

Master's Programme in Automation and Electrical Engineering

# Fine-tuning Open-source Large Language Models for Processing Open-vocabulary Commands for Robotic Navigation

---

**Joonas Palmulaakso**

© 2025

This work is licensed under a [Creative Commons](https://creativecommons.org/licenses/by-nc-sa/4.0/) “Attribution-NonCommercial-ShareAlike 4.0 International” license.



---

**Author** Joonas Palmulaakso

---

**Title** Fine-tuning Open-source Large Language Models for Processing  
Open-vocabulary Commands for Robotic Navigation

---

**Degree programme** Automation and Electrical Engineering

---

**Major** Control, Robotics and Autonomous Systems

---

**Supervisor** Prof. Ville Kyrki

---

**Advisors** Dr. Tsvetomila Mihaylova, Dr. Francesco Verdoja

---

**Date** 1 July 2025

**Number of pages** 51+1

**Language** English

---

### **Abstract**

This thesis investigates using fine-tuned open-source Large Language Models (LLMs) for interpreting open-vocabulary commands for robotic navigation tasks. In this study, this means retrieving objects from scene graphs based on freeform language instructions. This thesis seeks to address the following research questions: To what extent are open-source LLMs suitable for open-vocabulary tasks? How does model size influence performance? How effective are fine-tuned models when compared to their original, unmodified counterparts?

For this study, four different sized LLMs distilled from DeepSeek-R1 model were chosen. These chosen models were then fine-tuned on a training set created from data selected from VLA-3D dataset collection. During the fine-tuning process, a hyperparameter search was conducted. The best fine-tuned models for each size were then tested on another set created from VLA-3D dataset collection and statements from ViGiL3D-dataset.

It was observed that fine-tuned models improve significantly on statements that follow the style of statements included in VLA-3D dataset collection. When tested on more free-form statements from ViGiL3D-dataset, the improvements were less drastic but still notable. The results indicate that open-source LLMs can be used in the task presented in this thesis. The fine-tuned models achieved better results compared to the original models. It was also observed that in some cases smaller model could outperform bigger model.

---

**Keywords** Robotic Navigation, Large Language Models, Fine-tuning, Scene Graph

---

---

**Tekijä** Joonas Palmulaakso

---

**Työn nimi** Avoimen lähdekoodin suurten kielimallien hienosäätö avointen komentojen tulkitsemiseen robottinavigoinnissa

---

**Koulutusohjelma** Automaatio ja sähkötekniikka

---

**Pääaine** Sääntötekniikka, robotiikka ja autonomiset järjestelmät

---

**Työn valvoja** Prof. Ville Kyrki

---

**Työn ohjaajat** TkT Tsvetomila Mihaylova, TkT Francesco Verdoja

---

**Päivämäärä** 1.7.2025

**Sivumäärä** 51+1

**Kieli** englanti

---

### **Tiivistelmä**

Tämä työ tutkii hienosäädettyjen avoimen lähdekoodin suurten kielimallien (Large Language Models) käyttöä robottinavigointitehtävissä, joissa komentoina käytetään avointa kieltä. Tämän tutkimuksen kontekstissa tämä tarkoittaa esineiden hakemista tilaverkosta (Scene Graph) vapaamuotoisten ohjeiden perusteella. Tämä työ pyrkii vastaamaan seuraaviin kysymyksiin: Missä määrin avoimen lähdekoodin suuret kielimallit soveltuvat avoimen sanaston tehtäviin? Kuinka paljon kielimallien koot vaikuttavat niiden tehokkuuteen? Kuinka tehokkaita hienosäädetyt mallit ovat alkupe- räisiin malleihin verrattuna?

Tässä tutkimuksessa valittiin neljä erikokoista kielimallia, jotka ovat tislattu DeepSeek-R1-mallista. Nämä valitut mallit hienosäädettiin VLA-3D aineistokokoel- masta saadulla koulutusdatalla. Hienosäädön aikana suoritettiin hyperparametrihaku oppimismenopeudelle. Parhaiten hienosäädetyt mallit kustakin eri koosta testattiin erilli- sellä testidatalla, joka oli muodostettu VLA-3D-aineistosta sekä erillisistä ViGiL3D- aineiston vapaamuotoisista ohjeista.

Tutkimuksessa havaitsimme, että hienosäädetyt mallit paransivat merkittävästi suoriutumistaan ohjeissa, jotka noudattavat VLA-3D-aineiston ohjeiden tyyliä. Sen sijaan, kun malleja testattiin vapaamuotoisemmilla ViGiL3D-aineiston ohjeilla, paran- nukset eivät olleet yhtä merkittäviä, mutta silti huomionarvoisia. Tuloksista pystyttiin päättämään, että avoimen lähdekoodin kielimalleja voidaan käyttää tässä työssä esitetyssä tehtävässä. Hienosäädetyt mallit osoittivat parempaa suorituskyykyä alkupe- räisiin malleihin verrattuna. Tuloksista voitiin myös huomata, että joissain tilanteissa pienemmät kielimallit suorituiivat paremmin kuin suuremmat mallit.

---

**Avainsanat** Suuret kielimallit, suurten kielimallien hienosäätö, robottinavigaatio

---

# Contents

<b>Abstract</b>	<b>3</b>
<b>Abstract (in Finnish)</b>	<b>4</b>
<b>Contents</b>	<b>5</b>
<b>Abbreviations</b>	<b>7</b>
<b>1 Introduction</b>	<b>8</b>
<b>2 Literature review</b>	<b>10</b>
2.1 Large Language Models . . . . .	10
2.1.1 Origins . . . . .	10
2.1.2 Technical details . . . . .	11
2.1.3 LLM pre-training . . . . .	12
2.1.4 Fine-tuning . . . . .	13
2.1.5 Prompt engineering . . . . .	14
2.1.6 Open source LLMs . . . . .	14
2.2 Robotic Navigation . . . . .	15
2.2.1 Environment Representation with Scene Graphs . . . . .	16
2.2.2 Zero-Shot Object Navigation . . . . .	17
2.2.3 Planning problems . . . . .	20
2.2.4 Navigational problems . . . . .	22
2.3 Discussion . . . . .	23
<b>3 Research methods</b>	<b>25</b>
3.1 Problem formulation . . . . .	25
3.2 Models . . . . .	25
3.3 Dataset . . . . .	26
3.3.1 Filtering the data . . . . .	26
3.3.2 Creating the prompt . . . . .	30
3.3.3 Subsets . . . . .	30
3.4 Fine-tuning . . . . .	31
<b>4 Experiments</b>	<b>33</b>
4.1 Testing set . . . . .	33
4.2 Performance of the models . . . . .	35
4.2.1 General performance . . . . .	35
4.2.2 Performance on datasets . . . . .	36
4.2.3 Performance on relations . . . . .	39
4.2.4 Performance with and without distractors . . . . .	40
4.2.5 Performance on different scene sizes . . . . .	42
4.2.6 Performance on more free-form statements . . . . .	43

<b>5 Conclusions</b>	<b>45</b>
<b>Acknowledgements</b>	<b>46</b>
<b>References</b>	<b>47</b>
<b>A Prompt</b>	<b>52</b>

## Abbreviations

GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
LLM	Large Language Model
LoRA	Low-Rank Adaptation
ZSON	Zero-Shot Object Navigation

# 1 Introduction

The rise of Large Language Models (LLMs) has led to a growing interest in exploring how they can be employed in different fields. The strength of large language models lies in their ability to learn from vast amounts of text and generate seemingly coherent responses accordingly. Currently, LLMs are best known for their use in chatbots and coding assistants, but their potential applications are far more varied [1].

One area where the potential of large language models is still being explored is robotic navigation tasks. Robotic navigation is commonly understood as comprising three key components: localization, path planning, and motion execution. The proposed use cases vary from using LLMs for just retrieving keywords from instructions [2] to using LLMs for almost full end-to-end systems [3].

Other common tasks are task planning, and finding and locating a specific object. These tasks also have subtasks such as locating a specific object in an unknown environment or planning a sequence of actions. Furthermore, many proposed use cases integrate LLMs into specific stages of a robotic navigation pipeline, for example, using them for instruction interpretation, intermediate decision making, or scene understanding [3], [4], [5]. However, LLMs are originally made for natural language processing, which means everything passed to LLM needs to be formatted as text.

Current approaches in robotic navigation tasks mainly rely on closed-source LLMs provided by third parties, such as OpenAI's ChatGPT models [6]. Their strengths lie in their large scale of training data and parameter size, which often correlates with higher accuracy, and their user-friendly interfaces [7]. However, since these models are closed-source, users have limited access to their inner workings and customization. Another problem with closed-source LLMs lays in the need of sending data through third parties. This might become a problem if LLM is given sensitive data such as location information. There might also be down times or high latency for the service when the servers of third party are experiencing high load or technical problems.

One way to mitigate these problems is to use open-source models. Open-source models can be run on local computer or cluster, which mitigates potential down times of third party services and allows to keep data local if required. Although these models are parameter-wise smaller than their closed-source counterparts, they can reach similar performances in some tasks through fine-tuning [8].

Recent research by Chen et al. [9] focused on using fine-tuned open-source LLMs for robotic navigation tasks. However, in their research, the information of the scene is fine-tuned in to the model. This means the model would not work in other environments than what it has been trained on.

With this study, we aim to answer the following research questions: Can LLMs be fine-tuned to understand open-vocabulary natural language commands for robotic navigation? How does model size affect the performance of fine-tuned LLMs in this task? How well do fine-tuned LLMs perform in retrieving objects from scene graphs based on natural language input, compared to the original models?

This thesis will focus on the task of retrieving a specific object from a scene that is described by user's command. The goal of this study is to uncover how well suited fine-tuned open-source models are on a chosen robotic navigation task. This thesis

will fine-tune four different sized open-source language models. The training is done on a subset of a dataset collected by Zhang et al. [10] which includes multiple scenes consisting of scene graphs and referential statements. The models are evaluated on another subset of Zhang’s dataset and then compared with untrained models.

With this study, we aim to answer the following research questions: Can LLMs be fine-tuned to understand open-vocabulary natural language commands for robotic navigation? How does model size affect the performance of fine-tuned LLMs in this task? How well do fine-tuned LLMs perform in retrieving objects from scene graphs based on natural language input?

The rest of the thesis is divided as follows. Chapter 2 reviews the large language models, robotic navigation tasks and how large language models are being used in said robotic navigational tasks. Chapter 3 explains the methodology used in the thesis. Chapter 4 presents and discusses the results of the experimentation. Chapter 5 concludes the the thesis by summarizing previous chapters and suggesting directions for future work.

## 2 Literature review

The first subsection presents required background information about Large Language Models including training, fine-tuning and prompt engineering. It also introduces a range of open-source LLMs and highlights key model examples. The second subsection shifts focus to robotic navigation, reviewing relevant publications and categorizing existing work across various navigation-related tasks.

### 2.1 Large Language Models

Large Language Models (LLMs) have attracted significant attention since their emergence in the late 2010s [11]. At their core, LLMs are statistical models trained to estimate the probability of a sequence of words. In practice they model the probability of next word given the preceding context. This means that given a sequence of words, the model assigns probabilities to possible next words and selects the most likely one during generation. To encourage more diverse output sequences, a temperature parameter is often introduced [7].

These models consist of neural networks with vast number of parameters, which are adjustable weights that can be optimized during training. This allows LLMs to learn patterns in language, grammar, and semantics by being exposed to large corpora of text.

A common way to compare different LLMs is by their parameter sizes. More parameters often allow the model to capture more complex language patterns, though this also increases computational requirements for training, fine-tuning, and inference [12].

#### 2.1.1 Origins

The first major paper that introduced LLMs was *Attention is all you need* by researchers at Google in 2017 [13]. This paper introduced attention mechanism based transformer architecture. Attention mechanism allows models to predict words based on different parts of the input, not just based on the previous word. Transformer architecture made it possible to utilize attention mechanism in models and use nonsequential data processing. The nonsequential data processing has enabled fully utilizing parallel computing of modern Graphical Processing Units (GPUs). This has then allowed models to scale with better and larger GPUs and GPU clusters.

A key milestone in the growing public and research interest in LLMs was the introduction of Google's BERT model in 2018 [14]. One major breakthrough of this model was using bidirectional pre-training. This means the model could use words, both at left and right sides of word, when training. Another breakthrough was pre-training a model and then fine-tuning this pre-trained model for some specific task, such as question answering or sentiment analysis.

The most significant advancement in public AI usage likely came from OpenAI's ChatGPT, which is based on the GPT-3 model [15]. While OpenAI had already developed GPT-1 in 2018 [11] and GPT-2 in 2019 [16], it was the enhanced capabilities

of GPT-3 that enabled the creation of an interactive conversational model. The latest Chat-GPT models are based on GPT-4 models, that allows text and image inputs to create text outputs [6].

With GPT-models there is a visible trend on the parameter sizes of the models. The GPT-2 model has multiple sizes, smallest being 117 million parameter model and largest being 1.5 billion parameter model, and the GPT-3 model has hundred times more parameters being at 175 billion parameters [17]. There is no public information on size of GPT-4 model, but DeepSeek's comparable V3-model has 671 billion parameters [18]. With rising parameter sizes there is also rising need for computation power for training, fine-tuning and inferencing models. This computation power cost has led to interest in smaller models such as Microsoft's Phi-4 model, that has 14 billion parameters but can achieve similar performances to larger models on some benchmarks [19]. Larger models can also be distilled into smaller models by training the smaller models on the outputs of the larger models.

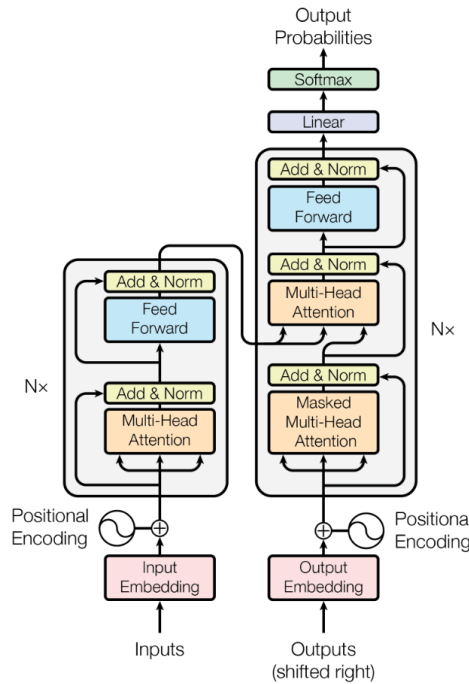
### 2.1.2 Technical details

The two most important breakthroughs enabling LLMs are attention mechanism and transformer architecture. Although the attention mechanism was introduced earlier, *Attention is all you need* paper created transformer solely on the attention, without relying on recurrence or convolutions.

These transformer models work by predicting outputs based on the inputs presented. In case of LLMs the inputs are most often words and sentences. These sentences are processed into predefined tokens, that are a numerical representations of words or strings of letters. For an example, a sentence of "dog is barking" could be divided into tokens of: "dog", "is", "bark", "ing". Tokenization usually also takes into account white spaces such as spaces and line breaks, and other characters. More exact token representations and total unique token count depends on the tokenizer of a specific LLM.

These tokens are then converted into embeddings, which are numerical vectors representing the meaning of the token. Then a positional encoding is added to the embeddings. Positional encoding is needed as all the tokens are processed simultaneously and not necessarily in order, to predict the next token. With the token embeddings and positional encodings, the model can fully utilize the self-attention mechanism, which allows the model to weight the relevance of different input tokens relative to each other based on their location in the input. This means a specific token might have different meaning if some other token is before it rather than after it. There are limitations on how many tokens a model can consider at once. This is called context window and the size of it differs between models.

The transformer architecture consists of multiple stacked layers that have a self-attention block and feedforward neural network. On top of these multiple residual connections and layer normalizations are included. These stacked layers are what allows the model to create deep connections in language understanding by progressively capturing complex patterns and contextual relationships in text. The original transformer architecture from [13] can be seen in figure 1.



**Figure 1:** Model architecture of the transformer. From [13]

### 2.1.3 LLM pre-training

For LLMs to be effective on any task, they need to go through pre-training. During the pre-training, the parameters of the model are adjusted based on massive amounts of text data. This pre-training allows the model to learn general patterns of language, grammar and other information included in the training data.

Pre-training is usually done with self-supervised learning, where the model predicts parts of the input based on other parts. Next-token prediction training is used in multiple models such as GPT-models. This next-token prediction works by comparing the predicted token with the true token and calculating loss based on this comparison, typically with cross-entropy loss. The weights of the model are updated through backpropagation to minimize this loss, improving its ability to make accurate predictions over time.

The training objective is to minimize the error between the model's predicted probability distribution and the correct target word, typically measured using cross-entropy loss:

$$\mathcal{L} = - \sum_{t=1}^T \log P(w_t | w_1, \dots, w_{t-1})$$

where  $T$  is the length of the sequence,  $w_t$  is the actual word at position  $t$ , and  $P(w_t | w_1, \dots, w_{t-1})$  is the model's predicted probability of that word given the preceding context. This loss penalizes the model more when it predicts low probability for the correct word, which leads to better understanding of language structure over

time.

Models can be pre-trained on any textual data, but to get best results, a diverse dataset of good quality is needed. Some datasets commonly used are Common Crawl, Wikipedia, books and other internet sources such as forums. After pre-training the base models are ready. These models can be then fine-tuned for diverse tasks such as acting as a chatbot.

#### 2.1.4 Fine-tuning

The pre-trained LLMs can be fine-tuned further to match some specific use case such as acting as chatbots, code assistants or task planners. This fine-tuning can be done in multiple different ways. Popular ways include such methods as full fine-tuning, half fine-tuning, or parameter-efficient fine-tuning, common method being LoRA [20].

Data amount needed for fine-tuning is usually much lower than when pre-training. However, the data needs to be higher quality, either being labeled data or focusing on a specific topic. When fine-tuning with unlabeled data, unsupervised fine-tuning is performed. With these methods the data used for fine-tuning is focused on specific topic, such as medical documents or legal texts. With labeled data supervised fine-tuning is performed. This method is commonly used when model is needed to behave in specific ways such as labeling data or acting in a formal manner [20].

With full fine-tuning, during the tuning process all parameters are being changed as needed. This method has been used for tuning GPT-3 models [15]. This method of fine-tuning is extremely computationally heavy, just as heavy as pre-training would be, but since starting with a base model, it requires significantly less time. Also, research has shown that modifying all the parameters during fine-tuning can lead to forgetting previously learned information [21].

Easy way to mitigate overfitting and lowering resource needs for fine-tuning is to use half fine-tuning proposed by Hui et al. [22]. This means using partition of total parameters for fine-tuning and freezing remaining parameters. The research done by Hui et al. found that half fine-tuning have comparable results with full fine-tuning, while maintaining learned knowledge and leading up to 30% reduce to training time.

Low Rank-Adaptation (LoRA) addresses the fine-tuning by introducing extra parameters that are only used while fine-tuning while keeping original parameters from pre-trained model frozen [23]. These extra parameters are introduced in trainable low-rank matrices that are added to the architecture of the original model. The rank of the LoRA-matrix is a new hyperparameter that needs to be chosen. LoRA-rank determines the dimensionality of the low-rank matrices injected into the model's attention and feed-forward layers, effectively controlling the capacity of the adaptation. Smaller rank means a smaller matrix and fewer parameters to tune leading to faster fine-tuning process, but too small can lead to inadequate learning. Too large rank however could increase training time while adding excess complexity to the architecture. In the publication by Hu et al. [23], rank of eight could be seen as good starting point for hyperparameter search, as it performs well while reducing trainable parameter count by significant amount. LoRA-alpha is another new parameter that needs to be chosen. LoRA-alpha is a scaling factor applied to the update matrices, influencing the strength

of the adaptation signal.

### **2.1.5 Prompt engineering**

Since LLMs generate text by predicting the next token in a sequence, crafting an effective prompt is essential to guide the model toward producing the desired output. Regardless, the extent to which prompt building is required highly depends on the complexity of the task. For simpler tasks a simple prompt may be adequate, but for larger tasks where desired output might be needed in specific format, a more premeditated prompt is needed [15].

For a simple task, such as asking for trivial information, a prompt consisting only of the question might be sufficient. This type of prompt is called a zero-shot prompt. For more demanding tasks, one-shot or few-shot prompts are needed. In these kind of prompts, one or more examples of desired behaviour are given. For instance, these examples could be useful for fetching data from CSV-file and present the results in a structured way.

Another way to create efficient prompts is making the model use chain of thought [24]. The idea of chain of thought is creating more tokens relating to the task before giving the final answer. With more tokens relating to the task at hand, the model has better potential to predict the correct output. The paper by Kojima et al. [24] even proposed two-stage prompting, where LLM is first used to get reasoning related to the original prompt. Then the original prompt and reasoning are combined as one prompt that is given again for the LLM.

One effective approach used by state-of-the-art language models to enhance their capabilities is Retrieval-Augmented Generation (RAG), which integrates external knowledge retrieval into the generation process [25]. RAG works by creating a query to pre-build database for relevant information and then adding retrieved information to the prompt. This way we can give up-to-date information for LLMs without updating the model weights.

### **2.1.6 Open source LLMs**

The public awareness of LLMs significantly increased with the introduction of OpenAI's GPT models, particularly GPT-3 and GPT-4. While many of the most powerful models remain closed-source, the open-source community has rapidly progressed, narrowing the performance gap and expanding accessibility. This has created growing interest in open-source alternatives, where researchers and developers could customize and control the models with more freedom. Open-source LLMs enable users to perform the entire LLM pipeline, including deployment, fine-tuning, and inference, entirely on their own hardware without relying on external APIs or cloud infrastructure. The parameter sizes of open-source LLMs ranges from a few million parameters to hundreds of billions parameters.

In this section, we highlight a selection of both models used in our experiments and other high-impact open-source LLMs. These models were chosen to illustrate the

diversity in scale, architecture, and performance among current LLMs, as well as to contextualize the capabilities of the models we selected.

**Mistral Large 2 [26]** This model was developed by the French company Mistral AI. The model is 123 billion parameters large and has a context window of 128 thousand tokens. The model has performance on the level of GPT-4o and Llama 3 405B, which is over three times larger parameter wise.

**Phi-4 [19]** Phi-4 is a "small" language model developed by Microsoft, with a parameter count of 14 billion. Although considered small by modern standards, it has demonstrated performance comparable to significantly larger models. On some benchmarks, it approaches the capabilities of cutting-edge models like GPT-4o.

**LLaMA 3 [27]** Llama 3 is model developed by Meta with parameter size of 405 billion and context length of 128 thousand tokens. At launch Llama 3 was a considerably powerful model surpassing GPT-4 model on multiple benchmarks. However, more efficient smaller models have surpassed or reached its performance e.g., the Mistral Large 2 model. Nevertheless, it gave researchers and public access to a larger LLM model.

**Qwen 2.5 [28]** Qwen2.5 includes base model and instruct-tuned models ranging in sizes from 0.5 billion to 72 billion parameters, developed by Alibaba. The technical report highlights performance in multilingual tasks, focusing on English and Chinese.

**DeepSeek-R1 [29]** DeepSeek-R1 is a reasoning model created by a Chinese research group DeepSeek. DeepSeek-R1 gained attention by achieving similar or better results when compared to OpenAI's current state of the art reasoning model, GPT-1o. This was notable, as R1-model was trained from scratch by a lesser known group.

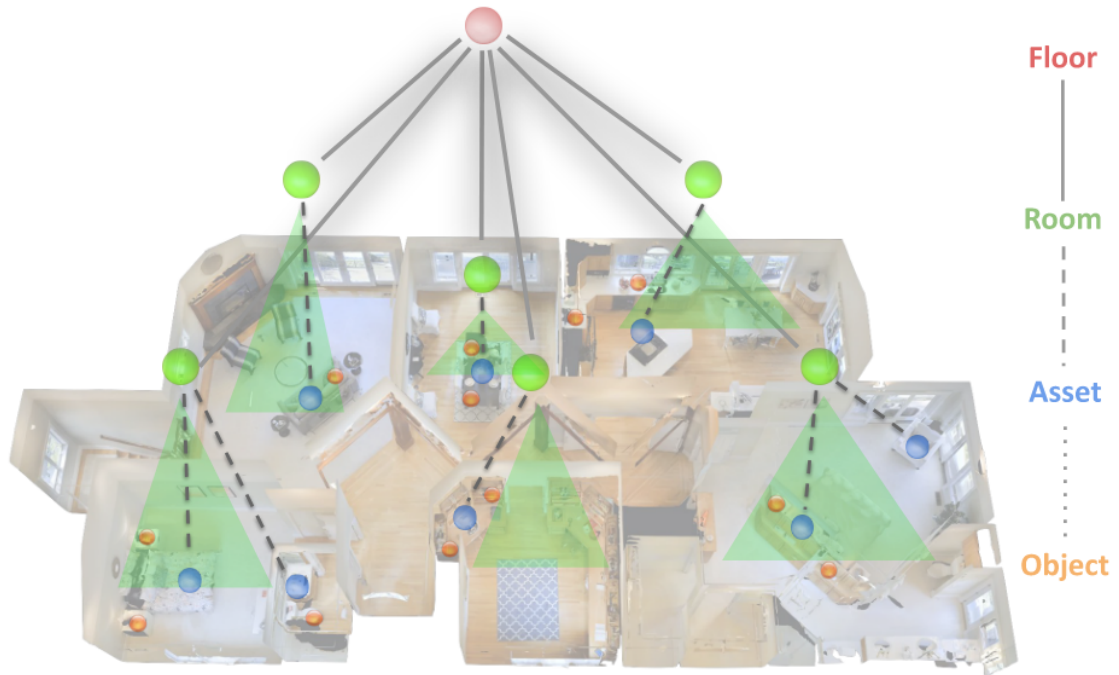
In this thesis we focus on using LLaMA and Qwen based models that have been distilled from DeepSeek-R1 model.

## 2.2 Robotic Navigation

Robotic navigation refers to a task in which the robot determines how to move through an environment to reach a specific destination or to position itself for executing a task, given a known or partially known map. The problem of localization of the robot is most often prerequisite for navigation but is not the focus of this work. Robotic navigation consists of several well-defined subproblems that researchers have studied extensively. For example, some researchers focus on problems of task planning, where the task of the robot is to create a sequence of actions which might include navigation to specific place or interaction with the environment. Some other researchers focus on Zero-Shot Object Navigation (ZSON) tasks, where the robot needs to retrieve an

object from an unknown environment. There is also a problem of more traditional navigation, where a robot needs to navigate from point to point.

The following sections present scene graphs and common problems in robotic navigation, and summarize proposed solutions. In reviewing these approaches, this thesis focuses on their general logic, how they leverage LLMs, and the evaluation methods used.



**Figure 2:** Example structure of 3D scene graph. From [3]

### 2.2.1 Environment Representation with Scene Graphs

Several papers discussed in the following sections, as well as this thesis, base their environmental grounding on scene graphs. 3D Scene Graphs were proposed by Armeni et al. [30] for grounding semantic information. 3D scene graphs can be seen as layered graphs. Each layer contains multiple entities of same type, which can be anything from buildings to objects. Figure 2 shows example of 3D scene graph structure from publication by Rana et al. [3]. The figure shows clear hierarchical structure, where on top level there are floors, floors can have multiple rooms that can have multiple assets that can have objects. For larger graphs, we could include buildings on top of floor layer. Usually, each element in each layer has some information attached to it such as precise location, bounding boxes or descriptions.

Scene graphs can be manually created, but this is not feasible for any graph larger than couple elements. For that reason, scene graphs are created automatically on data gathered from the environments. For 3D Scene Graphs this data is 3D mesh data and

RGB pictures. From these the objects are identified and labeled on the 3D mesh. These labeled items then act as nodes for the scene graph.

Scene graphs are used in various ways in research, but for this thesis we mainly focus on utilizing scene graphs for 3D scene understanding [31]. Scene graphs allow information to be provided to the LLM in a concise manner, while still preserving relationships between different items. This helps us to give more information about the environment while preserving context length of the model for other tasks.

## 2.2.2 Zero-Shot Object Navigation

This section reviews publications related to Zero-Shot Object Navigation (ZSON), a task that involves identifying and navigating to a specified object in an unfamiliar environment. While ZSON includes both object identification and navigation, this thesis focuses primarily on the object recognition and decision-making components of the task.

**OrionNav [32]** OrionNav is a system that focuses on navigating to an object in an unknown and changing environment based on natural language commands. The system works by creating a high level plan with LLM from user input, action primitives and hierarchical scene graph. The high level plan is then passed to another sub-system that creates low-level plan for the robot to execute. The robot has multiple sensors that are used to create semantic object maps that are then processed by a LLM to create updated hierarchical scene graphs.

LLMs are utilized in two places in the system. The LLM responsible for creating high level plans is GPT-4-Turbo. The LLM gets three things as inputs: scene graph, user input, and action primitives. Based on the user input, the LLM tries to retrieve the most likely place for some object to be in. Then the LLM gives the output consisting of the action primitives that tell the robot to go to a specific location, search a specific room, or search all around to gain more information about the environment. The other LLM used for updating the scene graph is Llama3 model. The object map has only information about locations of objects and the LLM is used to label those objects to specific rooms.

OrionNav is evaluated by testing the systems success rate in three different types of tasks. First task is to retrieve an object that is close by, second task is longer range object retrieval and third task is room navigation. The research shows 85 successes out of 96 experiments. For the failed experiments, an error causing it is stated.

**SayNav [33]** SayNav focuses on a problem of multi-object navigation. This can be seen as subproblem of ZSON, as the main goal is to navigate between multiple objects in an unknown environment. This system works by having an LLM as high-level dynamic planner that gets a graph of the environment and instructions as inputs and outputs short range goals for low-level planner that gives more grounded instructions for the robot. SayNav updates the graph based on sensory feedback of the robot and then uses an LLM to label the graph.

SayNav-system utilizes an LLM on different tasks. One of these tasks is high-level planning which dynamically generates step-by-step instructions during navigation based on an automatically created and parsed scene graph. Another usage of the LLM is to annotate and identify spatial entities at the room level in the graph. The graph is also used as supportive memory for LLMs, as investigated rooms will be annotated automatically. The LLM models used in the research are GPT-3.5 and GPT-4 models.

The system is evaluated with multiple configurations on multiple different metrics. The system is tested with building scene graph using ground truth or using visual observations, and with two different low-level planners, OrNav or PNav. The system is evaluated on three metrics: success rate, success weighted by path length, and Kendall Tau metric. With different configurations the success rate of the system varies from 60% to 95%. However, the higher success rates are achieved with scene graphs being built with ground truth, which means the environment is completely visible to the robot.

**VoroNav [34]** This work focuses on the ZSON-problem. The VoroNav-system works by utilizing three modules in the pipeline. One module is local policy module, that plans low-level actions for robot based on goals provided by another module. The robot interacts with the environment based on plans from local policy module. The robot senses the environment with a semantic mapping module, which includes pose detection and a RGB-D-camera. Information from these sensors is fused into a semantic map that is passed to the next module. The next module is a global decision module that is responsible for selecting the path to take. The semantic map is processed in two ways, one being creating node relations of the nodes in the map and another being creating path descriptions for possible paths. These path descriptions are completely text based to help LLM understand them better. These processed maps are fused into a prompt that also takes the goal into account. The prompt is also fused with image captions from surrounding images that have been processed by a BLIP-model [35]. The ready prompt is passed to LLM and the task for it is to retrieve the best path based on semantic, efficiency and exploration rewards.

VoroNav applies GPT-3.5-model in two ways in the global decision module. One application is creating path descriptions based on information from the semantic map. These path descriptions are then passed in the prompt for another application of LLM-model which is deciding the best path to take. This model is given multiple areas with descriptions of these areas and the paths leading to them. The goal of the model is to give predicted probabilities of the goal object being in one of these areas or around the paths. The most probable area is then passed to the local policy module.

Metrics used in the evaluation of VoroNav are success rate and success weighted by path length. The system is evaluated on two different datasets them being HM3D and HSSD. The success rate of VoroNav was 42% on HM3D and 41% on HSSD.

**L3MVN [36]** The research paper does not explicitly state that it focuses on ZSON-problem. However, the research says to tackle problem of visual target navigation in unknown environments, so it can be seen as ZSON-problem. L3MVN architecture

works by taking RGB-D images as an input and generates semantic map based on them. This semantic map is then processed to several frontiers that are passed for LLM along with goal objects. The LLM picks one frontier that is passed as long-term goal for controllers of the robot.

The research paper does not explicitly specify which LLM is used in the system. However, the paper presents two paradigms to LLM usage. One of the paradigms is zero-shot. With this approach the LLM scores which object category is best described by given query. Another approach is feed-forward where a query string is embedded by a pre-trained language model. After that, a fine-tuned neural network gives distribution over object categories for that embedding.

L3MVN system is evaluated with success rate, success weighted by path length and DTG. The DTG is distance to goal metric that is calculated from distance between the robot and the target goal after ending the test. These evaluations are then done on two datasets, Gibson and HM3D. The success rate with zero-shot approach was 76.1% on Gibson and 50.4% on HM3D datasets. With feed-forward approach the success rates were 76.9% on Gibson and 54.2% on HM3D.

**LGX [37]** The LGX approach focuses on L-ZSON task that stands for language-driven zero-shot object navigation. This problem involves a robot using freeform natural language description to find a goal object that it has not seen beforehand in an unknown environment. The LGX system consists of four components. The observation component gains RGB and depth images from the environment. Semantic extraction component detects objects from the RGB images and it captions for the images. At the same time vision-language model GLIP [38] is used to gain target object grounding scores from the images. If the score is high enough, the system moves to the target and stops. Otherwise the system passes object information and an image caption to an LLM decision making component. In this component a prompt is synthesized from the semantic information from images and the target object. Then the LLM gives the best target to move towards in order to find the final goal target. The final component is motion planning, which creates cost-map based on depth images from observations. Then the component moves the robot towards the goal target if GLIP-model is confident enough or towards another target based on the decision from LLM.

The LLM used by LGX is GPT-3. The LLM is used to find sub-targets that are most likely to lead towards the goal target. The research tries creating the prompt in two different ways. The RGB pictures are processed with two different systems, YOLO [39] and BLIP [35]. The YOLO gives list of objects detected in the image while BLIP gives captions to images.

The LGX is evaluated with different prompts created by either YOLO or BLIP outputs. The evaluation is done on RoboTHOR validation set with success rate and success weighted by path length. With prompts created by BLIP the LGX has 28.5% success rate. With YOLO prompts the system has 35% success rate.

### 2.2.3 Planning problems

Although this thesis does not focus on full task or motion planning, reviewing following approaches is relevant due to their use of LLMs for reasoning over scene information and interpreting open-vocabulary commands. These systems often use components such as object identification or subgoal selection, which overlap with the problem setting addressed in this thesis.

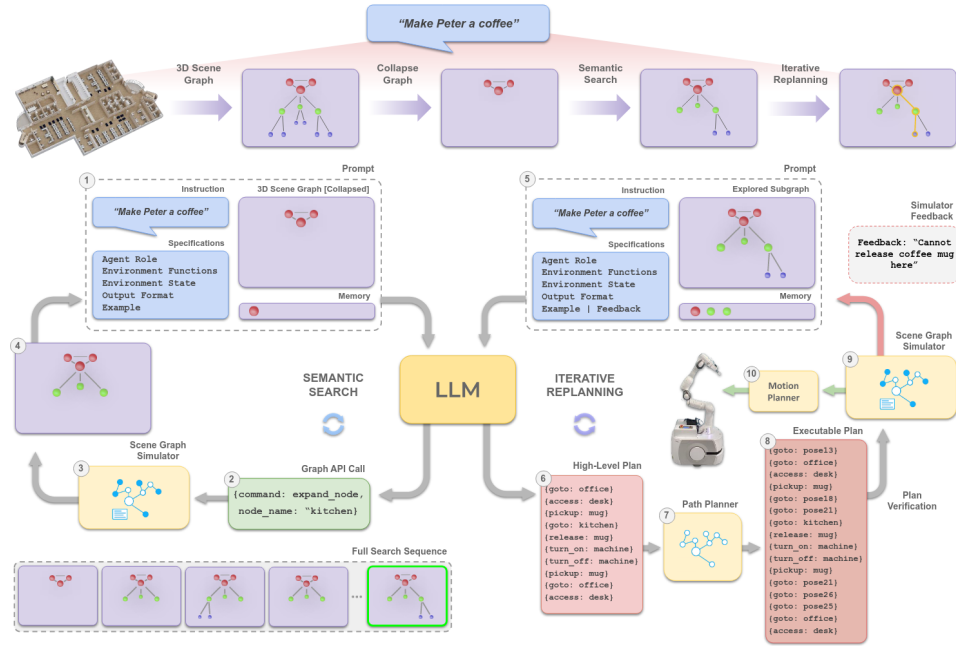


Figure 3: The logic of SayPlan. From [3]

**SayPlan [3]** This work focuses on using LLMs as grounded task planners with the help of 3D scene graphs. The problem addressed by the LLM system is long-range task planning for an autonomous agent. The system gets natural language instructions and needs to construct a plan to accomplish this instruction in a large scale environment. SayPlan achieves the plan creation by iteratively utilizing feedback loops around the LLM. The system starts by giving prompt that includes instruction, collapsed scene graph, specifications and memories of previous searches to the LLM. The LLM has then ability to expand the nodes and expose more of the scene. This is to reduce the memory usage and token limits if given the full graph directly. Then the old scene graph is replaced with this expanded one and information about this expansion is included in the memory. When the LLM thinks it has expanded nodes enough in order to accomplish the task, it can continue to creating high level plan. This high level plan is created from action API primitives and objects or locations. The plan is given to a conventional path planner that outputs an executable plan. This plan is then forwarded to a scene graph simulator, that either forwards it to a motion planner, or

gives feedback. If given feedback, the original prompt is again processed with the appended feedback. This whole logic can be seen more clearly in figure 3.

In the research GPT-3.5 and GPT-4 models were tested. The purpose of LLM in this system is to work as system's internal logic. The LLM makes decision, either to expand or to create high level plan from the available information. This is iterated as long as needed in order to create a feasible plan for the motion planner to execute.

In the research paper the system is evaluated on success rate of finding suitable subgraph, and correctness and executability on simple and long horizon planning tasks. The types of errors in the system are also specified. The SayPlan achieves success rates of 86.7% on simple search and 73.3% on complex search on subgraph search while using GPT-4 as model. On long horizon planning, the SayPlan achieves correctness of 73.3% and executability of 86.6%.

**3P-LLM [40]** 3P-LLM explores the possibilities of using LLMs for path planning tasks. The problem for the 3P-LLM is finding efficient robot path plans based on a natural language input. This method works by combining sensor data of the robot, description of goal state and possible actions into a natural language prompt that is given to the LLM. The LLM then proceeds to predict, which action would be the most efficient one. The outputs from LLM are then processed to robot control commands, after which the robot moves and charts the surroundings.

The research used GPT-3.5-turbo as the LLM model for its relatively real-time capabilities. The LLM is used as probabilistic predictor for feasibility of different actions. The LLM is given a state where it is and possible actions and following states. Then the LLM must predict, which one of these action state pairs leads to the most likely goal state.

In the experimental part, the research evaluated the model against A\* and RRT algorithms with processing time, average path length and path correctness. The GPT-3.5 Turbo had significantly lower processing time, but had the worst results in path correctness. However, it managed to find shorter paths than RRT on average and had almost as short path as solution by A\*.

**FASTNav [9]** This system focuses on using fine-tuned smaller LLMs for multi-point robot navigation based on open language commands. The navigation problem this system tackles is creating a sequence of target coordinates arranged in order based on an open language task description and map information. The paper does not explicitly explain in what format the map information is presented. However, the instructions are some natural language commands and an output is a list comprised of coordinates.

The research mostly focuses on using different small LLMs in the system. FASTNav is used with tau-0.5B, TinyLlama-1.1B, danube-1.8B, openllama-3B and mm4-3B models. All of these models range in sizes from 0.5 billion parameters to 3 billion parameters. The models are fine-tuned based on data that is generated using human-in-the-loop method. In this method some tasks are created by hand, after which LLM, in this case GPT-4, uses these as examples to create more examples. These LLM generated examples are scored by humans and then recreated if needed. Also,

teacher-student iteration is used during the fine-tuning process. This method utilizes the better semantic knowledge of a larger LLM to train another model. Larger model is used as teacher which creates suitable prompts for smaller model which is the student. The teacher adjusts the prompts based on results of the student.

Performances of different LLMs are evaluated on success rate, navigation error, average time, and moving time ratio. Navigation error tells average difference between estimated values and ground truths. Average time tells the average spent executing tasks. Moving time ratio tells how much of the execution time is spent on move. FASTNav got best performance on openllama-3B model, with success rate of 70%. For comparison GPT-4 model achieves 76.7% success rate. However, for the GPT-4 inference time is 15.4 seconds and for the FASTNav with openllama-3B inference time is 2.8 seconds.

#### 2.2.4 Navigational problems

This subsection presents recent language-based navigation systems that, while solving broader navigation tasks, include components closely related to this thesis. In particular, many of them use LLMs to extract and interpret object references from natural language input. Additionally, VELMA highlights the potential of fine-tuning LLMs for such grounded reasoning tasks.

**LM-Nav [2]** LM-Nav is a method that combines multiple models in to complete navigation tasks. The method combines a pre-trained navigation model ViNG, a pre-trained image-language association model CLIP and an LLM. The purpose of this method is to execute long-horizon navigation in outdoor environments based on natural language instructions. The system is given natural language instructions that contains landmarks. The landmarks are extracted from the instructions by the LLM. After retrieving the landmarks, the system grounds them through images of the environment and using an image-language model to check if the landmarks are reached. The navigation itself is done by a vision-only navigation model.

As the LLM, the research uses GPT-3 model. The purpose of the LLM is retrieving landmarks from natural language instructions and parsing them into a list. This instruction list is then used by the other models for the navigation task.

The research evaluates the system diversely. The system itself is evaluated on net success, efficiency, disengagements and planning. Different LLM candidates are evaluated on average extraction success and VLM candidates on detection rate. The system has a net success of 80% on small and large environments. The GPT-3 model had average of 100% extraction success in parsing open language instructions into landmarks. The CLIP model had 87% detection rate when detecting landmarks from visual observations.

**VELMA [41]** VELMA is also a method that combines multiple models, an LLM and CLIP model, to complete vision and language navigation task. VELMA is developed for vision and language navigation in urban environments in Street View.

This is achieved by leveraging the semantic knowledge of the LLM model and the visual capabilities of the CLIP-model. The model is given instructions that include open-ended references of landmarks and directional indicators, so that the model would follow the desired path. The instruction is passed to the LLM with descriptions of the environment created by the CLIP model. Then an action sequence is started, where LLM gives predictions of probable next move to take, and after that a new descriptions of the environment is passed to the model. This is repeated until the final goal is reached.

The LLM is utilized in two ways, extracting the landmarks from the original instruction and making choices based on descriptions of environment and the known landmarks. The CLIP model is used to create the descriptions. GPT-3 model was used for landmark extraction and for the decision making, multiple models were tried. These models include GPT-3 and GPT-4 models, LLaMA models, and Mistral models, with LLaMA-7b model being fine-tuned for this task.

Navigation performance was measured with three metrics, task completion rate, shortest path distance, and key point accuracy. Task completion rate can be seen as success rate used in other papers. The key point accuracy tells ratio of correct choices at key points, in other words, success rate at specific locations. The testing was done with two datasets, of which both are divided into development and test sets. A fine-tuned VELMA model achieved 79.6% key point accuracy and 47.5% task completion rate.

## 2.3 Discussion

In the previous sections, several systems developed for various robotic navigation purposes were described. This section discusses how those systems relate to the research presented in the thesis. While the research questions addressed here do not align exactly with any single system, each of the reviewed systems shares certain similarities with this work. Our research focuses on picking singular objects from a scene graph based on natural language command, while most other research focuses on more complex tasks. Systems such as OrionNav, SayPlan and SayNav try to utilize the semantic knowledge of the LLMs as internal logical unit. Other systems use the LLMs for simpler tasks, such as LM-Nav and VELMA using LLMs to extract landmarks from instruction, and L3MVN using LLM for assigning categories for objects.

Some of the systems do not utilize natural language commands when using LLMs. For example, some of the systems created for zero shot object navigation get only some object as input and afterwards automatically create prompt around that. Contrary for those, we have systems such as FASTNav and SayPlan that can receive free form natural language commands.

Multiple systems use scene graphs as their grounding method of the environment. SayPlan and SayNav systems shows the versatility of scene graphs by SayNav annotating the scene graph during the execution and SayPlan showing the possibility of expanding and contracting the scene graph to save space.

This thesis shares the most similarities with FASTNav, which focuses on fine-tuning smaller models for multi-point navigation based on open language commands. The

motives for that research is same as to our own, instead of relying on third party services that could use our data in malicious manner, we want to run local models. Research conducted in FASTNav paper focused on LLMs of sizes ranging from 0.5 billion parameters to 3 billion parameters, while our research focuses on models ranging from 1.5 billion parameters to 14 billion. However, that work fine-tunes the information of the environment into the model, which means the model needs to be fine-tuned for each environment separately. In our research we want to use any environment that is given in scene graph format without separate fine-tuning for them.

All of the methods employ interesting evaluation strategies, however, they all primarily rely on success rate or similar metrics. Some distinguish between task difficulty by reporting separate success rates for easier and more challenging tasks, such as OrionNav giving results for short and long range object navigation. However, many of the evaluation methods used in the reviewed papers are not directly applicable to this thesis. Several existing approaches define success using auxiliary factors such as path length or efficiency. These criteria are not consistent with the evaluation method used in this work. Instead, success rate is adopted, as it directly measures whether the intended goal is achieved. This provides a clearer and more objective indicator of task completion, especially in the context of this thesis.

## 3 Research methods

### 3.1 Problem formulation

In this research we focus on the problem of processing open-vocabulary natural language commands in the context of robotic navigation, with a specific focus on object retrieval from scene graphs. While many prior works use the semantic capabilities of LLMs for internal logic applications, they often ignore the processing of free-form natural language user input [34] [42] [43]. In contrast, our goal is to fine-tune model that uses natural language user query as input and can be applied to any system that utilizes scene graphs as the grounding representations of the environment.

To keep the scope feasible and well-defined, we limit the task to identifying a single object from a scene graph rather than creating long horizon plans. Fine-tuning for longer plans would require substantial manual creation of large and diverse dataset, which is beyond the current scope.

We can define the problem as follows: Find the object ID  $o$  from a scene graph  $G$  corresponding to an object referred to in a natural language statement  $S$ ,

$$LLM(G, S) \rightarrow o$$

Where  $LLM$  is Large Language Model that denotes a large language model that, given a prompt constructed from the scene graph  $G$  and the natural language statement  $S$ , outputs the identifier  $o$  of the referenced object. The scene graph is explained in section 2.2.1, the statement is a string that is referring to some object in the scene graph. The model is expected to output a single number corresponding to the object id of the predicted object. We extract the first sequence of digits from the output of the LLM and assume it is the predicted object id. During the fine-tuning we provide the true object id as the target label for the model.

Most evaluation metrics in the literature are tailored for more complex tasks, which means the metrics are not suitable in our use case. Therefore we adopt mainly success rate as the evaluation metric. We want to also see how success rates differ in different scenarios, i.e. with different statement types and scene sizes.

### 3.2 Models

In this research we used four different sized models distilled from DeepSeek-R1 model [29]. Choosing these models made the fine-tuning process and testing easier, as they are limited by same context lengths. Smallest model used is 1.5 billion parameter model with base model of Qwen2.5-Math-1.5B. Second smallest model is 7 billion parameter model with base model of Qwen2.5-Math-7B. The second largest model is based on Llama-3.1-8B model and is 8 billion parameters large. The largest model used is based on Qwen2.5-14B-model and is 14 billion parameters large. The Qwen2.5-Math-model [44], that 1.5 billion and 7 billion models are based on, is itself fine-tuned from Qwen2.5 base model [28]. This base model is continuation of Qwen2-model [45]. From now on, the models will be addressed based on their parameter size.

All of the models used have the same maximum amount of tokens accepted as inputs. This maximum model token length for the models is 16 384 tokens. While this token limit is quite large for more common LLM tasks such as working as chat bot, when we introduce vast amounts of information, namely scene graphs, the token limit quickly limits our possibilities to create prompts. Filtering of the scene graphs is discussed more in depth in the section 3.3.

### 3.3 Dataset

The main dataset used for training and testing the models is VLA-3D dataset collection collected by Zhang et al. [10]. The VLA-3D dataset collection is large collection consisting of six datasets. The datasets are 3RScan [46], ARKitScenes [47], Matterport [48], Scannet [49], and Unity [10]. Datasets 3RScan, ARKitScenes, Matterport and Scannet are based on real-world measurements, while Unity is simulation generated. These datasets have combined 11.5 thousand scanned rooms with corresponding ground-truth scene graphs including 23.5 million generated semantic relations between objects and 9.7 million automatically created semantic statements. These semantic statements are short natural language statements referring to singular object in a scene. The statements are generated based on information extracted from the scene graphs. More precisely, each scene has following data available: point cloud, list of objects with labels of semantic class, bounding box and colors, list of traversable free spaces, list of regions with semantic labels and bounding boxes, scene graph of spatial relations, and language statements [10].

For the use case of this work, the most important information from the dataset collection is information of the environment in a LLM-friendly way and open vocabulary statements. The scenes in datasets have two JSON-files which of one is used for scene graph and includes information such as bounding boxes, regions, objects in the regions, info of those objects including labels, locations and colors, and relationships between objects. An example segment of the scene graph is seen in figure 4. The other JSON-file includes referential statements created from scene graph. An example of referential statement can be seen in figure 5.

As said in section 3.2, the used models have token limit of 16 384 tokens. When trying to turn one complete scene graph to tokens, we almost always exceed this token limitation. That is one reason why we need to prune irrelevant information from the scene graph. Another reason is that when leaving out irrelevant information, the LLM has to focus on more essential parts of the scene graph.

#### 3.3.1 Filtering the data

One type of redundant information filtered out from the scene graph is bounding boxes of objects and regions. These also include center points, volumes and sizes of the bounding boxes. Each object in the scene graph has multiple different labels and color information attached to them as seen in figure 4. We only leave most prominent information from these namely object id, raw label and color labels, leaving out excessive labels, numerical color values and color percentages. All of the objects

---

```
{
  "scene_name": "scene0012_00",
  "regions": {
    "0": {
      "region_id": "0",
      "region_name": "Living_room/_Lounge",
      "region_bbox": [ ... ],
      "objects": [
        {
          "object_id": "0",
          "raw_label": "floor",
          "nyu_id": "11",
          "nyu40_id": "2",
          "nyu_label": "floor",
          "nyu40_label": "floor",
          "color_vals": [
            [47.0, 79.0, 79.0],
            [160.0, 82.0, 45.0],
            [0.0, 0.0, 0.0]
          ],
          "color_labels": ["gray", "brown", "black"],
          "color_percentages": [
            "0.36489132",
            "0.27524313",
            "0.22664969"
          ],
          "bbox": [ ... ],
          "center": [-0.06, 0.24, 0.04],
          "volume": 3.17,
          "size": [4.46, 3.24, 0.21]
        },
        ]
      ]
    }
  }
}
```

---

**Figure 4:** Section from a scene graph

---

```

{
  "scene_name": "scene0012_00",
  "regions": {
    "0": {
      "the_chair_that_is_between_the_coffee_table_and_the_door": [
        {
          "target_index": "4",
          "target_class": "chair",
          "target_position": [-1.20, -0.99, 0.39],
          "target_colors": ["gray", "black", "brown"],
          "target_size": 0.35,
          "target_color_used": "",
          "target_size_used": "",
          "distractor_ids": ["3", "5", "6"],
          "relation": "between",
          "relation_type": "ternary",
          "anchors": {
            "anchor_1": {
              "index": "9",
              "class": "coffee_table",
              "position": [-0.59, -1.11, 0.28],
              "color": ["gray", "N/A", "N/A"],
              "size": 0.15,
              "color_used": "",
              "size_used": ""
            },
            "anchor_2": {
              "index": "16",
              "class": "door",
              "position": [-1.64, -1.21, 0.84],
              "color": ["gray", "N/A", "N/A"],
              "size": 0.21,
              "color_used": "",
              "size_used": ""
            }
          }
        }
      ]
    }
  }
}

```

---

**Figure 5:** Section from referential statements

have also multiple relationships assigned to them. These include relationships: *above*, *below*, *closest*, *farthest*, *between*, *beside*, *near*, *in*, *on* and *hanging on*. Even though *closest* and *farthest* sounds as unambiguous information, objects have multiple other objects as their farthest and closest object. The relations of *farthest* and *closest* also take up a lot of space from token limit as objects occur in relations of multiple other objects. Therefore relationships of *farthest* and *closest* are left out from filtered data. After applying this filtering to each region and object in a scene, most of the scene graphs from datasets 3RScan, ARKitScenes and Scannet are able to fit inside of the token limitation. For example, a singular scene graph sized 100 000 tokens can be downsized to 8 000 tokens. While most of the scenes from Matterport and Unity remain too large for the models, we are still able to find enough statements for our subsets from these datasets.

For the next step, (query, object id) pairs are created based on the referential statement JSON-files. The most crucial information in these files are the referential statements themselves. These statements have multitude of other information attached to them as seen in figure 5. For prompt creation we only need information of the specific scene the statement is for, the statement itself and the target index that is our target id. The statements have conveniently relation types attached to them so we can filter out statements that are based on of relations of *farthest* and *closest*. We also need information on existence of distractors, that means existence of other same type of objects in the scene. Also we save a count of total number of object in scenes. This information is later used to determine the difficulty of a prompt based on existence of distractors and total number of objects. If figure 6 we can see filtered section of the same scene graph presented in figure 4.

---

```
{
  "scene_name": "scene0012_00",
  "regions": {
    "0": {
      "region_id": "0",
      "region_name": "Living_room/_Lounge",
      "objects": [
        {
          "object_id": "0",
          "raw_label": "floor",
          "color_labels": ["gray", "brown", "black"]
        }
      ]
    }
  }
}
```

---

**Figure 6:** Section from a filtered scene graph

### 3.3.2 Creating the prompt

Each prompt is constructed by pairing a natural language statement with the corresponding scene graph. The statement describes an object using open-vocabulary expressions, and the scene graph provides grounded information about the objects and their relations in the environment. These combined inputs are used as the base of the prompt.

The final prompt is divided into three parts: system prompt, user input, and location for output. Research has shown that doing one-shot or few-shot prompts can improve performance of LLMs [50]. For the use case of this research we use one example in the prompt before we give the real user statement and associated scene graph. To save token space for the scene graph, the example included in the prompt is as minimal as possible. In it we include short user prompt, scene graph with four items and eight relations, and wanted output.

After the system prompt including the example, the real user query is given with the scene graph. The scene graph is kept in JSON-format as scene graphs can be presented in a compact way. Other ways to present the graph include Markdown, as used in the research by He et al. [51], but this approach does not significantly improve performance compared to the JSON format in most cases. After the scene graph we end the prompt by including a location for the output. We end the prompt with "###Output:". This structure is designed to encourage the LLM to respond consistently by continuing with an object id, as demonstrated in the example given previously in the prompt. The prompt used can be seen in appendix A.

We are not training the model with end of sentence tokens, which means the fine-tuned model will not stop generating tokens after giving an object id. For this reason we operate the zero-shot models and fine-tuned models the same way: we allow the model to generate 10 tokens after our prompt, and then we retrieve first sequence of digits presented in the output and assume it is the desired object id. This is done under the assumption that the model will continue directly with tokens representing a number, with each token corresponding to a single digit or multiple digits. The 10-token limit is chosen to allow for the inclusion of extra tokens preceding first digit token while still providing enough tokens to represent the largest object IDs found in the scenes.

### 3.3.3 Subsets

We divided the datasets within VLA-3D into three subsets: training, evaluation and testing sets. The datasets 3RScan, ARKitScenes, Matterport, and Unity are used for training, evaluation and testing sets. The dataset Scannet is only used for testing. Training, evaluation and testing subsets are created by random sampling statements from the datasets. During the random sampling of the statements, the statements are transformed to full prompts. The prompts are then tokenized and the number of tokens is compared with max token length of the models. If number of tokens exceeds the limit, that statement is ignored and random sampling is continued.

For training set we pick 2 500 random natural language statements from each dataset

Dataset	Example Statement
3RScan	"The black chair that is next to the big table."
Matterport	"The table that is beneath the small table."
ARKitScenes	"The cabinet that is in between the other cabinet and the dishwasher."
Scannet	"The clothes that are on the desk."
Unity	"The computer that is in the middle of the headphones and the keyboard"
ViGiL3D	"Starting from the right of the fridge, this is the closed cabinet above the counter. It is not the middle one"

**Table 1:** Example natural language statements from different datasets.

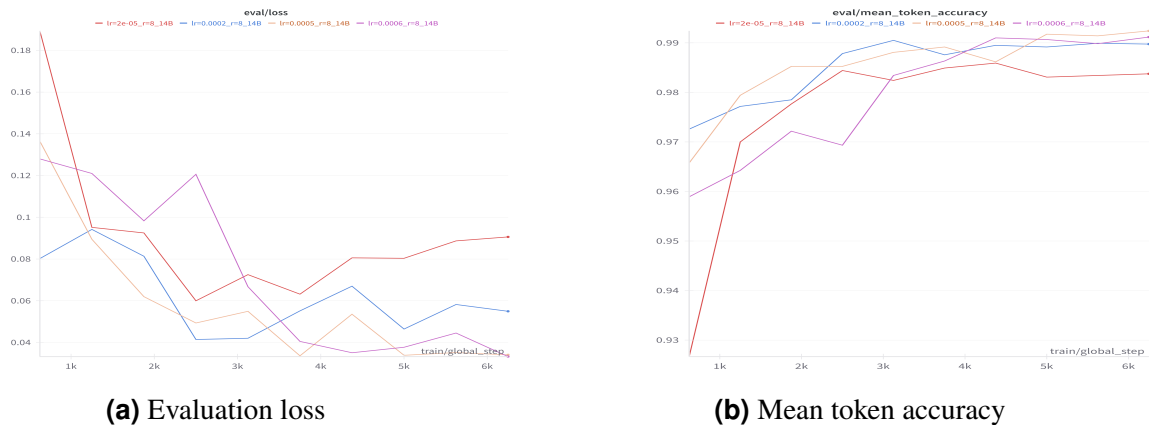
used for training with combined amount of 10 000 statements. The evaluation set is created from 250 random natural language statements from same datasets as training set. For the testing set 2 000 statements are randomly sampled from Scannet-dataset and 500 statements from other datasets with total of 4 000 statements. We do not check whether a single statement appears in multiple sets, as we are sampling fewer than 20,000 samples from 9.7 million possible samples, making the likelihood of significant overlap negligible.

On top of this training set, we utilize ViGiL3D-dataset [52] that includes more linguistically diverse statements that reflect real world statements much better than automatically created ones. The dataset has 239 statements referring to objects in Scannet-dataset. Some of these statements have multiple possible referred objects or statements that are not referring to any object in a scene. After filtering these statements, some statements led to oversized prompts for the models. After filtering these statements, we are left with 177 unique statements. We also found that the scene graphs in VLA-3D do not always include all of the objects presented in object list of that scene. Out of the 177 statements, additional 14 statements were removed, leaving us with final 163 statements.

Table 1 shows examples of statements from different datasets. As we can see, all datasets from VLA-3D dataset collection have similar statements, that do not make always sense. For example, Matterport has a statement of "table beneath small table" which does not make much sense in the real world. We can also see that ViGiL3D dataset has much more expressive statements. The difference results from statements in VLA-3D being procedurally generated, while those in ViGiL3D are manually created.

### 3.4 Fine-tuning

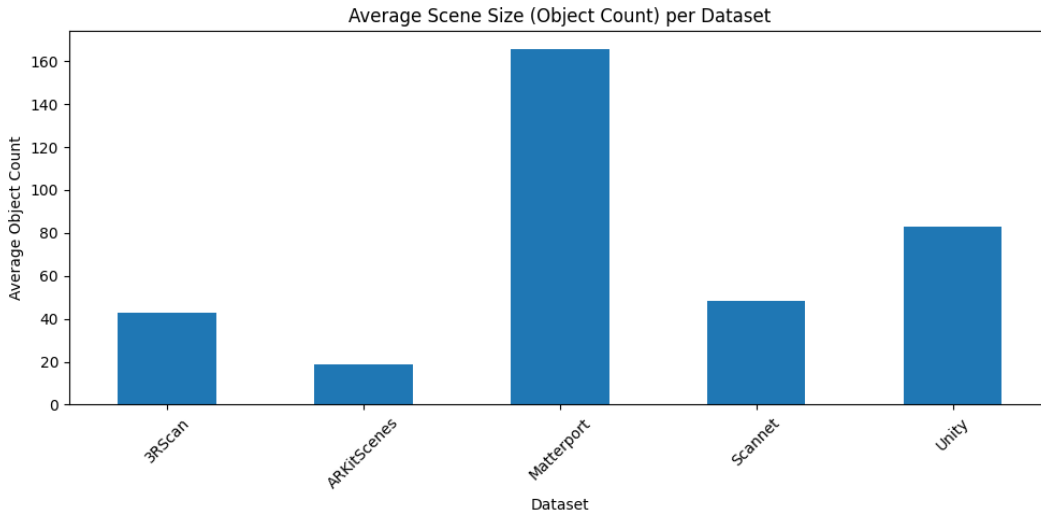
The fine-tuning is conducted on the training subset of 10 000 natural language statements converted to prompts and continued with the desired object ids. For fine-tuning Hugging Face’s Python libraries are utilized to simplify the process. During the fine-tuning, Weights & Biases is used to log training runs. Fine-tuning was conducted on Nvidia’s H100 and A100 GPUs with 80 GB of VRAM.



**Figure 7:** Plots of evaluation loss and mean token accuracy for 4 training runs for 14 billion parameter model

Before fine-tuning some essential training parameters were set. As LoRA is used in fine-tuning, LoRA-rank and LoRA-alpha need to be chosen. Rank of 8 is chosen as in the publication proposing the LoRA, this rank seems to perform well with low memory usage [23]. For LoRA-alpha, value of 16 was chosen. Batch size was set to 1 but to simulate a larger batch size within memory constraints, gradient accumulation steps were set to 8. This means gradients were accumulated over 8 forward passes before a single optimization step, yielding an effective batch size of 8. Training was conducted over 5 epochs. Checkpoints of the trained models were saved for each epoch.

During fine-tuning a hyperparameter search for the learning rate was conducted. A learning rate of 0.00002 was selected as starting point, as it is commonly used as a default value in widely adopted libraries. After a training run, loss and mean token accuracy graphs were used to determine whether the learning rate was sufficient. An example of these graphs can be seen in figures 7a and 7b. After training multiple learning rates, most promising learning rates were chosen based on similar plots to these. The optimal learning rate and checkpoint were selected by evaluating each model’s checkpoints, fine-tuned with different learning rates, on the evaluation subset. Hyperparameter tuning was performed separately for each model, with particular emphasis on the 1.5 billion and 14 billion parameter models. These models were given more training runs, as we were particularly interested in evaluating how well the smallest model could perform and how strong the best-performing model would be.



**Figure 8:** Average scene sizes for each dataset.

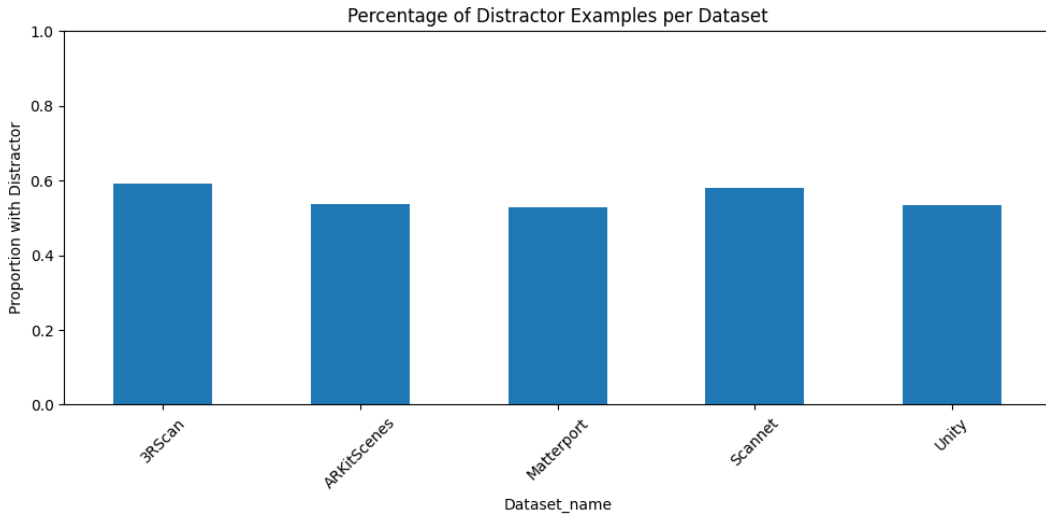
## 4 Experiments

As stated in section 3.3, the zero-shot models and the fine-tuned models were tested on subset based on VLA-3D dataset collection. This included 500 data points from each subset that were used for training, and 2000 data points from subset that was allocated only for testing. All of the models were run on Nvidia H100 GPUs with 80 GB of VRAM. The experiments aim to evaluate how fine-tuning influences model performance, how differences in model size impact results, and whether any of the tested models demonstrate sufficient capability for the task defined in this thesis. Next the testing subset is addressed more thoroughly and after that the performances of the models, both zero-shot and fine-tuned are presented.

### 4.1 Testing set

The testing subset consists of 4 000 referential statements. In figure 8 the average sizes of scenes in datasets of testing subset can be seen. It can be noticed that Matterport dataset has substantially larger scenes object wise with over 165 objects on average. On the other hand ARKitScenes dataset has on average under 20 objects in each scene. 3RScan and ScanNet have similar average object counts, with 42 and 48 objects per scene, respectively. On average, Unity scenes contain 83 objects, which is notably more than 3RScan and ScanNet, yet still only about half the size of Matterport scenes. From these scene sizes, it could be hypothesized that the statements from Matterport would be considerably harder for the models, as it has seemingly more complex scenes on average. Contrarily, statements from ARKitScenes could be trivial for the models with its small scenes.

Figure 9 shows the percentage of statements containing distractors for each dataset. If the scene contains objects of the same type to the goal object referred in the statement, the statement has a distractor. It could be considered that statements without

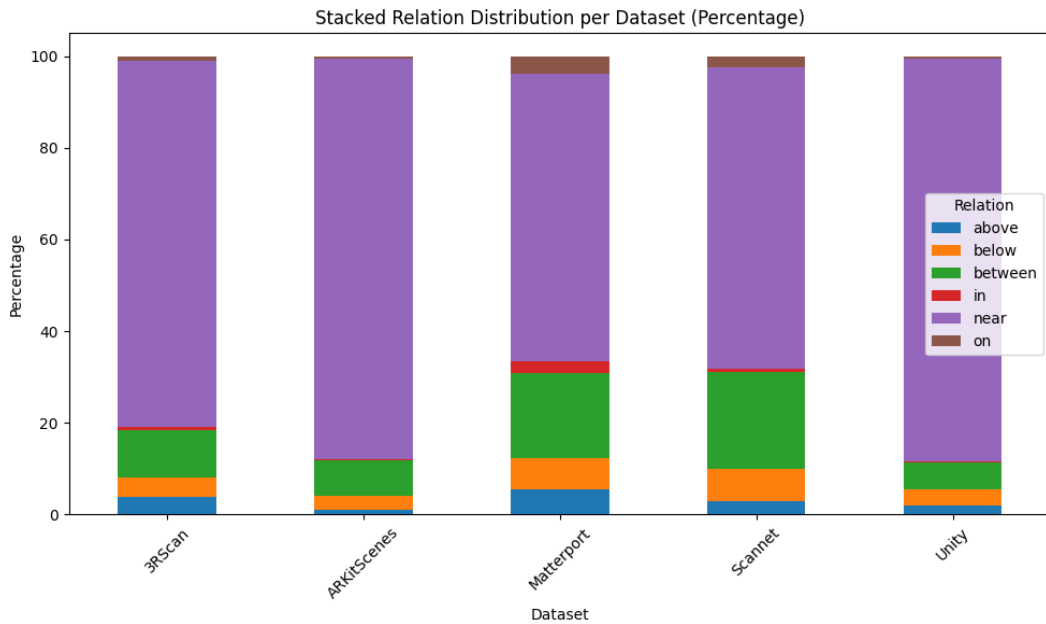


**Figure 9:** Percentage of statements involving distractor for each dataset.

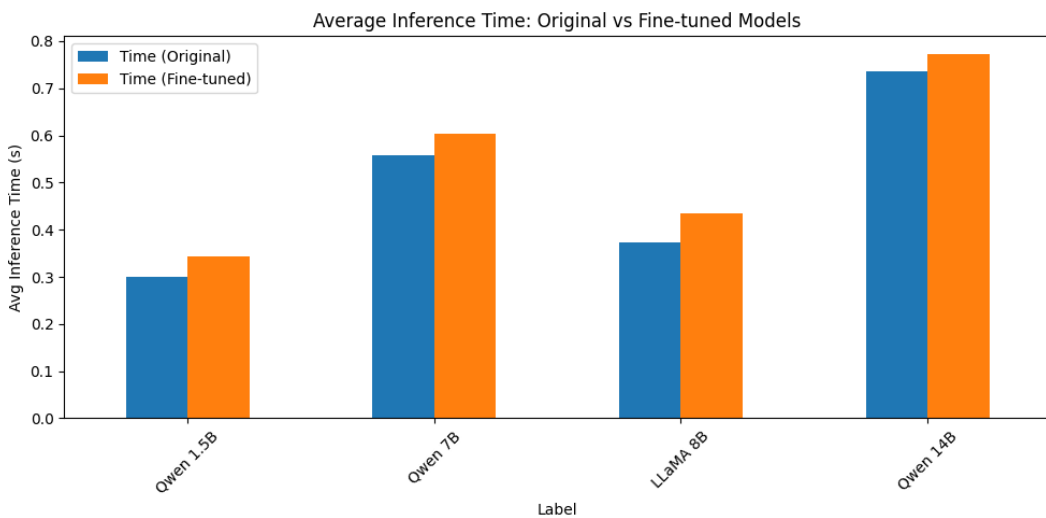
distractors would be trivial for the models, while models could have more difficulties with statements with distractors. In the figure 9 it can be seen that the different datasets have surprisingly similar percentages of statements including distractors. While the Matterport had the most objects on average per scene, only 52% of the statements have a distractor. One could have thought that the largest scenes would have the largest amount of objects and therefore having the most statements with distractors. From this information we can conclude scenes in Matterport having large amount of unique objects. 3RScan dataset has the largest percentage of distractors with 59% of the statements having them. Datasets ARKitScenes, Scannet and Unity have 54%, 58% and 53% of statements with distractors respectively.

Another important statistic of the testing subset is the distribution of relations for each dataset that can be seen in figure 10. The statements have been divided into six different relations: above, below, between, in, near and on. From the figure it can be seen that all of the datasets have highest proportion of statements of relation *near*. It can be noted that Matterport and Scannet have the most diverse selection of statements, when it comes to relations, with having only 63% and 66% of statements being of *near* relation. 3RScan has 80%, ARKitScenes 87%, and Unity 88% of their statements being of relation *near*. It can be also seen that the test subset has limited amount of statements of relations *on* and *in*. The testing subset has total of 67 instances of statements which have relation of *in*. Same number for relation of *on* is 78.

It is hard to hypothesize which dataset poses the greatest challenge for the models, as discussed further in section 4.2.2. While Matterport has the largest scenes on average, it has the least amount of statements with distractors by percentage. It also has most diverse set of relations, but this could either be easier as some relations such as *in*, *on* and *below* could be more trivial to solve, or harder as much more diverse set of statements are included. On the other hand, statements from ARKitScenes could be most trivial, as it has smallest scenes and most monotonous set of relations for its



**Figure 10:** Distribution of relations in each dataset.



**Figure 11:** Inference times of zero-shot and fine-tuned models.

statements.

## 4.2 Performance of the models

### 4.2.1 General performance

Starting with inference time, in figure 11 the average inference times of each model can be seen. For models of sizes 1.5B, 7B and 14B the inference times logically increase as the model size increases and more calculations are needed for each predicted token.

**Table 2:** Accuracies of different-sized models on the test subset.

Model	Orig. Acc. (%)	FT Acc. (%)	Abs. Improv. (%)	Rel. Improv. (%)
1.5B	5.5	53.1	47.6	865.5
7B	23.1	61.0	37.9	164.1
8B	40.6	94.9	54.3	133.7
14B	56.4	93.4	37.0	65.6

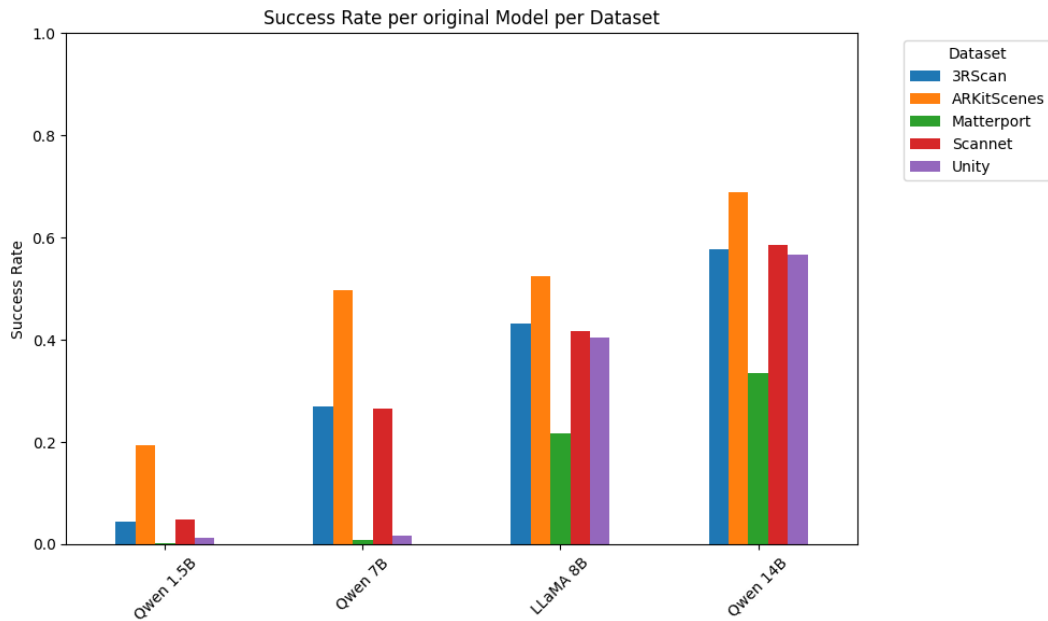
However, 8B model has closer inference times of that of 1.5B model than 7B model which seems odd. This can be attributed to 8B model being based on Llama model while 1.5B, 7B and 14B models are based on Qwen model. It can be as presumed that architecture of Llama is more efficient than that of Qwen’s. The figure also shows that fine-tuned models exhibit longer inference times compared to the zero-shot model. This can stem from the fact that models are fine-tuned with LoRA which adds complexity to the model in the form of additional parameters from the low-rank matrix as told in section 2.1.4.

In the table 2 the overall results on testing subset are shown. The results show that the fine-tuning process led to improvement across all models. The zero-shot models show a clear correlation between model size and the accuracy of the model. However, after fine-tuning the models, the 8B model shows the best accuracy. The 8B model has also seen the largest amount of absolute improvement, which also is quite interesting. On the other hand, the 1.5B has undoubtedly seen largest relative improvement from the zero-shot model, but it is no surprise as the starting point was fairly low. It is neither a surprise that the 14B model has seen the least amount of improvement, both absolutely and relatively, as it had clearly best results to start with. The unforeseen results of fine-tuned 8B model could be explained again with the fact that it is based on different model than the rest of the models. But more probable reason is that during hyperparameter search we happened to find better values for 8B model.

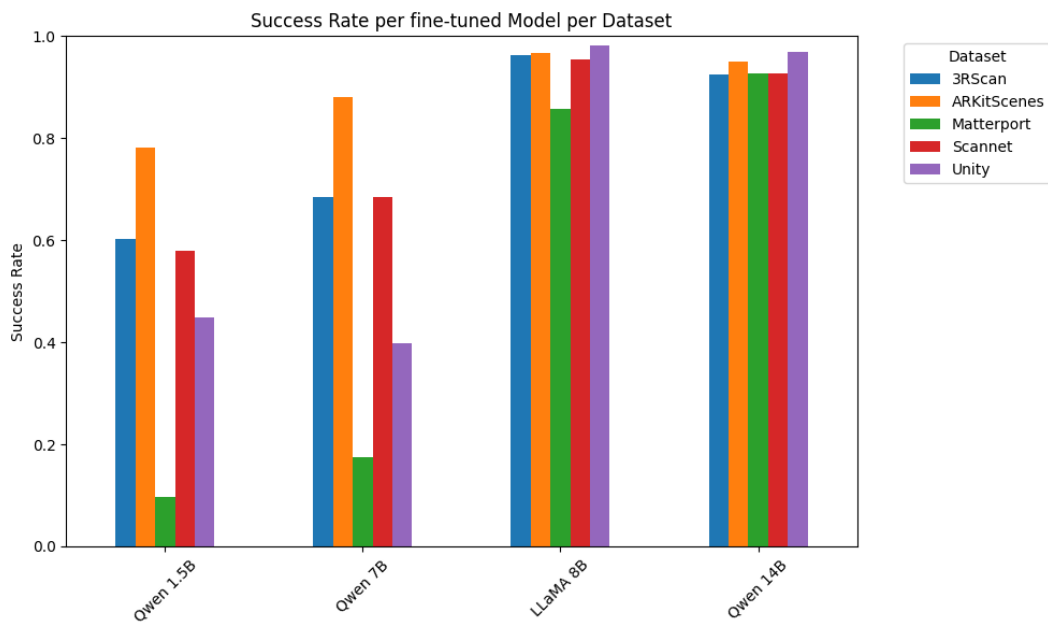
#### 4.2.2 Performance on datasets

The figure 12 shows more detailed information on success rates of zero-shot models for each dataset. For each zero-shot model the dataset ARKitScenes seemed to include the easiest statements as hypothesized in previous section 4.1. It can likewise be seen that the Matterport had the most challenging combination of statements and large scene graphs. The accuracies on each dataset seem to mostly correlate with the sizes of models, with the exception of Matterport and Unity, which have no apparent difference between 1.5B and 7B models.

It is interesting to see differences between 7B and 8B models. On ARKitScenes they have almost identical performances with 8B model being slightly better, as one could expect based on their size difference. However, it can be seen that 7B has performances close to those of 1.5B model on datasets Matterport and Unity. The results on datasets 3RScan and Scannet are not as drastic but still clear. It can be concluded that the 7B model exhibits greater performance degradation with increasing scene size compared to the 8B model.

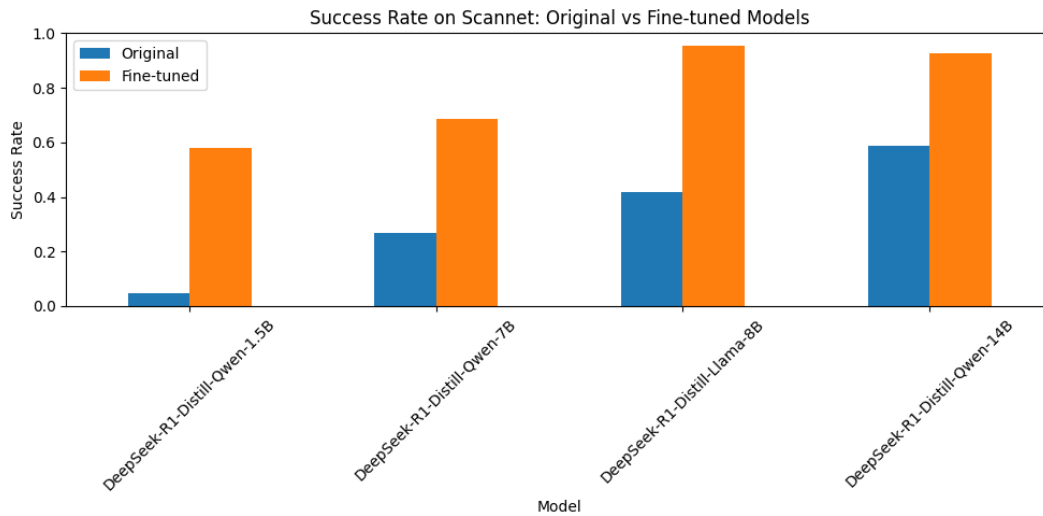


**Figure 12:** Success rates of zero-shot models for each dataset.



**Figure 13:** Success rates of fine-tuned models for each dataset.

Continuing to figure 13, the success rates of fine-tuned models for each dataset are shown and improvements can be seen all across the board. Overall, ARKitScenes keeps its position among the easiest dataset for most models. Now however, the highest absolute accuracy is found with Unity dataset by 8B model with accuracy of 98.2%. The second highest accuracy overall on single dataset is with 14B model, that has

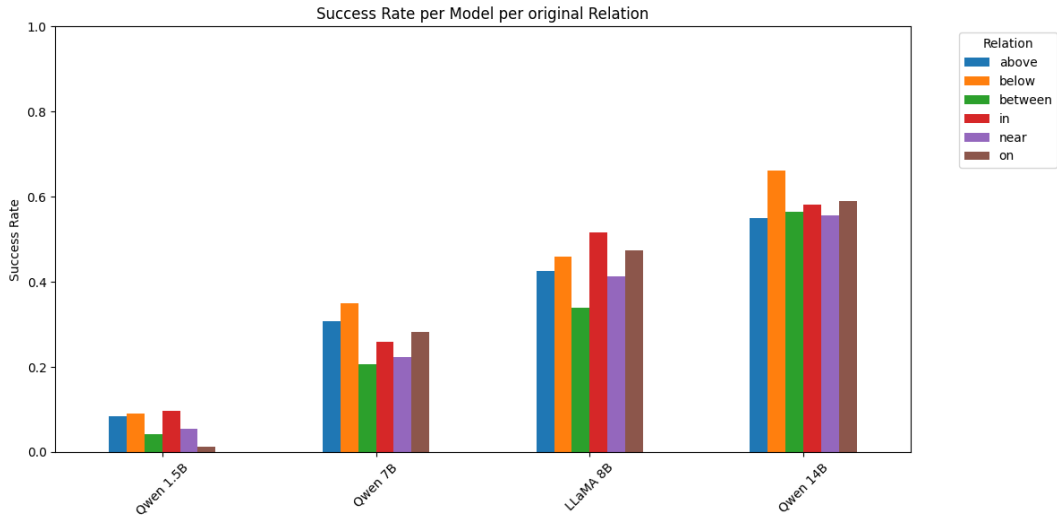


**Figure 14:** Direct comparison between zero-shot and fine-tuned models on Scannet dataset.

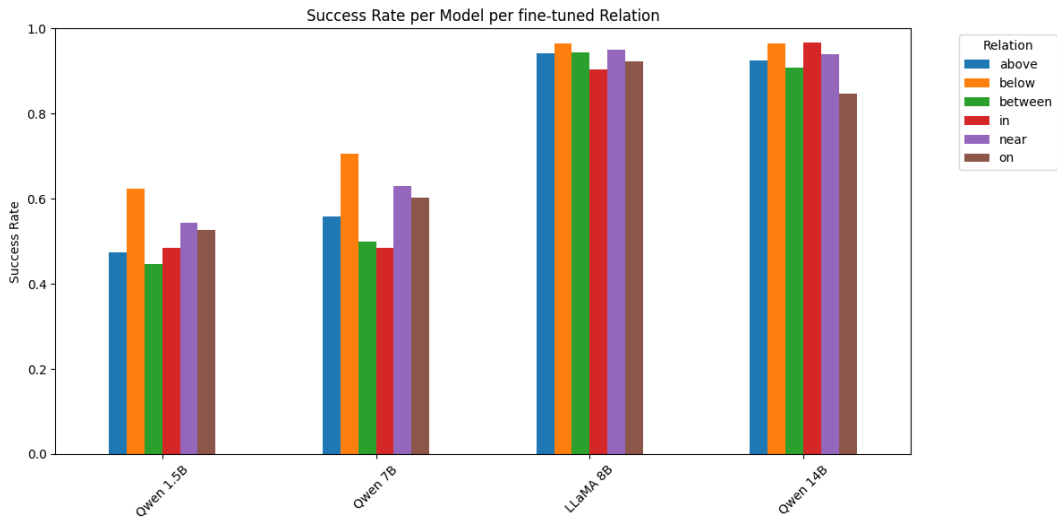
accuracy of 97.0% on this same Unity dataset. For each model, datasets 3RScan and Scannet have almost identical results. This can also be traced to scene sizes, as in figure 8 the similar average sizes for scenes of datasets 3Rscan and Scannet can be seen. Another interesting note is 1.5B model having better results than 7B model on Unity dataset. This is really interesting, as the 7B model otherwise has better performances on larger scenes with Matterport and smaller scenes with 3RScan, ARKitScenes and Scannet. Accuracies on the Matterport dataset have the lowest accuracies overall for a single dataset with accuracies of 9.6% on 1.5B model and 17.4% on 7B model.

Comparing figures 12 and 13 we can see interesting development between zero-shot models and fine-tuned ones. For 8B and 14B models the finetuning led to more constant performances across all datasets, while 1.5B and 7B retained more uneven performances between datasets. Even with fine-tuning 1.5B and 7B models struggle with larger scenes. While fine-tuning led to smaller models having better performances than larger ones on some datasets, the performances on Matterport dataset still improves as size of the models increase.

The results for Scannet dataset with both, zero-shot and fine-tuned models, are shown in figure 14. This is the dataset that was left out from the training data. Meaning, the models have never been trained on statements from scenes of this dataset. We can note, that the fine-tuning has positively affected the performances on this dataset. This is important, as we could use these fine-tuned models on any scenes that follow the scene graph format. However, while the scenes are new to the model, the statements follow the same pattern as models have seen in the training data. From this, we can only conclude that fine-tuning has not over trained the models to work on scenes from the training data.



**Figure 15:** Success rates on each relation with zero-shot models.



**Figure 16:** Success rates on each relation with fine-tuned models.

### 4.2.3 Performance on relations

Figure 15 shows performance of each zero-shot model on different relations. We can see from the figure that different models have varying performances with different relations. 1.5B model performs best with relation *in* with accuracy of 9.7% and worst with relation *on* with accuracy of 1.3%. 7B model has best accuracy of 35.1% with relation *below* and worst accuracy of 20.7% with relation *between*. 8B model is also performing worst on relation *between* with accuracy of 34.0% while best relation for it is *in* with accuracy of 51.6%. 14B model otherwise has even performances on different relations with accuracies of 55.0%, 56.6%, 55.5% for relations *above*, *between* and *near* respectively, but it clearly stands out on relation *below* with accuracy of 66.2%.

Figure 16 presents the performance of each fine-tuned model on different relation types. From the figure it can be noted that relation *below* is easiest for all models with accuracies 62.3% with 1.5B, 70.6% with 7B, 96.5% with both 8B and 14B models. However, the worst performing relations are different for almost all models. 1.5B model performs worst on relation *between* with accuracy of 44.7%. For both the 7B and 8B models, the relation *in* yields the lowest performance, with accuracies of 48.4% and 90.3%, respectively. The 14B model performs worst on relation *on* with accuracy of 84.6%.

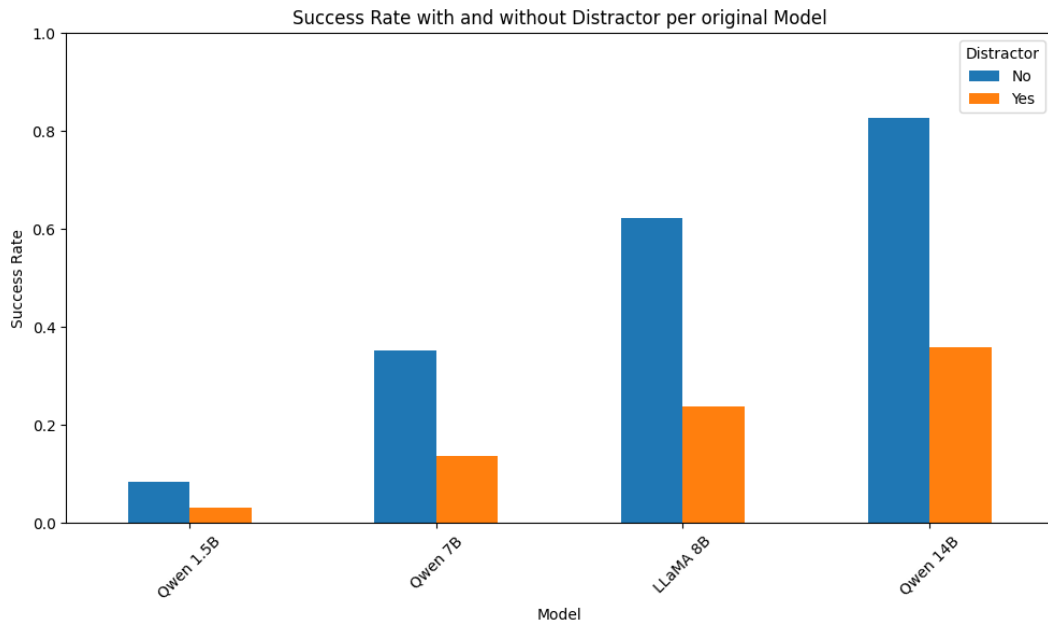
Finetuning process has improved performances of each model on all relations. Fine-tuning evened out performances of 8B model with 6.3% difference between best performing and worst performing relations while before fine-tuning the same value was 17.7%. Other models have not seen same kind of evening of performances. Fine-tuning brought the 1.5B and 7B models to similar performance levels on the relations *above*, *between*, and *in*, although the 7B model still clearly outperforms the 1.5B model on the remaining relations. While zero-shot 8B and 14B models have obvious differences in performances, fine-tuned version have almost identical performances with the exceptions of relations *in* that the 14B is better with and *on* that is better for 8B model.

#### 4.2.4 Performance with and without distractors

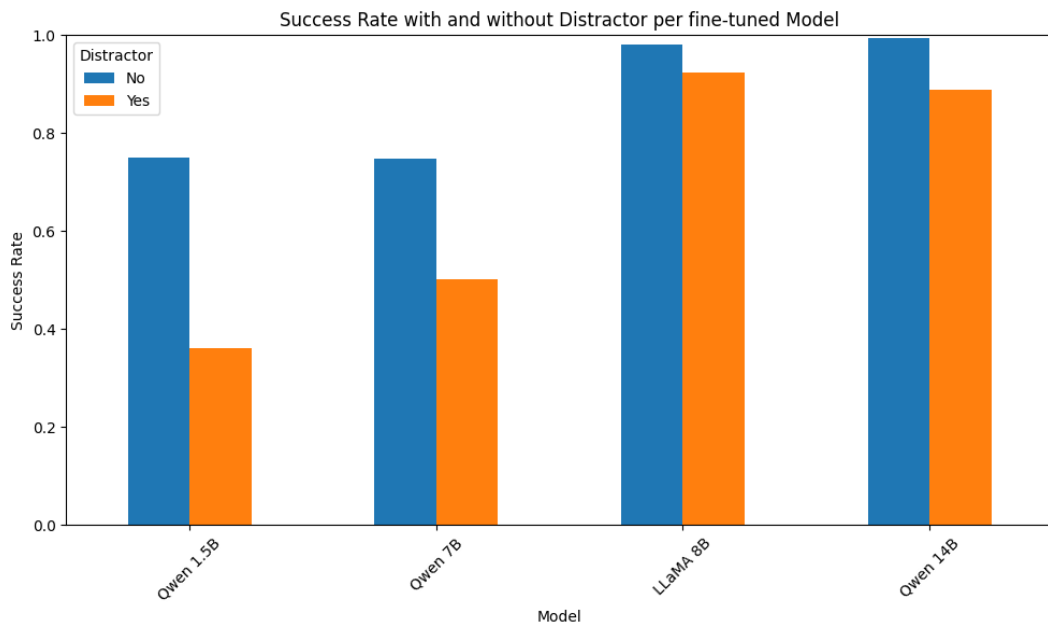
Figure 17a shows performances of zero-shot models on statements with and without distractors. As predicted in section 4.1, all models have more difficulties with statements including distractors. The 1.5B model has accuracy of 8.4% with statements without distractor and 3.3% with them. The 7B model has accuracy of 35.3% on statements without distractor and 13.8% with them. The 8B model follows the same pattern, having accuracies of 62.4% and 23.8% for statements with and without distractors respectively. Best results are with 14B model that has accuracies of 82.8% for statements without distractors and 36.0% with them. Clear correlation between model sizes and performances on statements both, with and without distractors, can be seen.

In figure 17b performances of fine-tuned models on statements with and without distractors can be seen. We can note that 1.5B and 7B models have similar performances on statements without distractors with accuracies of 75.0% and 74.8% respectively, while models 8B and 14B also have similar performances with accuracies of 98.2% and 99.3% respectively. As observed previously, statements containing distractors proved more difficult for all models, having accuracies of 36.1%, 50.3%, 92.3%, and 88.9% for the 1.5B, 7B, 8B, and 14B models, respectively.

Fine-tuning greatly improved performances of models on statements without distractors, making them almost trivial for two biggest models. The performance on statements with distractors was clearly enhanced as well. With fine-tuned models we can see smaller difference between statements with and without distractors when compared to zero-shot models, with fine-tuned 8B model almost having same performances on both. It is interesting to note that, while having best performance overall as seen in table 2, 8B model has worse performance on statements without distractors than the 14B model, even though those statements should be more trivial. Another



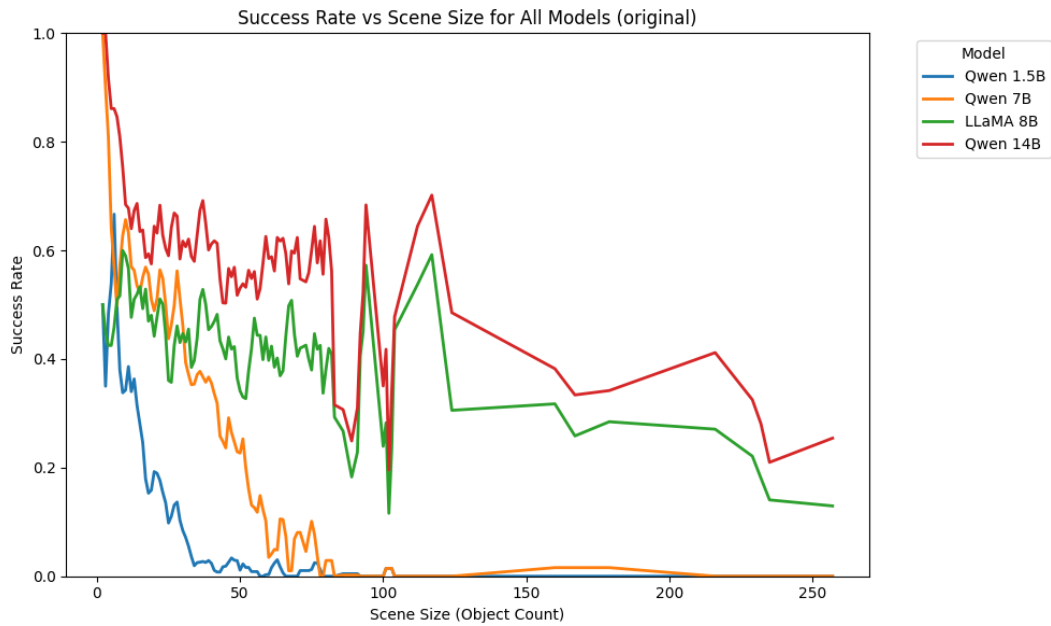
(a) Success rates with and without distractor present for zero-shot models.



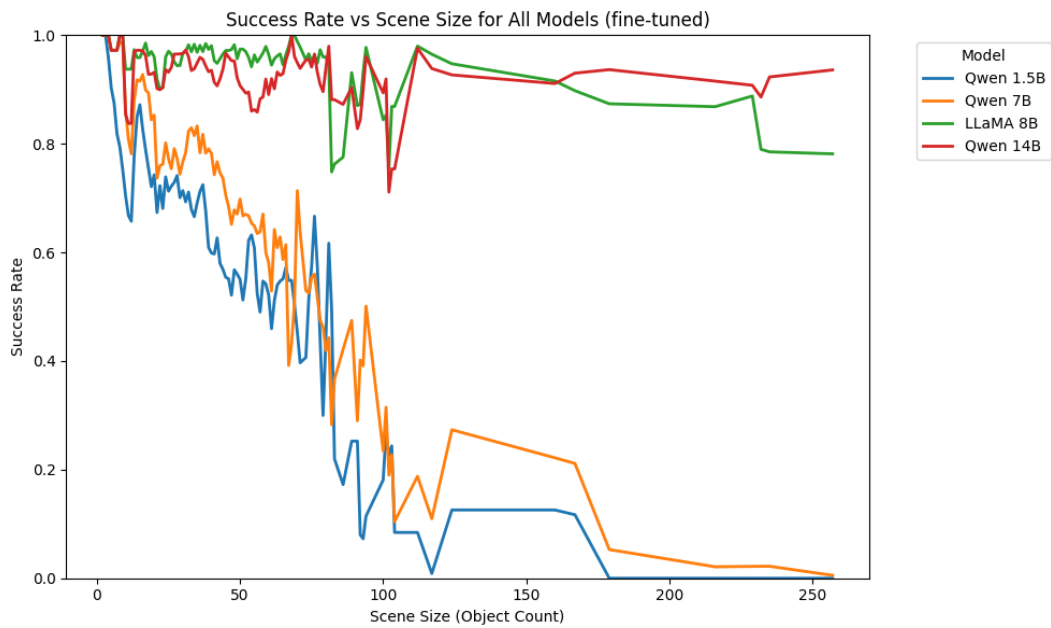
(b) Success rates with and without distractor present for fine-tuned models.

**Figure 17:** Comparison of model success rates with and without distractors for zero-shot and fine-tuned settings.

interesting observation is that the fine-tuned 1.5B model outperforms the 7B model on statements without distractors, while the performance gap becomes more pronounced on statements with distractors.



(a) Success rate compared with scene sizes for zero-shot models.



(b) Success rate compared with scene sizes for fine-tuned models.

**Figure 18:** Comparison of success rates versus scene sizes for zero-shot and fine-tuned models.

#### 4.2.5 Performance on different scene sizes

Figure 18a presents how the performance of the zero-shot models varies with scene size. The 1.5B model can be seen degrading in performance extremely quickly, reaching accuracies of under 10% well below 50 objects per scene. The 7B model

**Table 3:** Accuracies of different-sized models on more free-form statements.

Model	Orig. Acc. (%)	FT Acc. (%)	Abs. Improv. (%)	Rel. Improv. (%)
1.5B	4.9	22.7	17.8	363.3
7B	19.0	27.0	8	42.1
8B	20.2	27.0	6.8	33.7
14B	33.1	49.1	16.0	48.3

degrades similarly, but reaches below 10% accuracies only at over 50 objects per scene. The 8B and 14B models do not degrade as rapidly with increasing scene size and never reach under 10% accuracy for any scene size.

Figure 18b presents the performances of the fine-tuned models relatively to scene sizes. We can notice almost identical plots for 1.5B and 7B models, only having some offset. Their accuracies degrade rapidly when scene sizes get larger and reach almost zero with largest scenes. Models 8B and 14B seem to also have quite similar plots. They both have varying performances with scenes between 0 objects about 120 objects, after which performance of 8B model starts to slowly degrade, while performance of 14B seems to continue steadily.

One interesting note is performances on larger scenes. The model 1.5B shows no improvements at all on larger scenes after fine-tuning, while performance of fine-tuned 7B model still degrades rapidly with larger scenes as seen with zero-shot model. Considering only largest scenes, it seems like fine-tuning did not improve accuracies of 1.5B and 7B models at all. Another noteworthy finding is 14B model having clearly better performance over 8B model when dealing with statements from larger scenes even when 8B model has better performance overall. While in Figure 18a each zero-shot model appears to follow a distinct trend, the fine-tuned models in Figure 18b show much more overlap. Notably, the 1.5B and 7B models show overlapping patterns, as do the 8B and 14B models.

#### 4.2.6 Performance on more free-form statements

The table 3 shows the performances of all models, both zero-shot and fine-tuned, on statements from ViGiL3D-dataset, that are more free-form than the statements from VLA-3D dataset collection. We can instantly see that fine-tuning has improved the accuracies overall with the biggest absolute and relative improvements happening for the 1.5B model. However, the zero-shot 1.5B model has already clearly the poorest performance so it has most room to improve. The zero-shot 7B and 8B models have almost the same performances and 14B model having the best performance. We can see three fine-tuned models, 1.5B, 7B, 8B having similar performances with 7B and 8B having exactly same accuracies. The 14B fine-tuned model has clearly the best accuracy.

It is really interesting to see that fine-tuned 7B and 8B models have the exact same performance when with other testing subset their differences are so drastic as was seen in table 2. From this we can conclude that fine-tuning has improved the 8B model’s understanding of the scene graphs, but it may have over fitted to expecting simple

statements that are used in VLA-3D. Another explanation might be that 7B and 8B models do not have as good semantic capabilities as 14B model and it shows now when we are using more complicated statements.

One interesting question that arises when looking at these results is whether fine-tuning improved the understanding of the scene graphs of the models or have the models now just more likely to answer with any object id included. Sometimes zero-shot models did not instantly give numerical value and spend the limited token amount on "thinking". This might be prominent question for the smaller models.

## 5 Conclusions

This thesis focused on fine-tuning open-source LLMs for processing open vocabulary commands for robotic navigation. In practice this meant retrieving specific objects from scene graphs of environments based on natural language statements referring objects in the scene. We found that the open-source models can be used on task of retrieving objects from a scene graph.

From the results we could see that the fine-tuning was surprisingly efficient for the 8B model, which had accuracy of 95% on VLA-3D based statements. This suggests the model can be used reliably for inputs that closely follow the training structure. However, these results did not transfer to more free-form statements, where the largest, 14B model, had the best results with accuracy of 49%. While this accuracy is better than random guessing, it would not be reliable with more complex tasks as it could fail every other time. We could also note that with the largest scenes 14B model was always best one. We can conclude that if using statements similar to those used in VLA-3D, we could use our models in environments that have scene graphs available. We could also use the models with other type of statements, but with inferior results.

Our method was limited, by training data availability, to singular object retrieval task from scene graph. This also led to fine-tuned models being limited to specific kinds of statements. A more thorough parameter search across multiple hyperparameters, rather than focusing solely on the learning rate, was limited by the available computing resources. No statistical significance testing was conducted, so the reliability of close performance differences remains uncertain.

Our study showed that finetuned open-source LLMs can be utilized for robotic navigation tasks. While our finetuning improves performances on free-form statements, the models perform significantly better on statements that follow the formats of the statements from the training data. Based on these findings, several open questions remain for future research: How to construct more diverse datasets to support better generalization to more free-form language input? How different types of models perform under such conditions? How these models can be adapted for long-horizon planning tasks? Additionally, grounding techniques for aligning LLM outputs with physical environments remain a valuable direction for further exploration, particularly those that can operate effectively using as few tokens as possible.

## **Acknowledgements**

In this thesis, Large Language Models were utilized in the following ways:

- As coding assistants (GPT-4o, Copilot)
- For grammar checking (GPT-4o)
- For rephrasing sentences (GPT-4o)
- For reforming references (GPT-4o, Copilot)

## References

- [1] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, *Challenges and Applications of Large Language Models*, preprint, Jul. 19, 2023, arXiv: arXiv:2307.10169. doi: 10.48550/arXiv.2307.10169.
- [2] D. Shah, B. Osiński, B. Ichter, and S. Levine, *LM-Nav: Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action*, Proceedings of The 6th Conference on Robot Learning, PMLR, Mar. 2023, pp. 492–504. Accessed: Jan. 31, 2025. [Online]. Available: <https://proceedings.mlr.press/v205/shah23b.html>
- [3] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. D. Reid, and N. Sünderhauf, *SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Task Planning*, 7th Proc. Conf. Robot Learn. (CoRL), pp. 23-72, Nov. 2023, Atlanta, GA, USA.
- [4] S. Song, S. Kodagoda, A. Gunatilake, M. G. Carmichael, K. Thiyagarajan, and J. Martin, *Guide-LLM: An Embodied LLM Agent and Text-Based Topological Map for Robotic Guidance of People with Visual Impairments*, preprint, Oct. 28, 2024, arXiv: arXiv:2410.20666. doi: 10.48550/arXiv.2410.20666.
- [5] G. Zhou, Y. Hong, and Q. Wu, *NavGPT: Explicit Reasoning in Vision-and-Language Navigation with Large Language Models*, Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 7, Art. no. 7, Mar. 2024, doi: 10.1609/aaai.v38i7.28597.
- [6] OpenAI et al., *GPT-4 Technical Report*, preprint, Mar. 04, 2024, arXiv: arXiv:2303.08774. doi: 10.48550/arXiv.2303.08774.
- [7] H. Naveed, A. Ullah Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, *A Comprehensive Overview of Large Language Models*, ACM Trans. Intell. Syst. Technol., June 2025, doi: 10.1145/3744746.
- [8] J. Zhao et al., *LoRA Land: 310 Fine-tuned LLMs that Rival GPT-4, A Technical Report*, preprint, Apr. 29, 2024, arXiv: arXiv:2405.00732. doi: 10.48550/arXiv.2405.00732.
- [9] Y. Chen, Y. Han, and X. Li, *FASTNav: Fine-Tuned Adaptive Small-Language- Models Trained for Multi-Point Robot Navigation*, IEEE Robotics and Automation Letters, vol. 10, no. 1, pp. 390–397, Jan. 2025, doi: 10.1109/LRA.2024.3506280.
- [10] H. Zhang, N. Zantout, P. Kachana, Z. Wu, J. Zhang, and W. Wang, *VLA-3D: A Dataset for 3D Semantic Scene Understanding and Navigation*, preprint, Nov. 05, 2024, arXiv: arXiv:2411.03540. doi: 10.48550/arXiv.2411.03540.

- [11] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, *Improving Language Understanding by Generative Pre-Training*, Jun. 11, 2018, OpenAI Technical Report. Accessed: Apr. 28, 2025. [Online]. Available: [https://cdn.openai.com/research-covers/language-unsupervised/language\\_understanding\\_paper.pdf](https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf)
- [12] J. Kaplan et al., *Scaling Laws for Neural Language Models*, preprint, Jan. 23, 2020, arXiv: arXiv:2001.08361. doi: 10.48550/arXiv.2001.08361.
- [13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017.
- [14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), vol. 1 (Long and Short Papers), Minneapolis, MN, Jun. 2019, pp. 4171–4186, doi: 10.18653/v1/N19-1423.
- [15] T. B. Brown et al., *Language Models are Few-Shot Learners*, preprint, Jul. 22, 2020, arXiv: arXiv:2005.14165. doi: 10.48550/arXiv.2005.14165.
- [16] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, *Language Models are Unsupervised Multitask Learners*, Feb. 14, 2019, OpenAI Technical Report. Accessed: Apr. 28, 2025. [Online]. Available: [https://cdn.openai.com/better-language-models/language\\_models\\_are\\_unsupervised\\_multitask\\_learners.pdf](https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf)
- [17] K. S. Kalyan, *A Survey of GPT-3 Family Large Language Models Including ChatGPT and GPT-4*, preprint, Oct. 04, 2023, arXiv: arXiv:2310.12321. doi: 10.48550/arXiv.2310.12321.
- [18] DeepSeek-AI et al., *DeepSeek-V3 Technical Report*, preprint, Feb. 18, 2025, arXiv: arXiv:2412.19437. doi: 10.48550/arXiv.2412.19437.
- [19] M. Abdin et al., *Phi-4 Technical Report*, preprint, Dec. 12, 2024, arXiv: arXiv:2412.08905. doi: 10.48550/arXiv.2412.08905.
- [20] V. B. Parthasarathy, A. Zafar, A. Khan, and A. Shahid, *The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities*, preprint, Oct. 30, 2024, arXiv: arXiv:2408.13296. doi: 10.48550/arXiv.2408.13296.
- [21] B. Y. Lin et al., *The Unlocking Spell on Base LLMs: Rethinking Alignment via In-Context Learning*, Proc. 12th Int. Conf. Learn. Represent. (ICLR), May 2024.

- [22] T. Hui, Z. Zhang, S. Wang, W. Xu, Y. Sun, and H. Wu, *HFT: Half Fine-Tuning for Large Language Models*, Proc. 63rd Annu. Meeting Assoc. Comput. Linguist. (ACL), Jul. 2025.
- [23] E. J. Hu et al., *LoRA: Low-Rank Adaptation of Large Language Models*, Proc. 10th Int. Conf. Learn. Represent. (ICLR), pp. 9906–9930, Apr. 2022.
- [24] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, *Large Language Models are Zero-Shot Reasoners*, Proc. Adv. Neural Inf. Process. Syst. 35 (NeurIPS), pp. 22199–22213, 2022.
- [25] P. Lewis et al., *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, Proc. Adv. Neural Inf. Process. Syst. 33 (NeurIPS), pp. 9459–9474, 2020.
- [26] Mistral AI team, *Large Enough | Mistral AI*, Jul. 24, 2024, Accessed: May 24, 2025. [Online]. Available: <https://mistral.ai/news/mistral-large-2407>
- [27] A. Grattafiori et al., *The Llama 3 Herd of Models*, preprint, Nov. 23, 2024, arXiv: arXiv:2407.21783. doi: 10.48550/arXiv.2407.21783.
- [28] Qwen Team, *Qwen2.5: A Party of Foundation Models*, Sep. 2024, Accessed: May. 10, 2025. [Online]. Available: <https://qwenlm.github.io/blog/qwen2.5>
- [29] DeepSeek-AI et al., *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*, preprint, Jan. 22, 2025, arXiv: arXiv:2501.12948. doi: 10.48550/arXiv.2501.12948.
- [30] I. Armeni et al., *3D Scene Graph: A Structure for Unified Semantics, 3D Space, and Camera*, in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Oct. 2019, pp. 5663–5672. doi: 10.1109/ICCV.2019.00576.
- [31] X. Chang, P. Ren, P. Xu, Z. Li, X. Chen, and A. Hauptmann, *A Comprehensive Survey of Scene Graphs: Generation and Application* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 1, pp. 1–26, Jan. 2023, doi: 10.1109/TPAMI.2021.3137605.
- [32] V. N. Devarakonda et al., *OrionNav: Online Planning for Robot Autonomy with Context-Aware LLM and Open-Vocabulary Semantic Scene Graphs*, preprint, Oct. 23, 2024, arXiv: arXiv:2410.06239. doi: 10.48550/arXiv.2410.06239.
- [33] A. Rajvanshi, K. Sikka, X. Lin, B. Lee, H.-P. Chiu, and A. Velasquez, *SayNav: Grounding Large Language Models for Dynamic Planning to Navigation in New Environments*, Proceedings of the International Conference on Automated Planning and Scheduling, vol. 34, pp. 464–474, May 2024, doi: 10.1609/icaps.v34i1.31506.

- [34] P. Wu et al., *VoroNav: Voronoi-based Zero-shot Object Navigation with Large Language Model*, Proceedings of the 41st International Conference on Machine Learning (ICML 2024), 2024, Vienna, Austria.
- [35] J. Li, D. Li, C. Xiong, and S. Hoi, *BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation*, Proceedings of the 39th International Conference on Machine Learning (ICML), vol. 162, Baltimore, MD, Jul. 2022, pp. 12888–12900.
- [36] B. Yu, H. Kasaei, and M. Cao, *L3MVN: Leveraging Large Language Models for Visual Target Navigation*, 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2023, pp. 3554–3560. doi: 10.1109/IROS55552.2023.10342512.
- [37] V. S. Dorbala, J. F. Mullen, and D. Manocha, *Can an Embodied Agent Find Your ‘Cat-shaped Mug’? LLM-Based Zero-Shot Object Navigation*, IEEE Robotics and Automation Letters, vol. 9, no. 5, pp. 4083–4090, May 2024, doi: 10.1109/LRA.2023.3346800.
- [38] L. H. Li et al., *Grounded Language-Image Pre-training*, 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2022, pp. 10955–10965. doi: 10.1109/CVPR52688.2022.01069.
- [39] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2016, pp. 779–788. doi: 10.1109/CVPR.2016.91.
- [40] E. Latif, *3P-LLM: Probabilistic Path Planning using Large Language Model for Autonomous Robot Navigation*, preprint, Mar. 27, 2024, arXiv: arXiv:2403.18778. doi: 10.48550/arXiv.2403.18778.
- [41] R. Schumann, W. Zhu, W. Feng, T.-J. Fu, S. Riezler, and W. Y. Wang, *VELMA: Verbalization Embodiment of LLM Agents for Vision and Language Navigation in Street View*, Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 17, Art. no. 17, Mar. 2024, doi: 10.1609/aaai.v38i17.29858.
- [42] Y. Zhu et al., *ChatNav: Leveraging LLM to Zero-shot Semantic Reasoning in Object Navigation*, IEEE Transactions on Circuits and Systems for Video Technology, pp. 1–1, 2024, doi: 10.1109/TCSVT.2024.3485907.
- [43] H. Yin, X. Xu, Z. Wu, J. Zhou, and J. Lu, *SG-Nav: Online 3D Scene Graph Prompting for LLM-based Zero-shot Object Navigation*, Proc. Adv. Neural Inf. Process. Syst. 37 (NeurIPS), Dec. 2024, New Orleans, LA, USA. Curran Associates, Inc.
- [44] A. Yang et al., *Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement*, preprint, Sep. 18, 2024, arXiv: arXiv:2409.12122. doi: 10.48550/arXiv.2409.12122.

- [45] A. Yang et al., *Qwen2 Technical Report*, preprint, Sep. 10, 2024, arXiv: arXiv:2407.10671. doi: 10.48550/arXiv.2407.10671.
- [46] J. Wald, A. Avetisyan, N. Navab, F. Tombari, and M. Nießner, *RIO: 3D Object Instance Re-Localization in Changing Indoor Environments*, Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, Seoul, South Korea. IEEE.
- [47] G. Baruch et al., *ARKitScenes: A Diverse Real-World Dataset For 3D Indoor Scene Understanding Using Mobile RGB-D Data*, preprint, Jan. 12, 2022, arXiv: arXiv:2111.08897. doi: 10.48550/arXiv.2111.08897.
- [48] A. Chang et al., *Matterport3D: Learning from RGB-D Data in Indoor Environments*, 2017 International Conference on 3D Vision (3DV), Oct. 2017, pp. 667–676. doi: 10.1109/3DV.2017.00081.
- [49] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, *ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes*, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jul. 2017, pp. 2432–2443. doi: 10.1109/CVPR.2017.261.
- [50] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen, *What Makes Good In-Context Examples for GPT-3?*, Proc. Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, pp. 100–114, May 2022, Dublin, Ireland and Online. Assoc. for Computational Linguistics.
- [51] J. He, M. Rungta, D. Koleczek, A. Sekhon, F. X. Wang, and S. Hasan, *Does Prompt Formatting Have Any Impact on LLM Performance?*, preprint, Nov. 15, 2024, arXiv: arXiv:2411.10541. doi: 10.48550/arXiv.2411.10541.
- [52] A. T. Wang, Z. Gong, and A. X. Chang, *ViGiL3D: A Linguistically Diverse Dataset for 3D Visual Grounding*, preprint, Jan. 02, 2025, arXiv: arXiv:2501.01366. doi: 10.48550/arXiv.2501.01366.

## A Prompt

System: You are given a user prompt and a scene graph. Your task is to retrieve the object\_id from the scene graph described in the user prompt. The output should only be the desired object\_id. Here is an example:

Description of the object: The red book that is on the sofa. This is the graph:

---

```
{
  "scene_name": "example_scene",
  "regions": {
    "0": {
      "region_id": "0",
      "region_name": "living_room",
      "objects": [
        { "object_id": "0", "raw_label": "floor", "color_labels": ["brown", "N/A", "N/A"] },
        { "object_id": "1", "raw_label": "book", "color_labels": ["green", "N/A", "N/A"] },
        { "object_id": "2", "raw_label": "sofa", "color_labels": ["black", "white", "N/A"] },
        { "object_id": "3", "raw_label": "book", "color_labels": ["red", "N/A", "N/A"] }
      ],
      "relationships": {
        "above": { "0": ["1", "2"], "2": ["3"] },
        "on": { "0": ["1", "2"], "2": ["3"] },
        "near": { "2": ["1", "2"] }
      }
    }
  }
}
```

---

Output: 3

End of an example

User:

Description of the object:

...

This is the graph:

...

###Output: