



MENG INDIVIDUAL PROJECT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Large Language Models as Planners for Robots

Author:
Ruari Phipps

Supervisor:
Dr. Edward Johns

Second Marker:
Dr. Antoine Cully

June 20, 2022

Abstract

Traditional planners in Robotics rely on complex specifications of the environment and the goal, which make them difficult to create and understand. The most understandable way of defining goals is in natural language, but due to its nature this is often hard to comprehend in programs.

Recent developments in Large Language Models have enabled complex deep learning models to be able to model and represent language extremely well and have shown their ability to be used in many different tasks including text classification, text generation and translation. These models come pre-trained and can be used on downstream tasks with further fine tuning or by giving examples of the task in the input to the models.

In this project, the capabilities of these Large Language Models are explored by looking into if they can produce plans detailing how to complete a high-level instructions in mid-level instructions as well as in encoding those mid-level instructions in a way that robot can execute them.

Other implementations of language model based planning systems have grounded the language systems using additional systems to manage or convert the output into commands executable by their robot. In this project it is shown that, at least for a simple grab and place robot in a simple environment, the grounding of the robot can be done inside the input to language model by providing the abilities of the robot and a description of its environment.

A planning system is created that utilises this ability to plan to get a robot trained on simple mid-level tasks to achieve various actions. This planning system utilises many different abilities of the Large Language model to implement including translation between known and unknown objects, error correction and impossible task detection and can successfully apply these to various tasks across different scenarios.

Acknowledgements

Thank you to Norman Di Palo, who helped throughout the project explore plenty of ideas and gave me guidance when I was stuck.

I would also like to thank my family, who gave plenty of motivation and tips during the project which helped keep me keep driving me forward.

Thanks also to my friends, whose support and their ability to entertain has allowed me to keep focused and on track whenever times got tough.

And finally, I would like to thank my kettle, for all the cups of tea it made. I would never have made it through this without it.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 3 |
| 1.1 | Motivation | 3 |
| 1.2 | Objectives | 4 |
| 1.3 | Contributions | 4 |
| 2 | Preliminaries | 5 |
| 2.1 | Deep Learning | 5 |
| 2.1.1 | Convolutional Neural Networks | 5 |
| 2.1.2 | Variable Sized Inputs | 7 |
| 2.2 | Natural Language Processing | 9 |
| 2.2.1 | Word Embedding | 9 |
| 2.2.2 | Language Models | 11 |
| 2.3 | Training Robots | 13 |
| 3 | Related Work | 14 |
| 3.1 | Language to Control Robots | 14 |
| 3.2 | Traditional Planners for Robots | 14 |
| 3.3 | Using Language Models as Planners | 15 |
| 3.4 | Language models as Planners for Robots | 16 |
| 4 | Implementation | 17 |
| 4.1 | Overview | 17 |
| 4.2 | Creating the Robot Model | 17 |
| 4.2.1 | Single object | 18 |
| 4.2.2 | Multi-object | 22 |
| 4.2.3 | Multi-policy | 24 |
| 4.3 | Using the Language Model | 29 |
| 4.3.1 | API | 29 |
| 4.3.2 | Capabilities of the Model | 30 |
| 4.3.3 | Designing the input | 33 |
| 4.3.4 | Experiments | 35 |
| 4.4 | Robot Planning System | 36 |
| 4.4.1 | Detecting Failures | 37 |
| 4.4.2 | Context | 37 |
| 4.4.3 | Experiments | 38 |
| 4.4.4 | Evaluation | 39 |
| 5 | Conclusion | 41 |
| 5.1 | Insights gained | 41 |
| 5.2 | Future work | 41 |
| 5.3 | Final remarks | 42 |
| 6 | Ethical Issues | 43 |
| A | Detailed results for multi-policy model | 48 |
| B | LLM Test Results | 52 |

Chapter 1

Introduction

1.1 Motivation

Robot planning methods require a detailed model of the environment as well as understanding of how the robot's actions affects it (1). They also require a way of expressing the current state of the environment, with this state being used in both in the planning process and in expressing start states and goals. When these models are created, it allows for a reliable planning process which normally uses an optimisation method to the system figure out the steps needed to achieve it's goal. However, creating such a system is extremely difficult, especially when dealing with a robot in the real world. This is because the real world has too many variables, so being able to find a suitable state representation and create an accurate model is impossible and models with assumptions have to be made, which reduce reliability. Furthermore, goals are difficult to quantify in each plan and have to be manually engineered for the robot to recognise them. An easier way of representing these goals is in the form of natural language commands, such as "move the apple from the bowl to the plate", however as natural language is very complex it is difficult to convert this command into something that can be used inside a robot planner.

Recent advances in Natural Language Programming and deep learning (2) have allowed for the creation of models known as large language models (3, 4, 5, 6). These models are trained on vast amounts of data to be used on the very simple task of given current input, predict the next word. This task however has many uses across many different tasks including text generation, text classification and error correction. These models have also shown their capability as few-shot learners (5), where when given an example of how to perform a task in their input, they can then attempt to perform that task in the output utilising the example and the large quantity of general knowledge it gained during training.

Large Language Model Input:

English: My name is Ruari Phipps

Spanish: Mi nombre es Ruari Phipps

English: I am writing my final year project

Large Language Model Output:

Spanish: Estoy escribiendo mi proyecto de último año

Some work has been done in investigating if a large language model can use it's knowledge acquired by training , and it's ability as a few-shot learner to break down high-level commands, into a series of mid-level actions that relate to different action that a robot can take (7, 8, 9). These works showcase that the model is capable being used for such as task. However, these works use additional systems to ensure the model outputs actions relating to its current environment (grounds the model) rather than trying to use the models ability as a few-shot learner to ground the model using it's input. With robot systems being able to perform unseen actions based on text alone (10), these systems limit the model in it's output which makes them less able to adapt to new scenarios and environments as well as come up with creative solutions to solve tasks.

1.2 Objectives

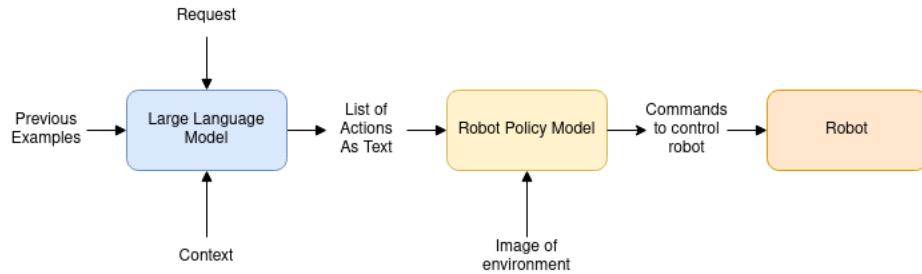


Figure 1.1: Overview of planning system

This project aims to achieve a variety of objectives:

- To create a grab and place robot that when given a text input can perform the related action.
- To investigate the ability for a large language model, grounded using only examples given in the input, to plan a series of mid-level instructions for a robot to perform to reach a high-level goal specified in natural language.
- To create a planning robot combining both the robot and the large language model to have it execute high-level commands across different scenario, seen in 1.1.

1.3 Contributions

While attempting to achieve these goals, several important insights and contributions were discovered on how language models could be used within robots and planning systems for robots.

- That large language models used in robot policy models allow for them to be more flexible and adapt to different inputs. Section 4.2
- Large language models can be grounded using just examples given in the input and can produce reliably planned outputs for simple tasks and tasks that involve some reasoning. Section 4.3
- Large language models can demonstrate this knowledge of the grounding by doing impossible task detection, where it won't output a plan if the high-level task is impossible in the context. Section 4.3.3
- The translation ability of large language models can be used to apply robots trained with one set of objects to a different set of objects by telling it how to relate similar objects. Section 4.3.3
- Large language models can be used as error recovery tools when combined with failure detection skills to help a robot complete a plan if it failed to execute one of the steps. Section 4.4.3

Chapter 2

Preliminaries

2.1 Deep Learning

Machine learning and the use of neural networks have allowed for the creation of models to approximate unknown functions using large portions of data. As computers became more capable and techniques developed, it allowed for these models to be more powerful and to represent more complicated functions.

However, in classical machine learning the data input is very structured and normally requires experts designing the set of features to be used as the data input for the models. For example, one feature could be whether an image contains a tree. A subset of the area of machine learning is deep learning. This differs by having the learning process also learn the features to extract from the data as well as the model. This is often achieved by having a larger model (i.e. a neural network with 3 or more hidden layers is considered deep (11)) or by adding more complexity to the model with additional techniques. The advantage of deep learning is it allows models to be more easily used on unstructured data such as text and images.

In this project, models will be required to take inputs with instructions for the robot as text and images of the robot's current environment in order to get the robot to undertake these actions. These models will employ deep learning techniques to allow it to process these unstructured inputs and to learn the model that will best fit the problem.

2.1.1 Convolutional Neural Networks

Convolutional neural networks are a type of network that contain several convolutional layers that are used to process unstructured data where the order and shape of the data is important, such as signals and images.

Convolutional layers apply a convolution function between the input f and a learned set of weights called a kernel g .

$$(f * g)(t) := \int_{-\infty}^{\infty} f(t - \tau)g(t - \tau)d\tau \quad (2.1)$$

The convolution is normally implemented by having the kernel be of a fixed size (by having values outside that size be 0) so the values of the kernel can be learned by back propagation of losses. It can be thought of as applying a weighted average of each element and its neighbours, as seen on a 2-dimensional input in figure 2.1.

A convolutional neural network is made up with a combination of these layers, as well as pooling layers, activation functions and fully connected layers as seen in traditional neural networks. Pooling functions, such as average pooling or max pooling, take local averages and maximums and allow the system to be more invariant to small shifts in the input. They can also allow the output to be down sampled to a smaller size which can help learning by forcing it to distil the input into a smaller number of features.

Simple convolution network architectures can be useful for simpler tasks, but it has been found that deeper architectures work better for harder challenges. AlexNet (12) and VGG (13) improved upon the state of the art by increasing the number of layers and thus improving performance across several image recognition challenges.

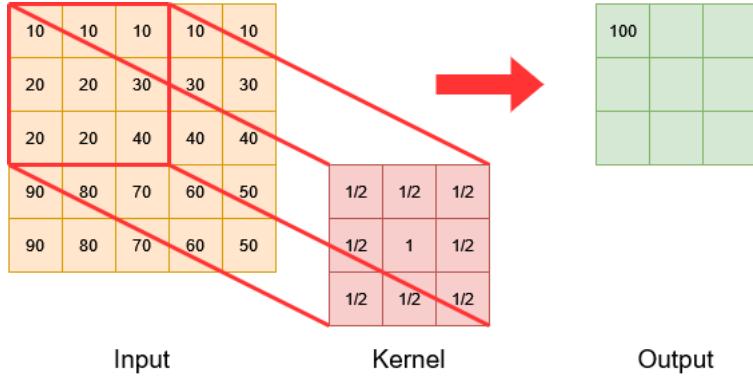


Figure 2.1: A convolution

ResNet An architecture that improved further on the state of the art is ResNet. This type of model uses a key innovation to improve, residual blocks. A block in a network is a pre-designed series of layers that can be repeated several times throughout a network to speed up development and allow for more interesting architectures.

Residual blocks were created with an aim to address a problem seen in traditional deep models, where increasing the number of layers can sometimes cause higher training and test errors on datasets (14). This is referred to as degradation of accuracy and is believed to be caused as current optimisation methods are unable to find better models in a cost-efficient time for deeper models (15). Residual blocks work by adding shortcut layers. This allows each block to estimate the mapping $H(x)$ by trying to estimate the residual function:

$$F(x) := H(x) - x \quad (2.2)$$

Thus, the output of each block would be $F(x) + x$. A network of these blocks would then be able to model function in a similar structure to a Taylor expansion, meaning that adding further blocks would be improving the approximation capabilities of the model.

Residual networks can be constructed using different types of blocks, but all take roughly the form of one of the following:

$$y = F(x, W_i) + x \quad y = F(x, W_i) + W_s x \quad (2.3)$$

The matrix W_s is used to map the input x to have the same dimension as the output y when the dimensions of the input and output don't match. The ResNet-18 architecture uses blocks seen in figure 2.2, which utilise the batch norm layer (16) as a form of regularisation (17), and the whole architecture in 2.3. However, the internal dynamics of each block isn't fixed and is often varied depending on performance on the required task.

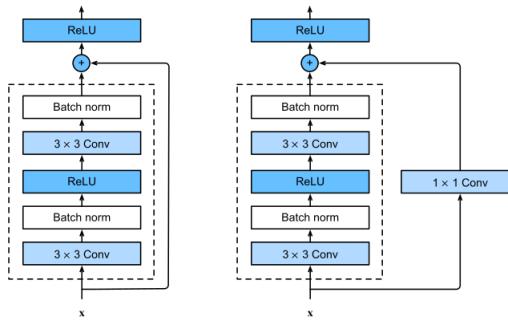


Figure 2.2: Example ResNet Block Architecture (18, p. 290)

Models that use this method are found to achieve better accuracy over non residual models (14) when tested on the Imagenet dataset and the deeper the models, the more accurate they were.

FiLM Most convolutional neural networks usually only have one input, being the signal/image that is convolved over. In some problems it might be required to have another input to condition

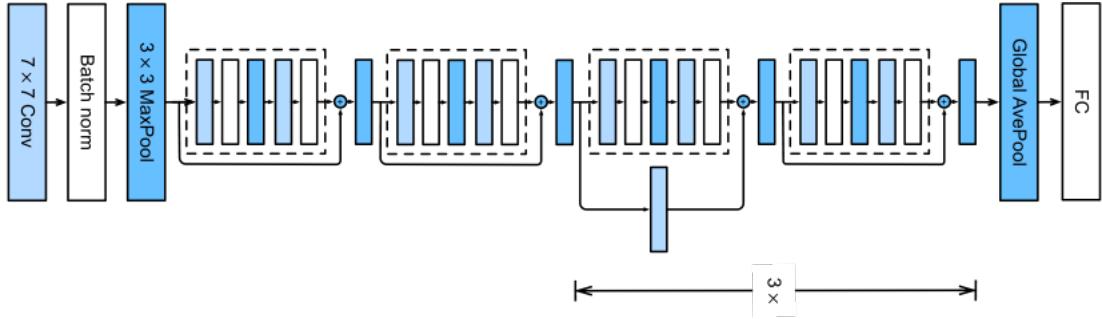


Figure 2.3: ResNet-18 Architecture (18, p. 292)

the main convolutions, for example being able to perform visual reasoning by answering questions about images.

A FiLM (Feature-wise Linear Modulation) (19) is a way to achieve this. A FiLM layer is a layer that can be added to general model architectures that does feature-wise affine transformations between the main input and the input it is being conditioned on.

To do this, the network learns the functions:

$$\gamma_{i,c} = f_c(x_i) \quad \beta_{i,c} = h_c(x_i) \quad (2.4)$$

Where x_i is the input to be conditioned on and c represents the c^{th} FiLM layer. The outputs of this function can be fed into the FiLM layer which will modulate a networks output $F_{i,c}$:

$$\text{FiLM}(F_{i,c}| \gamma_{i,c}, \beta_{i,c}) = \gamma_{i,c} F_{i,c} + \beta_{i,c} \quad (2.5)$$

The functions f and h can be estimated in any way, including with neural networks. These layers can be inserted throughout any type of model and have been shown to performs better than other approaches on visual reasoning test sets (19), such as CLEVR (20), when incorporated into the residual blocks in a ResNet based architecture.

2.1.2 Variable Sized Inputs

Classical neural networks and architectures, such as convolutional neural networks, only deal with fixed size inputs. For problems that can involve variable inputs, such as text processing, these architectures are not ideal. Some workarounds do exist, such as padding the input, so it is always a certain size, but processes like these struggle with performance and are still limited by a maximum input length.

Recurrent Neural Networks RNNs are an architecture used to process these variable input lengths. They work by learning a function that when sequentially taking an input x_i along with a hidden state c_i it will produce an output h_i as well as a hidden state for the next input c_{i+1} . This can be seen in figure 2.4. It can be trained in the same way as other neural networks, using back propagation, and can theoretically work with inputs of infinite length.

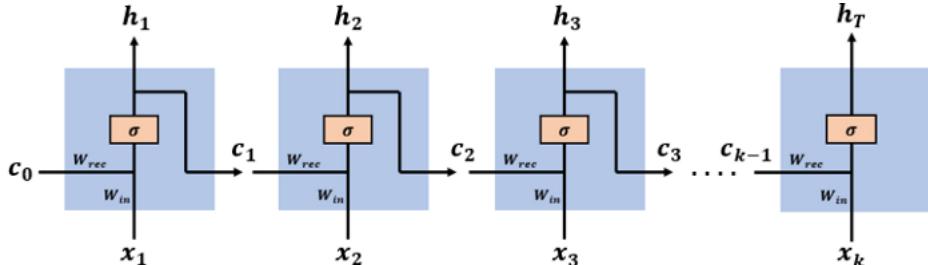


Figure 2.4: An RNN being run on an input sequence, (21)

Recurrent Neural Networks can be used in many different ways; in a many-to-one set-up, where many inputs are used to get one output, such as a text classifier; A one-to-many set-up, where

one output is used to produce many outputs, for example a text decoder that uses an input as a starting point to produce text; or in a many-to-many input, where each new input results in a new output, such as part of speech tagging.

RNNs do have several limitations', they struggle with vanishing gradient in the learning process when faced with long input sequences (21). Also, due to how inputs must be processed one by one, it limits the opportunity for parallelism and makes training time increase (2). Another problem is that they struggle to learn long range dependencies in inputs, as the data path between inputs can be long (2).

Gated models, such as Gated Recurrent Units (GRUs) (22) and Long Short Term Memory (LSTMs) (23), help with this problem by adding gates to the network. This allows for the network to better choose how and what data is carried through the network, improving their ability to learn long range dependencies, and also adding a path to always propagate a loss to reduce the chance of vanishing or exploding gradient.

One common set-up for RNNs is an encoder/decoder system. This system is good for problems such as translation or text generation from an initial input. In this system the input sequence is encoded input by input with the output of the final node being used as an encoded representation of the whole input. This can then be fed into a decoder network as the hidden state along with an initial input. Encoder/decoder networks have lots of uses, with many in Natural Language Processing.

Attention Attention is a mechanism that was created to be used in recurrent neural network encoders/decoders networks (22). In most RNN encoder/decoder networks, the input to the whole decoder would be the output of the last unit of the encoder. Attention allows the decoder to access all outputs of the encoder network and then decide and use the outputs it thinks are most relevant.

It works by having each bit of information, the "values" have an associated "key" which describes what's in each value. Then it will compare the keys to an input query and output the values associated with whatever keys are the best.

A common method of attention is called Scaled Dot-Product Attention" (2) and gives the output from K key V value Q query inputs using the following:

$$\text{ATTENTION}(Q, K, V) = \sigma \left(\frac{QK^T}{\sqrt{d_k}} \right) V \quad (2.6)$$

Where d_k is the dimension of K and σ is an appropriate activation function. Activation functions can be created to only output the value where the key and query match the most or an activation function, such as the softmax function, can be used to output a sum of the most likely values.

This mechanism helped recurrent networks by introducing a "memory" to help it query old information. This helped it with longer sequences where information might be lost from long data paths.

While initially used only in decoder networks, the attention mechanism was found very useful when encoding data as well.

Transformers Transformers were introduced by Vaswani et al. (2). In this paper they utilise attention in both the encoder and the decoder which created the ability to create a very powerful language model (discussed in 2.2.2).

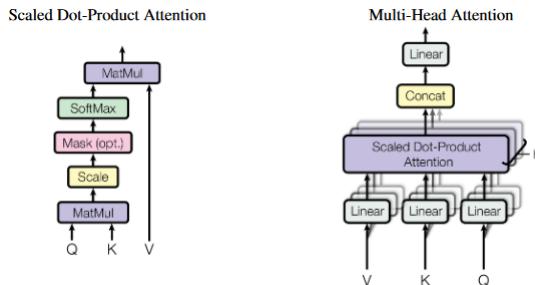


Figure 2.5: Attention layer, from (2)

A transformer encoder differs from usual RNN encoders in two ways, the first was that it no longer takes the input sequentially, but instead all at once, then using a positional encoder so it

can still work out the order. This meant it could be trained much more efficiently and created a smaller data path for long length inputs which helped reduce the chances of vanishing gradient. The other was using attention inside the encoder. It does this in a process called self-attention, where the keys, values and queries are produced from the same input, by mapping each input to its own key, value and query using some learned weights. This helped better learn the connections between different parts of the input.

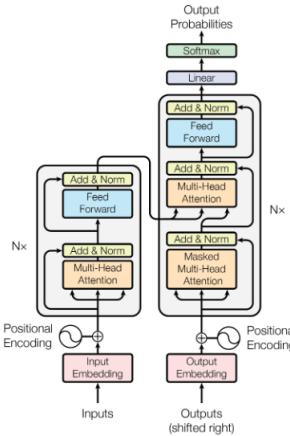


Figure 2.6: Transformer network architecture, from (2). The left side is the encoder, and the right side is the decoder

The decoder also works quite like a traditional decoder. It also uses some of the same techniques used in the encoder, such as the positional encoder and self-attention, but also uses a traditional attention mechanism for it to be able to query the encoded input. The decoding process works in a similar way to in a RNN, by producing one output at a time, but instead of feeding just that new input into the network it would feed all the current outputs which would allow it to reference what it has previously outputted using the self-attention network. The architectures of the transformer encoder and decoder can be seen in figure 2.6.

This network was able to establish itself as a state of the art, outperforming other approaches various tasks such as machine translation tasks (2). Larger and more complicated transformers would be produced, and it is currently considered the best way to model language problems.

2.2 Natural Language Processing

Natural Language Processing (NLP) is the field of using software to understand and manipulate natural language. Natural Language is a hard area to understand, despite most languages having implicit rules, these rules are normally very flexible in day-to-day expression. Even in the same language, different cultural expressions and quirks can make being able to understand a language a lot harder.

2.2.1 Word Embedding

Word Embedding is the process of representing words and sentences in a way the computer can understand. Normally this is in the form of a real space vector, which would allow it to be easily used in machine learning techniques.

Basic representations of sentences would be using a Bag of Words technique (24). This technique represents sentences as a vector with each element representing one word in a dictionary and each value being a metric such as that words frequency or TF-IDF value (25). This sort of technique normally results in large sparse vectors and can't represent meaning in text. Pre-processing and having a limited dictionary can help reduce the vector size; and grouping similar meaning words together using tools such as WordNet (26) or manually engineered features can also introduce some meaning to these embeddings. However, these techniques are time consuming and often ineffective.

It has been noted that words that have similar meanings will have similar contexts (27). With this fact, the core concept of deep learning could be used and have the computers learn their own word embeddings.

Word2Vec was introduced in 2013 in Mikolov et al. (28) and is a neural word embedder. It works in two ways, continuous bag of words (CBOW) and skip-gram (29). The networks used were fairly simple with one hidden layer, where the output of which would be used as the embedding of the word. The network is trained by either predicting a word based on the words around it (CBOW) or the other way around in skip-gram. This can be seen in figure 2.7.

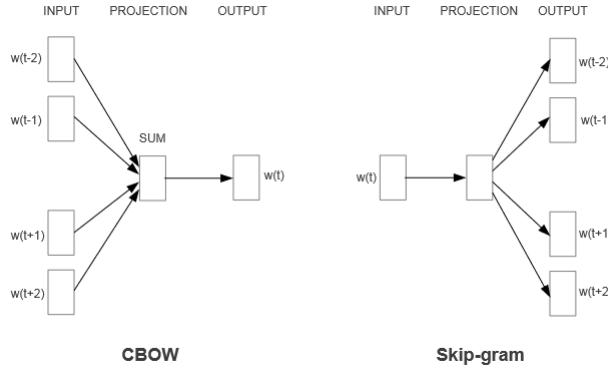


Figure 2.7: Architecture of CBOW and Skip-Gram models, from (28)

Word embeddings created can be compared using a metric known as cosine similarity. The lower the similarity the closer the meaning between the two embeddings will be.

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Embeddings created from Word2Vec were very powerful as it was able to represent meanings for words. This can be shown as adding/subtracting embeddings of words is able to show their relationships, for example: King - Man = Queen - Woman. However, Word2Vec was only able to work one word at a time, so homonyms would not be able to be represented well and it would be great at representing whole texts/sentences.

BERT

Bidirectional Encoder Representations from Transformers (BERT) was introduced in Devlin et al. (6) and is a word embedder that takes a word's current context into account by embedding each word along with the sentence it's in. Other similar models also existed, such as ELMo (30), however BERT takes advantage of the transformer architecture and takes a pre-training approach, which means it is trained for general goals and can be subsequently fine-tuned for downstream tasks.

BERT is a Masked Language Model (MLM), which refers to the task it was trained to perform in-order to produce the encodings. In this task, it takes in a sentence with some words masked and it is tasked with predicting what the original words were, which can be seen along with its structure in figure 2.8. Because of this, it is able to implement a Bidirectional architecture. Unlike in traditional language models (discussed in 2.2.2) where only words previous of the current can be used to predict, BERT uses both the words before and after each masked word. This means it can understand the context better of each word and thus produce better word embeddings.

Similar to Word2Vec, the hidden layers of BERT can be used as the embedding of each word. BERT models have various hidden layers so different combinations could be used. The creators of BERT found that the best embedding is produced is when the last 4 hidden layers are concatenated.

BERT can come in various sizes. The authors created pre-trained BERT models *BASE* and *LARGE*. The *BASE* version has 12 layers and 110m parameters and *LARGE* has 24 layers with 360m parameters. The *LARGE* model performs the best of the two but does require more training examples to fine tune for downstream tasks and would produce larger word-embeddings.

When tested on the SWAG dataset (32), which tests common sense reasoning, the BERT architecture caused a 66% error reduction over ELMo and 70% error reduction compared to non contextual models. This shows a significant improvement in performance which demonstrates the strength of transformer-based networks.

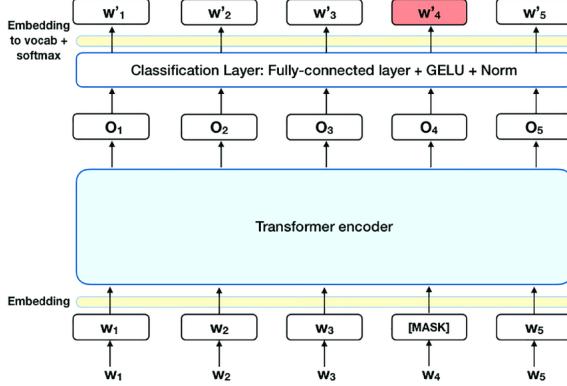


Figure 2.8: Diagram showing BERT structure, from (31)

A pre-trained BERT model is accessible and can be downloaded, used and trained for further tasks which makes them good resources to use as a base for NLP based tasks.

2.2.2 Language Models

A key part of NLP is language models. Language models are used to represent the probability distribution of text sequences. They have many uses, such as sentiment analysis, text completion, question answering, machine translation and text verification.

Language models can be used to calculate the probability the next word given the current history as well as finding the probability of a sequence of words appearing.

$$P(w|h) = P(w_n|w_1^{n-1}) \quad (2.7)$$

$$\begin{aligned} P(w_1, \dots, w_n) &= P(w_1)P(w_2|w_1)P(w_3|w_2, w_1)\dots P(w_n|w_1^{n-1}) \\ &= \prod_{i=1}^n P(w_i|w_1^{i-1}) \end{aligned} \quad (2.8)$$

A pure statistical way to produce language models can be done with n-grams. Text sequences can be theoretically infinitely long, so modelling a probability distribution for the whole history is infeasible, so instead they take a Markov assumption and assume the probability can be approximated by just look at the last n words.

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-n}^{k-1}) \quad (2.9)$$

The way to evaluate a language model, is by looking at its perplexity on a test dataset. The perplexity looks at how "surprised" it is at seeing a sequence. The lower the perplexity the better a model is at modelling the domain of the test set.

$$PPLX(W) = P(w_1, \dots, w_n)^{\frac{1}{n}} \quad (2.10)$$

These probabilities can then be calculated using their relative frequencies in a corpus. With larger, more representative corpus producing better models. These purely statistical models are simple to make but don't generalise well and often have to deal with sparse data for unlikely sentences.

Deep learning can help here, with large architectures they can be used to model this probability distribution when trained over a set of text. They can use simple neural architectures by taking in fixed length inputs like n-gram models or use a variable length model so they can look at any length sequences and better deal with relationships between words.

GPT

GPT (Generative Pre-Training) was a model created by OpenAI and introduced in Radford et al. (3). This was a language model that utilises the transformer architecture and a pre-training

approach. It was put forward that to train a language model from scratch, it requires a large amount of data in order to be used for each task, however a lot of that data is just needed to just train to have an understanding of the language as a whole rather than the task needed training for. GPT used a pre-training approach to develop a general task-agnostic model by training it on a general language model objective (given input predict next word), then it was further fine-tuned on several tasks such as text classification, question answering and similarity.

It was found that this pre-training approach and the power of the transformer model allowed the model to outperform various other specifically trained approaches in several tasks with less task specific training data. Thus, showcasing the strengths of this architecture and training techniques.

The architecture was then improved upon with the second iteration, GPT-2 (4). GPT-2 uses a much larger dataset and model and is again trained on a simple language model task. Instead of fine tuning this model for further tasks, it was shown that as the dataset already included examples of most of the common language tasks so it could perform these tasks well without specific training. They also showed the model could perform well in a zero-shot tasks (when no examples of the task are given in the input). Although, the performance on these tasks weren't as good as the state of the art, the level of performance further showed the strengths of training on a large general dataset.

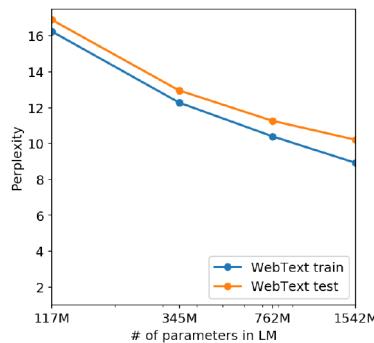


Figure 2.9: Graph showing how perplexity of a language model changes when changing size of a model trained on the WebText dataset, from (4)

Another result from the creation of GPT-2, was that it was showed that larger language models perform better. This is because over the several models trained, the ones with the largest number of parameters had a lower perplexity and therefore was better at modelling the language, this is shown in 2.9.

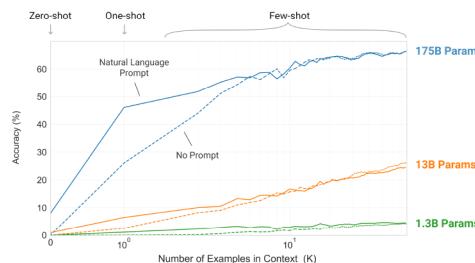


Figure 2.10: Graph showing accuracy of output of differently sized models when given different number of examples in the input. The larger model shows a better improvement going from zero-shot to one-shot demonstrating how it's able to better learn from context, from (5)

GPT-3 was the most recent model put out by OpenAI (5). GPT-3 is significantly larger than previous models, having 175 billion parameters and was trained on over 570GB of text compared to GPT-2's 1.5 billion parameters and 40GB of text. This model performed very well, especially as a few-shot learner, where 2-10 examples of the task are given along with the input as can be seen in figure 2.10. As a few-shot learner it was able to outperform or match some state-of-the-art models that were crafted specifically for a task.

A model like this is very powerful for many different tasks and has an API that can be used to access it to be used for developer's personal tasks.

2.3 Training Robots

For robots to learn how to perform actions, they have to learn policies. These policies define a function that given a representation of the robot's and environment's state s_t will map to an action a_t for the robot to take that will help it achieve its goal:

$$a_t = \pi(s_t) \quad (2.11)$$

Normally these actions refer to movements of the robot throughout the environment, which will in turn update the state.

There are various ways to create these policies. Model-based methods use the fact that the model of the environment is known, so it can use planning algorithms to find the optimal action at each state to achieve its goal. If the model of the environment is not known, it can be estimated by learning an approximation of the environment through interactions. Model-free methods can learn a policy without directly knowing the model, these methods include reinforcement learning and imitation learning. In the real world, a model of the environment is not known and very difficult to learn as there are too many variables to consider to create an accurate model. So, most robots policies are taught using a model-free approach.

Reinforcement Learning This method utilises a reward function and then creates a policy that picks the action at each state that will most likely lead to the best reward (33). The value each action is given at each state is represented by the state-action value function Q so the policy can be formulated at follows.

$$a_t^* = \pi(s_t) = \text{argmax}_{a \in A} Q(s_t, a_t) \quad (2.12)$$

The state-action value function can be calculated outright, if the exact dynamics of the model is known, but as in robotics this is not the case it can instead be estimated using deep neural networks with the state normally represented by various measurable variables or an image of the environment. The training data for these networks are supplied through experience, where the robot explores the environment using its currently known policy and some random movements to better estimate what actions it needs to take to achieve its goal.

This sort of learning has a few issues, one is defining a reward function. Defining and coding these reward functions can be difficult, especially when running on a robot in the real world. Secondly, the system would have to have a way to reset after each exploration, which again is difficult to apply to real world methods.

Imitation Learning This approach learns from demonstrations of the desired actions and uses the information learned to create a policy (34). There are several approaches to this method: behavioural cloning (BC), which tries to learn the policy directly from state action pairs created in the demonstrations using traditional machine learning techniques; bootstrapping which uses demonstrations to learn how to correct its current policy, and inverse reinforcement learning which uses demonstrations to estimate the reward function and then perform reinforcement learning to create a policy to achieve that reward.

While these methods can be effective they all have drawbacks, BC requires a large amount of training examples to be effective and bootstrapping and inverse reinforcement learning requires lots of human intervention for it to learn a policy. Other approaches do exist that can learn policies with a smaller number of examples (35) and human interventions, but these approaches might not be applicable to all tasks.

Chapter 3

Related Work

3.1 Language to Control Robots

For most policies created for robots the goal is specified explicitly, by either having only one task the model is trained upon (36), or by having the goal specified by hand designed parametrisations (37). These approaches, while successful, are inflexible to potentially new goals and are difficult to explain in a human way. A more explainable way to express the goals for the policy would be through natural language. Though as explained before, language is complicated, and meaning is often hard to distil from text, so this is quite a challenge.

In Stepputtis et al. (38) they create a network to control a simulated robot using natural language commands. They create this by splitting their system into two parts, a semantic model and a control model. The semantic model takes an input image of the situation and a language task specifying the goal and produces a representation vector. The control model takes the representation vector and the robot’s current state and produces a set of steps to achieve the goal. The semantic model encodes a sentence by taking word embeddings for a sentence up to 15 words long and then passes it through a GRU encoder. This output is compared to the results of an object detection model run on the input image to get the location of the appropriate object that is referred to in the text. This location along with the embedded representation of the input text is then combined to produce a representation for the task. The whole system performs well in the environment given when used with synthetic text, with successful actions on 84% of tasks, however performance drops when tested with language different to what it was trained on, dropping to 64%. This was likely due to the lack of generality created with their word and sentence encoder.

Large Language models can be utilised to help improve a model’s generality from text inputs. In Lynch and Sermanet (39) descriptions of actions are tied to movements collected from a robot’s exploration of environment during play (40). This description passed through a large language model is used to train a policy to control the robot. The human labelling of the actions and the embedding powers of the large language models led to the system being available to adapt to many different commands including in different languages. This generalisation power of language models was also explored in Jang et al. (10) which uses encoded human demonstrations and encoded language inputs to condition a robot’s policy using FiLM on a ResNet architecture. In a test environment, just the language input can be used to direct the robot for each action and when the model produced was tested it was able to successfully execute 40% of previously unseen actions from just the task description which shows the generalisation ability of large language models and their application in robotics.

3.2 Traditional Planners for Robots

There are two main types of planning systems for robots (1): motion Planning (MP) (41) as well as Task and Motion Planning (TAMP). MPs deal with how robots move through an environment to reach a goal while avoiding obstacles and can be solved effectively using sampling (42) or constrained optimisation methods (43). TAMP problems deal with planners that also involve actions that affect the environment. These are much harder to solve, as they not only require a way to represent the current state of the environment, as well as understanding of how each action changes this environment. TAMP planners have to use a form of discrete continuous search to look

through a set of actions through their state space that will get them from their start goal to end goal. There are many approaches to achieve this, either through searching trees and finite state trees (44) or utilising logic languages such as PDDL or ASP (45). These methods work and are able to be applied to complex systems such as grab and place systems (44) or even controlling various systems at NASA (46). If defined with a perfect representation of the environment, these planning systems can produce perfect plans for robots to execute, however defining perfect representations is almost impossible for real world robots as the world have so many variables. Assumptions about the world will have to be made to produce plans but this costs reliability. Furthermore, these systems are all hand created for a specific set of goals meaning they aren't very generalistic systems and can't be reused for other tasks. Finally, goals for each robot will have to be specified exactly in the state space, making them hard to use and implement without that exact knowledge of this space.

3.3 Using Language Models as Planners

A more understandable and flexible way to represent goals is using natural language. Some work has been done investigating if language models can be used for planning. In Jansen (8) the capabilities of a RNN based language model and a large language model, GPT-2, were tested. The paper evaluated these models on the ALFRED dataset (47), which contains various tasks for planners to try to create a set of actions for an agent to take around a human environment. The RNN model was trained directly, and the GPT-2 model was fine-tuned using example pairs of request and perfect sequences. The aim was for each output to follow a specific structure, and some hard coded methods were introduced to correct for any errors in each model's output. The result of the experiments show that the GPT-2 model outperforms the RNN model on these tasks and would get the majority of the steps in the sequence correct, but still only achieving a perfect plan 26% of the time. The models mostly failed on the first instruction, which usually relied on dataset specific tasks such as going to a location. When given the current visual context of the agent along with the input, the model doubled its accuracy. This paper showed that the large language models, such as GPT-2, had enough internal knowledge to create these plans and could also use the current context to boost its performance.

Huang et al. (9) furthers these ideas by utilising the more powerful GPT-3 language model and moved to testing in a zero-shot setting. They evaluated their model by using a human-like agent in a simulated VirtualHome environment (48) which had a set number of actions they could use. When testing on base GPT-3, they found that although it could produce a correct series of actions, these set of actions were rarely in an executable format for the agent. To improve the executability of the outputs, at each step they would sample various possible next steps, they would then compare each sample with all of the actions from the set that are currently executable and pick the action that has the lowest combination of similarity between sample and action and probability of output from the language model. They also included a threshold so if none of the samples produce an output that maps well to a possible action then the system would stop trying to find a plan. This method improved the executability of the outputs of the model from 8% to 79%. However, as this model was being run in a zero-shot environment, the base GPT-3 model would obviously perform poorly as it does not know the style of output or capabilities of the robot required. As GPT-3 is shown to be significantly stronger in a few-shot environment than in zero-shot; then it might be possible to improve its ability to produce executable outputs using more examples. Although in this setting, the number of possible actions are vast would take lots of previous examples which might be too much for the model to perform successfully.

These previous works use methods that deal with outputs dealing with more complicated actions that would be performed by a human-like robot, these often use complicated action statements that can be expressed in many ways. Current robots might be able to achieve some of these tasks but not all of them and will generally have a small subset of tasks it can perform in a very limited environment. This smaller subset of simpler actions means it could be easier to ground the output of the model to produce better outputs.

3.4 Language models as Planners for Robots

An implementation of one of these language planners with a physical robot was given in Ahn et al. (7). This implementation again used GPT-3 but this time gave the language model some examples to help teach the model how best to complete some tasks and to begin to ground it to the environment and the capabilities of the robot. Their implementation defines a set of possible actions that their robot is trained to perform $\pi \in \Pi$. Each of these actions were trained using behavioural cloning on a ResNet architecture that has been conditioned by FiLM on an encoded language description for each task l_π .

The policy executed at each step of the plan is calculated by a "SayCan" system. At each step in this system, the model calculates the probability of outputting each of the possible language commands $P(l_\pi|i)$ by using the output probabilities from GPT-3. By limiting the system here, it grounds the model in the capabilities of the robot. This system also calculates the probability that each policy will succeed given the current context of the model $P(success|s, \pi)$ which is given by an image from the robot. Once this is done, the action is chosen based on:

$$\pi^* = \operatorname{argmax}_{\pi \in \Pi} P(success|s, \pi) P(l_\pi|i) \quad (3.1)$$

This action is then executed, and the model progresses to picking the next step.

This system had good success in being able to execute these tasks, the "SayCan" system allowed for plans to be created by only looking at the possible commands rather than letting it generate any command on its own. It was able to execute simple requests such as "move the coke can to the bin" up to more abstract and long horizon tasks such as "bring me something to recover from a workout". However, a model such as this requires many examples to be able to train each additional policy, and lots of work has to be done even for extra objects, even if they are similar to current ones. This is because these examples need to be used to train both the actual policy and the *can* affordance model. Also, limiting of the possible actions might make the model struggle with some tasks where its current action set is insufficient as it could not even try to perform these tasks even if it could generalise with its current knowledge like in Jang et al. (10).

Chapter 4

Implementation

4.1 Overview

The raw output of language models can often range wildly from input to input, which is why many previous approaches have used a classification method to help control the output into grounded actionable tasks.

However, language models have many tools to help manage the output and could be shown how to produce specific outputs grounded in its current environment, especially if the environment is fairly simple with very mid-level commands. Furthermore, it has been shown that robots can be trained to perform actions on inputs utilising encodings from large language models. These large language models are able to map similarly meaning inputs together, which means a robot could become more generalised in its input and react to any slight changes to input structure given by the language model.

These two ideas will be tested, with the creation of robot policy conditioned on text embeddings from a BERT encoder in section 4.2 and the evaluation of the capabilities of the GPT-3 language model in producing grounded plans to find the best method to use it as a planner in 4.3.

Once a policy and method of using the language model have been established, a planning system can be created to execute these commands on the robot. This will be discussed in section 4.4.

4.2 Creating the Robot Model

The first part of the task is to create a robot that can be instructed using text commands. To reduce complexity a simple grab and place robot will be used. The project will also utilise a simulated environment, as it allowed for quick and easy resetting of the scenario and would also reduce the amount of noise and uncertainty that would come about with real robots allowing for easier training.

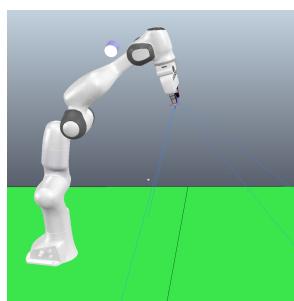


Figure 4.1: Panda arm used as robot

A simulator has access to all the details of the environment, including positional and orientation data of all objects. This means that for this task, hard coded policies for each possible action the robot can perform could be created, which would create an accurate and robust robot. However, as no real robot would have this data, only data from their own sensors such as a camera, it was decided to instead train models to execute these actions instead using only the data a real robot

would have as input. While this will hurt the reliability of the robot, as it is unlikely that a perfect model will be learned, the process for training this model would mimic how the process of training a real robot.

The simulator chosen for this task was CoppeliaSim (49). CoppeliaSim has a lot of tools and features that are very useful in testing simulated robots. The simulator will be interfaced with using PyRep (50). PyRep is a python library that allows for python code to control the simulation and the objects and robots inside of it. CoppeliaSim has several different robots that are available to be used and for this project the Panda Robot (seen in image 4.1) was chosen. This robot is an arm with a stationary base and a gripping mechanism is attached.

The model was created iteratively, starting off with a single action on just one object type and then expanded to multiple actions across a range of objects. This allowed for insight to be gained from each step; in understanding the capabilities of the model architecture used as well as techniques needed to get the best performance from training; which meant problems could be solved much easier and faster than they would have if the full implementation was attempted immediately.

4.2.1 Single object

The first task was to be able to create a model to pick up a single object. In order to do this, systems also have to be created to interact with the robot, gather data, train a model and then to run and test this trained model on the simulator. In this task the robot will be given a wrist camera attached to the gripper, this camera can give RGB images as well as a depth map. These outputs will be fed into the model as inputs and it will predict the movement needed at each step in order to pick up the object.

The object chosen for this task is a red cube. This is a very basic object that should show up clearly against the green table backdrop and will be of consistent size which should allow for an easier training process.

Robot API

For the model to be able to control the robot, it needs to be able to execute some commands. To do this, an API needed to be created that can take input from the trained model and actually move the robot in the simulation, as well as be able to gather data from the robot that will be needed to train and run the model.

Movement

- Move by:
 - Would take the input δ which would be in the form of a position vector $[x, y, z]$ and it would move the end effector of the robot by δ amount
- Move to:
 - Would take the input x which would be in the form of a position vector $[x, y, z]$ and it would move the end effector of the robot to that position x
- The *move to* command is very useful for setting up the robot in initial positions and for gathering data points. However, as the robot won't be aware of its current position when using the model, it will only interact with *move by*
- Throughout testing and training, it was noticed that when an action would move the robot into a position it can't be in, the simulation would crash. To prevent this the model is given bounds. If the robot is instructed with moving into these bounds, then the movement would be cut so the robot would remain inside the allowed area.

Rotation

- Both the robot and the gripper has an orientation and can rotate. If the arm rotates then the gripper would also rotate. However, the arm can only rotate a certain amount until the joint positions needed to achieve this rotation is unfeasible for the robot to be in. The gripper can rotate a full 360° an infinite number of times making it much more useful. So, the rotation

of the robot will be achieved by just rotating the gripper and ensuring the orientation of the arm is the same at all times.

- Rotate by:
 - Would take the input δ which would be in the form of a rotation vector $[x, y, z]$ and it would move the end effector of the robot by δ amount. Although the gripper could be rotated around all axis, in practice it will only be rotated in the z axis.
- Rotate to:
 - Would take the input x which would be in the form of a rotation vector $[x, y, z]$ and it would move the end effector of the robot to that rotation x
- Just like in movement, the model wouldn't know its exact current rotation, so would only want model to interact with rotate by using *rotate by*

Grabbing

- PyRep enables control over the robot's gripper by being able to set how wide the gripper is set and the speed it reaches that width is.
- Once the model has reached a position above the robot, then it can be possible for it to grasp it. The grasping action has been decided to be hard coded and not dependent on a trained model, as the process will be invariant on any input/object. The same applies to the releasing motion.
- Grab object
 - Move the arm straight down a set amount, then close the gripper as much as possible (by setting the required width to 0)
- Release object
 - Open the gripper (by setting the required width to its maximum) then move the arm up a set amount so the gripper is no longer around the object.

Data

- The robot's current position and the grippers orientation will be able to be collected to be used in the creation of demonstrations. These demonstrations will be used to train the model.
- The camera on the robot can output data an RGB image and a depth image. This will be collected and formatted in a way that will be easy to be entered into the model.

Utility

- Functions for both position and rotation will exist that will move the robot back to its "home" position/orientation
- These can be used at the end of each action, to ensure the robot starts at the same point.

Creating demonstrations

The model will be trained by a process called behavioural cloning (51). This is a training process where the model is trained using data from demonstrations of the action that is wanted to be performed. The data collected would be the image from the camera and the movement needed at each step. The camera for this process would output an 128x128 RGB image and as well as a 128x128 depth map.

The demonstration process would start by randomising the location of the cube in an area in front of the Robot. It would then also rotate the cube a random amount.

To help improve the randomness of the data, the robot would also start in a random start position. The random position chosen would be from within an area where it would still be in a position that it can see the object.

The target end position of the gripper would be a location above the cube. The height of this location was dependant on the height of the cube + how far down the gripper would move in the *grasp* command.

At each step, the movement delta would be calculated using the following.

$$step = \frac{target_position - current_position}{STEPS} \quad (4.1)$$

Where *STEPS* is a constant to force the robot to take multiple steps that decrease in size the closer they get to the robot. Through testing, the best value for this was decided to be 6.

The camera for the robot is attached to the gripper. This means that when the gripper rotates then so will the camera. As the movements are relative to the robot's frame and not camera's (i.e. a movement forward for the robot could be left in the camera) it would mean that if the gripper rotates then it would have to be able to tell its orientation to calculate movement in the robot's frame to be able to achieve its goal. The floor of the simulation environment is a solid colour which would mean the model would have no reference points to get this orientation and thus calculating movement with a rotated camera would be impossible.

To prevent this problem, the action would be split into two stages: first the robot moves to above the target position, then when it has reached this position, it will rotate to the target orientation before picking up the object. It will switch between the two stages when the distance moved in one step is less than a threshold. For the demonstrations this threshold was 0.0015 so when counting in its division by *STEPS* means it will be a distance of 0.009 from the target. The cube has a width of 0.05 so this small margin of error means that it will still be able to pick up the block.

The desired rotation for the gripper is so it lines up perfectly with the cube, so it will have two of its sides parallel to the gripper claws. To gather the data for the rotation, the cube was rotated a random amount between -45° and 45° as this would be able mimic all possible rotations of a cube and make the computation of the shortest rotation for the robot easier.

In a similar fashion to movement, the rotation taken would be calculated by:

$$step = \frac{target_rotation - current_rotation}{STEPS} \quad (4.2)$$

Here the *target_rotation* is the negative of how much the the cube was rotated by. This is because the gripper is upside down, so for it to match the rotation of the block it has to rotate in the opposite direction.

It was decided to collect the rotation data and the movement data every step. This is so the model can be trained to output both and that it can easily switch from one stage to the next when the threshold was met, rather than having to run it through a second model. To help with stopping the robot when it is close, the data recorded for movement during the rotation section is a zero vector.

Model Design

The model for this problem will have a Resnet-18 architecture (seen in figure 2.3). This is due to the successes shown of this architecture in this task shown by Ahn et al. (7) and Jang et al. (10).

The model will take in a $4 \times 128 \times 128$ tensor as input, relating to the $3 \times 128 \times 128$ RGB Image appended to the $1 \times 128 \times 128$ depth map and will output a vector of dimension 4. Three of those dimensions relate to the movement of the robot in the x , y , and z dimensions and the 4th refers to the rotation needed for the gripper.

Running the model

Once the model was trained, a way to run the model needed to be created. This would follow a similar structure to the demonstration loop, going from movement to rotation once a threshold was reached then not being able to return. A rotation threshold and a max limit for the number of steps were added to terminate the process when it was deemed to be close enough to the target or that the action has gone on too long. The average number of steps in the demonstration was around 35, so 60 was deemed a sensible amount.

Algorithm 1 Single Object Loop

```
mov ← True
X ← 0
while X ≤ 60 do
    X ← X + 1
    inp ← robot.get_image()
    δmov, δrot ← model(inp)
    if ||δmov||2 ≥ movement threshold AND mov then
        robot.move_by(δmov)
    else
        mov ← False
        if |δrot| ≥ rotation threshold then
            robot.rotate_by(δrot)
        else
            break
        end if
    end if
end while
robot.grab()
```

The movement threshold was decided to be 0.005, above the threshold used in the demonstrations. This point in the demonstrations would still be close enough to pick up the cube, but as the model will have some error in it, it gives the model some more leeway to make it more robust in picking up the cube.

Experiments

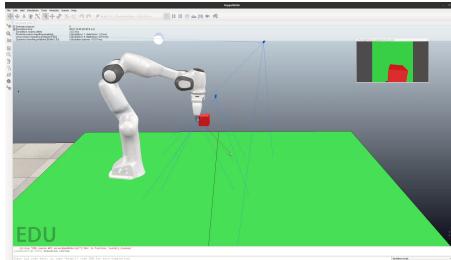


Figure 4.2: A model trained picking up a cube. Video: https://youtu.be/BfK6P_U2CV8

Several different models were trained on this training data but using a different number of demos as their training data. They were all trained using batch gradient descent, on a batch size of 128, a learning rate of $2e - 3$ and a weight decay of $1e - 3$. It was trained using the AdamW optimiser (52).

Once trained, the models were subsequently tested on the simulation. Each test started with creating a red cube randomly in front of the robot and moving the robot to a random location where it can still see the cube. Then the algorithm in 1 was executed and once it concluded it would attempt to pick up the cube. An iteration was then deemed to be a success if it was able to successfully pick up the cube. Each model was tested 50 times and the results are shown in Table 4.1.

| Model | Success Rate (%) |
|--------------------|------------------|
| 50 Demonstrations | 0 |
| 100 Demonstrations | 10 |
| 250 Demonstrations | 32 |
| 500 Demonstrations | 78 |

Table 4.1: Success rate of models

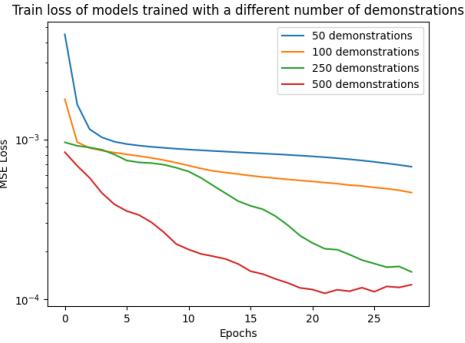


Figure 4.3: Training loss of the different trained models

The models trained on a smaller number of demonstrations fail to be able to pick up the cube most of the time. The smaller number of demonstrations means the model doesn't have enough data to be able to reliably predict the movement at each step, with the error often compounding so it doesn't reach the target. As more demonstrations are added, the model is able to learn the required function better, so succeeds much more often. Also, with more demonstrations come more generality, this means that it can better adapt to situations it hasn't seen before so is more likely to succeed in practice. A short video showing the best model in action can be found in figure 4.2.

4.2.2 Multi-object

The previous task showed it was possible to pick up an object successfully using this architecture and training methodology. The next step in building the model would be to get the model to be able to pick up a specific object from a collection of different objects. This object would be specified using a natural language prompt.

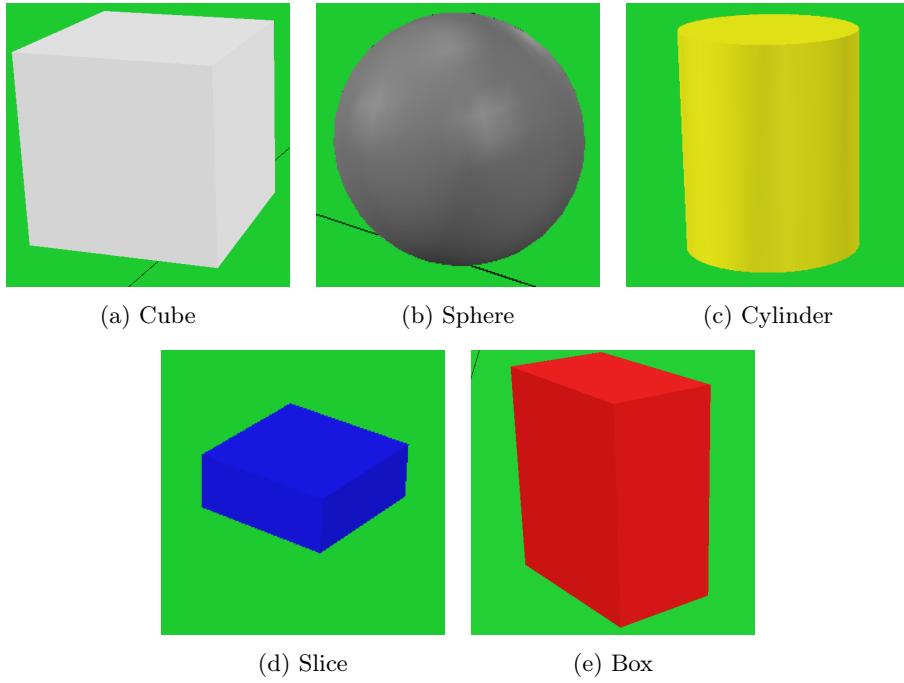


Figure 4.4: Different Objects

The possible shapes these objects could take are: a cube, a cylinder, a sphere, a box and a slice (seen in fig 4.4). To add variety to these objects, they could take one of 7 different colours: black, grey, white, red, yellow, green and blue. They were also scaled randomly from their usual size by a factor between 0.8 and 1.2.

Creating demonstrations

At the start of each demonstration, a random number (between 1 and 5) of objects were placed in front of the robot. Each object would be one randomly picked from the possible objects and then given a random colour and size. To simplify the creation of objects in the scene no colour would be repeated across the objects. This meant that two differently coloured cubes could exist but not two same-coloured objects. This was an easy way to ensure no duplicate objects existed in the scene.

When placing these objects, it was important to ensure they were placed in positions where they were not colliding with each other and there was enough space for the gripper to grab each object. To do this, each object is given a maximum radius which is given by

$$\max_radius_{ob} = \sqrt{ob_{width}^2 + ob_{height}^2}/2 \quad (4.3)$$

Then, when a new place for a new object is generated, if the distance from its centre to any other centre is less than $\max_radius_{new} + \max_radius_{ob} + gap$ then a new location will be generated for it. *gap* is a small constant that ensures the two objects can't be directly next to each other, so the gripper has enough room to grab the object.

The demonstration process is the almost exactly same as before, with the target object being randomly selected from the possible objects. There are only two differences with the process, the first is that each object has a different target orientation for the gripper; with the sphere and cylinder requiring no rotation, the box requiring rotation, so the gripper grips the object at the widest point and the cube and slice having the same procedure as before. The next main difference is that at each step the encoded name of the target object is also saved. The name is given by a combination from the colour and the name of the shape (e.g. "red cube" or "yellow sphere") and the encoding is given from a BERT encoder.

FiLM

The general model architecture for this task is mostly the same as the single-object variant, with the base being ResNet-18. However, as the model also needs to take in the encoded text as input the ResNet blocks are given a FiLM layer and several fully connected layers are created to process the encoded text into the *beta* and *gamma* parameters needed at each layer.

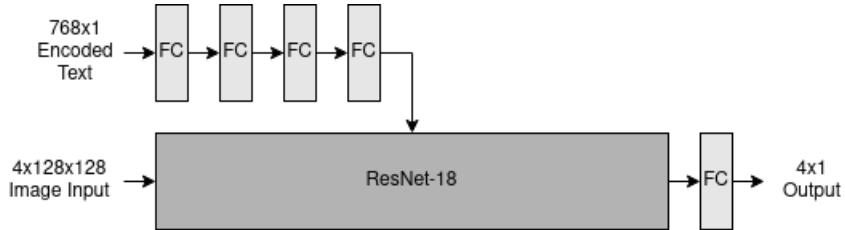


Figure 4.5: FiLM Resnet Architecture

Running the model

The main loop for running the model is almost identical to the one shown in 1. The major difference is at the beginning of each task, the name of the target item will be encoded using BERT and sent as an additional parameter each time the model is called to predict the movement for that step.

Experiments

For this test, two models were trained. One with 750 demonstrations and the other with 1500. They were both trained for 30 epochs with a learning rate of 5e-4 and a weight decay of 1e-3.

Once trained, each model was tested in the simulation environment. A test would involve randomly generating a selection of items and then selecting one as the target for the robot to pick up. Each model was tested for 250 iterations, ~50 iterations for each object. During testing, how successful the model was at picking up the correct object was recorded, as well as the successes per colour and per shape.

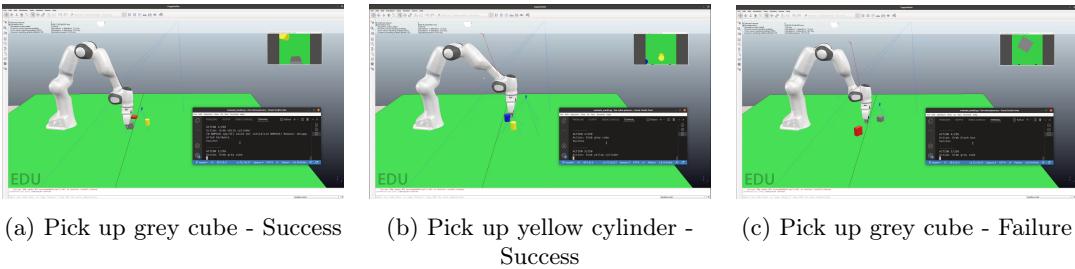


Figure 4.6: The model trained on 1500 demons being tested on various objects. Video: <https://youtu.be/RZQQ9-288kE>

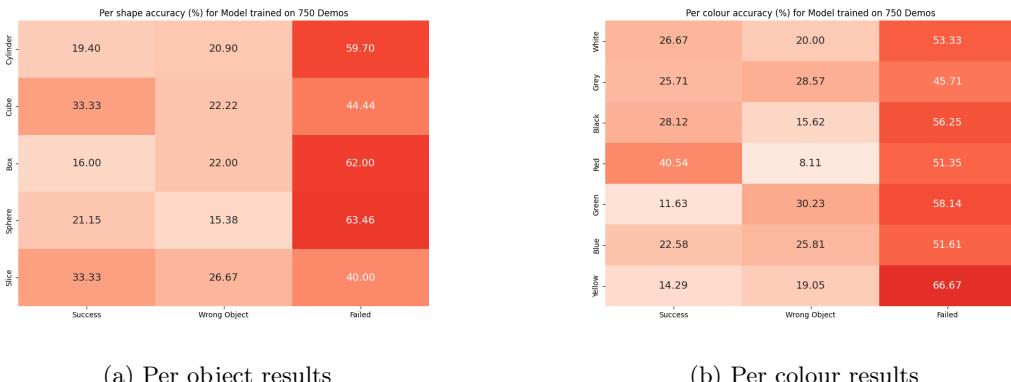


Figure 4.7: Model trained with 750 demos

As expected, the model trained on 750 demos performs worse, failing most of the events with an average success rate of 22%. The model trained on 1500 demos performs much better, with an average success rate of 55%. In the 1500 model the major problem is for failing to pick up the item this mainly occurs because it does not reach the threshold for pick up the object, despite reaching the location. Though investigating the different thresholds, the one used did give the best performance so this issue could be overcome with more demos or tweaking the hyper parameters. The main object it struggles with is the slice, this is likely due to it being similar to a cube, so thus getting confused between the distance it needs to stop at above the object. It also struggles the most with the colour red, which interestingly is the colour the 750-demo model performs best at, so this behaviour could just be a result of how the model spent its time learning.

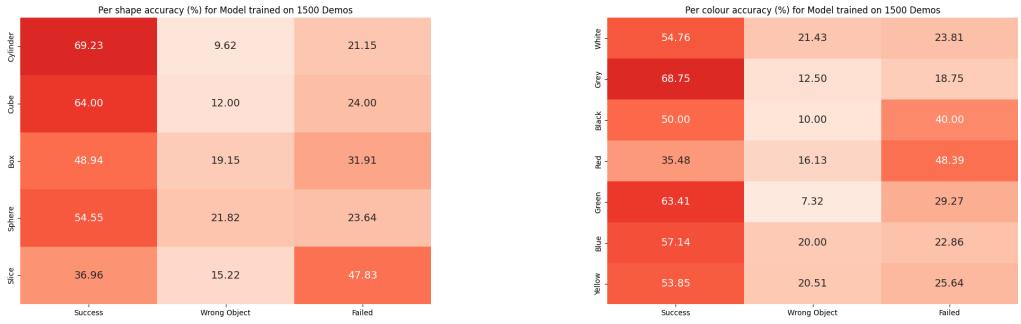
Overall, the 1500 model performs fairly well, and is able to pick up an object most of the time. Longer training times and more demos could lead to a better model but it does show that the Resnet-FiLM architecture combined with BERT encoding is able to learn how to differentiate between the objects and execute a successful policy. A video of this model in action can be seen in figure 4.6.

4.2.3 Multi-policy

The final task is for the robot to be able to execute several different actions, not just grabbing objects. This action would also be specified with a natural language prompt. The different actions that the robot needs to perform were grab, place on, place in and place by. These actions need to be able to be used on objects with the same shapes and colours as before. Also to be introduced in this task, are several other types of shapes, the plate and the bin (seen in figure 4.4). Both these objects won't be picked up and are only there to be used for the place commands (e.g. place on plate, place in bin).

Creating Demonstrations

With the addition of the two additional shapes, the object creation set up is changed slightly. A random selection grabable objects are generated as before, then some bins and plates will be added



(a) Per object results

(b) Per colour results

Figure 4.8: Model trained with 1500 demos

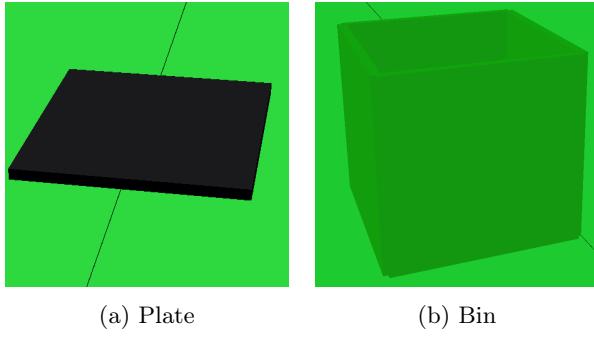


Figure 4.9: Plate and the bin

to the scene if possible. Unlike before where it was impossible for two objects to have the same colour, the bins and the plates are able to share colours with the different other objects, though still there won't be duplicate plates or bins.

To place an object, you first need to grab an object, so a demonstration would be a combination of a grab command followed by a place command. The grab commands in the demonstrations work the same as before. The place commands work like the grab commands, in selecting a target location then moving there using the formula in Eq.4.1. The target location for *place on*, would be a place slightly above the middle of the object and the target for *place in* would be above the centre of the opening of the bin. The target location for *place by* is more complicated however, as it has to pick a location next to the target object where no current object resides. This is done using the same algorithm as the one to ensure the objects aren't close together when initially placed in the scene. The place commands also do not have any rotation associated with them, and the objects are placed in the orientation from the gripper. Furthermore, every place command won't work with all objects; you will only be able to place on a cube, slice or plate and you will only be able to place in bins.

Along with each command, the natural language prompt needs to be saved. In order to give the model the ability to generalise to different possible sentences, this prompt will be generated by selecting a random template that is filled in with the appropriate information. Furthermore, some of the objects will be given other names that could be used instead. Examples of generated prompts are follows:

- Grab the red cube
- Pick up a yellow ball
- Place the green can on a blue plate
- Lay grey cylinder next to black cube
- Put in green bin

These prompts saved as both their encoded forms in BERT and as their raw text so different encoding methods could be tested.

Model Design

With the success of the architecture shown in the multi-object model, the same architecture could also be applied here. It would also use the same BERT encoder for the input of the actions. However, when looking at the encodings of the different actions that will be supplied to the model and the cosine differences between them, some of them are very similar:

```
cosine_difference("pick up red cube", "place by red cube") = 0.037
cosine_difference("pick up red cube", "place on top of red cube") = 0.014
cosine_difference("place by red cube", "place on top of red sphere") = 0.029
```

While the model could be able to work out the small differences between the different actions, there is no guarantee. To aid this, fine tuning the BERT could help improve the representations of the text and thus the model.

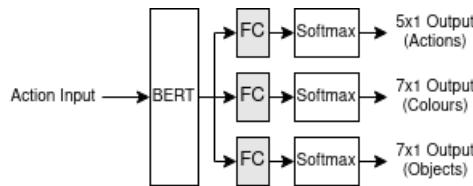


Figure 4.10: Training architecture for fine tuning BERT. Once training had finished, just the BERT model was saved and then used to create the encodings to be used in the architecture used for multi-object seen in 4.5

There are two ways that this can be approached, fine tuning the BERT model outside the model, or fine tuning the BERT model as part of the model. In the approach where it is fine-tuned outside the model, the BERT model is trained on a classification task. In this task, the model is given a sentence and must classify the action, the colour of the object and object itself. This is run for a small number of epochs with a low learning rate and a high weight decay as it is important that the model is still able to apply to general concepts and isn't over fitted to the training set. The BERT model fine-tuned here can then be used to encode the actions before input to the model. This process makes the encodings of each of the tasks more unique, so the cosine difference is much larger:

```
cosine_difference("pick up red cube", "place by red cube") = 0.30
cosine_difference("pick up red cube", "place on top of red cube") = 0.32
cosine_difference("place by red cube", "place on top of red sphere") = 0.36
```

The other approach is to include the BERT model as part of policy model, this way the training process will train the policy model but also fine tune the BERT model at the same time. This could allow the BERT model to encode the sentences better, but it does mean the model has many more parameters to train, so the training process might take longer and there is no guarantee it will learn correct encodings of the sentences.

The three approaches, base BERT, fine-tuned BERT and BERT in the model, will be trained on the same dataset then evaluated to ensure the best approach is used.

Running the model

To use the newly trained model some small adjustments needed to me made. The first was making sure the text was encoded with the specific model's BERT model or, in the case of the BERT model included in the whole model, having the action supplied as raw text rather than in its encoded form. The second adjustment was to the grabbing procedure. Before, when the action was done it would always grab. As some actions don't involve grabbing this had to change. Here, the gripper

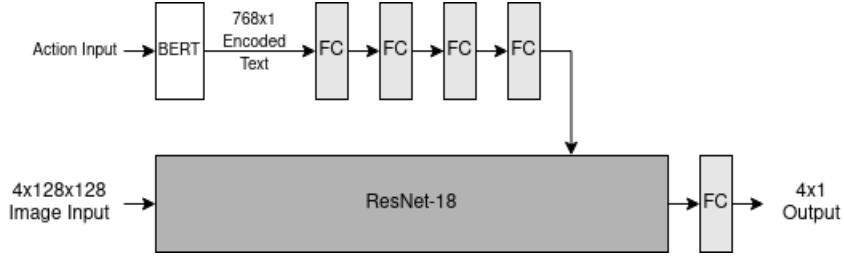


Figure 4.11: BERT in policy model Architecture

just toggles itself on and off. This means if it is currently holding an object, it will release it, and if it isn't it will attempt to grab.

Experiments

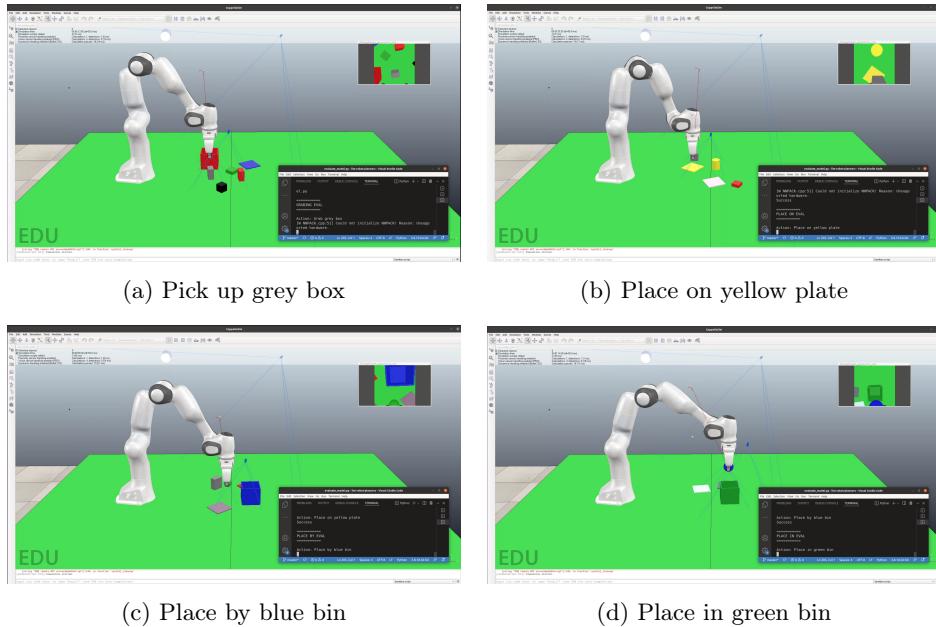


Figure 4.12: The pre trained BERT being tested on the different actions. Video: <https://youtu.be/1kgJur4qEZU>

Each model was trained for 30 epochs on 3000 demos, with a learning rate of 5e-4 and weight decay of 1e-3. Then they were evaluated by using the models on the simulator to perform each action. Each action was performed 50 times for each shape it could be done upon, so 250 for grabs, 150 for place on, 350 for place by and 50 for place in. The performance of each task was recorded, and the results are shown in table 4.2.

It can be seen that the model that used the fine-tuned BERT trained before the policy performs better than the two alternatives. Achieving good results across the different actions. From looking at the per actions breakdown for this model in figure 4.13. It rarely interacts with the wrong object or performs the wrong action which shows that this method of pre training BERT really helps the model. Furthermore, the system was not trained with placing next to bins, due to bug in the demo selection code, however this model was able to infer how to perform those actions with it's representation which shows it can generalise to unseen scenarios. Because of this better performance, this is the model that will be used for the final system. A video showcasing this model performing the actions can be seen in figure 4.12

The base BERT model does still perform well, achieving most of the tasks to a good standard. It does get slightly confused on some actions, either picking the wrong object or action in each situation, this is likely because of how close the encodings were that sometimes it wasn't enough to distinguish them. However, that situation seems rare and the policy model is still able to learn

| Model | Grab Success (%) | Place On Success (%) | Place By Success (%) | Place In Success (%) |
|---------------------------|------------------|----------------------|----------------------|----------------------|
| Base BERT | 91.2 | 72.6 | 90.9 | 84 |
| Pre Fined-tuned BERT | 94.4 | 93.4 | 92.6 | 98 |
| Fine-tuned BERT in Policy | 39.6 | 20.3 | 85.4 | 40 |

Table 4.2: Results from running the evaluation on the different models. More detailed results including per shape and per actions breakdowns can be found in the appendix A

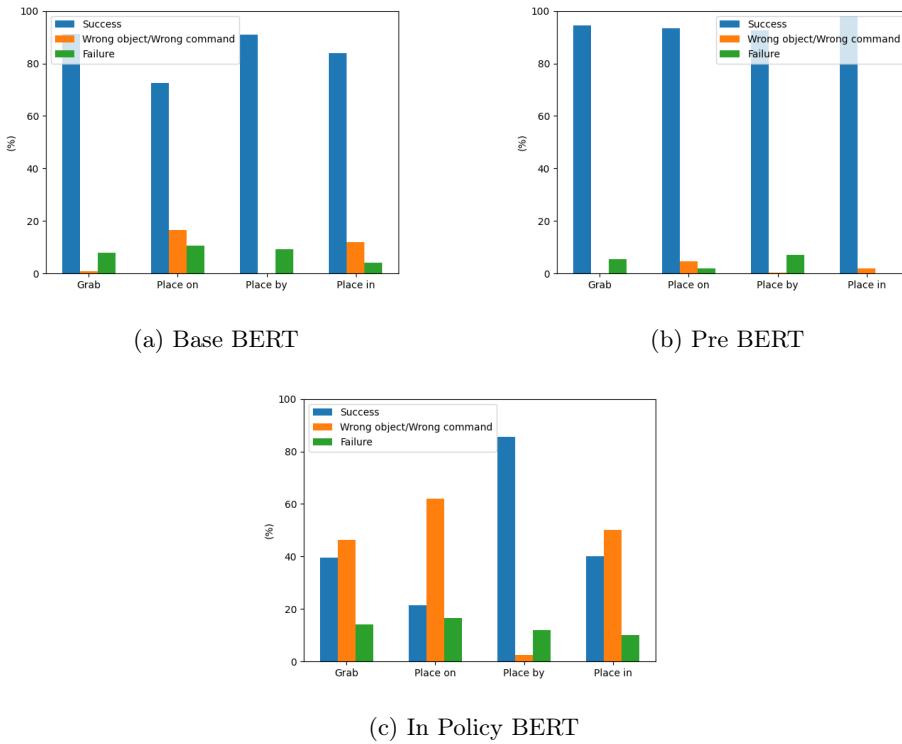


Figure 4.13: Breakdowns of results per action

how to use them quite well.

The fine-tuned BERT in the policy model does struggle, it is successful in the place by category but in the other actions it isn't as good. This is likely because the BERT model is so big, and the task is very unspecified that it was unable to fine tune the model in a way that can best achieve the task. Potentially longer training could help improve this model to the level of the other methods produce better results faster. The model's failure to train both BERT and the policy can be seen with the encodings given by its BERT model. The difference it gives between actions is small, showing it wasn't able to learn to differentiate them better:

```

cosine_difference("pick up red cube", "place by red cube") = 1.2e - 7
cosine_difference("pick up red cube", "place on top of red cube") = 6.0e - 8
cosine_difference("place by red cube", "place on top of red sphere") = 6.0e - 8

```

4.3 Using the Language Model

For the planning system, the language model to be used is GPT-3 as it has strong capabilities as a few-shot learner and has shown its ability as a planner before. To be able to use it in the system, understanding the best approaches to use it as well as what it can and can't perform is important.

This model isn't available to be loaded in and can only be accessed by an API supplied by OpenAI. This API still allows access for many features and tuning tools to have it act in a way that best suit the task.

Unlike in the Ahn et al. (7) approach, where it picked the most likely step from a discrete set, the system will use the steps generated directly from the model. This could allow the model to possibly predict something out of the capabilities of the robot, but through careful tuning of hyperparameters and utilising the power of the model as a many-shot learner when designing previous examples, the chance of this can be minimised. Also, as the robot model is being trained on a range of inputs using a BERT encoder it will hopefully be flexible enough so it can still function if the output of the model isn't exactly what is expected. This approach also allows the demonstration of the power of this model, by showing how it can be forced to construct only eligible outputs without being manually hard coded.

4.3.1 API

The model can be used through a Python package supplied by OpenAI which allows access to several models including GPT-3. This language model can be used in several ways: completion, edits, classification, and question answering. While classification could be used to select the best next possible action from a discrete set this would require quite at least one training for each possible command (112), this wouldn't be utilising the ability of the model as a few-shot learner. It would also not allow for flexibility in the model which is a desired trait in this task. Because of this, the completion mode will be selected. In this mode it will take an input, then complete the text until the model feels it has finished. This mode will allow the model to be more flexible in the output, which can allow it to generalise to unseen data and could be applied easily to different robots.

Hyperparameters

There are several different hyperparameters that can be set when using the model

- Model: There are several different models available for use, with some being better than others. The most recent model, and the best, is text-davinci-02 which is the model that will be used.
- Temperature: This defines how creative the model is when giving an output. 0 means the model is deterministic and picks the most likely token each time, 1 means the model is more creative with the output. For this task, the output of the model should be the same every time, so a temperature of 0 will be used.
- Top-p: Controls what tokens are considered for each token produced. A top-p of 0.1 considers only the tokens that make up 10% of the probability mass. It is recommended that only one of temperature or Top-p is changed, so for this task this parameter will be left as default.
- Stop: This parameter gives a list of tokens that once the model sees, it will stop predicting. This will be useful for controlling the output of the model.
- Presence Penalty/Frequency Penalty: These parameters penalise new tokens based on if they already exist in the input or how frequently they exist in the input respectively. These values could be tweaked to make it more likely to produce outputs that more tightly fit the desired output structure.
- N: You can get the model to output N outputs per input. This could be useful to get a different output if the first one generated fails. However, there will be no way of evaluating which output is best until it is performed, so 1 output is what will be used.

4.3.2 Capabilities of the Model

To best use the model for this task, the capabilities of the model need to be understood. This is so the size and type of tasks that this model could be used for can be seen and the input to the language model for this task could be designed to maximise success.

Zero-Shot/One-Shot/Many-Shot

There are a few different ways of specifying the task to the model: Zero-shot, One-shot and Few-Shot. These refer to the number of examples of the task that is given to the model beforehand. While few-shot may seem advantageous, some models struggle to handle long range dependencies in inputs, so when constructing the output, it might not consider all of the input leading to shorter inputs giving better results.

Zero-Shot In zero shot, no previous examples of the task is given. Because of this, it might be difficult to get the model to output the correct information in the format necessary.

Input: What steps would be taken to put an apple next to a banana?

Output: The apple would be picked from the tree and the banana would be picked from the bunch. They would then be placed next to each other.

However, giving more information in the input can help guide the output.

Input: What steps would a robot take to put an apple next to a banana? Write steps in one line separated by a comma and ending in done

Output: Pick up the apple in its hand, move hand to the banana, put the apple down next to the banana, done

This output is a lot better and more structured showing that the model can be instructed from the text input. But the model output is still very variable, and not always applicable to the situation. To make it better, more detailed descriptions of how the output should be structured is needed. However, designing this sort of specific description could be avoided if the model instead learns the semantics from an example.

One-Shot A One-Shot task is when one previous example is given. This allows the model to infer the output rather than having to specifically guided to it. In the following examples, "R" is used to specify a request and "S" the steps taken.

Example

R: Put the apple next to the banana

S: Pick up apple, place next to banana

Input R: Stack a red cube on a blue cube on a yellow cube

Output S: Pick up red cube, place on blue cube, pick up yellow cube, place on top of red cube

The results from this model show that it works very well and is able to apply the style of the first task into different tasks with different type of object. However, just one example means if the model encounters something unexpected then it doesn't have enough examples to learn what to do in those situations

Example

R: Put the apple next to the banana

S: Pick up apple, place next to banana

Input R: There is two apples and a plate, put the orange onto the plate

Output S: Pick up apple, place next to other apple, pick up orange, place on plate

Few-Shot In Few-Shot several examples are given. This means more work has to be done to prep the model, but it will allow for better examples and to give it the power to deal with edge cases.

Examples

R: There is an apple and a banana. Put the apple next to the banana

S: Pick up apple, place next to banana

R: There is an apple and a plate. Place orange on a plate

S: Can't

R: There is an orange, pear and banana. Place the fruit in the bin

S: Can't

Input R: There is a sandwich and a chocolate bar and a plate. Place the apple on the plate

Output S: Can't

Having an example for each edge case could mean a very large input, instead, like in zero-shot, these could be described with text.

Examples

R: There are two apples and a bin, place both in the bin

S: Pick up apple, place in bin, pick up apple, place in bin

R: There are two apples and a plate, put the orange onto the plate

S: Can't

Pick up a plate or bin = can't

Input R: There is an apple a bin and a plate. move bin onto the plate

Output S: Can't

This combination of techniques can help the model learn how it should approach solving each task, either implying this information from previous examples or being explicitly told the rules.

Context

In several of the previous inputs, the current condition of the environment was specified just before the input. It is important to know whether the model can use facts and information given to it, such as this context, to infer the right solution. Fortunately, the model is able to do this, it is able to bring forward facts and modify them as the text evolves. In the following examples "C" is used to specify context of each situation, where each situation is a new paragraph.

Examples

A fruit bowl contains one apple, one oranges and a banana

C: There is a banana, a melon and a plate

R: Put a banana and melon on the plate

S: Pick up the banana, place on the plate, pick up melon, place on the plate

C: There is a fruit bowl, a plate and a bag

R: Put one apple in a bag

S: Pick up apple, and place in bag

Input R: Put rest of fruit on a plate

Output S: Pick up orange, place on plate, pick up banana, place on plate

However, the ability of it to track these facts is limited. When approached with situations that involves counting and numbers, rather than a binary "in the bowl"/"out of the bowl" it fails to

keep track.

Examples

A fruit bowl contains two apples, one pear and a peach

C: There is a banana, a melon and a plate

R: Put a banana and melon on the plate

S: Pick up the banana, place on the plate, pick up melon, place on the plate

C: There is a fruit bowl, a plate and a bag

R: Put one apple in a bag

S: Pick up apple, and place in bag

Input R: Put rest of fruit on a plate

Output S: Pick up pear, place on plate, pick up peach, place on plate

Through testing, the model does have the ability to keep track of the count of objects but only when this is the specific task.

Examples

A fruit bowl contains two apples, one pear and a peach

C: There is a fruit bowl, a plate and a bag

R: Put one apple in a bag

S: Pick up apple, and place in bag

Input What fruit is in the fruit bowl?

Output There is an apple, a pear and a peach in the fruit bowl.

So when this task is part of some other task the model isn't able to do achieve this and having to keep track of two tasks might be too complicated for it to reason over.

Abstraction

The large language model is trained on a large number of concepts and topics that it can apply to the current task, it doesn't just have to rely on the information given to it in the few examples.

Examples

C: There is a blue cube, a yellow cube and a red sphere

R: Stack the cubes

S: Grab yellow cube, Place on Blue cube

Input C: There is two slices of bread, a slice of ham, a slice of cheese, a tomato and a plate

R: Make a vegetarian sandwich on the plate

Output S: Grab one slice of bread, Place on plate, Grab slice of cheese, Place on bread, Grab tomato, Place on cheese, Grab second slice of bread, Place on tomato

Because of this, the language model can be applied to a lot more human concepts that might be more natural for us to give as commands. It is noted that sometimes the model struggles to completely relate all these ideas in a way that the model can ensure a well-structured output. More examples of the abstract task helps, as the model does perform better when running in a way that is closer to what it has seen before, but giving too many examples may make the model struggle with a large input.

Translation

The robot policy model is only trained to interact with specific names of objects, such as "slice" or "cube". To apply it to real world objects, even if they look the same, would require retraining.

Large language models have strong translation capabilities between real world languages and can also be taught new translations/word associations as well as when to use them.

Examples

Coca Cola = red cylinder
Sprite = green cylinder
Apples = green sphere
French Fries = red box
Bread = yellow slice
Ham = red slice

C: There is a red cylinder, a green cylinder, yellow slice

R: Put the coke next to the spite

S: Pick up red cylinder, place next to green cylinder, done

C: There is a green cylinder, a red cylinder, a red box, a green sphere and a bin

Input R: Put the drinks in the bin

Output S: Pick up green cylinder, place in bin, pick up red cylinder, place in bin

C: There is two yellow slices, a red slice and a plate

Input R: Make a sandwich on the plate

Output S: Pick up one yellow slice, place on plate, pick up red slice, place on yellow slice, pick up other yellow slice, place on top of red slice

This capability makes the model very useful. It will allow it to be convert many different inputs into the output domain which could allow for the robot to be made applicable to new settings without being retrained.

4.3.3 Designing the input

The GPT-3 language model is powerful in many ways, but also needs to be supplied with the right information to succeed reliably. To ensure the model will be successful in this project, the examples given to it need be designed carefully to ensure it will be robust and take advantage of the whole power of the language model. The examples given need to be enough so that the model will be confident in the output and that it will be correct, but as longer texts can perform poorly it will also need to be concise.

The input needs to be able to teach the model:

- The structure of the output - Each step separated by commas for easy processing of output
- The possible commands - What moves the model can do: Grab, Place next to, Place on, Place in
- The limitations of the robot - What it can't do. For example: pick up two objects at once, pick up a bin, place on top of a sphere, use an object that isn't there or it can't access (under another object/in a bin)
- How to use translations - To translate from the input domain to the output domain
- How to deal with failure - If a model crashes or fails a step, what steps can it do to fix it?

These concepts will be taught to the robot using a few-shot input, with multiple examples and text explaining some rules. This input will also have to be able to be combined with the context of the current scenario, any additional facts that might be needed for that specific scenario and the new request for the robot, in a way that makes sense to the model and allows it to produce the best output.

With these considerations, the following 2 inputs were created as potential inputs to the model.

Input 1

Possible commands: Pick up, Place on top of, Place next to, Place in, Can't, Done

Only place in bin

Pick up object then Place object

All objects, bins, plates used must be in C

Pick up any bin = Can't

Pick up item in any bin = Can't

Place on any sphere = Can't

Place on any box = Can't

Place on any cylinder = Can't

If X placed on Y, can't pick up Y

C: There is a green cylinder, grey plate and a yellow cube

R: Place the green cylinder on the grey plate?

S: Pick up green cylinder, Place on grey plate, Done

C: There is a green sphere and yellow box

R: Place blue sphere in yellow bin

S: Can't

C: There is a red cube, blue cube and green cube

R: Stack the cubes

S: Pick up red cube, Failure-wrong object, Place, Pick up red cube, place on blue cube, Pick up green cube, Place on blue cube, Failure-wrong location, Pick up green cube, Place on blue cube, Done

Input 2

- C: There is a red cube, green sphere and a blue plate
R: Place the cube on the plate and the sphere next to it
S: Pick up cube, place on blue plate, pick up green sphere, place next to blue plate, done
- C: There is a yellow cube, green box and black sphere
R: Move the green object onto the cube
S: Pick up green box, place on yellow cube, done
- C: There is a red sphere, blue sphere and blue bin
R: Put the items in the bin
S: Pick up red sphere, place in blue bin, pick up blue sphere, place in blue bin, done
- C: There is a green box, white box and black bin
R: Put a box on the plate
S: Can't
- C: There is a red cube, white cylinder and yellow plate
R: Put the red cube on the white plate
S: Can't
- C: There is a blue cylinder, black sphere and white box
R: Place the blue cylinder on the sphere
S: Can't
R: Place the black sphere on the box
S: Can't
R: Put the box on top of the blue cylinder
S: Can't
- C: There is a plate and a bin
R: Put plate in bin
S: Can't
R: Put bin on plate
S: Can't
- C: There is a green slice, a red cube and a black plate
R: Put the red cube on the slice
S: Pick up red cube, place on green slice, done
R: Pick up green slice
S: Can't
- C: There is a red cube, blue cube and green cube
R: Stack the cubes
S: Pick up red cube, Failure wrong object, Place, Pick up red cube, place on blue cube, Pick up green cube, Place on blue cube, Failure-Wrong location, Pick up green cube, Place on blue cube, done

The first input focused on telling limitations of the model through the being told and the first input was focused on letting the model infer these rules. For entering facts or translations in both these model, these would be entered just before any new context/request/steps input.

4.3.4 Experiments

The two different inputs were evaluated on a set of 50 inputs, and their performance was ranked on whether the model would be likely to complete each task. The set (seen in the appendix B along with the full results) contains simple in domain requests, more abstract requests involving tasks/objects not seen before as well as tests involving translations, extra facts, recovering from failure and recognising impossible tasks.

Each output was judged manually by giving it one of three marks: Correct plan, semi-correct and wrong plan. The results of each language model in this task can be seen in table 4.3. These

| Type | Input 1 | | | Input 2 | | | Total |
|-----------------|---------|--------------|---------|---------|--------------|---------|-------|
| | Wrong | Semi-Correct | Correct | Wrong | Semi-Correct | Correct | |
| Normal | 0 | 0 | 11 | 0 | 1 | 10 | 11 |
| Abstract | 1 | 3 | 6 | 1 | 4 | 5 | 10 |
| Can't | 5 | 0 | 7 | 7 | 0 | 5 | 12 |
| Failure Recover | 0 | 2 | 4 | 0 | 2 | 4 | 6 |
| Translations | 2 | 1 | 3 | 1 | 1 | 4 | 6 |
| Facts | 2 | 0 | 3 | 2 | 0 | 3 | 5 |
| Total | 10 | 6 | 34 | 11 | 8 | 31 | 50 |

Table 4.3: Results of testing inputs on example tasks

results show that the two types of models perform similarly. With input 1 slightly outperforming input 2. Most of the errors for each model came in the Can't factor, with both failing to properly grasp some of the rules given to them. The other errors were mainly in translations and facts, where possibly better-defined tasks and facts could lead to stronger results. In normal and abstract tasks ,the model performed well, and quite well with error recovery too, with only dubious results coming in the harder tasks. This implies that as long as the model is used in a fairly standard way it should be able to make a tasks and recover from any problems it might encounter.

The input used for the robot planning system will be input 1. Not only did it outperform input 2 but it is smaller, therefore it will likely work better on when given longer inputs. However, some examples will be added, to reinforce some of the limitations given in the input that it struggled on during the tests and input 2, where examples were given, didn't. The additional examples were:

C: There is a red plate and a blue bin
R: Put red plate next to bin
S: Can't
R: Put blue bin on plate
S: Can't

C: There is a green sphere and yellow cylinder
R: Place green sphere on the cylinder
S: Can't

When tested, these inputs didn't change the performance in any other category but did move two tasks in Can't from Wrong to Correct meaning this input had a 76% success rate over the whole test set.

Furthermore, the example used to demonstrate error correction might confuse the model on how to achieve normal tasks and isn't needed unless the model has experienced an error. So, this example could instead only be inserted into the previous example when the system does experience an error.

4.4 Robot Planning System

Now each of the two main parts of the system are established, they can be combined to make a system that can execute commands. This system will first gather the information needed to get the language model to plan the command given to the system. This model would then return a list of instructions that will then have to be processed and fed to the robot command by command so it can be executed. The language model has shown its capabilities for identifying impossible instructions as well as failure recovery which will be useful tools to exploit. If the model says it can't run a high-level command, then it can just get the user to enter a different action and if an action fails, then the model could be reused to figure out the best steps to correct the situation. A flow chart showcasing the flow through this system is shown in figure 4.14

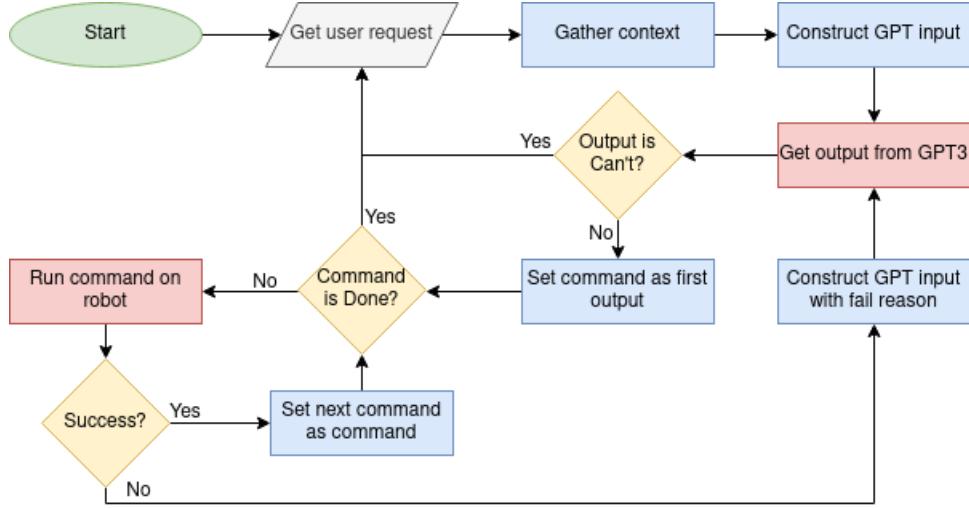


Figure 4.14: Flowchart detailing system flow

4.4.1 Detecting Failures

One key part of this system is detecting failures in actions. As shown in 4.2.3, the policy used to control the robot is not 100% successful in executing its actions, so it is important for the system to be able to retry actions that it has failed. This will mean the system will be much more likely in achieving the objective as it will aim to ensure each action is completed.

The aim of this project was to be able to implement all the parts of the model without getting the explicit details of the environment from the simulation. This was to mimic how the process would be done with a real robot. However, due to time constraints of the project, an implementation that could work in this way was not able to be produced. Though, to demonstrate how a system like this would operate within the robot planning system, a method was still created. If time was available, the planned method to do this would involve training a network that when given before and after images, as well as the instruction, state whether this task has been achieved and potentially why it went wrong.

The method created utilises two tools that were already constructed in the creation of the model. The first was the BERT classifier created in 4.2.3 with the architecture seen in figure 4.10. This classifier could be used to figure out the exact meaning of the input: the action being executed and the current target. This data could then be passed onto a system to evaluate the results, where the logic will be reused from the process used to evaluate the robot's policy model. This could work out if the model failed so any further action could be created.

This procedure should help the system be able to complete more high-level commands. It wouldn't be a perfect system; however, the large language model isn't reliable at detecting impossible commands and the robot could create a scenario where it is impossible to recover from. To stop the chances of the model being stuck in an infinite loop because of these possibilities, the number of times that this system will be activated will be capped at 5 per high level command.

4.4.2 Context

The other key part of the system is describing the robot's current environment, and then supplying this to the robot as context. While the system could operate without context, the domain of requests that could be given to the language model is decreased. Furthermore, the model can use this information to decide when a request can't be performed which is a very useful skill.

Unfortunately, much like the system to detect failures, a system to get the current context of the model without it having to interact with the simulation couldn't be developed in the time frame for the project. This sort of system could have been implemented using a Region-Based CNN (RCNN) to extract regions of the image containing potential objects (53) then have a classifier network trained to identify what object is in each region.

While this system could be replaced by human input, so still adhering to the design principles of the project, it was decided that just collecting the names and colours of the objects generated

for the simulation and using them as the context would be easier and less prone to human error.

4.4.3 Experiments

The whole model was deployed in a various different scenario and given a range of different tasks. Different scenarios were created for the robot by utilising the translation ability of the language model to map the objects the robot has been trained to interact with with real world objects applicable to the scenario.

The different scenarios that were tested over were:

- Commands using standard objects, For example, Group by shape, place this object here, put cubes in bins.
- Fruit and Junk food: Using the translation feature, look at sorting different types of food and organising them by how healthy they are.
- Making sandwiches: A more specific scenario. This will utilise two of one object, for the bread slices, so will be interesting how the model handles it as it wasn't trained like it. It also involves slices, which is has been shown to be the hardest item for it pick up.
- Making tea and coffee: A more abstract scenario, as liquids can't be simulated they will be represented by cubes which could be a hard problem
- Assortment of objects: Also using translations, this will place the robot in a scenario where it is surrounded by lots of different sorts of items that it then has to reason about.

The full list of tasks used is available in the appendix ??

| Type of Task | Completed (%) | Completed (%) + Error correction | Completed (%) + Error correction + In domain example | Can't Accuracy (%) (+ In domain example) |
|---------------------|---------------|-------------------------------------|--|--|
| Traditional | 70 | 83 | N/A | 67 |
| Healthy food | 50 | 50 | 66 | 33 |
| Sandwiches | 14 | 18 | 40 | 20 |
| Tea and Coffee | 58 | 67 | 83 | 50 |
| Assortment of Items | 63 | 63 | 79 | 38 |

Table 4.4: Results from running the system on different scenarios

Each task was run multiple times with varying the objects and inputs, but each with the same objective. The results were plotted in 4.4. Overall, the system performed well, especially when presented with simpler tasks and concepts. The more complicated tasks and scenarios were still able to see success, but that success was limited sometimes by the capabilities of either the language model or the robot. The more abstract tasks results were mixed, with some tasks being limited by the abilities of the robot and some because it was too difficult for the language model to reason about those objects.

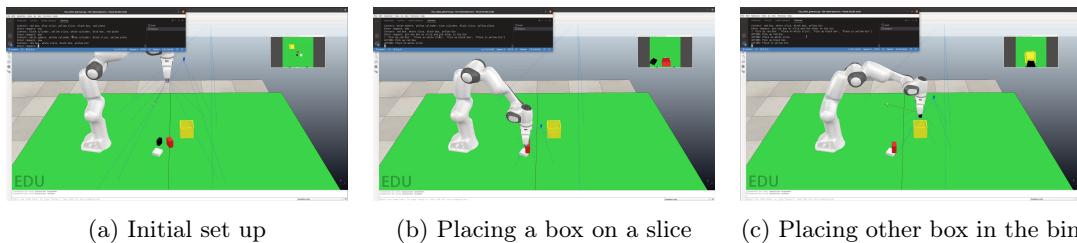


Figure 4.15: The model executing "put one box on slice and other in the bin". Video: <https://youtu.be/qTuC5kixCEY>

In the tests involving translation, the model were also tested when given an example to help the language model learn to output in these scenarios. These helped improve performance massively, as it allowed the model to get an understanding of how to map these scenarios to the objects the robot was trained upon and some concepts and facts given to the model. The error correction

did not work as well in this situation, with the few times it was created it struggled to produce a correct output.

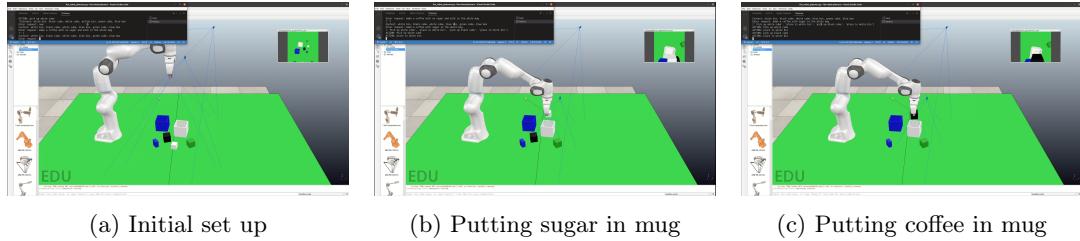


Figure 4.16: The model executing "make a coffee with sugar in the white mug". In this scenario the mugs are translated to bins, coffee to black cubes, sugar to white cubes, tea to green cubes and milk to blue boxes. Video: <https://youtu.be/XTqhpC-Fub8>

In multiple tests the error correction was used, this usually had good results and was able to correct and fix any mistakes. Most cases of this happened when the robot failed to pick up an object, usually a slice, or when placing on top of an object that had not been balanced correctly which causes these objects to fall. To help stop it from being stuck in a loop where it can't do one action, the start position of the robot was randomised in case approaching from a new angle would help.

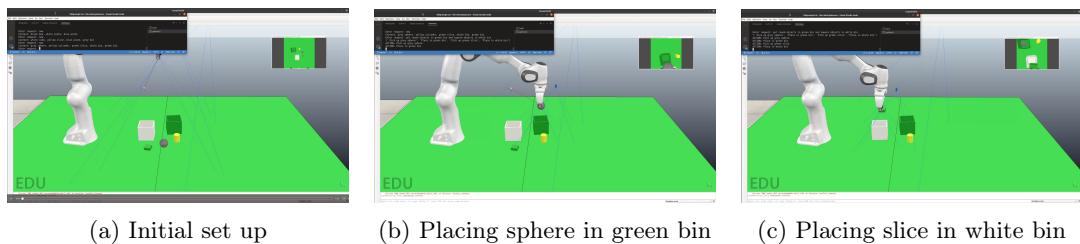


Figure 4.17: The model executing "place the round objects in the green bin and the square objects in the white bin". This shows the model completing the task, but it does assume that cylinders aren't round. Video: https://youtu.be/RcKa91V_txU

Also during testing some small quirks of the model were found through how it gave answers to some tasks. These were normally in tasks where it has to rely on some knowledge that it has learned from the dataset it was trained on. Some examples of this are: asking it to make tea and coffee would result in the addition of sugar but no milk, cylinders aren't considered round and cans of coke are considered healthy.

4.4.4 Evaluation

Successes of the system

The model performed best when having to manipulate objects using easy to understand tasks without having to perform any translation or abstract reasoning. Additionally with some tasks that involve some reasoning such as "group", "stack" or "organise" the model was still able to reliably produce successful inputs. With the robot's good performance on the different skills, most well planned tasks were able to be executed successfully without any failures. In all of these tests, the output of the language model would follow the output structure shown to it almost perfectly, this showed that giving the system the context as well as capabilities of the model allows it to be grounded easily in the environment without the addition of other approaches that involved a middle step such as the "can" system in Ahn et al. (7) or k-nearest neighbours in Huang et al. (9).

Furthermore, with simple tasks in the translated environments, the system can apply what it knows to these environments and complete tasks using names of objects not known by the robot. This is a good strength as it demonstrates language models are a tool that would allow robot systems such as these to operate on objects that it hasn't been explicitly trained on but told how to relate it to ones it does know. The model was also able to use natural text facts to guide the system on additional information important to the scenario. This also shows how the approach of

the model can be updated with new information and it can also be used to tweak the model to produce better outputs in each scenario.

The impossible task detection model, while not completely reliable, is still able to work for more obvious statements and, on occasion, some more complex tasks and translation tasks. This capability of the language model worked very well as, throughout the experiments, none of the possible tasks were ever misclassified, which shows the ones it did predict that were impossible it was very confident in doing so. In some of the impossible task detection where it didn't classify it as impossible, it instead related it to the available objects to achieve a similar goal. For example, in one task when presented with the task "place cube on white plate" where no white plate existed, it resulted in placing the cube on a white slice that did exist. Although wrong, this re-imagining of the task to use commands that were available also shows the language model can ground itself using just previous examples.

The system also performed well when it had to perform error correction, managing to correct the system the most of the time. The previous examples given to the system was able to help guide how it should react in each fail safe, and it understood how to continue the plan reusing past generated inputs no matter how long the plan already is.

Limitations of the system

The system and models used still do have limitations and failures.

The robot policy that was created was trained with a dataset that had no shared colours amongst the grabbable objects, this has led to a case where when tested on that exact scenario the model would not be able to pick the correct object up. This does not show any limitations of the method, as the model is still able to differentiate between objects of the same shape by colour and differentiate between objects and non-grabbable objects of the same colour, but rather a failure to of the training set to accurately represent the possible environments it will face.

The ability for the model to recognise impossible outputs is also not completely consistent. The system got confused if an object to interact with in the task, that didn't exist in the context, shared a colour with an object in the context quite often. Also, with more everyday items, such as bins and plates, the language model would sometimes just assume their existence resulting in sequences being produced that can't be created. With extremely abstract tasks that imply other abilities the robot can't perform; the model isn't able to identify that this scenario isn't possible as it is out of the scope of the examples. This problem likely won't be solvable by more examples, as the number of actions the robot can't do is much larger than can be defined in the examples.

Finally, sometimes there is unreliability in the output in larger more complicated outputs. If the situation is too abstract for the model it could un-ground itself from the robot and produce a set of actions that, although correct, aren't executable by the robot. Sometimes it would keep its grounding, but the series of actions produced don't complete the goal but are still executable. These situations would occur no matter the implementation of the planning system, as it is due to the language model's failure to reason a good output. It could be solved with fine tuning the model or giving more detailed examples, but especially for abstract and translation tasks there might be too many possibilities to distil in these. However, these issues only occur when the model is being pushed to its limit in terms of input and output length and reasoning skills on much harder topics which might not always be the type of inputs a system like this would regularly get if implemented in the real world.

Chapter 5

Conclusion

5.1 Insights gained

In this project the ability and possible applications of language models as planners have been assessed. Also, how language models can be used to train robot policies have been investigated and implemented and a system to combine these two aspects have been designed, implemented and evaluated. The results of these investigations and experiments on the different applications of language models has lead to a variety of different insights that can be gained from the process.

- Large language models can work well as planners. While they may not be as reliable as traditional robot planners, their large knowledge base and excellent few-shot learning abilities make it much easier for them to be created and applied to a variety of situations without having to recreate models by hand.
- These large language models can be constrained without the need for additional systems with good designing of the output domain and the examples given to the language model. As language models struggle with larger text lengths, this sort of methodology for planners might only be applicable if the range of possible actions is limited and the environment is well defined.
- These planners also are able to use the its context and facts given to it to better plan and reason about its environment. This is useful for when the type of objects surrounding it can change or its objectives can change so it can easily be moved from scenario to scenario. However, when it has to perform more complicated tasks or reasoning then sometimes it fails to output a sensible output.
- The translation abilities of the language model can be utilised to adapt the planning system to various objects and situations. Instead of retraining the policies of the robot to be used for similar objects, the language model could be told how to adapt the current situation to the known situation and it can produce reliable outputs.
- The language model is very adaptable in its plans and understands how errors work and can also plan how to recover from them. This is extremely useful and allows for plans to be adapted in-case the robot fails to execute any of the commands.
- The detection of whether a task is possible or not is able to be reasoned about by a language model, however only with simpler scenarios.
- Finally, language models such as BERT work really well as models to be used to control robots. Their generality allows for various different commands to be imputed and it can still reliably execute the right command when given an input that wasn't expected.

5.2 Future work

This project does leave potential for further improvements and additional ideas to be explored. Some improvements to the robot policy model could be introduced, such as a better training set

including duplicate colours and objects as well as in more varied locations (on top of a block, in bin etc.). This could help the model perform better in several scenarios that were tested. Also, the complexity of the robot could be increased by adding additional actions, objects and maybe the ability to move between various locations which would widen the scope of the investigation in the ability for language models to plan.

For the language model, more work could be done in investigating different styles of input as well as previous examples to enable the model to perform the best in more scenarios. Additionally, the model showed it can struggle when given two tasks at once, so splitting the impossible task detection to its own separate task for the language model could increase performance.

An implemented context collection mechanism could be extended to also indicate locations of objects (in hand, on plate). This could then be used to help the model keep track of the environment by having the language model predict step by step then updating the context. Also, the current context could be used to dynamically update the previous examples given to the model so that they would be more applicable to the current task.

Additionally, this whole system could be applied to a real robot. This could be done using sim-to-real techniques (54) or by retraining using the same process with real world demonstrations. With more variability in the real world, such as lighting conditions as well as more complicated physics and environments, this might require a much larger number of demonstrations.

Finally, a memory system could be implemented to the robot model. If tasks involve the manipulation of duplicate objects, such as move both red cubes onto the plate, having the ability to remember which objects were previously moved could improve its robustness. This could be implemented by encoding the model's history using a recurrent neural network, but the process of training this network would be difficult and might require the actions to be trained in whole planned sequences.

5.3 Final remarks

Overall the project manages to reach many of its objectives, and is able to produce a system that can plan for a grab and place robot rather well. Using natural language for goal setting does make the task more difficult to solve classically but does allow for a better ease of use in defining an environment and switching between them. It is also a much more human way of defining these goals which make their application in everyday robots and systems more likely.

Chapter 6

Ethical Issues

This project presents few ethical issues that needs to be considered. Although the project used a simulated environment and did not use any personal information, the implications of Large Language Models used in this way in the realm of robotics is important to investigate.

The Large Language models that will be used in this project, such as BERT and GPT-3, are trained from large corpus of data found on the web. This collection is vast and almost impossible to verify will contain no biases that can affect its performance and outputs. Some Language Models, such as the word2vec embedder, have been found to contain sexist biases (55) as a result of underlying bias in the data that it was trained on. Biases such as these could affect how the language models behaves when being asked to plan different tasks. Most tasks used in this project was very object based so weren't be affected by these, but some tasks such as "sort items into X and Y" use intuition in the language model learned in training which has the chance of echoing potentially harmful stereotypes.

Furthermore, this project only tested a small number of environments and scenarios. As GPT-3 is trained on an array of data then it can work in a large number of contexts. Not all of these contexts were able to be covered during testing, so it is impossible to assume this method will be able to create safe outputs for all scenarios, which is important if this method is implemented in the real world especially around humans.

This uncertainty around how these models will act is further compounded by the black box nature of them. GPT-3 contains 175 billion parameters (5) so it will be near impossible to predict what any input will give. This also means it's hard to get explanations to why it has taken certain actions, which could be important if used in a medical environment.

A potential use for a system like the one being tested in this project, is in a military capacity. Having a system that can help robots plan their future actions could aid in the development of fully autonomous weapons. Fully autonomous weapons are "weapons systems that use artificial intelligence (AI) to identify, select, and kill human targets without human intervention" (56). For many, these sorts of weapons are considered unethical, and attempts have been taken to ban these from use, including an open letter from many top members of industry (57) and robotics, as well as in recent discussions at the UN (58).

Bibliography

- [1] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *arXiv preprint arXiv:2010.01083*, 2020.
- [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
- [3] Alec Radford, Karthik Narasimham, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.
- [4] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [5] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- [6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [7] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.
- [8] Peter A Jansen. Visually-grounded planning without vision: Language models infer detailed plans from high-level instructions. *arXiv preprint arXiv:2009.14259*, 2020.
- [9] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.
- [10] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.
- [11] Eda Kavlakoglu. Ai vs. machine learning vs. deep learning vs. neural networks: What's the difference? *IBM*, 2020.
- [12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [14] Kaiming He, X Zhang, S Ren, and J Sun. Deep residual learning for image recognition." computer vision and pattern recognition (2015). *Google Scholar There is no corresponding record for this reference*, pages 770–778, 2015.
- [15] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. corr abs/1412.1710 (2014). *arXiv preprint arXiv:1412.1710*, 2014.

- [16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.
- [17] Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanlin Peng. Towards understanding regularization in batch normalization. *arXiv preprint arXiv:1809.00846*, 2018.
- [18] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.
- [19] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [20] J Johnson, B Hariharan, L van der Maaten, L Fei-Fei, CL Zitnick, and RB Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. corr, abs/1612.06890, 2016.
- [21] Nir Arbel. How lstm networks solve the problem of vanishing gradients, May 2020. URL <https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577>.
- [22] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [24] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.
- [25] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.
- [26] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.
- [27] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.
- [28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [30] Justyna Sarzynska-Wawer, Aleksander Wawer, Aleksandra Pawlak, Julia Szymanowska, Izabela Stefaniak, Michal Jarkiewicz, and Lukasz Okruszek. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304:114135, 2021.
- [31] Faiza Khan Khattak, Serena Jeblee, Chloé Pou-Prom, Mohamed Abdalla, Christopher Meaney, and Frank Rudzicz. A survey of word embeddings for clinical text. *Journal of Biomedical Informatics*, 100:100057, 2019.
- [32] Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 93–104, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1009. URL <https://aclanthology.org/D18-1009>.
- [33] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.
- [34] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

- [35] Edward Johns. Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.
- [36] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4693–4700. IEEE, 2018.
- [37] Jonathan Chang, Nishanth Kumar, Sean Hastings, Aaron Gokaslan, Diego Romeres, Devesh Jha, Daniel Nikovski, George Konidaris, and Stefanie Tellex. Learning deep parameterized skills from demonstration for re-targetable visuomotor control. *arXiv preprint arXiv:1910.10628*, 2019.
- [38] Simon Stepputtis, Joseph Campbell, Mariano Phiellip, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *Advances in Neural Information Processing Systems*, 33:13139–13150, 2020.
- [39] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *Proceedings of Robotics: Science and Systems*. doi, 10, 2021.
- [40] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pages 1113–1132. PMLR, 2020.
- [41] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.
- [42] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.
- [43] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494. IEEE, 2009.
- [44] Jinzhong Li, Ming Cong, Dong Liu, and Yu Du. Integrated classical planning and motion planning for complex robot tasks. In *International Conference on Intelligent Robotics and Applications*, pages 70–78. Springer, 2021.
- [45] Yuqian Jiang, Shiqi Zhang, Piyush Khandelwal, and Peter Stone. Task planning in robotics: an empirical comparison of pddl-based and asp-based systems. *arXiv preprint arXiv:1804.08229*, 2018.
- [46] Jeremy D Frank. Artificial intelligence: Powering human exploration of the moon and mars. In *ASCEND 2020*, page 4164. 2020.
- [47] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.
- [48] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018.
- [49] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. www.coppeliarobotics.com.
- [50] Stephen James, Marc Freese, and Andrew J Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*, 2019.
- [51] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

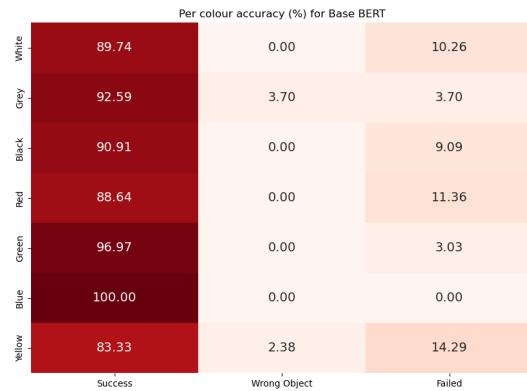
- [52] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [53] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [54] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744. IEEE, 2020.
- [55] Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *Advances in neural information processing systems*, 29:4349–4357, 2016.
- [56] An introduction to the issue of lethal autonomous weapons, Dec 2021. URL <https://futureoflife.org/2021/11/30/an-introduction-to-the-issue-of-lethal-autonomous-weapons/>.
- [57] S Hawking, E Musk, S Wozniak, et al. Autonomous weapons: an open letter from ai & robotics researchers. future of life institute, 2015.
- [58] Emma Farge. U.n. talks adjourn without deal to regulate 'killer robots'. *Reuters*, Dec 2021. URL <https://www.reuters.com/article/us-un-disarmament-idAFKBN2IW1UJ>.

Appendix A

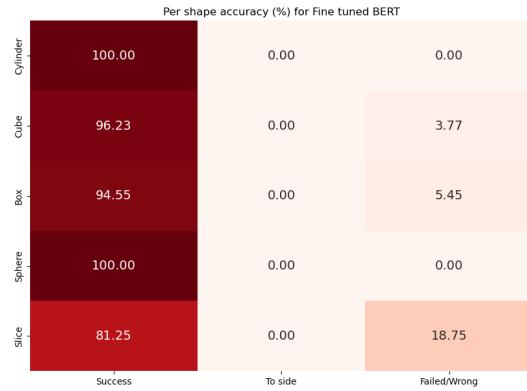
Detailed results for multi-policy model



(a) Per object results - Base BERT



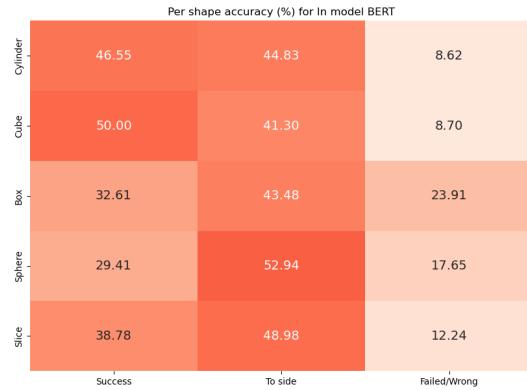
(b) Per colour results - Base BERT



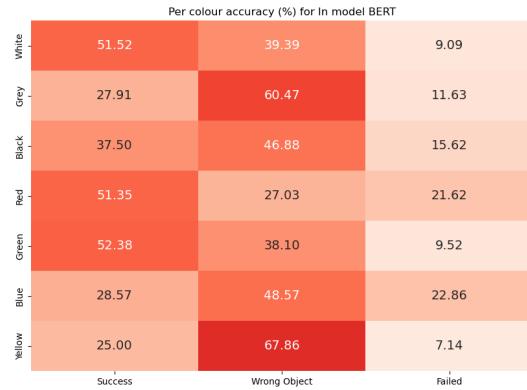
(c) Per object results - Fine tuned BERT



(d) Per colour results - Fine tuned BERT



(e) Per object results - In model BERT



(f) Per colour results- In model BERT

Figure A.1: Grab Results for each model

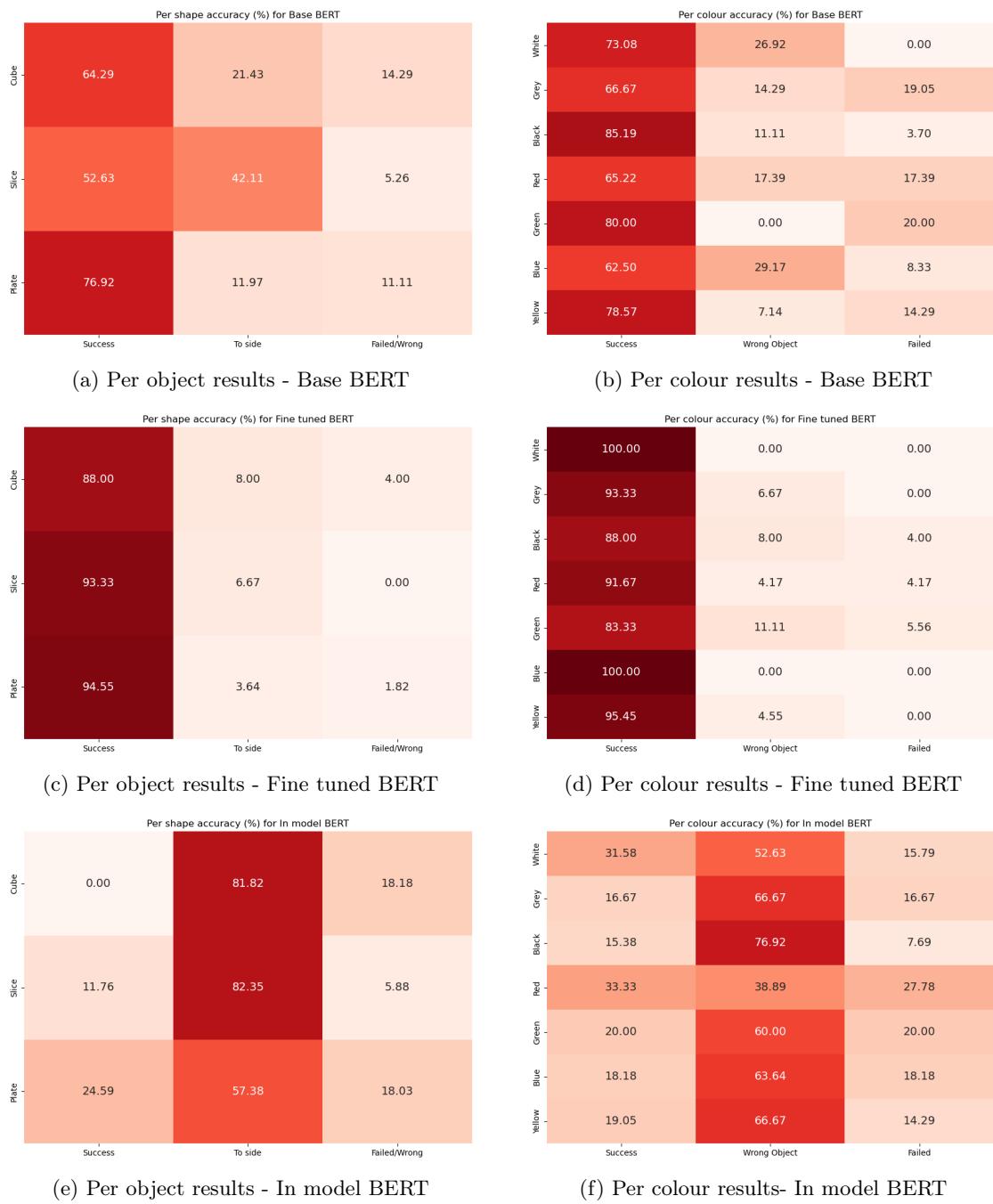


Figure A.2: Place on Results for each model

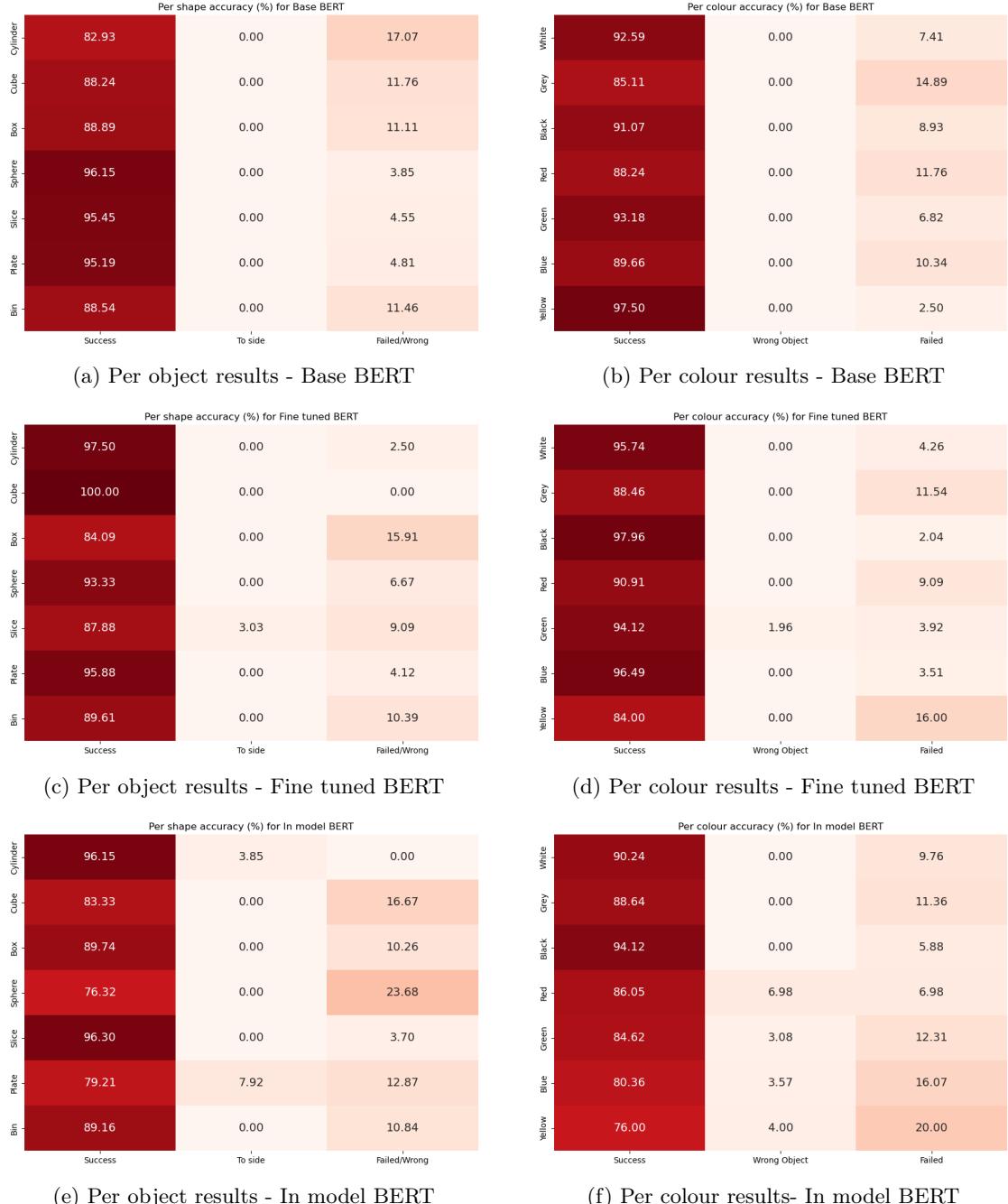


Figure A.3: Place by results for each model

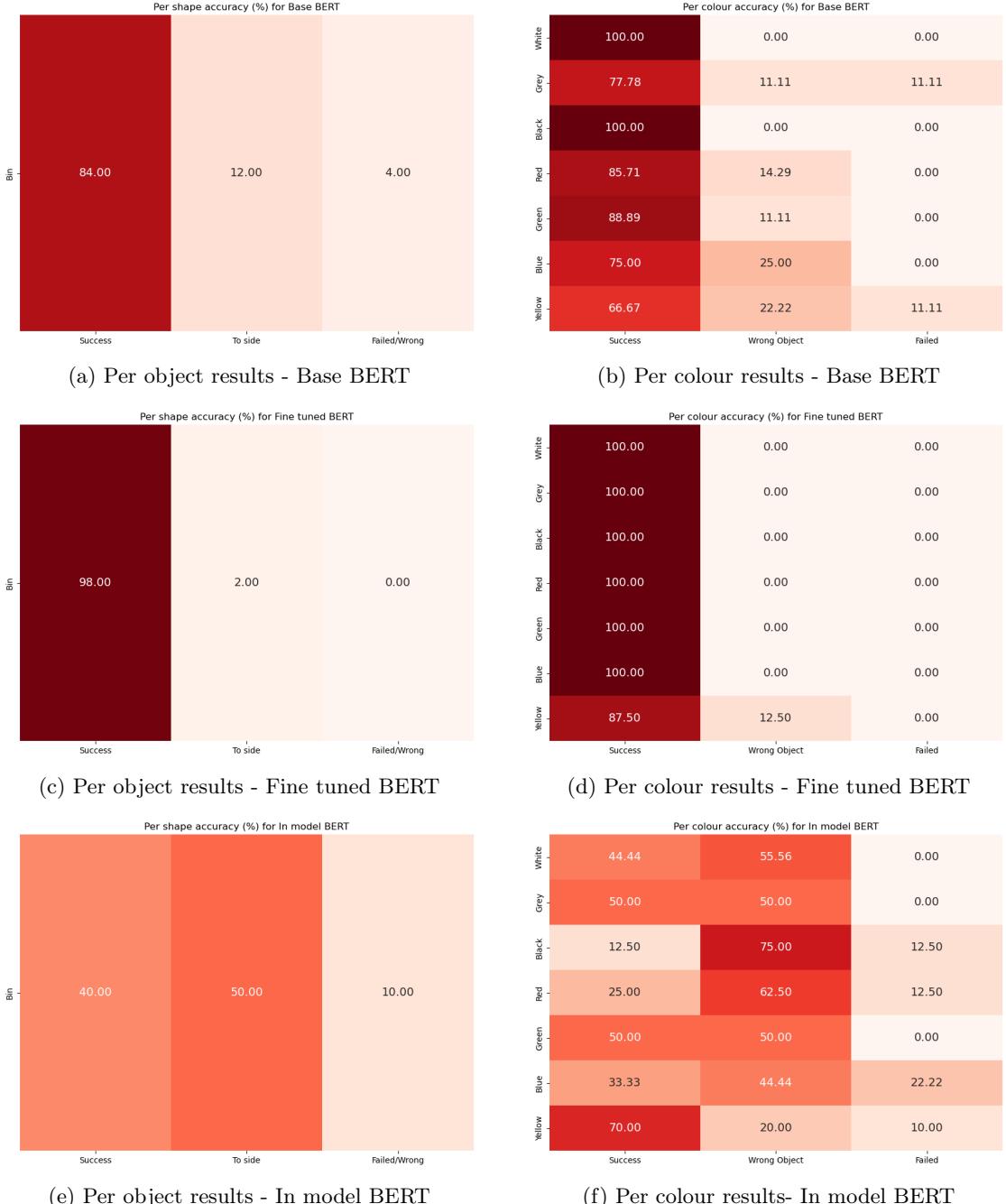


Figure A.4: Place in results for each model

Appendix B

LLM Test Results

| Type | Extra info | Context | Request | 1 score | 2 score |
|---------------------|---|---|--|---------|---------|
| 1 Normal | | There is a red cube, blue cube and a green plate | Place blue cube on plate | 2 | 2 |
| 2 Normal | | There is a blue sphere, green sphere and grey plate | Put spheres next to plate | 2 | 2 |
| 3 Normal | | There is a yellow box, grey box and yellow cube | Put yellow objects next to the grey box | 2 | 2 |
| 4 Normal | | There is a green sphere, blue box and white bin | Put everything in the bin | 2 | 2 |
| 5 Normal | | There is a yellow box, yellow sphere, red sphere, red cube, yellow bin, red bin | Sort items into their colour bins | 2 | 2 |
| 6 Normal | | There is a black cube, red cube and white cube | Make a tower from the cubes | 2 | 2 |
| 7 Normal | | There is a black sphere, white slice and green plate | Put sphere on slice | 2 | 2 |
| 8 Normal | | There is a black box, red box, green cylinder and red cylinder | Put the same shapes next to each other | 2 | 0 |
| 9 Normal | | There is a black box, red box, yellow sphere and a white plate | Put boxes on plate | 2 | 2 |
| 10 Normal | | There is a green cylinder, red cylinder a black bin and a black plate | Put green cylinder on plate and red cylinder in bin | 2 | 2 |
| 11 Normal | | There are two green cylinders and a black sphere | Move green cylinders next to sphere | 2 | 1 |
| 12 Can't | | There is a blue sphere, red box and green plate | Place yellow cube on plate | 2 | 2 |
| 13 Can't | | There is a blue sphere and green cylinder | Place sphere in black bin | 2 | 2 |
| 14 Can't | | There is a black cube and blue plate | Throw cube in bin | 2 | 0 |
| 15 Can't | | There is a yellow cylinder and red sphere | Place sphere on green plate | 2 | 2 |
| 16 Can't | | There is a blue sphere, green bin | Put sphere in bin S: Pick up sphere, place in bin, done R: | | |
| 17 Can't | | There is a blue plate and green plate | Put sphere next to bin | 2 | 2 |
| 18 Can't | | There is a blue bin and black bin | Place green plate on other plate | 0 | 0 |
| 19 Can't | | The is a red sphere | Throw blue bin in black bin | 2 | 2 |
| 20 Can't | | The is a red slice, black slice | Roll the sphere | 0 | 0 |
| 21 Can't | | There is a red cube, blue sphere and blue plate | Put red slice on black slice S: Pick up red slice, place on black slice, done R: Put red slice next to black slice | 0 | 0 |
| 22 Can't | | There is a green sphere and blue cylinder | Place objects in blue bin | 0 | 0 |
| 23 Can't | | There is a red sphere and blue cylinder | Place sphere on cylinder | 2 | 0 |
| 24 Abstract | | There is an apple, a banana, an orange and bin | Place cylinder on sphere | 0 | 0 |
| 25 Abstract | Books are clean in a stack | There is a red book, yellow book and green book | Put fruit in bin | 2 | 2 |
| 26 Abstract | | There are two slices of bread, a slice of ham and a slice of cheese | Clean the books | 2 | 1 |
| 27 Abstract | | There are two slices of bread, a slice of ham and a slice of cheese | Make a sandwich | 2 | 1 |
| 28 Abstract | | There is a box of cereal, a box of biscuits, an apple and a banana | Make a vegetarian sandwich | 1 | 1 |
| 29 Abstract | | There is a can of coke, a can of sprite, a bottle of water and a bin | Put fruit next to each other and everything else next to each other | 2 | 2 |
| 30 Abstract | | There is a coke can, a raspberry, a green apple and a plate | Throw away fizzy drinks | 2 | 2 |
| 31 Abstract | | There are two pens, one pencil, one green cup and on yellow cups | Put red objects on plate | 2 | 0 |
| 32 Abstract | | There are two blue blocks, one red block and three yellow blocks | Sort items into their own cup | 0 | 1 |
| 33 Abstract | | There is an apple, a banana, an onion, a biscuit and a bowl | Build a yellow tower | 1 | 2 |
| 34 Translations | Slice = Book, Blue=Sci-fi, Red=Romance, Green=Mystery , Yellow=Non-fiction | There are two blue slices a green slice and a yellow slice | Make a fruit bowl | 1 | 2 |
| 35 Translations | Green sphere = Apple, Red sphere=Peach, Red box = fries | There is a red sphere, green sphere, red box and a bin | Stack fiction books | 0 | 0 |
| 36 Translations | Yellow slice = bread, red slice = ham, green slice = lettuce | There are two yellow slices, one red slice and a plate | Put non fruit in bin | 2 | 2 |
| 37 Translations | Cup = White bin, Coffee = black cube, Sugar = white cube | There is a white bin, black cube and white cube | Put blue cube next to green plate S: Pick up blue cube, place next to green plate, fall-placed on | 0 | 2 |
| 38 Translations | White sphere = Football, Black sphere = bowling ball, yellow sphere = volley ball | There is a yellow, white, black sphere and bin | Make sandwich on the plate | 2 | 2 |
| 39 Translations | White slice = Thesis, Red cylinder = Empty can, yellow box = empty box | There is a white slice, two red cylinders, yellow box and a bin | Make coffee with no sugar | 2 | 2 |
| 40 Error Correction | | There is a red sphere and a blue cube | Put blue cube next to red cube S: Pick up blue cube, place next to red cube, fall-placed next to | 2 | 2 |
| 41 Error Correction | | There is a red sphere and a blue cube | Put red sphere next to blue cube S: Pick up red sphere, fail - wrong object, | 2 | 2 |
| 42 Error Correction | | There is a blue cube and green plate | Put red sphere next to blue cube S: Pick up red sphere, fail - didn't grab, | 2 | 2 |
| 43 Error Correction | | There is a blue cube and green plate | Put blue cube next to green plate S: Pick up blue cube, place next to green plate, fall-placed on | 2 | 2 |
| 44 Error Correction | | There is a blue cube and green bin | Put blue cube next to green plate S: Pick up blue cube, place on green plate, fall-placed next to | 2 | 2 |
| 45 Error Correction | | There is a blue cube and red cube | Put blue cube next to green bin S: Pick up blue cube, place next to green bin, fall-placed in | 1 | 1 |
| 46 Facts | Can no longer pick up blue objects | There is a red cube, a blue cube and plate | Place blue cube next to red cube S: Pick up blue cube, fail - wrong object, place, pick up blue cube, fail-nothing grabbed | 1 | 1 |
| 47 Facts | Towers are only made of yellow slices | There are three yellow slices and a red cube | Put all objects on the plate | 0 | 2 |
| 48 Facts | All items in C are doubled | There is a red cube and a blue bin | Make a tower | 2 | 0 |
| 49 Facts | Spheres go in the blue bin | There is a red cube, a white sphere, a green bin and a blue bin | Put all cubes in the bin | 2 | 0 |
| 50 Facts | Blue objects are actually green | There is a blue cube, grey sphere, green cylinder and plate | Put all the objects in a bin | 0 | 2 |
| | | | Put green objects on the plate | 2 | 2 |