task is first tokenised and then fed into an encoder. The words "first" and "then" are added to the vocabulary so that the tokeniser can recognise them without converting them into <UNK> tokens.

Oracle. The oracle module is responsible for both the navigation teacher (during training) as well as the advisor (during training and testing). At any time while the agent is performing a task, the oracle keeps a state of whether the agent has reached a goal in the first (higher-priority) subtask. It also keeps track of the current subtask that the agent is performing. At every step, the oracle checks whether the agent's current viewpoint is one of the goals of the first subtask. If so, the oracle changes the current subtask that it keeps track of to the second (lower-priority) subtask. Note that the agent is oblivious to the state of the oracle. During training, the oracle serves as the navigation teacher. The navigation teacher overrides the agent's actions so as to navigate to the correct goal when the agent requests for help. During both training and testing, the oracle serves as the advisor, which

provides the agent with help in the form of prepended subgoal instructions when requested by the agent. Every time when the oracle provides a subgoal instruction, the instruction contains a few fine-grained steps to reach the nearest goal in the current subtask. If the agent has yet to complete the first subtask, the oracle will direct the agent to go to the nearest goal in the first subtask. After fulfilling the first subtask, the oracle switches to instructing the agent to navigate towards the second subtask's goal. Just like the original VNLA, the only information that the agent can possibly receive from the advisor (oracle) during testing is the prepended instructions when the agent asks the advisor for help. Thus, the agent does not have direct knowledge of the states of which the oracle keeps track. The agent has to figure out which subtask it needs to perform currently and when to move on to the next subtask (with language assistance if it so requests).

Agent. The agent is trained to output a placeholder action of `(0, 0, 0)`, which corresponds to the agent action <`first_goal`>, when reaching a goal in the first (higher-priority) subtask.

Hyperparameters. Comparing with the original VNLA, we more than double the hyperparameter `max_episode_length` from 25 to 60. `max_episode_length` is the maximum number of actions that the agent can take when completing a task. This hyperparameter affects the time budget assigned to each task. Since all our tasks can be completed within 50 steps (see Section 4.4.2 on Data Generation), a maximum limit of 60 steps is appropriate, considering that the agent may not always take the shortest path.

Data generation. Since the task formats are now different from the original VNLA, completely new datasets need to be generated to suit the modifications. Please refer to Section 4.4.2 for the detailed data generation processes.

4.4 Experiments

4.4.1 Setup

We train the multi-priority VNLA agent on the newly generated training datasets for explicit and implicit multi-priority tasks respectively. We then design and evaluate separate sets of performance metrics for the two types of multi-priority tasks.

All experiments are conducted on a single NVIDIA GTX 1080Ti with CUDA 10.0 in Ubuntu 16.04 LTS. Code is written in Python 3.7.6 and PyTorch 1.0.2. A full training session takes around 48 hours to run on our machine.

4.4.2 Data Generation

Our datasets with multi-priority tasks are generated based on the original VNLA datasets. In this subsection, we will first summarise the data generation process of the original VNLA (Nguyen et al., 2019), before detailing how we generate the datasets for explicit and implicit multi-priority tasks respectively.

Original VNLA dataset. After processing the annotations from the Matterport3D dataset, there are 289 object labels and 26 room labels. Each data point represents a single task, and is defined as a tuple (*house*, *start pose*, *goal viewpoints*, *instruction*). An instruction has the format "Find [object] in [room]". [object] is preceded with "a/an" if singular. [room] is replaced with "the [room label]" is there is only one such room in the house, or "one of the `pluralize`([room label])" if there are multiple rooms with the same room label in the house. There are five datasets: train, validation seen, validation unseen, test seen, and test unseen. The seen sets contain data points that are generated in the same houses from the training set but these data points do not appear in the training set. The unseen sets contain data points generated in the validation houses or test houses.

Multi-priority datasets. The data generation processes for explicit and implicit multi-priority tasks differ only in start pose selection, instruction formulation and validity

checking. This paragraph introduces the common procedure for both datasets. The next two paragraphs describe their differences. The houses in Matterport3D are divided into 56 houses for the train set, 10 houses for the validation sets and 16 houses for the test sets. Based on the original VNLA datasets, data points from the same houses are grouped together. To generate a new multi-priority task T_{mp}, two original data points, T_1 and T_2, are selected from the same house. T_1 and T_2 combine to form T_{mp}, which is defined as a tuple (*house*, *start pose*, *first goal viewpoints*, *second goal viewpoints*, *instruction*). The new multi-priority task is constructed in the following way:

$$T_{MP}[house] := T_1[house](= T_2[house]) \tag{4.1}$$

$$T_{MP}[first\ goal\ viewpoints] := T_1[goal\ viewpoints] \tag{4.2}$$

$$T_{MP}[second\ goal\ viewpoints] := T_2[goal\ viewpoints] \tag{4.3}$$

Explicit multi-priority dataset. The start pose for T_{MP} is selected by $T_{MP}[start\ pose] := T_1[start\ pose]$. The instruction for T_{MP} is constructed as "First $T_1[instruction]$, then $T_2[instruction]$." For example, if T_1's instruction is "find a chair in the dining room" and T_2's instruction is "find a coffee table in the familyroom", then T_{MP}'s instruction will be "First find a chair in the dining room, then find a coffee table in the familyroom." In an explicit multi-priority task, the subtask from T_1 has a higher priority than the subtask from T_2. In addition, we exclude data points which fulfil any one of the following conditions: (1) the starting viewpoint is adjacent to one of the first goal viewpoints; (2) any first goal viewpoint is adjacent to one of the second goal viewpoints; (3) any of the paths require fewer than 10 or more than 50 actions to complete the entire task (i.e., passing through one of the first goal viewpoints and reaching one of the second goal viewpoints). To ensure that the datasets have a balanced distribution of data points from different houses, each house is limited to 3000 data points. We accumulate all the valid data points, randomly shuffle them and sample a fraction to construct a dataset. Table 4.4.1 shows some

statistics of all the multi-priority datasets.

Type	Dataset	Number of data points	Average trajectory length (steps)
Original VNLA	Train	94,798	15.60
	Validation seen	4,874	15.29
	Validation unseen	5,005	15.26
	Test seen	4,917	15.04
	Test unseen	5,001	14.95
Explicit Multi-priority	Train	162,837	32.74
	Validation seen	5,000	34.17
	Validation unseen	5,000	34.28
	Test seen	5,000	34.04
	Test unseen	5,000	33.82
Implicit Multi-priority	Train	145,671	30.32
	Validation seen	5,000	30.28
	Validation unseen	5,000	28.49
	Test seen	5,000	30.18
	Test unseen	5,000	28.38

Table 4.4.1: Statistics on the explicit and implicit multi-priority datasets. The original VNLA dataset is also presented here for comparison. A data point represents a task with two subtasks. Note that, on average, multi-priority tasks require twice as many actions to complete as compared to original tasks.

Implicit multi-priority datasets. In these datasets, the priority levels of subtasks need to be implicitly inferred by the agent based on the perceived distance of each subtask such that the nearer subtask should be performed first. To facilitate distance determination, both the subtasks in an implicit multi-priority task have only one possible goal viewpoint each. Then, to ensure that the higher-priority subtask T_1 is nearer to the starting viewpoint as compared to the lower-priority subtask T_2, we enforce that the starting viewpoint of T_{MP} is in the same region (same room) as T_1's goal viewpoint while it has to be in a different region (different room) from T_2's goal viewpoint. Therefore,

the first goal viewpoint of T_{MP} is in the **same** region (same room) as the starting viewpoint of T_{MP}, and the second goal viewpoint of T_{MP} is in a **different** region (different room) from the starting viewpoint. The instruction for T_{MP} is simply a concatenation of $T_1[instruction]$ and $T_2[instruction]$. In half of the data points, $T_1[instruction]$ precedes $T_2[instruction]$ whereas in the other half $T_2[instruction]$ precedes $T_1[instruction]$. This formulation ensures that the agent cannot obtain explicit information on the priority levels of the subtasks through verbal hints (i.e., words like "first" and "then") or the ordering of the subtask instructions. For example, if T_1's instruction is "find a chair in the dining room" and T_2's instruction is "find a coffee table in the familyroom", then T_{MP}'s instruction can possibly be "Find a coffee table in the familyroom. Find a chair in the dining room". In addition, we exclude data points which fulfil any one of the following conditions: (1) the starting viewpoint is adjacent to the first goal viewpoint; (2) the first goal viewpoint is adjacent to the second goal viewpoint; (3) the task's path requires fewer than 10 or more than 50 actions to complete the entire task (i.e., passing through the first goal viewpoint and reaching the second goal viewpoint); (4) the first goal viewpoint is on the shortest path between the starting viewpoint and the second goal viewpoint (in order to prevent the scenario where the agent "accidentally" completes the nearer subtask even though it is navigating directly to the farther subtask); (5) the total number of steps required to complete the task in the order "start -> higher-priority goal -> lower-priority goal" is more than or equal to the number of steps for the order "start -> lower-priority goal -> higher-priority goal". Similar to the explicit multi-priority datasets, to ensure that the datasets have a balanced distribution of data points from different houses, each house is limited to 3000 data points. We accumulate all the valid data points, randomly shuffle them and sample a fraction to construct a dataset. Table 4.4.1 shows some statistics of all the multi-priority datasets.

4.4.3 Evaluation Metrics

Explicit multi-priority tasks. To evaluate the navigational performance of the agent performing explicit tasks, the following metrics in Table 4.4.2 are used. The main metric that we focus on is the task success rate. A task is considered successful if the agent passes within d metres from *any* of the goal viewpoints of the higher-priority subtask and stops at a location within d metres from *any* of the goal viewpoints of the lower-priority subtask. Here, d is the success radius, a task-specific hyperparameter. In both the original work and this project, d is set at 2 metres. Also, the agent has to complete a task within the assigned time budget. All tasks are designed such that they can be completed within their respective time budgets.

Metric Name	Description
Task success rate	Fraction of test set on which the agent passes within d metres from any of the goals of the higher-priority subtask and stops at a point within d metres from any of the goals of the lower-priority subtask
First success rate	Fraction of test set on which the agent passes within d metres from any of the goals of the higher-priority subtask during its trajectory.
Second success rate	Fraction of test set on which the agent stops at a point within d metres from any of the goals of the lower-priority subtask
First-succeed-second-fail rate	Fraction of test set on which the agent passes within d metres from any of the goals of the higher-priority subtask but fails to reach a goal in the lower-priority subtask
First-fail-second-succeed rate	Fraction of test set on which the agent fails to pass within d metres from any of the goals of the higher-priority subtask but succeeds in stopping at a point within d metres from any of the goals of the lower-priority subtask
Both-fail rate	Fraction of test set on which the agent fails to pass within d metres from any of the goals of the higher-priority subtask and fails to stop at a point within d metres from any of the goals of the lower-priority subtask
First room success rate	Fraction of test set on which the agent passes through any room containing a goal in the higher-priority subtask
Second room success rate	Fraction of test set on which the agent stops at any room containing a goal in the lower-priority subtask
Length	Average distance of the entire path that the agent covers when completing a task
Steps	Average number of steps the agent takes on the trajectory when completing a task

Table 4.4.2: Navigational metrics to evaluate explicit multi-priority VNLA tasks.

Implicit multi-priority tasks. To evaluate the navigational performance of the agent

performing implicit tasks, the following metrics in Table 4.4.3 are used. The main metrics that we focus on are the task success rate and successful step savings. A task is considered successful if the agent passes within d metres from the goal viewpoint of the nearer subtask and stops at a location within d metres from the goal viewpoint of the farther subtask. Here, d is the success radius, a task-specific hyperparameter. In both the original work and this project, d is set at 2 metres. Also, the agent has to complete a task within the assigned time budget. All tasks are designed such that they can be completed within their respective time budgets.

Metric Name	Description
Task success rate	Fraction of test set on which the agent passes within d metres from the goal of the higher-priority (nearer) subtask and stops at a point within d metres from the goal of the lower-priority (farther) subtask
First success rate	Fraction of test set on which the agent passes within d metres from the goal of the higher-priority (nearer) subtask during its trajectory
Second success rate	Fraction of test set on which the agent stops at a point within d metres from the goal of the lower-priority (farther) subtask
Wrong-order success rate	Fraction of test set on which the agent passes within d metres from the goal of the lower-priority (farther) subtask and stops at a point within d metres from the goal of the higher-priority (nearer) subtask (i.e., completes the task in the wrong order)
Successful steps	Among tasks where the agent completes both subtasks in the correct order (i.e., defined in "task succss rate"), the average number of steps the agent takes on the trajectory when completing a task
Successful step savings	Among tasks where the agent completes both subtasks in the correct order (i.e., defined in "task succss rate"), the average number of steps the agent saves by following its current trajectory instead of completing the task in the order "start -> farther subtask -> nearer subtask" via the shortest path. In other words, this metric equals the agent's "successful steps" subtracted from the number of steps the agent would take *if* the agent were to complete the task in the wrong order. A positive value indicates that the agent manages to save some steps by ordering the subtasks based on their relative distances inferred.

Table 4.4.3: Navigational metrics to evaluate implicit multi-priority VNLA tasks.

4.4.4 Results and Discussion

Displayed below are the experiment results of running the multi-priority VNLA agent on both the explicit (Table 4.4.4) and implicit (Table 4.4.5) multi-priority datasets. We

evaluate the agent with five different random seeds and report the mean of each metric.

Metric	Test Seen	Test Unseen
Task Success Rate (%)	**26.40**	**18.28**
First Success Rate (%)	63.36	61.61
Second Success Rate (%)	36.28	26.32
First-succeed-second-fail Rate (%)	36.96	43.33
First-fail-second-succeed Rate (%)	9.88	8.04
Both-fail Rate (%)	26.76	30.34
First Room Success Rate (%)	68.77	68.73
Second Room Success Rate (%)	42.61	33.78
Length (m)	26.69	24.26
Steps	40.90	42.21

Table 4.4.4: Results for multi-priority VNLA agent performing explicit multi-priority tasks.

Explicit multi-priority VNLA. In this experiment, we aim to investigate how well the agent can handle multi-priority tasks with explicit priorities. With reference to Table 4.4.4, when the query ratio (a hyperparameter that determines the asking budget) is set at 0.4, which is the same as the original VNLA, the task success rates are relatively low at 26.40% and 18.28% respectively. Nevertheless, these values are comparable with expected results based on the original baseline. If we assume that completing the subtasks is independent, then the task success rate would be $49.50\% * 49.50\% = 24.50\% < 26.40\%$ for seen houses and $33.94\% * 33.94\% = 11.52\% < 18.28\%$ for unseen houses (49.50% and 33.94% are the success rates in seen and unseen houses for the original VNLA baseline when re-run in our compute server)[1]. The agent is considered to generalise reasonably well to handle multi-priority tasks with explicit priorities.

[1]The success rates here are slightly lower than reported in the original paper. The exact reason is unclear but we assume that it is due to different initial randomisation.

Metric	Test Seen	Test Unseen
Task Success Rate (%)	**41.56**	**13.86**
First Success Rate (%)	92.46	74.90
Second Success Rate (%)	44.64	18.59
Wrong-order Success Rate (%)	0.18	0.30
Successful Steps	26.37	26.23
Successful Step Savings	**10.89**	**0.67**

Table 4.4.5: Results for multi-priority VNLA agent performing implicit multi-priority tasks.

Implicit multi-priority VNLA. In this experiment, we aim to find out whether the agent can determine priorities by inferring the relative distances of subtasks without explicit task instruction, and how well it can perform such tasks. With reference to Table 4.4.5, the agent performs well in seen houses. The task success rate is relatively high and the step savings are 10.89 steps on average. The agent is able to discern which subtask is nearer and should be attempted first. However, the agent performs significantly poorly in unseen houses, with a task success rate only one-third of that in the seen houses. Moreover, the step savings are only 0.67 steps on average (Although it is still a positive value, meaning that the agent manages to save some steps). The significant gap between seen and unseen performances can be due to the fact that the agent struggles to recognise in which room type it is currently located in unfamiliar houses. Consequently, the agent cannot determine which subtask has its goal in the current room. Thus, the agent fails to infer priority information from the task's instruction and performs poorly in completing the subtasks in the correct order. Another observation is that the wrong-order success rates (where the agent completes both tasks but in the wrong order) are extremely low at 0.18% and 0.30% only. The main reason for this result is that it would take significantly more steps to complete a task in the wrong order (by navigating to the farther subtask first) as compared to the correct order. As the agent operates under a tight time budget for each task, the agent would have exceeded its time budget before being able to complete

the entire task, should it have chosen the wrong subtask to start doing first.

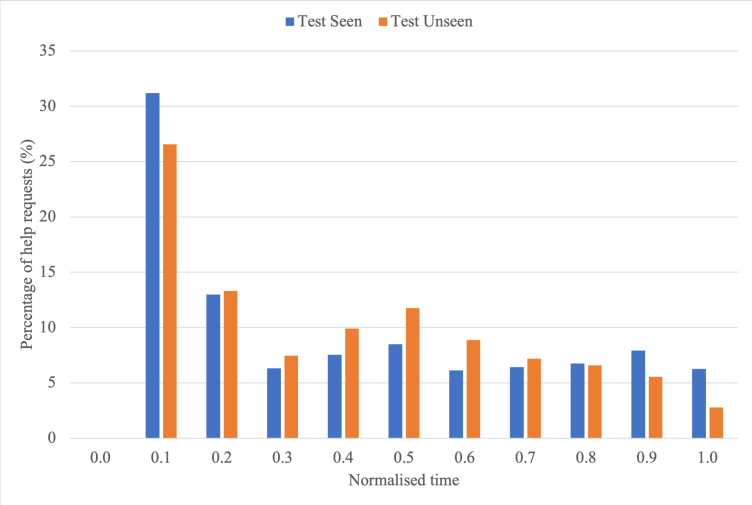


Figure 4.4.1: Percentage of help requests over normalised time for original VNLA, presented here for comparison.

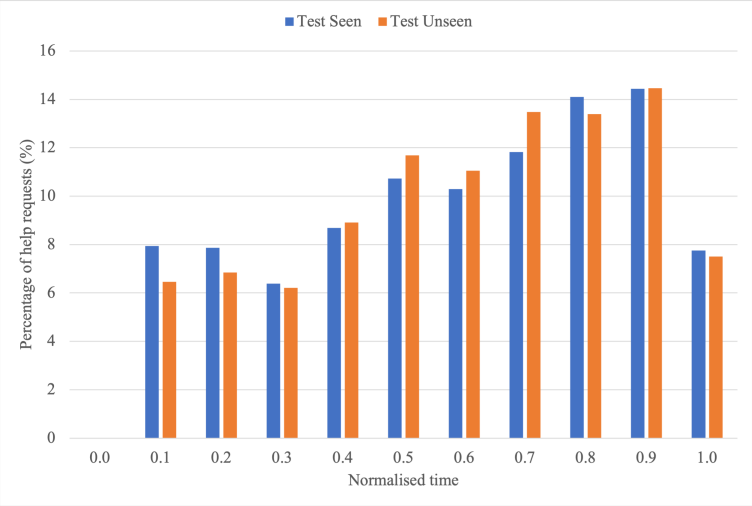


Figure 4.4.2: Percentage of help requests over normalised time for explicit multi-priority tasks.

Distribution of help requests for explicit tasks. To obtain more insights into how the agent's help requesting behaviours may affect its performance, we plot a graph of the fraction of help requests made over normalised time each for both explicit and implicit

multi-priority tasks. Normalised time refers to the time step at which a help request is made, divided by the length of the entire trajectory of the task. On the horizontal axis of the graph, the label represents an interval of 0.1 in normalised time. For example, the data point for "0.3" refers to the fraction of help requests made between 0.2 (exclusive) and 0.3 (inclusive) over the total number of help requests. All the bars of the same colour in the same graph add up to 100%. Fig. 4.4.2 shows an interesting observation that the multi-priority agent has learned to avoid requesting help too early so as to save some asking budget for later steps in a trajectory. This trend is consistent in both seen and unseen test sets. This behaviour is in stark contrast to the original VNLA agent which tends to ask for help early (Fig. 4.4.1). The reason could be because, if the multi-priority agent spends all the budget in completing the first subtask, it will be left with no budget to request for help when completing the second subtask. It has learned to explore first and hold off asking for help until the second subtask. In contrast, it makes sense for the original agent to ask for help early so as to get an initial general direction to start exploring.

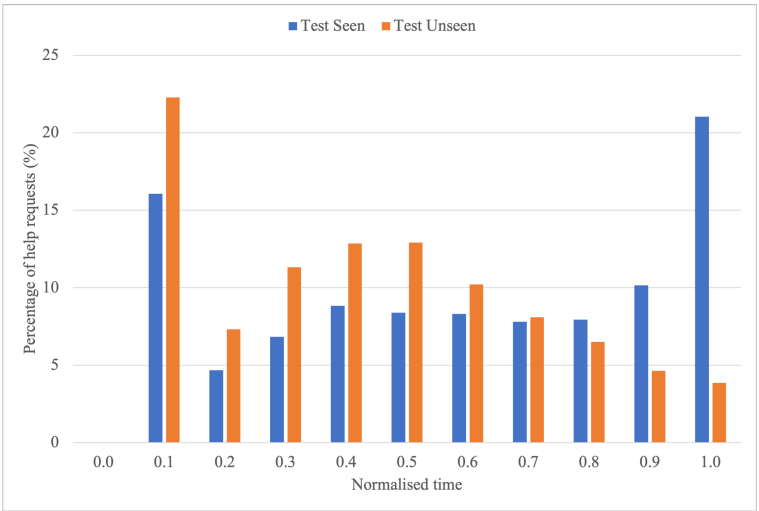


Figure 4.4.3: Percentage of help requests over normalised time for implicit multi-priority tasks.

Distribution of help requests for implicit tasks. Fig. 4.4.3 shows that the help requesting distribution for implicit multi-priority tasks exhibits opposite trends for seen

and unseen test sets. This behaviour aligns with the huge performance gap between seen and unseen environments (Table 4.4.5). The agent tends to ask for help early in unseen houses as it is confused and unable to recognise the current room type. As a result, it cannot identify which of the subtasks is supposed to be higher-priority. This confusion leads the agent to ask for help a lot more in the initial steps in unseen houses as compared to seen ones.

Dataset	Number of data points	Average trajectory length (steps)
Test seen	5000	34.46
Test unseen	5000	33.49

Table 4.4.6: Statistics on NoRoom datasets for explicit multi-priority tasks without room type information. Note that we only generate test sets but not train and validation sets, because our transfer-learning experiment only requires NoRoom test sets for evaluation.

Explicit multi-priority tasks without room labels. So far, all the experiments for multi-priority VNLA deal with task instructions that specify the room labels. It is possible that the agent might have only learned to navigate to the rooms specified in the task instruction and happened to come close to the target objects "by luck", since there are only a few viewpoints in a room. In order to verify that the agent has learned to identify objects instead of simply recognising the rooms, we conduct a transfer-learning experiment on explicit multi-priority tasks, using a similar approach as the no-room experiment in the original work (Nguyen et al., 2019). In this experiment, an agent trained to perform explicit multi-priority tasks with room types is directly evaluated with such tasks without room type information. We follow a data generation procedure similar to the one described in Subsection 4.4.2 to generate a NoRoom dataset. The NoRoom dataset contains explicit multi-priority tasks whose instructions are in the format "First find [object1], then find [object2]". Finding any instance of the requested object type satisfies the subtask. Table 4.4.6 shows some statistics on this NoRoom dataset. The results of this experiment are presented in Table 4.4.7, which shows that our multi-priority agent has

successfully learned to recognise objects despite being trained on tasks with room type information. The agent even achieves significantly better results in both seen and unseen houses (+15.50% on Test Seen and +15.84% on Test Unseen) as compared to explicit multi-priority tasks with room types. The improvement in results can also be because it is easier to locate any instance of an object type than an instance from a specific room.

Metric	Test Seen	Test Unseen
Task Success Rate (%)	**41.90**	**34.12**
First Success Rate (%)	71.34	69.18
Second Success Rate (%)	57.18	47.56
First-succeed-second-fail Rate (%)	29.44	35.06
First-fail-second-succeed Rate (%)	15.28	13.44
Both-fail Rate (%)	13.38	17.38
Length (m)	29.79	26.46
Steps	47.30	48.01

Table 4.4.7: Results for multi-priority VNLA agent performing explicit multi-priority tasks on NoRoom datasets.

4.5 Conclusion

In this line of inquiry, we successfully extend VNLA to generalise to multi-priority tasks with two subtasks. The tasks can have either explicit or implicit instructions for the priority levels of the constituent subtasks. More work needs to be done to devise ways to improve its performance, reduce the performance gap between seen and unseen houses especially for implicit multi-priority tasks, and generalise the agent to handle tasks comprising more than two subtasks.

Chapter 5

Conclusion

In this project, we conduct two lines of inquiry to improve and extend VNLA. Specifically, we (1) explore the use of object co-occurrence information in improving the performance of VNLA, and (2) generalise VNLA to handle tasks with two priorities, where the priority levels are either explicitly specified by the requester or implicitly inferred by the agent.

In our first line of inquiry, we conclude that our current methods of incorporating object co-occurrence information does not result in significant improvement in the agent's performance. Nevertheless, our experiment results offer insights into how object co-occurrence information can affect the agent's performance.

In our second line of inquiry, we successfully extend VNLA to generalise to both explicit and implicit multi-priority tasks with two subtasks. This extension takes VNLA a step closer towards real-world applications of more versatile mobile agents that can assist humans in efficiently locating multiple objects in a house.

For future work, we can explore more methods of designing the COM and incorporating it effectively into the network so as to improve the performance of VNLA. Also, more work needs to be done to improve the performance of the multi-priority agent, reduce the performance gap between seen and unseen houses, and generalise the agent to handle a greater variety of tasks, bringing the virtual agent closer to real-world applications in general household environments.

References

Agrawal, A., Lu, J., Antol, S., Mitchell, M., Zitnick, C. L., Batra, D., & Parikh, D. (2015). Vqa: Visual question answering. https://doi.org/10.48550/ARXIV.1505.00468

Bengio, S., Dean, J., Erhan, D., Ie, E., Le, Q., Rabinovich, A., Shlens, J., & Singer, Y. (2013). Using web co-occurrence statistics for improving image categorization.

Bermudez, L. (2021). Overview of embodied artificial intelligence. https://medium.com/machinevision/overview-of-embodied-artificial-intelligence-b7f19d18022

Chang, A., Dai, A., Funkhouser, T., Halber, M., Nießner, M., Savva, M., Song, S., Zeng, A., & Zhang, Y. (2017). Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*.

Das, A., Datta, S., Gkioxari, G., Lee, S., Parikh, D., & Batra, D. (2018). Embodied question answering. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

Edunov, S., Ott, M., Auli, M., & Grangier, D. (2018). Understanding back-translation at scale. https://doi.org/10.48550/ARXIV.1808.09381

Grauman, K., & Batra, D. (2020). New milestones in embodied ai. https://ai.facebook.com/blog/new-milestones-in-embodied-ai

Han, Y. (2020). Vision-language navigation with q&a interactions. *CP3106 Project Report, National University of Singapore*.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2014). Microsoft coco: Common objects in context. https://doi.org/10.48550/ARXIV.1405.0312

Nguyen, K., Dey, D., Brockett, C., & Dolan, B. (2019). Vision-based navigation with language-based assistance via imitation learning with indirect intervention. *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. *Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. http://www.aclweb.org/anthology/D14-1162

Rabinovich, A., Vedaldi, A., Galleguillos, C., Wiewiora, E., & Belongie, S. (2007). Objects in context. *Proceedings / IEEE International Conference on Computer Vision*. https://doi.org/10.1109/ICCV.2007.4408986

Smith, L., & Gasser, M. (2005). The development of embodied cognition: Six lessons from babies. *Artif Life*, *11*, 13–29. https://doi.org/10.1162/1064546053278973

Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., & Funkhouser, T. (2017). Semantic scene completion from a single depth image. *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*.

Sultana, F., Sufian, A., & Dutta, P. (2020). Evolution of image segmentation using deep convolutional neural network: A survey. *Knowledge-Based Systems*, *201-202*, 106062. https://doi.org/10.1016/j.knosys.2020.106062

Wortsman, M., Ilharco, G., Gadre, S. Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A. S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., & Schmidt, L. (2022). Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. https://doi.org/10.48550/ARXIV.2203.05482

Yu, L., Chen, X., Gkioxari, G., Bansal, M., Berg, T. L., & Batra, D. (2019). Multi-target embodied question answering. *CoRR*, *abs/1904.04686*. http://arxiv.org/abs/1904.04686