



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

---

# Embodied Crossmodal Language Learning Using Neurocognitively Plausible Mechanisms

## Dissertation

submitted to the University of Hamburg  
with the aim of achieving a doctoral degree at the  
Faculty of Mathematics, Informatics and Natural Sciences,  
Department of Informatics

**Ozan Özdemir**

Hamburg, 2024

---

Day of submission:

17 September 2024

Day of oral defence:

5 December 2024

Dissertation Committee:

Prof. Dr. Stefan Wermter (reviewer, advisor)

Dept. of Informatics

University of Hamburg, Germany

Prof. Dr. Loo Chu Kiong (reviewer)

Dept. of Artificial Intelligence

Universiti Malaya, Malaysia

Prof. Dr. Chris Biemann (chair)

Dept. of Informatics

University of Hamburg, Germany

Prof. Dr.-Ing. Timo Gerkmann (deputy chair)

Dept. of Informatics

University of Hamburg, Germany

---

## Abstract

Despite the remarkable progress made by advanced foundation models in computer vision and natural language processing, the research on language-conditioned robotic object manipulation continues to lag behind, underscoring the need for more in-depth exploration. As evidenced by the research on early childhood development, embodiment, i.e. actively changing the environment while perceiving it via multiple senses, is crucial for language learning in human infants. Inspired by this phenomenon, embodied language learning for artificial agents tries to emulate the process of early language acquisition with computational models. As most approaches focus on language-instructed action execution but cannot produce language themselves, language learning for robots remains a challenge.

This doctoral work aims to achieve unrestricted language learning with embodied artificial agents via crossmodal binding between linguistic, visual and motor signals, exploiting neurocognitively plausible mechanisms such as multimodal fusion, channel separation and environmental feedback. We begin with an existing bidirectional action-language translation model that can only map one robot action to one language description. To relax this strict mapping, we integrate the stochastic variational Bayes method into the hidden space, enabling one-to-many associations between actions and language descriptions. We also utilise channel separation in vision to distinguish object colours more accurately. However, the resulting approach can only work with predefined grammar-based instructions. By employing a pre-trained language model, we allow the model to accept natural language instructions as input. Since this model needs to be adjusted based on the desired translation direction, we replace the implicit loss-based binding mechanism between the action and language streams with an explicit gated-network-based multimodal fusion technique, allowing training-time consistent full model utilisation during inference without expert intervention. When we train this novel model with increasingly unlabelled data, its performance in language production and action execution deteriorates. To counter this drop in performance, we adopt a self-attention-based multimodal fusion network to learn efficiently in a predominantly unsupervised fashion.

We also tackle generalisation to continuous action spaces in various object manipulation tasks by developing a two-stage learning concept, asymmetrically combining supervised learning with reinforcement learning. This novel approach improves action precision on various object manipulation tasks, but it is limited to robotic task-related language. To address this limitation, we introduce a novel multimodality fusion technique incorporating our bidirectional action-language translation model into a large language model (LLM), combining the language skills embedded in LLMs and the sensorimotor capabilities of robotic object manipulation. Lastly, for speech processing and object-agnostic robotic manipulation in the real world, we devise a modular human-robot interaction approach primarily composed of pre-trained foundation models such as vision-language, speech recognition, text-to-speech and object localisation, leading to free-flowing open-ended

conversation, scene understanding and task-specific open-vocabulary object manipulation skills in the real world.

To summarise, this thesis makes several key contributions. It builds a flexible action-language translation model architecture that remains consistent during training and testing. By combining multiple learning paradigms, it enhances the execution of language-instructed actions. The thesis enables free-flowing human-robot dialogue and motor control by developing a modular approach composed of foundation models. Furthermore, it introduces a data-efficient intra-LLM fusion technique that endows action-language models with LLM capabilities. Additionally, it demonstrates the efficiency of self-attention-based multimodal fusion in action-language associations. As a result, this doctoral research is a valuable step towards fully autonomous embodied companion agents capable of assisting humans in their everyday tasks through advanced communication and sensorimotor skills.



## Zusammenfassung

Trotz der bemerkenswerten Fortschritte, die durch fortschrittliche Basismodelle im Bereich der Bildverarbeitung und der Verarbeitung natürlicher Sprache erzielt wurden, hinkt die Forschung zur sprachlich bedingten Manipulation von Robotern mit Objekten weiterhin hinterher, was die Notwendigkeit einer tiefergehenden Erforschung unterstreicht. Wie die Forschung zur frühkindlichen Entwicklung zeigt, ist die Verkörperung, d. h. die aktive Veränderung der Umgebung bei gleichzeitiger Wahrnehmung mit mehreren Sinnen, für den Spracherwerb bei Kleinkindern von entscheidender Bedeutung. Inspiriert von diesem Phänomen versucht das verkörperte Sprachenlernen für künstliche Agenten, den Prozess des frühen Spracherwerbs mit Computermodellen nachzuahmen. Da sich die meisten Methoden auf die Ausführung von Handlungen unter Anleitung von Sprache konzentrieren, aber selbst keine Sprache produzieren können, bleibt das Sprachenlernen für Roboter eine Herausforderung.

Diese Doktorarbeit zielt darauf ab, uneingeschränktes Sprachenlernen mit verkörperten künstlichen Agenten über eine multimodale Verbindung zwischen sprachlichen, visuellen und motorischen Signalen zu erreichen, indem neurokognitiv plausible Mechanismen wie multimodale Fusion, Trennung der Verarbeitungs Kanäle und Feedback von der Umgebung genutzt werden. Wir beginnen mit einem bestehenden bidirektionalen Aktions-Sprach-Übersetzungsmodell, das nur eine Roboteraktion auf eine Sprachbeschreibung abbilden kann. Um diese strenge Zuordnung zu lockern, integrieren wir die stochastische Variations-Bayes-Methode in dem latenten Raum und ermöglichen so ein-zu-viele Assoziationen zwischen Aktionen und Sprachbeschreibungen. Wir nutzen auch die Trennung der Kanäle in der Bildverarbeitung, um Objektfarben genauer zu unterscheiden. Der daraus resultierende Ansatz kann jedoch nur mit vordefinierten grammatikbasierten Anweisungen arbeiten.

Durch den Einsatz eines vortrainierten Sprachmodells können wir dem Modell erlauben, Anweisungen in natürlicher Sprache als Eingabe zu akzeptieren. Da dieses Modell je nach gewünschter Übersetzungsrichtung angepasst werden muss, ersetzen wir den impliziten, verlustbasierten Bindungsmechanismus zwischen den Aktions- und Sprachströmen durch eine explizite, auf einem Gated-Network basierende multimodale Fusionstechnik, die während der Trainingszeit eine konsistente vollständige Modellnutzung während der Inferenz ohne Eingreifen eines Experten ermöglicht. Wenn wir dieses neuartige Modell mit zunehmend unmarkierten Daten trainieren, verschlechtert sich seine Leistung bei der Sprachproduktion und Handlungsausführung. Um diesem Leistungsabfall entgegenzuwirken, setzen wir ein auf Selbstbeobachtung basierendes multimodales Fusionsnetzwerk ein, um überwiegend unüberwacht auf effiziente Weise lernen.

Wir befassen uns auch mit der Verallgemeinerung auf kontinuierliche Handlungsräume bei verschiedenen Objektmanipulationsaufgaben, indem wir ein zweistufiges Lernkonzept entwickeln, das überwachtes Lernen und Verstärkungslernen asymmetrisch kombiniert. Dieser neuartige Ansatz verbessert die Handlungspräzi-

sion bei verschiedenen Objektmanipulationsaufgaben, ist aber auf roboterbezogene Sprache beschränkt. Um diese Einschränkung zu beheben, führen wir eine neuartige multimodale Fusionstechnik ein, die unser bidirektionales Aktions-Sprach-Übersetzungsmodell in ein großes Sprachmodell (LLM) integriert und die in LLMs eingebetteten Sprachfähigkeiten mit den sensomotorischen Fähigkeiten der robotischen Objektmanipulation kombiniert. Schließlich entwickeln wir für die Sprachverarbeitung und objektagnostische Roboteranwendung in der realen Welt einen modularen Ansatz für die Mensch-Roboter-Interaktion, der in erster Linie aus vortrainierten Basismodellen wie Bild-Text-Verarbeitung, Spracherkennung, Text-to-Speech und Objektlokalisierung besteht und zu einer frei fließenden, offenen Konversation, Szenenverständnis und aufgabenspezifischen Objektmanipulationsfähigkeiten in der realen Welt führt.

Zusammenfassend lässt sich feststellen, dass diese Arbeit mehrere wichtige Beiträge liefert. Sie entwickelt eine flexible Architektur für die Übersetzung von Aktionen und Sprache, die während des Trainings und der Tests konsistent bleibt. Durch die Kombination mehrerer Lernparadigmen wird die Ausführung von sprachgesteuerten Aktionen verbessert. Die Arbeit ermöglicht einen frei fließenden Dialog zwischen Mensch und Roboter und motorische Kontrolle durch die Entwicklung eines modularen Ansatzes, der aus Basismodellen besteht. Außerdem wird eine dateneffiziente Intra-LLM-Fusionstechnik vorgestellt, die Aktions-Sprachmodelle mit LLM-Fähigkeiten ausstattet. Darüber hinaus wird die Effizienz der auf Selbstaufmerksamkeit basierenden multimodalen Fusion bei der Verknüpfung von Aktion und Sprache demonstriert. Im Ergebnis ist diese Doktorarbeit ein wertvoller Schritt auf dem Weg zu vollständig autonomen, verkörperten Agenten, die den Menschen durch fortgeschrittene Kommunikation und sensomotorische Fähigkeiten bei seinen alltäglichen Aufgaben unterstützen können.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Research Objectives . . . . .	2
1.3	Overview of Main Approaches and Novel Contributions . . . . .	4
1.4	Thesis Organisation . . . . .	6
<b>2</b>	<b>Related Work</b>	<b>7</b>
2.1	Language-to-Action Translation . . . . .	8
2.2	Action-to-Language Translation . . . . .	18
2.3	Bidirectional Translation . . . . .	20
<b>3</b>	<b>One-to-Many Action-to-Language Translation through Variation in Latent Space</b>	<b>27</b>
3.1	Introduction . . . . .	27
3.2	Proposed Method: Paired Variational Autoencoders (PVAE) . . . . .	29
3.2.1	Model Architecture . . . . .	30
3.2.2	Language Autoencoder . . . . .	31
3.2.3	Action Autoencoder . . . . .	31
3.2.4	Visual Feature Extraction . . . . .	32
3.2.5	Sampling and Binding . . . . .	32
3.2.6	Loss Function . . . . .	33
3.2.7	Training Details . . . . .	33
3.3	Experiments and Results . . . . .	33
3.3.1	Experiment 1: Three Colour Alternatives . . . . .	35
3.3.2	Experiment 2: Six Colour Alternatives . . . . .	36
3.4	Discussion . . . . .	36
3.5	Conclusion . . . . .	37
<b>4</b>	<b>Advanced Language Comprehension via a Pre-trained Language Model</b>	<b>39</b>
4.1	Introduction . . . . .	39
4.2	Proposed Method: PVAE-BERT . . . . .	42
4.2.1	Language Variational Autoencoder . . . . .	43
4.2.2	Action Variational Autoencoder . . . . .	44
4.2.3	Visual Feature Extraction . . . . .	45
4.2.4	Sampling and Binding . . . . .	46

4.2.5	Loss Function . . . . .	46
4.2.6	Transformer-Based Language Encoder . . . . .	47
4.2.7	Training Details . . . . .	47
4.3	Experiments and Results . . . . .	48
4.3.1	Experiment 1: Action-to-Language Translation with Different Colours and Shapes . . . . .	50
4.3.2	Experiment 2: Bidirectional Translations with BERT . . . . .	53
4.3.3	Principal Component Analysis on Hidden Representations . . . . .	56
4.4	Discussion . . . . .	57
4.5	Conclusion . . . . .	59
<b>5</b>	<b>Learning Flexible Translation Between Action and Language through Explicit Multimodal Fusion</b>	<b>61</b>
5.1	Introduction . . . . .	61
5.2	Proposed Method: Paired Gated Autoencoders (PGAE) . . . . .	64
5.2.1	Language Autoencoder . . . . .	64
5.2.2	Action Autoencoder . . . . .	65
5.2.3	Gated Multimodal Unit (GMU) Bottleneck . . . . .	66
5.2.4	Task Signals . . . . .	66
5.2.5	Visual Feature Extraction . . . . .	67
5.2.6	Loss Function . . . . .	67
5.2.7	Training Details . . . . .	68
5.3	Experiments and Results . . . . .	68
5.3.1	Action-to-Language Translation . . . . .	69
5.3.2	Language-to-Action Translation . . . . .	71
5.3.3	Language-to-Language and Action-to-Action Translations . . . . .	71
5.4	Discussion . . . . .	71
5.5	Conclusion . . . . .	72
<b>6</b>	<b>Efficient and Neurocognitively Plausible Translation via Crossmodal Attention</b>	<b>75</b>
6.1	Introduction . . . . .	75
6.2	Proposed Method: Paired Transformed Autoencoders (PTAE) . . . . .	78
6.2.1	Crossmodal Transformer . . . . .	80
6.2.2	Language Decoder . . . . .	81
6.2.3	Action Decoder . . . . .	81
6.2.4	Visual Feature Extraction . . . . .	82
6.2.5	Loss Function . . . . .	82
6.2.6	Training Details . . . . .	83
6.3	Experiments and Results . . . . .	83
6.3.1	Task Signals . . . . .	84
6.3.2	Reduction of Supervised Training . . . . .	84
6.3.3	Exposure to Conflicting Input Modalities . . . . .	88
6.4	Discussion . . . . .	90
6.5	Conclusion . . . . .	92

<b>7</b>	<b>Asymmetrical Combination of Learning Paradigms for Superior Action Execution</b>	<b>95</b>
7.1	Introduction . . . . .	95
7.2	Background: Fine-Tuning with Reinforcement Learning . . . . .	99
7.2.1	Large Language Model (LLM) Fine-Tuning . . . . .	99
7.2.2	Reinforcement Learning Pre-training . . . . .	100
7.2.3	Supervised Pre-training . . . . .	100
7.3	Proposed Method: Crossmodal Bidirectional Transformer (XBiT) . . . . .	101
7.3.1	Language Encoder . . . . .	101
7.3.2	Action Encoder . . . . .	102
7.3.3	Crossmodal Transformer Bottleneck . . . . .	102
7.3.4	Language Decoder . . . . .	103
7.3.5	Action Decoder . . . . .	103
7.3.6	Supervised Learning Loss Functions . . . . .	103
7.3.7	Reinforcement Learning Fine-Tuning . . . . .	104
7.3.8	Training Details . . . . .	105
7.4	Experiments and Results . . . . .	106
7.4.1	Action-to-Language Translation . . . . .	107
7.4.2	Language-to-Action Translation . . . . .	107
7.5	Discussion . . . . .	111
7.6	Conclusion . . . . .	114
<b>8</b>	<b>Capturing Strengths of Large Language Models for Bidirectional Action-Language Translation</b>	<b>115</b>
8.1	Introduction . . . . .	116
8.2	Background: Types of Crossmodality . . . . .	119
8.2.1	Leveraging LLMs for Vision-to-Language Tasks . . . . .	119
8.2.2	Action Execution from Multimodal Input . . . . .	120
8.2.3	Action-Centric Crossmodality . . . . .	120
8.2.4	Full Crossmodality . . . . .	121
8.3	Proposed Method: CrossT5 . . . . .	121
8.3.1	Model Architecture . . . . .	122
8.3.2	Crossmodal Language-Action & Natural Translation Dataset . . . . .	123
8.3.3	Training Setup . . . . .	124
8.3.4	Loss Calculation . . . . .	127
8.4	Experiments and Results . . . . .	129
8.4.1	Action Execution Evaluation in Simulation . . . . .	132
8.4.2	Language Robustness . . . . .	133
8.5	Discussion . . . . .	135
8.6	Conclusion . . . . .	136
<b>9</b>	<b>Seamless Integration of Foundation Models for Full-Fledged Dialogue in Robotic Manipulation</b>	<b>139</b>
9.1	Introduction . . . . .	139
9.2	Background: Leveraging LLMs for Robotics . . . . .	141

9.3	Proposed Method: ELMiRA . . . . .	141
9.3.1	Vision-Language Model . . . . .	141
9.3.2	Object Detection . . . . .	143
9.3.3	Visuospatial Coordinate Transfer . . . . .	143
9.3.4	Motion Planner . . . . .	143
9.3.5	Inverse Kinematics Solver . . . . .	143
9.4	Experiments and Results . . . . .	144
9.4.1	Mode Selection Experiments . . . . .	144
9.4.2	Action Execution Experiments . . . . .	144
9.5	Discussion . . . . .	145
9.6	Conclusion . . . . .	146
<b>10</b>	<b>Conclusion</b>	<b>147</b>
10.1	Discussion . . . . .	148
10.2	Addressing the Research Question . . . . .	151
10.3	Future Research . . . . .	153
10.4	Final Remarks . . . . .	154
<b>A</b>	<b>Nomenclature</b>	<b>157</b>
<b>B</b>	<b>Code Snippets</b>	<b>159</b>
<b>C</b>	<b>Publications Originating from this Thesis</b>	<b>175</b>
<b>D</b>	<b>Acknowledgements</b>	<b>177</b>
	<b>Bibliography</b>	<b>179</b>

# List of Figures

2.1	Overview of Language-to-Action Models . . . . .	8
2.2	Conventional Language-to-Action Model Architectures in Practice .	10
2.3	Learning from Human Demonstrations in Practice . . . . .	14
2.4	LLM-Based Language-to-Action Approaches . . . . .	16
2.5	Overview of Action-to-Language Models . . . . .	19
2.6	Overview of Bidirectional Models . . . . .	20
2.7	Conventional Bidirectional Architectures in Practice . . . . .	22
2.8	LLM-based Bidirectional Architectures in Practice . . . . .	24
3.1	NICO Robot in Simulation . . . . .	28
3.2	Paired Variational Autoencoder (PVAE) Architecture . . . . .	30
4.1	NICO with Toy Objects in Simulation . . . . .	40
4.2	PVAE and PVAE-BERT Architectures . . . . .	42
4.3	Joint Angle Trajectories for PUSH-LEFT-SLOWLY . . . . .	52
4.4	Joint Angle Trajectories for PULL-LEFT-SLOWLY . . . . .	53
4.5	Summed Error Margin for PUSH-LEFT-SLOWLY . . . . .	54
4.6	Joint Angle Trajectories for Description Variations by PVAE-BERT	55
4.7	Hidden Features of Language and Action . . . . .	57
5.1	Object Manipulation Scenario with Primary and Secondary Agents	62
5.2	Schematic Architecture Depicting Information Flow . . . . .	63
5.3	Paired Gated Autoencoder (PGAE) Architecture . . . . .	64
6.1	Tabletop Object Manipulation Scenario . . . . .	77
6.2	Paired Transformed Autoencoder (PTAE) Architecture . . . . .	79
6.3	Crossmodal Transformer (CMT) Architecture . . . . .	80
6.4	Action-to-Language Translation Performance Regarding Ratio of Crossmodal Translation Iterations . . . . .	85
6.5	Action-to-Language Translation Performance Regarding Ratio of Labelled Samples . . . . .	86
6.6	Language-To-Action Translation Performance Regarding Ratio of Labelled Samples . . . . .	87
6.7	Action-to-Language & Language-to-Language Performance with Conflicting Extra Input . . . . .	89
6.8	Action-to-Action & Language-to-Action Performance with Conflicting Extra Input . . . . .	90

7.1	Training Stages of Crossmodal Bidirectional Transformer (XBiT)	97
7.2	XBiT Tasks in Simulation	98
7.3	XBiT Architecture	101
7.4	Action Execution Accuracies on PushButton	108
7.5	Action Execution Accuracies for 1K-Epoch Checkpoint After Longer RL Training	109
7.6	Action Execution Accuracies on PushButtons	109
7.7	Action Execution Accuracies on All Tasks for SL-Only and RL-Fine-Tuned Versions	110
8.1	Multimodal Integration Architectures	117
8.2	NICO with Three Cubes	118
8.3	CrossT5 Architecture	122
8.4	Sample from Crossmodal Language-Action and Natural Translation (CLANT) Dataset	125
8.5	Language Performance in Different Loss Modes	129
8.6	CrossT5 Language Performance	130
8.7	CrossT5 Action Performance	131
9.1	NICO in Real-World Scenario	140
9.2	ELMiRA Architecture	142
9.3	Object Displacements	145



# List of Tables

3.1	Dataset Vocabulary . . . . .	34
3.2	Experiment 1: Action-to-Description Accuracy with Three Colours .	35
3.3	Experiment 2: Action-to-Description Accuracy with Six Colours . .	36
4.1	Channel-Separated Convolutional Autoencoder Architecture . . . . .	45
4.2	Vocabulary with Original and Alternative Words . . . . .	49
4.3	Action-to-Language Translation Accuracies at Sentence Level . . . .	51
4.4	Sentence Translation Accuracies . . . . .	51
4.5	Description Variations and Language-to-Language Performance . . .	55
5.1	Translation Results for Dataset without Second Agent . . . . .	70
5.2	Translation Results for Dataset with Second Agent . . . . .	70
7.1	Dataset Splits Per Task . . . . .	106
7.2	Target Selection and Action Precision Accuracies . . . . .	111
8.1	Vocabulary with Original and Alternative Words . . . . .	123
8.2	Language Performance with Different Shares of Translate Signal . .	129
8.3	Action Execution Quality in Simulation . . . . .	133
8.4	Language Robustness Patterns . . . . .	134
8.5	Action Execution Quality in Different Robustness Modes . . . . .	134
9.1	Mode Detection Success . . . . .	144
9.2	Action Execution Success . . . . .	144



# Introduction

---

Language learning relies on two complementary processes: language comprehension and language production. Language comprehension means understanding spoken or written language, while language production involves generating spoken or written language. The former can be considered a prerequisite for the latter; humans cannot produce language without comprehending it in the first place. Moreover, human language development commences with more concrete and basic phenomena rather than abstract and complex concepts. Human infants learn about their surroundings and interact while listening to their caregivers. Only after recognising the utterances of their caregivers can they begin to speak.

Similarly, when teaching language to a robot which is supposed to interact with humans naturally, it is wise for a language teacher to start with concrete examples rather than abstract concepts. When the robot is seated at a table with toy objects, naming each object and its physical features, e.g. colour, shape and weight, is a natural starting point. The next step may involve showing some simple actions and describing them, e.g. pushing the objects around while naming the directions in which they move. In this passive observation phase, our robot should watch your actions and listen to the relevant words and phrases. After learning the objects, object properties and action types, since our embodied agent has arms for interacting with and changing its environment, letting it name these objects and perform the actions it has observed forms the following phase. This second phase can be considered the active learning phase, where language is grounded in the environment through embodied actions. The active learning phase should help our robot reinforce its linguistic knowledge acquired during the first phase by actively participating in the environment. In this vein, a humanoid robot capable of full-fledged conversation must first understand and then generate language.

## 1.1 Motivation

Despite the recent success of deep learning approaches in language modelling and computer vision, language learning for embodied artificial agents remains challenging. Teaching robots language via computational models like artificial neural

networks (ANNs) as they interact with objects in the environment and perceive linguistic input is not trivial. At the same time, embodied language learning can enhance language acquisition in robots, using multiple modalities such as visual and sensorimotor input. Bisk et al. [19] argue that embodiment, i.e. action taking in the environment, is the necessary step after perception, i.e. using multimodal input, in language acquisition and production. An embodied agent with linguistic capabilities must be able to transform language into control and vice versa since control and action open up new dimensions to understanding and actively learning about the world. Consequently, executing simple actions in the environment according to natural language commands is essential for embodied language learning.

Crossmodal binding between language, action and vision has proven beneficial in language learning for embodied agents. The state-of-the-art approaches in the field rely on tabletop environments where a robot interacts with objects according to given instructions [22, 78, 79, 127, 158, 160]. These approaches rely on ANNs to ground language using multiple modalities such as audio, text, vision and proprioception. They associate language and vision with proprioception; they translate from language commands to robot actions, relying on the image processing capabilities of visual networks like convolutional neural networks (CNNs) [92, 93, 94] or vision Transformers (ViTs) [38]. The ability to conduct robot actions based on user instructions is only one side of the coin and does not suffice for comprehensive language proficiency. The other side of the coin is to produce language given an action; some approaches translate from robot actions to language descriptions [42, 43, 69]. It is only by combining the two skill sets – learning to comprehend and produce language – that it becomes possible to develop approaches truly proficient in language. There also exist approaches that can translate in both directions, from language to action and vice versa [21, 41, 146]. However, most of these bidirectional approaches learn in a supervised fashion, which requires large quantities of labelled robotic manipulation data that is not readily available. Moreover, they rely on large models with billions of parameters, necessitating substantial computational resources for training. Therefore, it is imperative to introduce novel methods to learn language in an embodied and crossmodal fashion, using different learning paradigms such as supervised, unsupervised and reinforcement learning.

## 1.2 Research Objectives

To take the first step towards developing an embodied artificial agent that can seamlessly interact with humans in a free-flowing conversation, we need to endow our robot with specific language comprehension and production capabilities grounded in the environment. Therefore, this research work sets out the following research objectives:

- **Objective 1:** advancing the language comprehension and production capabilities of bidirectional action-language modelling by enabling a model to interpret free-form language instructions and describe robot actions variously,

- **Objective 2:** building an action-language translation model architecture capable of flexibly translating from a given modality into a desired modality without any change in the model configuration during testing and generalising to different perspectives, e.g. recognising and repeating the actions of a second agent,
- **Objective 3:** emulating the characteristics of developmental human infant learning through predominantly unsupervised learning with minimal supervision and scrutinising the consequent model behaviour when it encounters contradictory inputs from different modalities, drawing analogies with human behaviour in psychology literature,
- **Objective 4:** modelling robust robotic action execution through a combination of data-driven supervised learning and active learning in the environment via sparse feedback, i.e. reinforcement learning,
- **Objective 5:** endowing crossmodal bidirectional action-language translation modelling with broad linguistic capabilities of pre-trained large language models (LLMs) involving unscripted dialogue, scene understanding and object manipulation in simulation and the real world.

These research objectives result in an overarching research question: **how to achieve more robust, free-form language learning, which is not restricted to a set of predefined descriptions, for a humanoid robot utilising neurocognitively plausible mechanisms.** Therefore, this thesis explores several neurocognitively plausible mechanisms, including variational autoencoders, channel separation in visual feature extraction, multimodal fusion, crossmodal attention and learning with direct feedback as rewards from the environment.

In order to address the research objectives and the overall research question, we have taken specific steps which culminated in developing a robot that executes actions according to given natural language instructions and describes those actions, simulated in a virtual environment and demonstrated in the real physical world. The main steps taken during this doctoral research can be summarised as follows. I) We have created multimodal datasets of textual descriptions and observations of actions taking place in a tabletop object manipulation scenario involving NICO (Nico2Blocks, NICO CoppeliaSim datasets) as well as Franka Panda robots (PushButton, PushButtons, SlideBlockToTarget and PickUpCup datasets) since multimodal datasets are a prerequisite for crossmodal learning. II) We have developed distinct novel neural network models (see [Section 1.3](#)) trained on these multimodal datasets for flexible action-language translations. III) We have increased the complexity of our tabletop scenario by introducing different types of objects and colours and by tackling multiple object manipulation tasks (e.g. moving cubes and pressing buttons) to test the scalability of our novel approaches. IV) We have utilised large-scale pre-trained language models like BERT [35] and Clip text encoder [140] as well as LLMs like T5 [143] and GPT-4 [130] to embed

linguistic input for transfer learning [107] to scale up our approaches to comprehend virtually unconstrained language instructions. V) We have simulated the language learning process of children to develop and exploit neurocognitively plausible mechanisms consistent with intricacies in developmental language learning. VI) We have adapted a versatile robot learning environment, i.e. RLBench [77], to explore generalisation to a continuous action space via reinforcement learning. VII) We have adopted a text-only LLM, i.e. T5, by incorporating it into our cross-modal architecture to augment bidirectional action-language translation skills with natural language skills such as neural machine translation. VIII) We have devised a modular approach predominantly composed of off-the-shelf speech recognition, object detection, vision-language and text-to-speech models aimed at closed-loop, free-flowing human-robot interaction (HRI). IX) The performance of our novel approaches has been consistently evaluated in terms of success with metrics like the percentage of accurately translated descriptions and executed actions, as well as the normalised root-mean-square error (nRMSE) between original and predicted action trajectories.

### 1.3 Overview of Main Approaches and Novel Contributions

Within the scope of this dissertation, we have introduced a series of novel neural network architectures aimed at modelling embodied crossmodal language learning. These architectures, along with the novel contributions made to the domain of language grounding and the research objectives each architecture addresses, are as follows:

1. Paired Variational Autoencoders (PVAE) [136] facilitates one-to-many mappings between robot actions and language descriptions by opting for a Bayesian method (i.e. variational autoencoder); its handling of visual observations via a channel-separated convolutional autoencoder leads to a superior distinction of object colours. As a result, our contributions with PVAE involve **i) modelling one-to-many action-to-language associations** and **ii) developing a more colour-accurate visual feature extraction mechanism**. The one-to-many action-language translation capability of the PVAE architecture partially addresses **Objective 1**.
2. PVAE-BERT [133] augments the action-language translation capabilities by employing a pre-trained language model, i.e. BERT, for an upscaled dataset of unconstrained language vocabulary and an increased number of colours and various objects. Thus, the contribution made by introducing PVAE-BERT is **advancing the language comprehension capabilities of a bidirectional action-language model via transfer learning**. Leveraging the pre-trained BERT model for understanding unconstrained language input fulfils **Objective 1**.

3. Paired Gated Autoencoders (PGAE) [135] overcomes the obstacle of having to change the model configuration during inference based on the desired translation direction by making use of signal word prefixes and a flexible multimodal fusion technique (i.e. Gated Multimodal Unit (GMU) [13]); it can also recognise and imitate actions of an opposite-sitting agent via robot demonstrations. Therefore, our contributions with PGAE are **i) modelling a flexible training-and-test-time-consistent translation between robot actions and language descriptions through explicit binding via a multimodal fusion mechanism** and **ii) recognising and imitating the actions performed by a second agent**. The PGAE architecture addresses **Objective 2** with its flexible translation between action and language and its recognition and imitation of second-agent actions.
4. Paired Transformed Autoencoders (PTAE) [134] utilises a Transformer-based crossmodal attention to compensate for the lack of labelled samples in the training data; it is also tested with conflicting multimodal input to compare with the incongruence experiments in psychology. Our contributions to the field with PTAE are **i) emulating the mechanisms behind human infant language learning by utilising unsupervised learning to overcome the lack of labelled samples** and **ii) exhibiting biologically plausible behaviour when provided with conflicting multisensory information**. PTAE’s utilisation of chiefly unlabelled data through its Transformer-based attention mechanism and its testing with incongruent multimodal information addresses **Objective 3**.
5. Crossmodal Bidirectional Transformer (XBiT) [137] masters continuous dexterous robotic object manipulation by employing reinforcement learning fine-tuning, which leads to moderate improvements in the object selection problem but substantial performance boosts in action precision. Consequently, with XBiT, we contribute to the field by **being able to asymmetrically combine supervised learning with reinforcement learning to improve language-instructed robotic action performance while suppressing the cons of either learning paradigm**. The introduction of the XBiT model, which is initially trained through supervision and then fine-tuned via reinforcement learning, addresses **Objective 4**.
6. CrossT5 [24] brings together the text-only capabilities of an LLM and sensorimotor control skills of a bidirectional action-language model by fusing their internal representations within a crossmodal network, being rapidly trained on a small-scale natural language and robotic object manipulation dataset while requiring a small amount of computational resources. With CrossT5, we contribute to the field by **i) introducing a novel method of integration for combining LLMs with robotic object manipulation models, which can reliably learn low-level action execution while retaining the inbuilt dialogue capacity from a tiny dataset compared with the**

vast scale of data LLMs are trained on and ii) demonstrating advanced language comprehension ability in the face of more natural diverse user instructions, resulting in robust action execution performance. The seamless integration of an LLM with a bidirectional action-language model in simulation by the CrossT5 architecture partially addresses **Objective 5**.

7. Finally, ELMiRA [50] is a modular approach developed for diverse HRI scenarios, featuring a free-flowing, open-ended dialogue with a tabletop object manipulation task. While utilising minimal robotic platform- and task-specific modules, it primarily leverages general pre-trained foundation models of vision language, speech recognition, speech synthesis and zero-shot object detection, which do not require further scenario-specific learning, i.e. fine-tuning. The contributions we make by introducing ELMiRA are **i) bringing advanced capabilities of foundation models such as open-ended conversation, open-vocabulary object detection and image captioning into the robotic domain and ii) creating a recipe for facilitating full-fledged dialogue and sensorimotor action control applicable to diverse robotic platforms and tasks**. Therefore, the introduction of ELMiRA fulfils **Objective 5**.

## 1.4 Thesis Organisation

The rest of this dissertation is divided into the following chapters: in the next chapter, we will probe more deeply into state-of-the-art (SOTA) approaches in language-instructed robotic object manipulation, compare and categorise them and explain their shortcomings; [Chapters 3 to 9](#) will detail the novel architectures we developed as well as the various experiments we conducted with each model; lastly, in [Chapter 10](#), we will discuss the proposed approaches, highlighting their originality and the novel contributions they bring to the field, as well as the conclusions drawn while addressing the research question, and we will conclude this work with what it entails for the future.



## Related Work

---

Translation between language and robot action has been a topic of interest. Some approaches learn the general mapping between objects and language as well as attributes like colour, texture and size [61, 161], and there exist approaches that learn complex manipulation concepts [107, 158]. The state-of-the-art approaches in embodied language learning mostly rely on tabletop environments [61, 69, 158, 161, 185] or interactive play environments [107] where a robot interacts with various objects according to given instructions. Our focus lies on the approaches that exploit tabletop scenarios. We group these approaches into three categories: those that translate from language to action, those that translate from action to language [43, 69] and those that can translate in both directions [1, 11, 128, 136, 185], i.e. bidirectional approaches. Bidirectional approaches allow greater exploitation of available data as training in both directions can be interpreted as multitask learning, which ultimately leads to more capable and powerful models independent of the translation direction. By maximising the use of shared weights for multiple translation directions, such models should theoretically be more efficient than individual unidirectional networks regarding data utilisation and model size.

When it comes to training, most recent approaches in language-oriented robotic object manipulation require large quantities of teacher trajectories, as they rely on supervised learning (SL) and do not consider reinforcement learning (RL) as an efficient paradigm for robot learning. Much of the research addresses the challenge of generalisation to new objects, environments and even robots by expanding and diversifying datasets. These approaches use an imitation learning method called behavioural cloning (BC) that learns from pre-collected trajectories in a supervised manner. BC is limited because it cannot explore the environment on its own and suffers from the distributional shift, where the cases encountered during inference diverge from the training set. In the following sections, we will delve deeper into various approaches in language-guided robotic object manipulation, divided into three main parts: language-to-action, action-to-language and bidirectional translation.

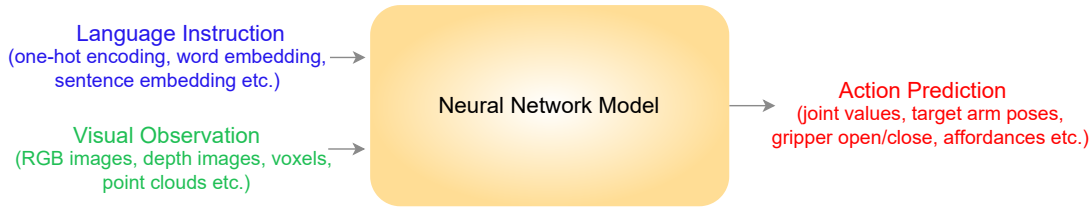


Figure 2.1: Overview of language-to-action models. Language-to-action models accept language instructions and visual observations as input and output action predictions such as joint angle values or target arm poses.

## 2.1 Language-to-Action Translation

Most language-instructed robotic object manipulation approaches can only translate from language commands to robotic action execution because their model architectures are not designed for bidirectional translation. As recent developments in language processing allow natural language to control robots for object manipulation tasks, translating from language to action is the most common form in embodied language learning. As can be seen in [Figure 2.1](#), language-to-action translation models accept language instructions as well as visual observations as input, while they produce robot actions in different forms, e.g. joint angle values or target end-effector poses. Some approaches utilise modular architectures combining separate vision and language components such as object detection and speech recognition. For example, Hatori et al. [61] introduce a neural network architecture for moving objects given the visual input and language instructions, as their work focuses on the interaction of a human operator with the computational neural system that picks and places 100 miscellaneous items as per verbal commands. In the setup, they distribute many items of different shapes and sizes (e.g. toys and bottles) across four bins with many of them occluded – hence, the scene is very complex and cluttered – and a robot is instructed to pick up an object and move it to a specific bin. Given an instruction from the human operator, the robot executes a pick-and-place action after confirming that the desired object is unambiguous. Otherwise, the robot asks the human operator to clarify the desired object. The network receives a verbal command from the operator and an RGB image from the environment, and it has separate object recognition and language understanding modules, which they train jointly to learn the names and attributes of the objects. Moreover, it learns different properties of an object, e.g. its colour, texture or size. Since it depends on the successful functioning of many different modules like object detection, target object selection and ambiguity resolution, the overall approach is brittle.

Similarly, Shridhar and Hsu [161] propose a comprehensive system for a robotic arm to pick up objects based on visual and linguistic input. The system, named INGRESS (Interactive Visual Grounding of Referring Expressions), consists of multiple modules, i.e. manipulation, perception and a neural network architecture.

INGRESS has two network streams (self-referential and relational) trained on large datasets to generate a definitive expression for each object in the scene based on the input image. The generated expression is compared against the language input to identify the desired object. INGRESS is, therefore, responsible for grounding language by learning object names and attributes via manipulation. INGRESS has the freedom of numerous object categories for the robot to interact with diverse objects available for everyday use. The approach can resolve ambiguities about which object to lift by asking confirmation questions to the user. As a modular approach relying on multiple modules such as the outdated object localisation network, Faster R-CNN [148], INGRESS has many potential sources of error.

**Conventional Approaches** A large majority of approaches train with BC on a precollected dataset of paired visuolinguistic sensorimotor data,  $D = \{(o_t^k, a_t^k)_{t=0}^T, l^k\}_{k=0}^K$ , where  $o \in O$  represents state observations,  $a \in A$  represents actions and  $l \in L$  is a language command, to learn language-instructed robotic object manipulation. Architecturally, they include language and vision encoders, a backbone policy network and an action decoder. These can be formulated as follows:

$$s = f_\sigma(o), \quad (2.1)$$

$$d = f_\phi(l), \quad (2.2)$$

$$\pi_\theta(a|s, d), \quad (2.3)$$

$$p = f_\psi(a), \quad (2.4)$$

where  $f_\sigma$ ,  $f_\phi$  and  $f_\psi$  are vision encoder, language encoder and action decoder respectively, while  $\pi_\theta$  is the policy network.  $s$ ,  $d$  and  $p$  are state representations (i.e. visual features), embedded language and practical action output (e.g. joint angle value or end-effector pose predictions) respectively. Figure 2.2 shows the general framework of language-to-action translation architectures. For example, the DreamCell architecture introduced by Paxton et al. [138] seeks to learn hidden representations to transform human-generated natural language instructions into sequences of executable subgoals for a pick-and-place task in the domain of simulated robotic object manipulation. Given a language command and the current state observation as input, DreamCell predicts not only the next action but also the next state and subgoal. It utilises an autoencoder network to encode the current state observation, including the RGB image, into a latent code. Then, it uses this code and a given goal to construct the predicted future state. Predicting future states makes the model understand the environment dynamics better and results in more robust language-to-action modelling. Dividing a plan into executable low-level actions helps in handling complex human instructions. However, the approach is not tested in the real world and minor errors in the low-level control over multiple steps may lead to failure in executing a high-level plan.

As a prime example of conventional language-to-action approaches, Jang et al. [78] propose BC-Z leveraging a large multi-task dataset, including 100 tasks,

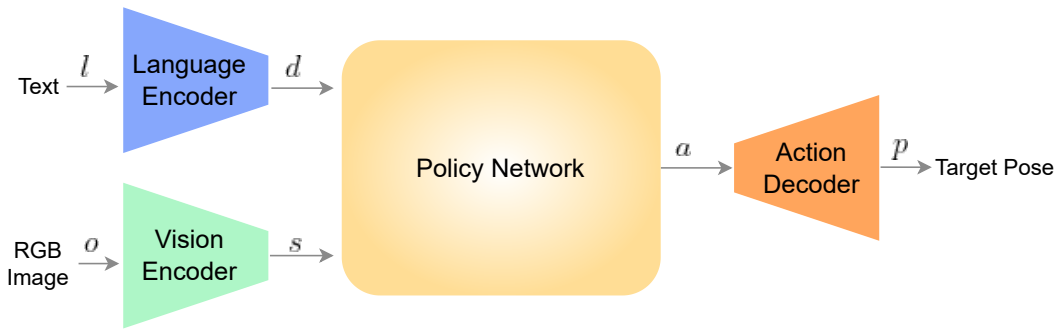


Figure 2.2: Conventional language-to-action model architectures used in practice. Most language-to-action models receive language instructions in textual format and visual observations as RGB images. They encode the text and image inputs via their modality-specific networks: language and vision encoders. The policy network is the backbone of the architecture and uses encoded representations to choose the best action. Finally, the action decoder outputs action predictions as target arm poses.

to train a single policy which is supervised with BC to match the actions demonstrated by humans in the dataset. To generalise to new tasks, they condition the policy on a task description, i.e. a joint embedding of a video demonstration and a language instruction, allowing passing either the video command or the language command to the policy when trained to match the actions in a demonstration. BC-Z performs relatively poorly on new tasks and requires a large collection of human demonstrations. It also relies on human intervention to avoid unsafe situations and to correct mistakes.

Lynch and Sermanet [107] introduce the LangLfP (language learning from play) approach using multi-context imitation to train a single policy based on multiple modalities. Specifically, they train the policy on image and language goals, enabling the approach to follow natural language instructions during evaluation. LangLfP extends the previous work, LfP [106], by pairing robot actions with hindsight language instructions. During training, fewer than 1% of the tasks are labelled with instructions because it suffices to train the policy for more than 99% of the cases with goal images alone. Furthermore, they utilise a Transformer-based [173] multilingual language encoder, Multilingual Universal Sentence Encoder [186], to encode linguistic input to handle unseen language input like synonyms and instructions in 16 languages. Despite being reduced to 1% of overall training samples, labelling even a tiny portion of a large dataset with around 10M samples requires several human annotators, which is expensive. Based on the same learning method, Mees et al. [113] present the HULC (Hierarchical Universal Language Conditioned Policies) framework aimed at improving the language-conditioned imitation learning performance on long-horizon<sup>1</sup> tabletop object manipulation tasks. In addition to

<sup>1</sup>Long-horizon tasks are those that require high-level long-term goal planning [56].

a multimodal Transformer network, HULC employs a contrastive loss to align visual and linguistic representations semantically. Following LangLfP, the model is trained on action trajectories produced by human teleoperators controlling the robotic arm in the environment. HULC learns a single policy in a task-agnostic manner and achieves promising results on the robotic manipulation benchmark CALVIN [114]. In a follow-up work, HULC++ [112] extends upon HULC by integrating a language-conditioned visual affordance model [20] and an LLM to perform long-horizon tasks in the real world. Another work extending upon LangLfP and HULC is the Skill Prior-based Imitation Learning (SPIL) framework by Zhou et al. [197], exploiting primitive actions such as rotation and grasping to generalise to unseen environments. Instead of learning a policy to act directly based on state observations and language instructions, SPIL learns an intermediate-level policy to select the appropriate primitive action.

Inspired by the two-stream theory in cognitive psychology, CLIPort [159] combines the CLIP model [140], for pre-trained vision-language representations, with the Transporter model [190], for robotic manipulation tasks, to predict pick- and place-coordinates of objects. Transporter takes an action-centric approach to perception by detecting actions, rather than objects, and then learns a policy, which allows CLIPort to exploit geometric symmetries for efficient representation learning. On multiple object manipulation tasks, CLIPort outperforms CLIP and Transporter alone. Further, CLIPort trained on multiple tasks performs better in most cases than CLIPort trained only on particular tasks. This supports the hypothesis that language-conditioned task-learning skills can be transferred from one task to another. However, the approach is only realised with a relatively simple gripper as it does not output joint angle values but 2D-pixel affordance predictions. The actual action execution relies on the calibration between the robotic arm base and the RGB-D camera.

A more recent approach introduced by the same authors, Perceiver-Actor (PERACT) [160], is designed to efficiently learn multi-task robotic manipulations according to given language input by utilising voxel grids extracted from RGB-D images. The backbone of the model is the Transformer-based Perceiver IO [76] which uses latent vectors to tackle the processing of very long sequences. After Perceiver IO processes the appended language and voxel encodings, the voxels are decoded again to generate discrete actions using linear transformations. PERACT outperforms baselines and achieves promising results in multiple tasks such as opening a drawer, turning a tap and sliding blocks. However, its dependence on voxelisation for generalising to different object positions via data augmentation renders PERACT expensive to train.

Introduced by Brohan et al. [22] as the first iteration of a series of Transformer-based robotic manipulation models, Robotics Transformer 1 (RT-1) can navigate through a kitchen environment and interact with objects according to language instructions. RT-1 is trained with BC on a large multi-robot dataset. The results show that RT-1 can generalise to unseen tasks, handle challenging situations with many distractors and different backgrounds and respond to more realistic instructions. It outperforms previous language-prompted robotic manipulation models.

The authors also leverage SayCan [6] to divide long-horizon tasks into executable steps. Although it partially generalises to unseen tasks and outperforms previous language-prompted robotic manipulation models, as another BC approach, RT-1’s performance is strictly bound to the quality of the expert demonstrations. RT-1 is also limited to the objects it learns during training. Stone et al. [164] extend RT-1 to a virtually unlimited number of objects by leveraging a pre-trained open-vocabulary object detection network (OWL-ViT [119]). They query OWL-ViT with the initial scene image and the desired object names to produce bounding boxes marking the target objects. The pixels at the centre of bounding boxes alongside histories of images and encoded language instructions are passed to the RT-1 network. The approach shows a significantly better object generalisation performance and robustness against different backgrounds, distractors and environments.

The encoder-decoder-based VisuoMotor Attention (VIMA) model [79] handles robot action generation from multimodal prompts by interleaving language and image or video frame tokens at the input level. In contrast to the approaches that process raw pixels, VIMA uses an object detection module to extract objects and bounding boxes from visual input to use as object tokens. The object tokens are then interleaved with the language tokens and processed by the pre-trained T5 model [143], which is used as the encoder. On the decoder end, the approach uses a causal Transformer decoder which consists of cross- and self-attention layers and autoregressively generates actions based on the history of actions and the multimodal prompt. VIMA outperforms state-of-the-art approaches, including GATO [146], on numerous increasingly difficult object manipulation tasks, involving zero-shot generalisation with unseen objects and their combinations. However, including images as part of the instructions given to a robot is impractical in real-world scenarios. As a modular approach, another apparent weakness of VIMA is its reliance on the accuracy of an off-the-self object detector, i.e. Mask R-CNN [64].

Lynch et al. [108] introduce an interactive language-guided robots paradigm with which a robotic arm can be instructed to manipulate objects in real time. The main contribution is about how they collect and annotate real-world and simulation data. The dataset is collected in two steps: first, teleoperated control, where human operators are hired to remotely control a robotic arm to manipulate objects on a tabletop; second, event-selectable relabelling, where humans annotate the recorded object manipulation episodes with the freedom of choosing any interval to describe a meaningful action event. A novel neural network, named LAVA (Language Attends to Vision to Act), is trained on the collected data with BC. The experiments show that success in long-horizon goals heavily depends on sufficient real-time feedback rather than open-loop instructions. The proposed approach can also be used to instruct multiple robots simultaneously. However, it does not address intention detection, non-verbal communication and physically collaborative task completion. The action space is also relatively simple, with a robotic arm moving in two dimensions. A follow-up approach by Jin et al. [81] employs LAVA as the low-level action execution model in a closed-loop high-level multi-step reasoning setup, utilising a multimodal LLM capable of updating its plans based on visual feedback. This framework, named CogLoop,



outperforms open-loop models (e.g. SayCan) and closed-loop with language feedback approaches (e.g. Text2Motion [99]) on a simulated block-placement task. The results show that even with a smaller and less powerful LLM, i.e. MiniGPT-4 [198], it is possible to integrate visual capabilities into language modelling, with instant visual feedback, for high-level task planning. Nevertheless, CogLoop’s success rate of 23.5% is low for reliable long-horizon task execution, and the approach is only tested on a block manipulation task in simulation.

Another attempt at robotic generalisation is the Octo model [127], which is made up of a Transformer architecture [173], a pre-trained language encoder, a shallow convolutional visual encoder and diffusion-based action heads. Octo is trained on 800k episodes from the Open X-Embodiment dataset [175]. Thanks to its flexible architecture, leveraging a diffusion policy [29], Octo can be fine-tuned on different tasks and robotic platforms after pre-training. Despite being a radically smaller model compared to state-of-the-art robotic action models, Octo’s performance relies on the careful curation of the vast amounts of pre-training data. Moreover, its performance drops considerably when it is conditioned on language annotations instead of goal images. Lastly, the novel OpenVLA (Open-Source Vision-Language-Action) [88] model, based on two separate vision encoders (SigLIP [191] and DINOv2 [132]) and an LLM (Llama 2 [169]), is fine-tuned on the Open X-Embodiment dataset to bring about a generalised robotic object manipulation approach. After concatenating visual features extracted by the two encoders, they are projected to the input space of the LLM and passed as input alongside the language tokens. The LLM outputs action tokens which are detokenised to control a robotic arm continuously. OpenVLA outperforms SOTA approaches, including Octo, in different aspects of generalisation across multiple robotic platforms. However, it has 7B parameters compared to Octo’s 86M parameters. Both Octo and OpenVLA can only produce actions like the aforementioned approaches in this category, which are designed to act upon a given language input as in textual or verbal commands.

**Learning from Human Demonstrations** Some approaches leverage labelled human-action datasets for learning the basics of action execution via imitation learning (see Figure 2.3). For example, Shao et al. [158] put forward a robot learning framework, Concept2Robot, for learning manipulation concepts from human video demonstrations in two stages. In the first stage, they use RL, while in the subsequent stage, they utilise imitation learning. The architecture consists of three main parts: a semantic context network, a policy network and action classification. The model receives as input a natural language description for each task alongside an RGB image of the initial scene. In return, it is expected to produce the parameters of a motion trajectory to accomplish the task in the given environment. Since it uses the output of the action classification model as a proxy reward, Concept2Robot utilises human demonstration videos from the “something something” database [53]. Similar to Concept2Robot, the LOReL (Language-conditioned Offline Reward Learning) approach [122] maps language instructions to rewards by

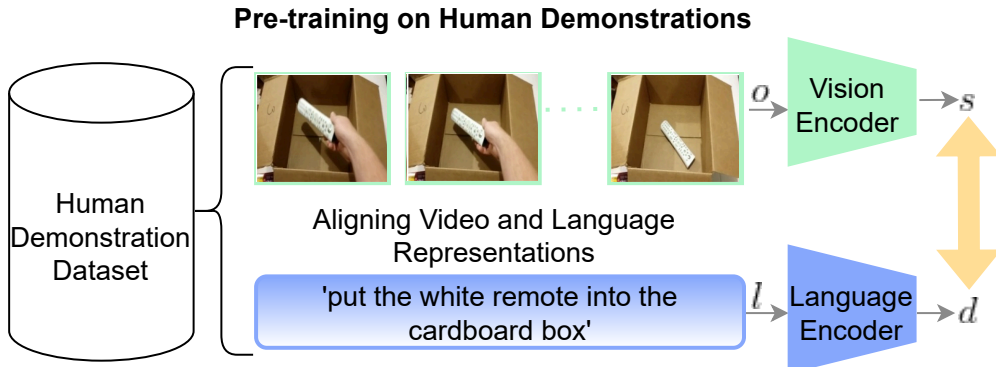


Figure 2.3: The approaches that learn from human demonstrations in practice. These approaches usually pre-train on annotated human demonstration videos to learn manipulation concepts in advance. In addition to other learning objectives, they try to align videos and their corresponding textual task representations after embedding them with modality-specific encoders. This pre-training helps models learn robotic object manipulation tasks later. The exemplary video frames and the corresponding annotation have been adopted from Goyal et al. [53].

learning language-conditioned behaviour from offline video datasets annotated by humans in natural language. LOReL has a convolutional vision encoder and employs the pre-trained language model DistilBERT [153] to embed language. Experiments show that utilising language-conditioned rewards accelerates learning and improves generalisation to new tasks. Nonetheless, both Concept2Robot and LOReL require tens of thousands of human demonstrations to learn the basics of object manipulation.

Another approach utilising human demonstrations from videos is R3M (reusable representation for robotic manipulation) by Nair et al. [123]. The Ego4D egocentric video dataset [54] with language annotations is used to pre-train the R3M perception network to learn multimodal representations that can be transferred to downstream robotic manipulation tasks. This pre-training involves time-contrastive learning (i.e. aligning temporally closer states in the latent space), video-language alignment (i.e. pairing specific frames of visual observations with corresponding annotations) and a sparsity penalty favouring sparse representations over dense ones. After pre-training, R3M is frozen and used as a perception module for object manipulation tasks with a robot. R3M employs a convolutional ResNet [65] architecture for visual processing. The simulated and real-world experiments across multiple benchmarks involving several manipulation tasks show that the approach outperforms the state-of-the-art (SOTA) visual representation models like CLIP [140]. However, R3M is limited to learning with BC as the authors admit that it may not be beneficial for RL. Similarly, introduced by Ma et al. [109], LIV (Language-Image Value Learning) can train on arbitrary human action videos that are paired with language annotations to produce zero-shot multimodal vision-



language representations. Built upon Value-Implicit Pretraining (VIP) [110] and CLIP, LIV can be fine-tuned on smaller robotic manipulation datasets for context-specific language grounding. It outperforms SOTA approaches such as R3M and VIP in various object manipulation tasks across different environments. However, unless fine-tuned with robot videos, it performs poorly on manipulation tasks.

**RL Approaches** Another group of approaches train via RL algorithms. For example, Sodhani et al. [163] adapt CMDP (contextual Markov decision process) [59] into the multi-task RL domain in order to leverage language-oriented contextual information as a task definition. The proposed model, CARE (Contextual Attention-Based Representation Learning), employs several state encoders to model diverse state representations, while the language input is encoded by a pre-trained language model (RoBERTa [103]) followed by an MLP, which results in a contextual representation. Afterwards, the contextual representation attends to the state representations to arrive at a unified state encoding, which is then concatenated with the contextual representation to form the policy input. The policy is trained with the Soft Actor-Critic (SAC) [57] RL algorithm. The approach outperforms SOTA methods in multi-task RL in the Meta-World benchmark [188]. However, it uses state observations from the simulator instead of visual inputs.

Introduced by Silva et al. [162], LanCon-Learn utilises natural language instructions instead of one-hot task IDs to enable skill transfer between semantically related tasks in multi-task object manipulation settings. LanCon-Learn uses GloVe [139] word representations and a bidirectional LSTM to embed language input. Trained with either RL (SAC algorithm) or imitation learning (DaGger [150] algorithm), LanCon-Learn displays some zero-shot learning capabilities for unseen tasks in the Meta-World benchmark. However, similar to CARE, it is provided with direct environment states instead of visual observations. Therefore, it cannot work under realistic circumstances where the internal states of a simulator are not available.

A similar approach, MILLION [18], focuses on skill learning based on natural language instructions to master multi-task learning with a trial-and-error RL approach in Meta-World. Like LanCon-Learn, GloVe is employed to extract language representations that define tasks. MILLION divides the learning into two stages, namely instruction and trial phases. In the instruction phase, the agent acts in the environment based on the given language instruction while ignoring the rewards provided by the environment, whereas in the trial phase, language instructions are no longer provided as the agent acts in the environment and receives rewards. MILLION is trained with an on-policy RL algorithm and outperforms SOTA approaches in terms of average success rate in multiple tasks. A more recent approach built upon MILLION by Yao et al. [187] exploits linguistic and action-space symmetries between tasks to learn faster and more successfully. For example, the open-drawer task is a mirrored version of the close-drawer task as the push-left task is the mirrored version of the push-right task. Learning one task of each pair thus helps in learning the other task. The approach uses a concrete

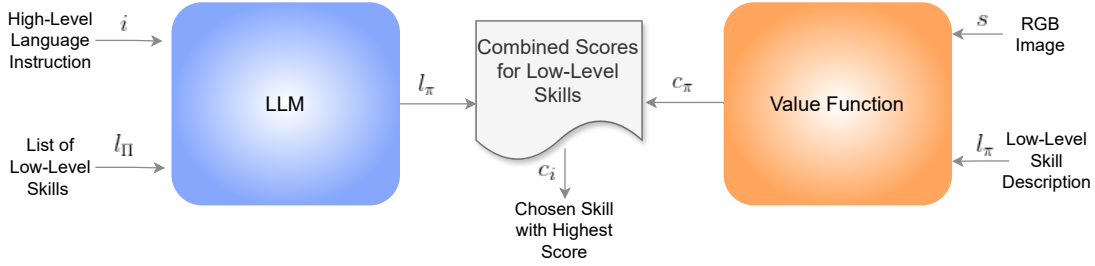


Figure 2.4: LLM-based language-to-action approaches like SayCan in practice. An LLM is used to choose a low-level skill from a high-level language instruction and a list of available low-level skills. They also use a value function to calculate the affordance value of low-level skills according to the current state observation from an RGB image. The low-level action with the highest combined score by the LLM and value function is chosen to be carried out.

syntax three method [82] to parse the input sentence into verb and noun phrases so that it can comprehend it semantically. The approach outperforms MILLION on symmetrical test tasks by a great margin showing good generalisation capabilities. Nevertheless, both approaches use direct state observations from the simulator instead of modelling visual representations as they focus on meta-RL.

An RL paradigm, Language-Goal-Behaviour (LGB), introduced by Akakzia et al. [7], decouples skill learning from language grounding in a 3D tabletop object manipulation task with a robotic arm stacking differently coloured blocks. In the first stage, the LGB agent learns primitive object manipulation skills by interacting in the simulation environment with curiosity and exploring possible block configurations. In the second stage, language grounding in actions is realised via feedback from a social partner in the form of hindsight descriptions of the performed action. LGB paradigm leads to robust and diverse behaviours in goal-conditioned RL.

**LLM-Based Approaches** LLM-based approaches calculate two kinds of probabilities, the first of which is the task grounding probability that is modelled by an LLM,  $p(l_\pi|i)$ , and the second is the world grounding probability that is formalised by a value function,  $p(c_\pi|s, l_\pi)$ . The selection of the low-level action is determined by the combined score of these two probabilities, which is calculated by multiplying them:  $p(c_i|i, s, l_\pi) = p(c_\pi|s, l_\pi) \cdot p(l_\pi|i)$ , where  $i$  is a high-level language instruction,  $s$  is a state,  $l_\pi \in l_\Pi$  is a low-level action skill,  $c_\pi$  is an affordance value and  $c_i$  is the chosen skill. Figure 2.4 displays this procedure in detail.

As a pioneering work, SayCan [6] employs LLMs to provide task-grounding capabilities to the agent, which is capable of executing low-level actions. The use of LLMs helps to ground these capabilities in the real world using value functions of the agent in order to produce feasible and useful instructions. An LLM is utilised to assign affordance probabilities to these skills according to a given high-level user instruction. The way these skills are defined in language (the wording, the length, etc.) can affect the overall performance, e.g. LLMs tend to favour

shorter phrases over longer ones. The follow-up work, Inner Monologue [74], brings SayCan into closed-loop planning as the employed LLM adapts its output according to various sources of feedback, including success detection, scene description and human response. This closed-loop versatile feedback in the form of language to the LLM increases the success rate of plan completion on simulated as well as real-world tabletop object manipulation and mobile manipulation tasks. However, neither SayCan nor Inner Monologue account for the geometric dependencies in a given scene based on the high-level plan; LLMs can generate plans composed of semantically viable steps, which may still involve spatially infeasible actions. In order to address the geometric feasibility issue, Lin et al. [99] propose Text2Motion that utilises a geometry-dependent long-horizon planner (STAP [5]); the LLM plans new actions until STAP determines the LLM-produced action as adhering to geometric dependencies. Text2Motion outperforms the previous approaches in challenging long-horizon tasks requiring geometric acumen. A more recent approach, ViLa (Robotic Vision-Language Planning) [72] employs the pre-trained VLM GPT4-V [131] to decompose high-level natural language instructions to low-level executable action steps for object manipulation. Due to its utilisation of a VLM, instead of separately handling language and vision which may lead to imprecise and inaccurate task planning, jointly processing vision and language results in a task-focused understanding of the current scene based on the given instruction. ViLa outperforms SayCan in common-sense-related tasks by a large margin. It also works with different modalities in the high-level instructions such as providing a goal image or a combined image-language goal. Nevertheless, all of these approaches are limited to the set of skills that the agent can possess in the environment.

Another example of leveraging LLMs is the approach proposed by Ren et al. [147] which employs an LLM (i.e. GPT-3 [23]) for tool manipulation. The novel tool learning paradigm ATLA, short for Accelerated Learning of Tool Manipulation with Language, utilises LLMs in two ways: first, to generate descriptions of tools (including affordances and shapes of tools), and second, to obtain feature representations. ATLA consists of two phases, namely meta learning and meta testing. During meta learning, GPT-3 is used to collect descriptions of tools; a predefined sentence template is used to prompt GPT-3 to describe the uses and shapes of tools. The resulting language description is fed to a pre-trained BERT to extract features. In the meta-testing phase, GPT-3 creates descriptions and tries to generalise to new tools. The results of the actions with test tools show that language information indeed helps adapt to new tools. Moreover, meta learning seems to help learn unseen tools better as it supports the gain from language information. However, when using a smaller version of BERT as the language encoder, the adaptation to new tools deteriorates, which indicates that richer, more detailed representations of language are needed for better generalisation. In a similar vein, Tang et al. [166] utilise an LLM for real-world object grasping; the approach, named GraspGPT, uses the LLM to generate multiple descriptions of given objects (e.g. a cup and a pitcher) and action types (e.g. pouring, and spraying), which are then used to generalise to unseen objects and action types for grasping. These

generated descriptions are passed to a Transformer decoder-based grasp evaluator alongside point-cloud embeddings of the object and the attempted grasp pose to predict the grasp score.

Palo et al. [37] also leverage an LLM to devise a comprehensive RL approach covering efficient language-guided exploration, experience reuse from offline data, timely execution of actions and learning human expert demonstrations for a pick-and-place object stacking task. They use the CLIP model as their VLM, which they fine-tune for their task, to bridge object manipulation actions with language descriptions via the dot product operation between the language embedding and the embedding of each video frame. This CLIP-based vision-language mapping serves as an internal reward model during RL. As the LLM, the authors employ FLAN-T5 [33], which was fine-tuned on language instructions, to generate executable subgoals for a given stacking task. The robot executes the subgoal in the environment, which is shown to the VLM to determine successful action execution. If the action is successful, the robot continues with the next subgoal generated by the LLM. In addition, the approach also makes use of successful action sequences from offline data collected on other tasks to bootstrap the RL policy. Compared with the baseline using only sparse environment rewards, the approach shows improved performance in stacking two or three cubes. However, it uses the object positions provided by the simulator as input, and it outputs actions only in X and Y coordinates, bypassing the height dimension. Besides, fine-tuning a large pre-trained VLM like CLIP requires large amounts of data, not least when scaling up the approach with more objects and real-world data.

All in all, language-to-action translation approaches can recognise commands and execute desired actions. However, they cannot describe the actions that they perform because their model architectures are not suitable for processing sensorimotor input or generating language output.

## 2.2 Action-to-Language Translation

Another class of approaches in embodied language learning translates from action to language. Visualised in Figure 2.5, these models produce language outputs like verbal or textual descriptions and answers to yes-no questions as they are fed visual (ranging from RGB images to point clouds) and proprioceptive (joint angle values, end effector poses, etc.) observations. Some of them even accept language as input in the form of hindsight descriptions and yes-no questions.

As an exemplary method for action-to-language translation, Heinrich et al. [69] introduce an embodied crossmodal neurocognitive architecture, the adaptive multiple timescale recurrent neural network (adaptive MTRNN), enabling a robot to acquire language by listening to commands while interacting with objects in a playground environment, emulating language development conditions for human infants. The adaptive MTRNN has auditory, sensorimotor and visual perception capabilities. As neurons at multiple timescales facilitate the emergence of hierarchical representations, the results indicate good generalisation and hierarchical

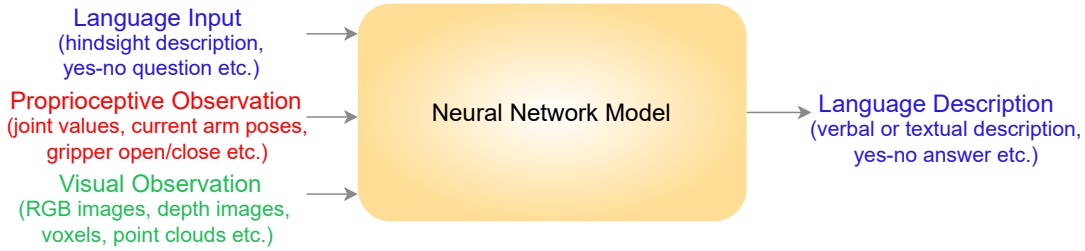


Figure 2.5: Overview of action-to-language models. Action-to-language models accept as input language, proprioceptive and visual observations and output verbal or textual descriptions of actions and answers to yes-no questions.

concept decomposition within the network. However, the approach is tested on a limited-scale dataset [66] and achieves at most 64% accuracy in describing actions. Furthermore, it uses a specialised type of recurrent neural network (RNN) model that has been superseded by Transformers in natural language processing.

As another action-to-language approach, Eisermann et al. [43] study the problem of compositional generalisation, in which they conduct numerous experiments on a tabletop scenario where a robotic arm manipulates various objects. They utilise a simple LSTM-based (long short-term memory [70]) network to describe the actions performed on the objects in hindsight – the model accepts visual and proprioceptive input and produces textual descriptions. The results show that, with the inclusion of proprioceptive input, i.e. joint angles, and training on more data, the model performance on compositional generalisation improves significantly. Nevertheless, like the adaptive MTRNN, the approach employs an old-fashioned RNN for language production. Also, it cannot achieve more than 67% accuracy on the challenging compositional generalisation test sets.

As an action success detection method, which can be considered yet another action-to-language approach, Du et al. [42] employ a VLM, i.e. the Flamingo model [8], to detect successful action execution in three domains, namely the simulated household environment, real-world robotic object manipulation and egocentric human videos. They treat success detection as a visual question answering (VQA) task; Flamingo is given a sequence of images showing a performed action alongside an accompanying yes-no question. The model then answers with yes or no depending on the outcome of the action carried out in the sequence. Flamingo is fine-tuned on this VQA task for each domain while its language layers are kept frozen. The results show that pre-trained VLMs can be used as a reward model or an oracle to evaluate the performance of a robotic object manipulation approach. Nevertheless, the approach struggles with ‘in-the-wild’ human videos as it cannot generalise well to unseen demonstrations in that domain.

Analogous to language-to-action translation, the action-to-language translation methods operate only in one direction: they generally describe the actions performed in the environment. They are unable to execute a desired action given by the human user. From the robotics perspective, it is still desirable to have mod-

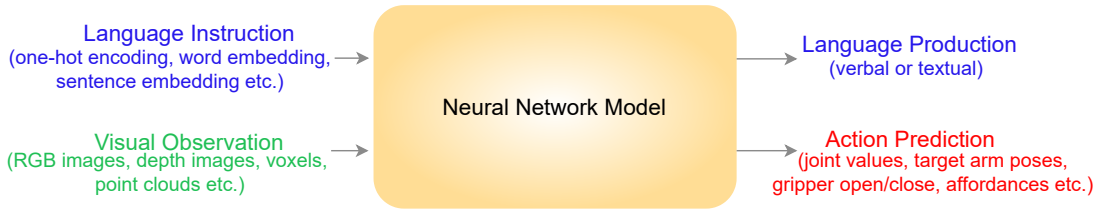


Figure 2.6: Overview of bidirectional models. Bidirectional models can translate from language to action as well as from action to language. They accept as input language instructions (as one-hot-encoded, word- or sentence-embedded) and visual observations (e.g. RGB or depth images, voxels and point clouds) and output language in verbal or textual form and action predictions, ranging from joint angle values to affordances.

els that can also translate from action to language and not just execute verbal commands; such robots can explain their actions by verbalising an ongoing action, which paves the way for more interpretable systems.

## 2.3 Bidirectional Translation

Only few approaches [1, 11, 128, 185] are capable of bidirectional translation, i.e. they have the ability to translate a given action into language as well as to translate a given language description into an action. As depicted in Figure 2.6, bidirectional models accept language instructions and visual observations as input and can produce both action and language. While unidirectional approaches are feasible for smaller datasets, we aim to research architectures that can serve as large-scale multimodal foundation models and solve multiple tasks in different modalities. By generating a discrete set of words, bidirectional models can also provide feedback to a user about the information contained within its continuous variables. By providing rich language descriptions, rather than only performing actions, such models can contribute to explainable AI (XAI) [2] for non-experts.

In one of the early examples of bidirectional translation, Ogata et al. [128] present an RNN-based model, RNNPB, that is aimed at articulation and allocation of arm movements by using a parametric bias to bind motion and language. The method enables the robot to move its arms according to given sentences and to generate sentences according to given arm motions. RNNPB enables bidirectional translation between compound sentences and robotic arm motions. Artificial bias vectors are used to bind the two modalities, which have separate RNNs, to enable flexible translation between them. The model shows generalisation towards motions and sentences that it is not trained with. However, it fails to handle complex sentences.

Exploiting the notion of multiple timescales like adaptive MTRNN, Antunes et al. [11] introduce the multiple timescale long short-term memory (MT-LSTM) model in which the slowest layer establishes a bidirectional connection between



action and language. The MT-LSTM consists of two components, namely language and action streams, each of which is divided into three layers with varying timescales. The two components are bound by a slower meaning layer that allows translation from action to language and vice versa. However, MT-LSTM shows limited generalisation capabilities.

**Conventional Approaches** The conventional bidirectional approaches can be formulated as follows:

$$s = f_{\sigma}(o), \quad (2.5)$$

$$d = f_{\phi}(l), \quad (2.6)$$

$$h = \text{MMF}(s, d), \quad (2.7)$$

$$a_t = f_{\psi}(h, a_{t-1}), \quad (2.8)$$

$$l_t = f_{\lambda}(h, l_{t-1}), \quad (2.9)$$

where  $f_{\sigma}$ ,  $f_{\phi}$ ,  $f_{\psi}$  and  $f_{\lambda}$  are vision encoder, language encoder, action decoder and language decoder respectively, while MMF stands for the multimodal fusion network.  $o$ ,  $l$ ,  $s$ ,  $d$ ,  $h$  and  $a$  denote visual observations, language descriptions, state representations (i.e. visual features), embedded language, common hidden representations and action output (e.g. joint angle value or end-effector pose predictions) respectively. Figure 2.7 illustrates the general framework of these bidirectional architectures.

As an ambitious project to go beyond modelling architectures only capable of following instructions, Abramson et al. [1] propose a complex paradigm combining supervised learning, reinforcement learning (RL) and imitation learning in order to solve the problem of intelligently interacting in an abstract 3D play environment while using language. In the environment, two agents communicate, with one agent (the setter) asking questions to or instructing the other (the solver) which answers questions and interacts with objects according to a given instruction. The scenario is abstract as the objects are unrealistically interacted with. As the actions are abstract, there is no need for proprioception, i.e. actual action execution. Therefore, the transfer of the approach from simulation to the real world is non-trivial.

To map robotic action and natural language bidirectionally, Yamada et al. [185] propose the paired recurrent autoencoder (PRAE) architecture comprising two recurrent autoencoders: action and description. The action autoencoder receives joint angle trajectories with visual features as input and reconstructs the original joint angle trajectories. The description autoencoder, on the other hand, reads and then reconstructs the language descriptions. The dataset consists of pairs of simple robot actions (including sequences of images and joint angle values) and their textual descriptions, e.g. ‘pushing away the blue cube’. The model is trained end-to-end, with both autoencoders reconstructing language and action, whilst there is no explicit neural connection between the two. A loss term that aligns the hidden representations of paired actions and descriptions, i.e. binding loss, enables the crossmodal pairing between action and description autoencoders. The binding

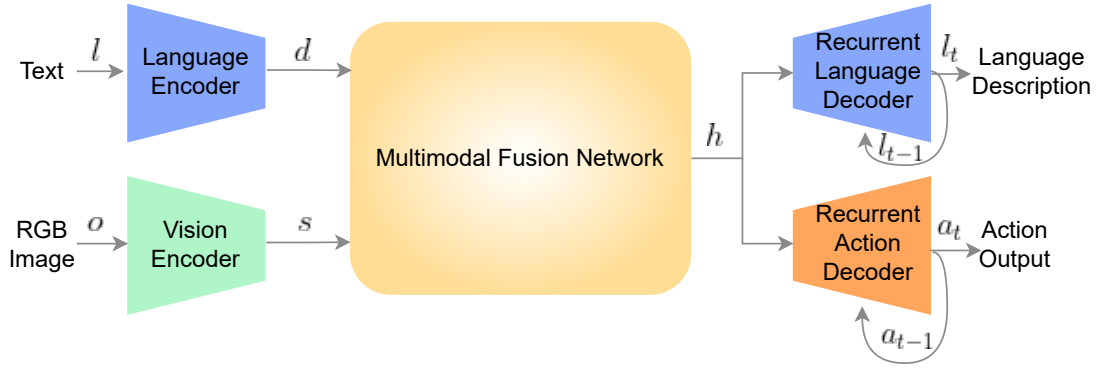


Figure 2.7: Conventional bidirectional model architectures used in practice. Most bidirectional models take language instructions in textual format and visual observations as RGB images as input. They encode the text and image inputs via their modality-specific networks: language and vision encoders. The multimodal fusion network is the backbone of the architecture and combines linguistic and visual features to arrive at a common hidden vector. Finally, the common hidden vector is used to output language and action predictions via the respective language and action decoders.

loss allows PRAE to execute actions given instructions as well as translate actions into descriptions. As a bidirectional approach, PRAE is biologically plausible to some extent, since humans can execute given commands and also describe these actions linguistically. Bidirectionality is essential to imitate human-like language recognition and production. However, due to its use of standard autoencoders, PRAE can only bind a robot action with a particular description in a one-to-one way even though actions can be expressed in language in various ways.

To map each robot action to multiple description alternatives, we introduce the PVAE (paired variational autoencoders) approach [136] which utilises variational autoencoders (VAEs) to randomise the latent representation space and thereby allows one-to-many translation between action and language. Specifically, we enable one-to-many translation between actions and descriptions by utilising the stochastic gradient variational Bayes-based sampling (SGVB) [90] that randomises the hidden representation space so that descriptions that are equivalent in meaning are represented tightly together, whereas those that have different meanings are represented far from each other. A review by Marino [111] highlights similarities between VAEs and predictive coding from neuroscience in terms of model formulations and inference approaches. PVAE extends the PRAE architecture [185] by replacing regular autoencoders with variational autoencoders so that it can learn one-to-many mappings between robot actions and instructions. Furthermore, it modifies the visual feature extraction by separately training the channels of the convolutional autoencoder, which leads to a more accurate recognition of object colours. In Chapter 3, we detail the PVAE model and the experiments conducted to test its one-to-many action-language translation capabilities. Inspired by the



TransferLangLFP paradigm by Lynch and Sermanet [107], we propose to use the PVAE with a pre-trained BERT language model [35] to facilitate the comprehension of unconstrained language instructions from human users. Furthermore, we perform experiments with PVAE-BERT on our dataset for various use cases and examine the internal representations for the first time. The details of the PVAE-BERT model and the experiments are given in Chapter 4.

Like PRAE, PVAE and PVAE-BERT employ a binding loss to map descriptions and actions. Therefore, due to its artificial nature in its multimodality fusion, PVAE too must be in a certain configuration according to the desired translation direction. To lift this constraint and allow flexible use of the model triggered by a verbally provided signal, we introduce the Paired Gated Autoencoders (PGAE) [135] model in Chapter 5. PGAE makes use of signal word prefixes and a flexible multimodal fusion technique, i.e. GMU. It is also able to recognise and imitate the actions of an opposite-sitting agent via robot demonstrations.

When we test PGAE with a more realistic case of limited labelled data, we observe a significant drop in performance. To address the issue of reliance on large quantities of labelled supervised samples, we introduce the Paired Transformed Autoencoders (PTAE) approach [134] that uses Transformer-based crossmodal attention to learn action-language mappings from mostly unsupervised data in an efficient manner. Chapter 6 details the PTAE architecture and the experiments conducted to prove the merits of PTAE in embodied language learning.

**LLM-Based Approaches** The bidirectional approaches that exploit LLMs in their architectures are modelled by optimising model parameters  $\theta$  based on the following log probabilities:

$$\log p_{\theta}(t_0, \dots, t_I) = \sum_{i=0}^{I-1} \log p_{\theta}(t_i | t_0, \dots, t_{i-1}), \quad (2.10)$$

where the model output token  $t_i$  depends on the previous tokens  $t_0, \dots, t_{i-1}$ . Figure 2.8 shows the architecture of such models.

For example, GATO [146] is a single multi-task, multi-embodiment model that is general and performs well on hundreds of tasks in various domains such as playing Atari games, manipulating objects and image captioning. Irrespective of the modality (e.g. vision, proprioception or language), the input is flattened and embedded before it is provided to the model. The model is a large Transformer decoder with the same weights and architecture for all tasks and is trained solely in a supervised manner. However, despite performing moderately in each task, the approach cannot compete with specialised approaches in various tasks. Besides, it is tested on a limited number of real-world robotics tasks.

Similarly, Driess et al. [41] introduce an embodied robotic task planning approach, PaLM-E, based on the PaLM LLM [32]. PaLM-E can work with multiple modalities including vision and states as all modalities are embedded with modality-specific encoders before being fed to the LLM. Inputs to the model are

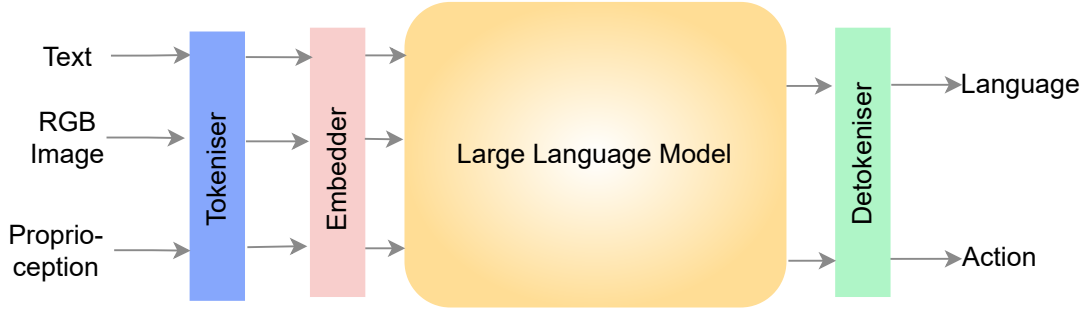


Figure 2.8: LLM-based bidirectional model architectures used in practice. These models first tokenise all modalities in order to have a unified representation of inputs. These tokenised inputs are embedded by encoders specific to each modality before being fed to the LLM. The LLM forms the backbone of the architecture and outputs target tokens, which are then detokenised accordingly. The model outputs are language (e.g. text) and action (e.g. end-effector poses).

multimodal sentences that interleave text with images and other embedded modalities. The model then outputs text specific to the task such as planning steps, answers to questions and descriptions of images. PaLM-E does not only perform well on robotic manipulation planning tasks but also on various VQA tasks. Although the model achieves impressive robotic manipulation planning performance, the approach requires an extra module to supply the robot with low-level outputs like joint angles or end-effector poses for action execution. In [Chapter 8](#), we introduce the CrossT5 architecture, utilising the intra-LLM integration (i.e. the encoder-decoder-based LLM employed is split in the middle to allow the integration of the action representations via a crossmodal network), which can output joint angle values and process potentially unrestricted language. Following that, in [Chapter 9](#), we present a minimally trained modular approach, leveraging out-of-the-box foundation models, including a powerful VLM, which combines communication skills and sensorimotor capabilities for a real-world HRI scenario.

As the successor of RT-1, RT-2 [\[21\]](#) leverages a VLM trained on large-scale Internet image-text data (i.e. PaLM-E or PaLI-X [\[28\]](#)). The approach fine-tunes the VLM on robotic tasks alongside vision-language tasks. This co-fine-tuning approach allows RT-2 to not only execute low-level actions but also generate high-level action plans. RT-2 outperforms previous SOTA models such as its predecessor RT-1, displaying emergent capabilities inherent in large VLMs. However, RT-2 is a substantially large model with billions of parameters. It requires plenty of computational resources and hence it is not possible to run it on a personal computer with a single GPU. Furthermore, RT-2 requires a pre-trained large VL model as a backbone, which itself is not widely available and trivial to train. More recently, Open X-Embodiment Collaboration has introduced the RT-X models [\[175\]](#), as well as a collection of 60 datasets that include 22 different robots and 527 robotic skills. RT-X models have the same architecture as RT-1 and RT-2 but are trained on var-

ious robotic manipulators. This extension leads to a significant performance boost across all tasks over task-specific models. Nevertheless, as they solely use SL and do not benefit from RL, these models rely on large datasets and they can struggle to generalise to cases where supervised data is scarce or unavailable. To address the problem of overreliance on large datasets of labelled robot actions, we propose the Crossmodal Bidirectional Transformer (XBiT) approach in [Chapter 7](#). XBiT first trains supervised on a relatively small dataset. After this pre-training, XBiT is fine-tuned in the simulation environment via a simple online RL algorithm. This two-stage training scheme, utilising supervised and reinforcement learning, improves performance in action execution on multiple object manipulation tasks.



# One-to-Many Action-to-Language Translation through Variation in Latent Space

---

Language acquisition is an integral part of developmental robotics, which seeks to understand the key components in human development and learn to utilise them in artificial agents. Like human infants, robots can learn language while interacting with objects in their environments and receiving linguistic input. This process, also known as embodied language learning, can enhance language acquisition in robots via multiple modalities such as visual and sensorimotor input. In this chapter, we explore ways to translate each simple action in a tabletop environment into various linguistic commands, based on an existing approach which exploits the idea of multiple autoencoders. While the existing approach focuses on strict one-to-one mappings between actions and descriptions by implicitly binding two standard autoencoders in the latent space, we propose a variational autoencoder model to facilitate one-to-many mapping between actions and descriptions. Additionally, we employ a channel-separated convolutional autoencoder to extract visual features more effectively. The results show that our model outperforms the existing approach in associating multiple commands with the corresponding action<sup>1</sup>.

## 3.1 Introduction

The linguistic capabilities of robots are still substantially inferior to humans although there have been many attempts at natural human-robot communication in recent years. Despite the recent success of LLMs in NLP applications such as virtual assistants and chatbots, language learning requires multisensory capabilities beyond processing text and images. Human infants, for example, use multiple senses, i.e. hearing, sight, touch, smell and taste, when they learn language. According to Bisk et al. [19], embodiment (action taking in the environment) is the

---

<sup>1</sup>The code belonging to this chapter is available at <https://github.com/oo222bs/PVAE>.

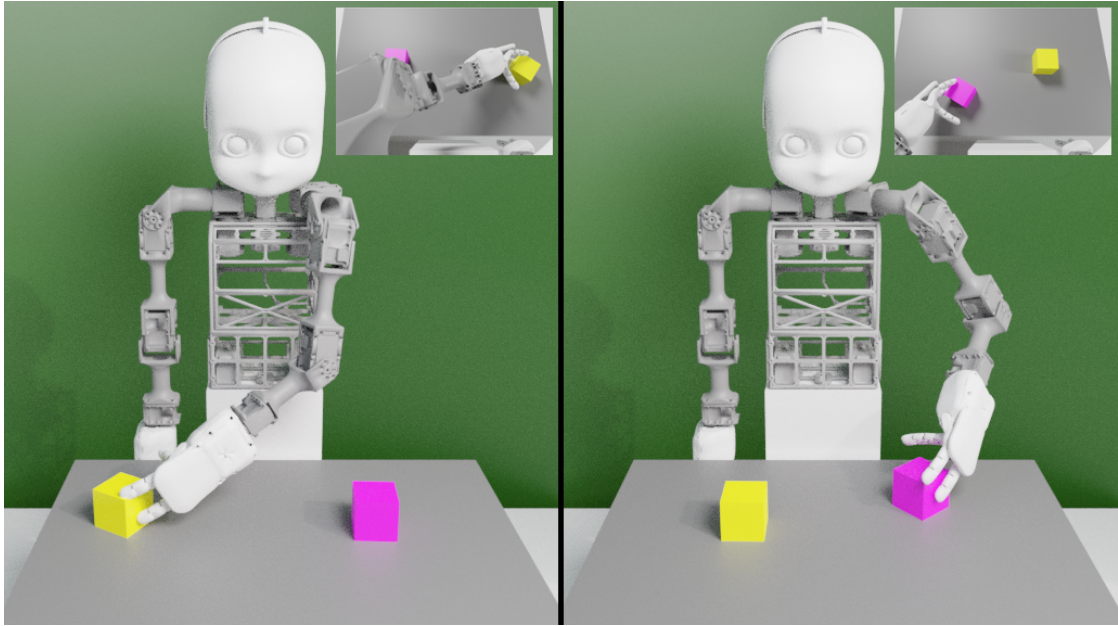


Figure 3.1: The NICO robot in the simulation environment: on the left, NICO is sliding the yellow cube; on the right, NICO is pulling the violet cube. In both segments, NICO’s field of view is shown in the top right insets.

needed next step after perception (using multimodal input) in language acquisition and production. An embodied agent must be able to relate language to physical control via sensory perception, as action and control open up new dimensions to understanding and actively learning about the world [19]. With embodiment, natural language processing can be brought to a level at which it can be deployed in realistic HRI contexts [19].

Embodied language learning is one of the main research topics in embodied robotics [66, 67, 68, 125]. In a typical scenario, a robot would execute an action and receive a description of the action. In well-structured environments, the complexity and ambiguity of language can be overcome by strictly defining the corpus, with each word having a distinct meaning. In this fashion, it is possible to translate actions into descriptions. Nevertheless, this strict one-to-one mapping between robot actions and linguistic descriptions is unnatural in human-to-human communication since we may use different words to describe the same action. In order to break the premise of one-to-one binding, alternative descriptions can be used to define an action, thereby facilitating one-to-many binding between action and language.

In our scenario, we have a robot, i.e. NICO (Neuro-Inspired COmpanion [85, 86]), interacting with cubes of different colours on a table in a simulation environment using its arm and hand while descriptions of the actions are provided (Figure 3.1). In this setup, we control the complexity of language and motion with appropriate actions and descriptions (e.g. “push the red cube fast”, “pull the green cube slowly”). Inspired by the work of Yamada et al. [185] with paired recurrent

autoencoders (PRAE), we propose a novel paired variational autoencoder (PVAE) architecture to enable our robot to translate from action to language in a one-to-many fashion. Our architecture is composed of two variational autoencoders: one for language and one for action. These two autoencoders, which consist of long short-term memory networks (LSTMs) [70], are integrated using Yamada et al.’s [185] binding loss in the latent space. PVAE extends the action-to-description translation capability of PRAE [185] by being capable of producing alternative versions of a description from an action (one-to-one vs one-to-many association). Aiming to address the research question of one-to-many association of actions and descriptions, i.e. an action can be translated into different variations of a description, the novelty of our PVAE model is exploiting a Bayesian method, i.e. variational autoencoders [90], to deal with the inexactness of relationships between actions and descriptions. PVAE learns from a dataset<sup>2</sup> that pairs visual observations and kinematics of actions with their corresponding textual descriptions. The robot actions are represented as sequences of joint angle values, and the visual input gathered from the egocentric perspective of the robot is extracted via a novel channel-separated convolutional autoencoder (CS-CAE). Besides, the textual descriptions are fed into the network word by word as sentences with one-hot encoding.

Our contribution, by introducing PVAE, is two-fold:

1. we show that employing variational autoencoders instead of standard autoencoders leads to a better one-to-many action-to-description translation accuracy, especially with a larger corpus and more data, hence addresses the linguistic ambiguity between an action and its probable descriptions;
2. the experiment results also indicate the superiority of channel separation (i.e. channel-separated CAE) in visual feature extraction, leading to a more accurate recognition of object colours where the objects cover only a small portion of the visual field.

The rest of this chapter is divided into the following parts: in the next section, we describe the PVAE architecture and its training in detail. In [Section 3.3](#), we explain the conducted experiments and present their results, comparing PVAE with PRAE and channel-separated CAE with standard CAE. In [Section 3.4](#), these results are discussed, giving reasons for PVAE’s superior performance. Finally, [Section 3.5](#) summarises the chapter while hinting at future work.

## 3.2 Proposed Method: Paired Variational Autoencoders (PVAE)

Following the PRAE approach [185], we use two recurrent autoencoders to learn robot actions and the corresponding descriptions alongside the mapping between

---

<sup>2</sup><https://www.inf.uni-hamburg.de/en/inst/ab/wtm/research/corpora.html>



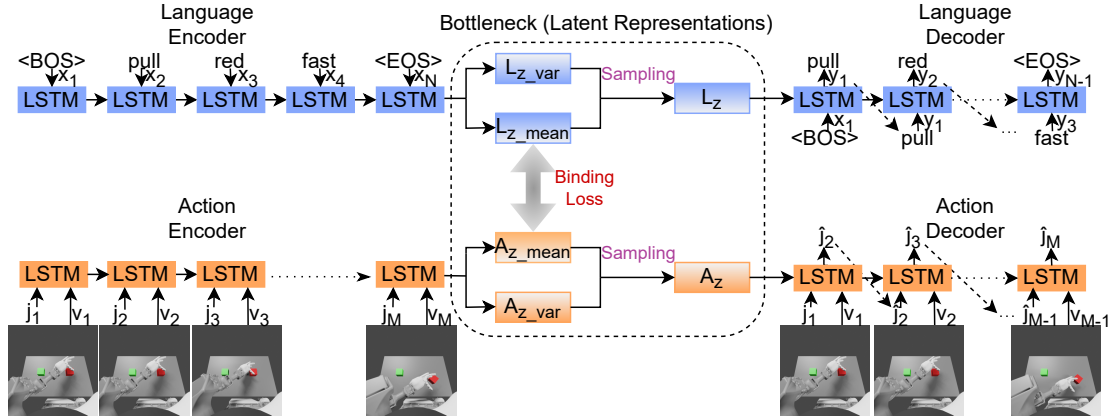


Figure 3.2: The architecture of the PVAE model. The language VAE (depicted with blue rectangles which denote unfolded LSTMs) is responsible for reconstructing descriptions and the action VAE (depicted with orange rectangles which denote unfolded LSTMs) is responsible for reconstructing the joint angles at each time step. The input to the language VAE is a one-hot-encoded word of a given description at a time, whereas the action VAE takes as input joint angle values and visual features at a time. The two VAEs are implicitly bound via a binding loss between their latent representations.

them in the latent space. Both autoencoders have a very similar architecture with LSTMs for temporal sequence processing. Different from PRAE [185], instead of regular autoencoders, we employ variational recurrent autoencoders (VRAE) [45], in which latent vectors are randomly sampled from a normal distribution over latent variables. Combining the capabilities of LSTMs and the Stochastic Gradient Variational Bayes (SGVB) [90] allows us to have efficient unsupervised learning on sequential data [45]. Moreover, following the advice of Yamada et al. [185] to use a Bayesian method, we overcome the inevitable ambiguity of relations between actions and descriptions. Thus, our approach can link an action with multiple language instructions. The two variational autoencoders that form the PVAE architecture are the language VAE and action VAE. The language VAE learns descriptions in an unsupervised manner, while the action VAE learns joint angles conditioned on the visual input similarly.

### 3.2.1 Model Architecture

As shown in Figure 3.2, the architecture consists of two VAEs: language and action VAE. The input to the language VAE is a sentence describing a robot action. The sentence is fed into the language encoder word by word using one-hot encoding. After the encoding phase, the encoded representation is used to extract latent representations using the reparameterisation trick [90], following the VRAE approach [45]. These latent representations are exploited in the decoder to reproduce the sentence describing the action.



The action VAE has two types of input: robot joint angles and visual input from the perspective of the robot. After the encoding, similar to the language VAE, latent representations are extracted from encoded actions in the bottleneck. The action decoder reproduces the joint angles from the latent representations, conditioned on the visual features. The two VAEs have no explicit connection, but they are integrated with a binding loss reducing the distance between two latent variables, binding actions to descriptions bidirectionally.

### 3.2.2 Language Autoencoder

The language VAE encodes descriptions so that they can be reproduced by decoding. It has two components which are the language encoder and decoder. The language encoder embeds a description of length  $N$ ,  $(x_1, x_2, \dots, x_N)$ , into two fixed-dimensional vectors,  $z_{\text{mean}}$  and  $z_{\text{sigma}}$ , with the following equations:

$$h_t^{\text{enc}} = \text{EncCell}(x_t, h_{t-1}^{\text{enc}}) \quad (1 \leq t \leq N), \quad (3.1)$$

$$z_{\text{mean}} = W_{\text{mean}}^{\text{enc}} \cdot h_N + b_{\text{mean}}^{\text{enc}}, \quad (3.2)$$

$$z_{\text{sigma}} = W_{\text{sigma}}^{\text{enc}} \cdot h_N + b_{\text{sigma}}^{\text{enc}}, \quad (3.3)$$

$$z_{\text{lang}} = z_{\text{mean}} + z_{\text{sigma}} \cdot \mathcal{N}(\mu, \sigma^2), \quad (3.4)$$

where EncCell is an LSTM,  $h_t$  is the state of the LSTM at time step  $t$ , with  $h_0$  set as a zero vector.  $\mathcal{N}$  is a Gaussian distribution with  $\mu$  and  $\sigma$  as its mean and standard deviation respectively.  $z_{\text{lang}}$  is the latent representation of a description. The language decoder generates a sequence by recursively expanding  $z_{\text{lang}}$ :

$$h_0^{\text{dec}} = W^{\text{dec}} \cdot z_{\text{lang}} + b^{\text{dec}}, \quad (3.5)$$

$$h_t^{\text{dec}} = \text{DecCell}(y_{t-1}, h_{t-1}^{\text{dec}}) \quad (1 \leq t \leq N - 1), \quad (3.6)$$

$$y_t = s(W^{\text{out}} \cdot h_t^{\text{dec}} + b^{\text{out}}) \quad (1 \leq t \leq N - 1), \quad (3.7)$$

where DecCell is an LSTM and  $s$  is the softmax activation function.  $y_0$  is given as a first symbol indicating the beginning of the sentence.

### 3.2.3 Action Autoencoder

The action VAE encodes robot actions with its encoder, allowing its decoder to reproduce them. Similar to the language VAE, it has two components: action encoder and action decoder. The action encoder encodes a sequence of length  $M$ ,  $((j_1, v_1), (j_2, v_2), \dots, (j_M, v_M))$ , that concatenates joint angles,  $j$ , with visual features,  $v$ , which are extracted from images by the channel-separated convolutional autoencoder:

$$h_t^{\text{enc}} = \text{EncCell}(v_t, j_t, h_{t-1}^{\text{enc}}) \quad (1 \leq t \leq M), \quad (3.8)$$

$$z_{\text{mean}} = W_{\text{mean}}^{\text{enc}} \cdot h_M + b_{\text{mean}}^{\text{enc}}, \quad (3.9)$$

$$z_{\text{sigma}} = W_{\text{sigma}}^{\text{enc}} \cdot h_M + b_{\text{sigma}}^{\text{enc}}, \quad (3.10)$$

$$z_{\text{act}} = z_{\text{mean}} + z_{\text{sigma}} \cdot \mathcal{N}(\mu, \sigma^2), \quad (3.11)$$

where EncCell is an LSTM,  $h_t$  is the state of the LSTM at time step  $t$ , with  $h_0$  set as a zero vector.  $\mathcal{N}$  is a Gaussian distribution with  $\mu$  and  $\sigma$  as its mean and standard deviation respectively.  $z_{\text{act}}$  is the latent representation of a robot action. The action decoder reconstructs the joint angles:

$$h_0^{\text{dec}} = W^{\text{dec}} \cdot z_{\text{act}} + b^{\text{dec}}, \quad (3.12)$$

$$h_t^{\text{dec}} = \text{DecCell}(v_t, \hat{j}_t, h_{t-1}^{\text{dec}}) \quad (1 \leq t \leq M-1), \quad (3.13)$$

$$\hat{j}_{t+1} = \tanh(W^{\text{out}} \cdot h_t^{\text{dec}} + b^{\text{out}}) \quad (1 \leq t \leq M-1), \quad (3.14)$$

where DecCell is an LSTM,  $\tanh$  is the hyperbolic tangent activation function and  $\hat{j}_1$  is equal to  $j_1$ .

### 3.2.4 Visual Feature Extraction

We follow the visual feature extraction architecture provided by Yamada et al. [185]. Accordingly, our CS-CAE accepts  $120 \times 160$  RGB images gathered from the egocentric view of the robot; it consists of a convolutional [92, 93, 94] encoder, a fully connected bottleneck (from which latent representations are extracted) and a deconvolutional [189] decoder. However, our CS-CAE is trained separately for each channel (red, green and blue) to recognise different colours more accurately, which we define as channel separation. After training, we extract the visual features of each image for all channels from the bottleneck positioned between the encoder and the decoder. The visual features extracted from each channel are then concatenated to form the ultimate visual features  $v$ .

### 3.2.5 Sampling and Binding

The two VAEs have identical random sampling procedures. After producing the latent variables,  $z_{\text{mean}}$  and  $z_{\text{sigma}}$ , using fully connected layers, a normal distribution,  $\mathcal{N}(\mu, \sigma^2)$ , is used to derive random latent representations, which are, in turn, used with  $z_{\text{mean}}$  and  $z_{\text{sigma}}$  to arrive at the ultimate sample latent representation  $z$  [90]:

$$z = z_{\text{mean}} + z_{\text{sigma}} \cdot \epsilon \quad (3.15)$$

where  $\epsilon$  is the approximation of  $\mathcal{N}(\mu, \sigma^2)$  with  $\mu = 0$  and  $\sigma^2 = 0.01$ .

Similar to PRAE [185], in order to bind the encodings of the language and action VAEs, we use an extra loss term that brings  $z_{\text{mean}}$  values of the two VAEs closer. This allows the network to bidirectionally translate actions to descriptions and vice versa after training, without an explicit fusion of the two modalities. The loss term is given below:

$$L_{\text{binding}} = \sum_i^B \psi(z_{\text{mean}_i}^{\text{lang}}, z_{\text{mean}_i}^{\text{act}}) + \sum_i^B \sum_{j \neq i}^B \max \left\{ 0, \Delta + \psi(z_{\text{mean}_i}^{\text{lang}}, z_{\text{mean}_i}^{\text{act}}) - \psi(z_{\text{mean}_j}^{\text{lang}}, z_{\text{mean}_i}^{\text{act}}) \right\}, \quad (3.16)$$

where  $B$  stands for the batch size and  $\psi$  is the Euclidean distance. The first term in the equation aligns the corresponding instructions and actions whereas the second term helps distinguish irrelevant actions from descriptions. Hyperparameter  $\Delta$  is used to adjust the second term.

### 3.2.6 Loss Function

The total loss function has three main components: reconstruction, regularisation and binding loss. The binding loss is calculated for both VAEs together. In contrast, the reconstruction and regularisation losses are calculated independently for each VAE. The respective reconstruction loss for the language VAE,  $L_{\text{lang}}$ , and action VAE,  $L_{\text{act}}$ , are defined as:

$$L_{\text{lang}} = \frac{1}{N-1} \sum_{t=1}^{N-1} \left( - \sum_w x_{t+1}(w) \log y_t(w) \right), \quad (3.17)$$

$$L_{\text{act}} = \frac{1}{M-1} \sum_{t=1}^{M-1} \|j_{t+1} - \hat{j}_{t+1}\|_2^2, \quad (3.18)$$

where  $W$  is the vocabulary size. The regularisation loss, which is specific to variational autoencoders, is defined as Kullback–Leibler divergence for language  $D_{\text{KL}_{\text{lang}}}$  and action  $D_{\text{KL}_{\text{act}}}$ . Therefore, the overall loss function can be defined as:

$$L_{\text{all}} = \alpha L_{\text{lang}} + \beta L_{\text{act}} + \gamma L_{\text{binding}} + \alpha D_{\text{KL}_{\text{lang}}} + \beta D_{\text{KL}_{\text{act}}}, \quad (3.19)$$

where  $\alpha$ ,  $\beta$  and  $\gamma$  are weighting factors for different terms in the loss function. In our experiments,  $\alpha$  and  $\beta$  are set to 1 and  $\gamma$  is set to 2 to bind the two modalities sufficiently.

### 3.2.7 Training Details

The model was trained with both networks (the language and action VAEs) together for 15,000 iterations and the gradient descent method was employed to update the weights using the Adam optimiser [89]. The learning rate was set to  $10^{-4}$  and the batch size was chosen as 100 (100 description and action pairs) after preliminary studies.

## 3.3 Experiments and Results

The proposed PVAE is evaluated with two experiments of varying data sizes to display the advantage of using variational autoencoders over regular autoencoders, i.e. PRAE, and the benefit of using the channel separation technique in visual feature extraction. We also analyse the impact of a larger corpus on action-to-language translation. Therefore, in both experiments, our PVAE and Yamada et al.’s PRAE [185] are trained on the same datasets and we evaluate their accuracy

in translating actions into descriptions. In addition, to see the effect of channel separation on the overall translation accuracy, we train our architecture with visual features provided by a regular CAE without channel separation. In all experiments, two cubes of different colours are placed on a table where the robot is seated to interact with the cubes. For the first experiment, each cube is one of three colours (red, green, blue) and, for the second, one of six colours (red, green, blue, yellow, cyan, violet). The words (vocabulary) that make up the descriptions are given in Table 3.1. Two-word phrases like ‘move up’ are hyphenated (i.e. ‘move-up’) and hence considered a single word for the one-hot encoding. We introduce a more diverse vocabulary by adding an alternative word for every word belonging to the original vocabulary.

Each cube arrangement has two cubes and these cubes are never of the same colour. There are three action types (‘PUSH’, ‘PULL’, ‘SLIDE’), two positions (‘L’, ‘R’) and two speed settings (‘SLOW’, ‘FAST’): 12 possible actions. Each sentence has three words (excluding the <BOS/EOS> tags which indicate the beginning or end of a sentence) with the first word indicating the action, the second the cube colour and the last the speed at which the action is taken (e.g. “push green slowly”). Therefore, without the alternative words, there are 18 possible sentences (3 action verbs  $\times$  3 colours  $\times$  2 adverbs). As a result, our dataset consists of six cube arrangements (12 for the second experiment),  $18 \times 8 = 144$  possible sentences ( $36 \times 8 = 288$  for the second experiment – the factor of eight because of eight alternatives for each sentence) and 12 actions ( $3 \times 2 \times 2$ ). We have 72 patterns for the first experiment (12 actions with six cube arrangements each) and 144 patterns for the second. Following Yamada et al. [185], we choose the patterns (action-description-arrangement combinations) rigorously ensuring that combinations of action, description and cube arrangements selected for the test set do not exist in the training set although every action, description and cube arrangement is shown during training. Therefore, 54 patterns are used for training while the remaining 18 are for testing (second experiment: 108 for training, 36 for testing). Each pattern is collected six times in the simulation with random variations on the action execution resulting in different joint trajectories. Additionally, we use 4-fold cross-validation to provide more reliable results.

Table 3.1: Dataset Vocabulary with Original Words and Their Alternatives

Original	Alternative	Original	Alternative
push	move-up	yellow	blonde
pull	move-down	cyan	greenish-blue
slide	move-sideways	violet	purple
red	scarlet	slowly	unhurriedly
green	harlequin	fast	quickly
blue	azure		

Table 3.2: Experiment 1: Action-to-Description Accuracy with Three Colours

Method	Training Acc.	Test Acc.
PRAE + regular CAE	33.33 $\pm$ 1.31%	33.56 $\pm$ 3.03%
PVAE + regular CAE	66.6 $\pm$ 1.31%	65.28 $\pm$ 6.05%
PVAE + CS-CAE	<b>100.0 <math>\pm</math> 0.0%</b>	<b>90.28 <math>\pm</math> 4.61%</b>

The robot used in our experiments is NICO (Neuro-Inspired COmpanion) [85, 86] in a virtual environment created with the Blender software – see Figure 3.1. NICO is a humanoid robot, approximately one metre tall and weighs approximately 20 kg. We use the left arm of NICO to interact with the objects utilising five joints. Actions are realised with an inverse kinematics solver. NICO has a camera in each of its eyes, which is used to extract egocentric visual images.

### 3.3.1 Experiment 1: Three Colour Alternatives

We use the same instructions and actions as in the PRAE paper [185], e.g. “PUSH-R-SLOW” which can be interpreted as “push the right object slowly”. Similarly, we use three colour options for the cubes following PRAE. In addition, the instructions are extended by adding an alternative for each word in the vocabulary. Hence, the vocabulary size of 9 is extended to 17 (we do not add an alternative for <BOS/EOS> tags). As every sentence is composed of three words, we extend the number of sentences by a factor of eight ( $2^3 = 8$ ).

After training our PVAE and PRAE, we test them for action-to-description translation. For the reproduced description to count as correct, all three words (plus the <BOS/EOS> tag) must be correctly predicted. As each description has seven more alternatives, predicting any of the eight correct descriptions is considered successful. As can be seen in Table 3.2, our model translates approximately 90% of the patterns in the test set (last row), while PRAE translates only one third of the patterns. Thus, our model outperforms PRAE in one-to-many mapping.

We also test the impact of channel separation on the translation accuracy by training our model with visual features extracted with the regular CAE as described in Yamada et al.’s approach [185]. In Table 3.2, we can see that using our variational approach alone significantly increases the accuracy. Nevertheless, using PVAE with channel-separated CAE improves the results further, indicating the superiority of channel separation in our tabletop setting. Therefore, our approach with variational autoencoders and a channel-separated CAE is superior to both PRAE and PVAE with regular visual feature extraction in this experiment, involving three colours.

Table 3.3: Experiment 2: Action-to-Description Accuracy with Six Colours

Method	Training Acc.	Test Acc.
PRAE + regular CAE	33.64 $\pm$ 1.13%	33.3 $\pm$ 0.98%
PVAE + regular CAE	69.60 $\pm$ 0.46%	61.57 $\pm$ 2.01%
PVAE + CS-CAE	<b>100.0 <math>\pm</math> 0.0%</b>	<b>100.0 <math>\pm</math> 0.0%</b>

### 3.3.2 Experiment 2: Six Colour Alternatives

To test the limits of our PVAE and the impact of more data with a larger corpus, we add three more colour options for the cubes: yellow, cyan and violet. These secondary colours are combined amongst themselves for the arrangements in addition to the colour combinations used in the first experiment, i.e. a cube of a primary colour and a cube of a secondary colour do not co-occur. Therefore, this experiment has 12 arrangements. Moreover, the vocabulary size is extended to 23 compared with 17 in Experiment 1 (two alternative words for each colour – see Table 3.1). As in the first experiment, each sentence has eight alternative ways to be described.

We train both PVAE and PRAE on the extended dataset from scratch and test both architectures. As shown in Table 3.3, PVAE succeeds in performing 100% by translating every pattern from action to description correctly, even for the test set. In contrast, PRAE performs poorly in this setting and manages to translate only one third of the descriptions correctly in the test set. Compared with the accuracy values reached in the first experiment with less data and a smaller corpus, extending the dataset helps PVAE to perform better in translation, while PRAE is not able to take advantage of more data.

As in Experiment 1, we also test the influence of channel separation on translation accuracy by training PVAE with visual features provided by a regular CAE. In this setting, PVAE only achieves around 61% of accuracy in the test set. This highlights again the importance of channel separation in visual feature extraction for our setup. While the improvement by using our PVAE over PRAE is significant, further improvement is made by utilising CS-CAE.

## 3.4 Discussion

The results from both experiments show that our variational autoencoder approach with a channel-separated CAE visual feature extraction outperforms the standard autoencoder approach, i.e. PRAE, in the one-to-many translation of actions into language commands. Our approach not only proved more successful in the case of three colour alternatives per cube but also in the case of six colour alternatives by a large margin. Specifically, when the dataset and the corpus were extended, our PVAE model performed better, proving that a Bayesian method like variational

autoencoders can scale up with more data for generalisation; on the contrary, standard autoencoders cannot capitalise on more data. Moreover, standard autoencoders are fairly limited when it comes to handling ambiguity in linguistic input. In contrast, variational autoencoders yield remarkably better results in one-to-many mapping between actions and descriptions, because stochastic generation (random normal distribution) within the latent feature extraction allows latent representations to slightly vary, leading to VAEs learning not only one specific description but various descriptions for each action. Moreover, analysing the specific case in which we train our PVAE with visual features extracted by the standard CAE demonstrates that separating the channels of CAE helps to increase distinguishing objects of different colours significantly with a visual input in our setup in which the objects cover only a modest portion of the visual field.

On the one hand, the translation accuracy results of PRAE show that adding more data (i.e. more colour options) does not help regular autoencoders bind actions with descriptions more successfully in the case of one-to-many mapping. In fact, the results of PRAE on both experiments are very similar, which also indicates that regular autoencoders are not suitable for this task as they do not respond well to data with more variations.

On the other hand, our PVAE performs even better with more colour alternatives when trained with visual features extracted by the channel-separated CAE. This shows the importance of larger and more varied data in one-to-many mapping. When compared with our PVAE and channel-separated CAE approach, the ‘PVAE + regular CAE’ option yields significantly lower translation accuracy, which exhibits the importance of visual input since the channel-separated CAE performs a more accurate visual feature extraction than a standard CAE in our setup. A significant number of errors for the ‘PVAE + regular CAE’ case was caused by failing to distinguish colours, e.g. “push green slowly” instead of “push red slowly”.

## 3.5 Conclusion

PVAE has extended one-to-one mapping between actions and descriptions to one-to-many mapping by employing variational autoencoders to tackle the ambiguity of various descriptions describing the same action. Besides, we have shown that in tabletop scenarios with the objects covering only a small portion of the visual field, it is plausible to use a channel-separated CAE to distinguish objects of different colours. The experimental results show that our approach with variational and channel-separated autoencoders outperforms the standard autoencoder approach PRAE in one-to-many action-to-description translation. Moreover, our PVAE network performs even better with more data and a larger corpus, suggesting that Bayesian methods like VAEs may scale up well with more data. In real-world applications, exploiting Bayesian methods may lead to profiting from larger and more complex data.

In the next chapter, we will introduce the PVAE-BERT model which extends the PVAE architecture with a pre-trained language model, i.e. BERT. PVAE-

BERT furthers the linguistic capabilities of action-language modelling by allowing the network to comprehend more diverse and natural language instructions such as full commands and polite requests.



# Advanced Language Comprehension via a Pre-trained Language Model

---

Human infants learn language while interacting with their environment in which their caregivers may describe the objects and actions they perform. Similar to human infants, artificial agents can learn language while interacting with their environment. In this chapter, first, we present a neural model that bidirectionally binds robot actions and their matching language descriptions in a simple object manipulation scenario. Building on our previous Paired Variational Autoencoders (PVAE) model, we demonstrate the superiority of the variational autoencoder over standard autoencoders by experimenting with cubes of different colours and enabling the production of alternative vocabularies. Additional experiments show that the model’s channel-separated visual feature extraction module can cope with objects of varying shapes. Next, we introduce PVAE-BERT, which equips the model with a pre-trained large-scale language model, i.e. Bidirectional Encoder Representations from Transformers (BERT), enabling the model to go beyond comprehending only the predefined descriptions that the network has been trained on; the recognition of action descriptions generalises to unconstrained natural language as the model becomes capable of understanding unlimited variations of the same descriptions. Our experiments suggest that using a pre-trained language model as the language encoder allows our approach to scale up for real-world scenarios with instructions from human users<sup>1</sup>.

## 4.1 Introduction

Humans use language as a means to understand and to be understood by their interlocutors. Although we can communicate effortlessly in our native language, language is a sophisticated form of interaction which requires comprehension and

---

<sup>1</sup>The code of this chapter is available at <https://github.com/oo222bs/PVAE-BERT>.

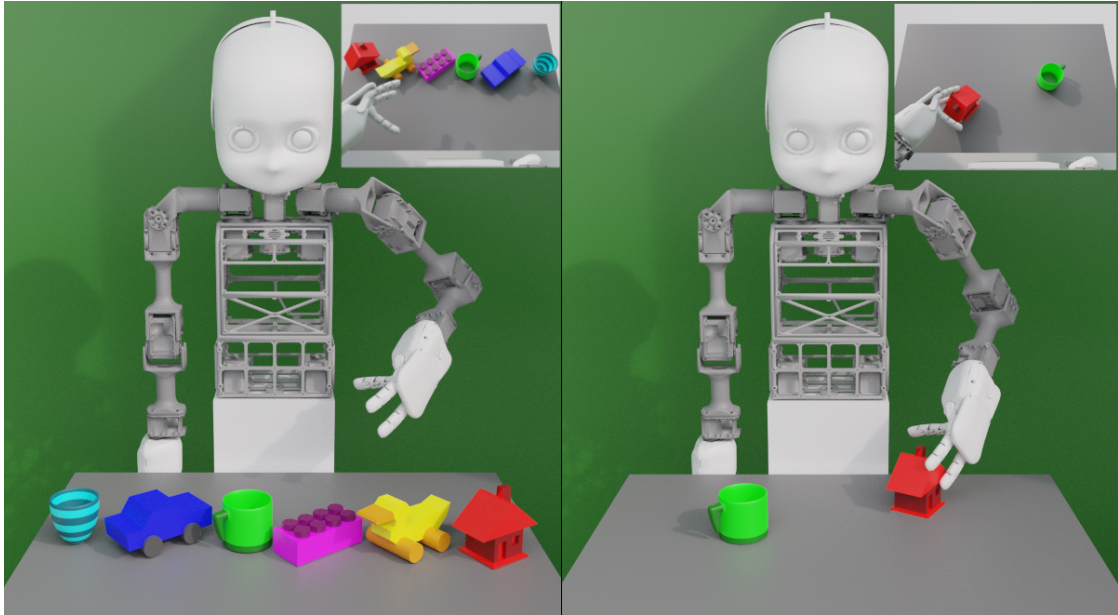


Figure 4.1: Our tabletop object manipulation scenario in the simulation environment: the NICO robot interacts with toy objects. In the left panel, NICO views all the toy objects; on the right, NICO pulls the red house. In both panels, NICO’s field of view is given in the top right inset. Note that the left panel displays all the toy objects for illustrative purposes. In all our experiments, only two objects are placed on the table, as exemplified in the right panel.

production skills. Understanding language depends also on the context, because words can have multiple meanings and a situation can be explained in many ways. As it is not always possible to describe a situation only in language or understand it only with the medium of language, we benefit from other modalities such as vision and proprioception. Similarly, artificial agents can utilise the concept of embodiment (i.e. acting in the environment) in addition to perception (i.e. using multimodal input like audio and vision) for better comprehension and production of language [19]. Human infants learn language in their environment while their caregivers describe the properties of the objects, which they interact with, and the actions, which are performed on those objects. In a similar vein, artificial agents can be taught language; different modalities such as audio, touch, proprioception and vision can be employed towards learning language in the environment.

The field of artificial intelligence has recently seen many approaches attempting to learn language in an embodied fashion [7, 26, 68, 107, 125]. In this work, we bidirectionally map language with robot actions by employing three distinct modalities, i.e. text, proprioception and vision. In our robotic scenario, two objects are placed on a table as the NICO (Neuro-Inspired COmpanion) robot [86] physically interacts with them (see Figure 4.1). NICO moves objects along the table surface according to given textual descriptions and recognises the actions by translating them to corresponding descriptions. The possibility of bidirectional

translation between language and control was realised with a paired recurrent autoencoder (PRAE) architecture by Yamada et al. [185], which aligns the two modalities that are each processed by an autoencoder. We extended this approach (PRAE) with the Paired Variational Autoencoders (PVAE) [136] model, which enriches the language used to describe the actions taken by the robot: instead of mapping a distinct description to each action [185], PVAE maps multiple descriptions, which are equivalent in meaning, to each action. Hence, we have transcended the strict one-to-one mapping between control and language since our variational autoencoder-based model can associate each robot action with multiple description alternatives. PVAE is composed of two variational autoencoders (VAEs), one for language, the other for action, and both of them consist of an LSTM (long short-term memory) [70] encoder and decoder which are suitable for sequential data. The dataset<sup>2</sup> that we train our model on consists of paired textual descriptions and corresponding joint angle values with egocentric images. The language VAE reconstructs descriptions, whereas the action VAE reconstructs joint angle values that are conditioned on the visual features extracted in advance by the channel-separated convolutional autoencoder (CS-CAE) [136] from egocentric images. The two autoencoders are implicitly bound together with an extra loss term which aligns actions with their corresponding descriptions and separates unrelated actions and descriptions in the hidden vector space.

However, even with multiple descriptions mapped to a robot action as implemented in our previous work [136] and described in Chapter 3, replacing each word with its alternative does not lift the grammar restrictions on the language input. In order to process unconstrained language input, we equip the PVAE architecture with the Bidirectional Encoder Representations from Transformers (BERT) language model [35] that has been pre-trained on large-scale text corpora to enable the recognition of unconstrained natural language commands by human users. To this end, we replace the LSTM language encoder with a pre-trained BERT model so that PVAE can recognise different commands corresponding to the same actions as the predefined descriptions given the same object combinations on the table. This new model variant, which we call PVAE-BERT, can handle not only the descriptions it is trained on but also various descriptions equivalent in meaning with different word order and/or filler words (‘please’, ‘could’, ‘the’, etc.) as our analysis shows. We capitalise on transfer learning by utilising a pre-trained language model and thus benefit from large unlabelled textual data.

Our contributions in this chapter can be summarised as follows.

1. In Chapter 3, we showed that variational autoencoders facilitate better one-to-many action-to-language translation and that channel separation in visual feature extraction, i.e. training RGB channels separately, results in more accurate recognition of object colours in our object manipulation scenario. Here, we extend our dataset with different shapes and show that our PVAE with the channel separation approach is able to translate from action to language while manipulating different objects.

<sup>2</sup><https://www.inf.uni-hamburg.de/en/inst/ab/wtm/research/corpora.html>

2. In this chapter, we introduce PVAE-BERT, which, by using pre-trained BERT, indicates the potential of our approach to be scaled up for unconstrained instructions from human users.
3. The principal component analysis (PCA) shows that language and action representation vectors are arranged according to the semantics of the language descriptions.

The remainder of this chapter is organised as follows: the next section presents the architecture of the PVAE and PVAE-BERT models. In Section 4.3 various experiments and their results are given. Section 4.4 discusses the results and their implications. The last section concludes the paper with final remarks.

## 4.2 Proposed Method: PVAE-BERT

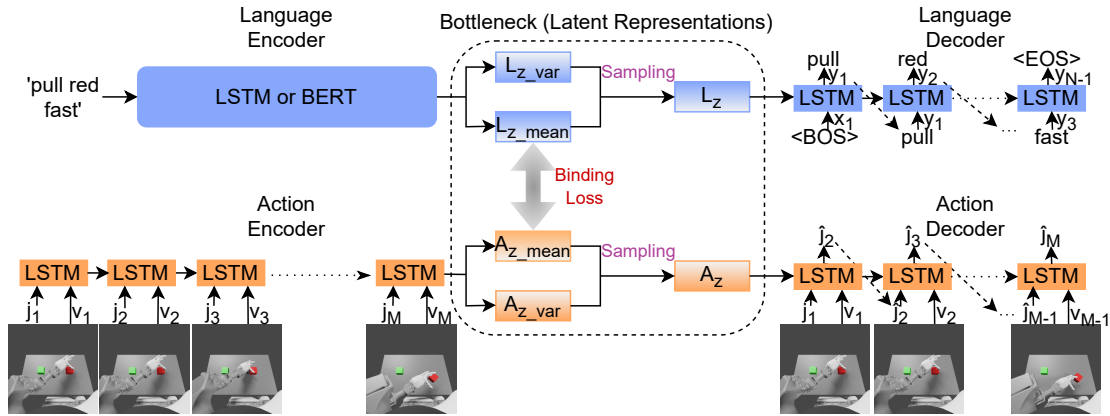


Figure 4.2: The architecture of the PVAE and PVAE-BERT models. The language VAE (blue rectangles) processes descriptions, while the action VAE (orange rectangles) processes joint angles and images at each time step. The input to the language VAE is the given description  $x$ , whereas the action VAE takes as input joint angle values  $j$  and visual features  $v$ . The two VAEs are implicitly bound via a binding loss in the latent representation space.  $\langle \text{BOS} \rangle$  and  $\langle \text{EOS} \rangle$  denote *beginning of sentence* and *end of sentence* tags respectively. The two models differ solely in the language encoder used: PVAE employs an LSTM, whereas PVAE-BERT utilises a pre-trained BERT model.

As can be seen in Figure 4.2, the PVAE model consists of two variational autoencoders: a language VAE and an action VAE. The former learns to generate descriptions matching original descriptions, while the latter learns to reconstruct joint angle values with conditioning on the visual input. The two autoencoders do not have any explicit neural connection between them; instead, they are implicitly aligned by the binding loss, which brings the two autoencoders closer to each other in the latent space over the course of learning by reducing the distance between

the two latent variables. First, action and language encoders map the input to the latent code, i.e. the language encoder accepts one-hot encoded descriptions word by word as input and produces the encoded descriptions, whereas the action encoder accepts corresponding arm trajectories and visual features as input and produces the encoded actions. Next, the encoded representations are used to extract latent representations by randomly sampling from a Gaussian distribution separately for language and action modalities. Finally, from the latent representations, language and action decoders reconstruct the descriptions and joint angle values respectively.

Our model is a bidirectional approach, i.e. after training, translation is possible in both directions, action-to-language and language-to-action. The PVAE model transforms robot actions into descriptions in a one-to-many fashion by appropriately randomising the latent space. PVAE-BERT additionally handles variability in language input by using pre-trained BERT as the language encoder module. As part of the action encoder, the visual input features are extracted in advance using a channel-separated CAE, which improves the ability of the approach to distinguish the colours of cubes. The details of each model component are given in the following subsections.

### 4.2.1 Language Variational Autoencoder

The language VAE accepts as input one-hot encoded matrix of a description word by word in the case of PVAE or the complete description altogether for PVAE-BERT, and for both PVAE and PVAE-BERT, it is responsible for reproducing the original description. It consists of an encoder, a decoder and latent layers (in the bottleneck) where latent representations are extracted via sampling. For PVAE, the language encoder embeds a description of length  $N$ ,  $(x_1, x_2, \dots, x_N)$ , into two fixed-dimensional vectors,  $z_{\text{mean}}$  and  $z_{\text{sigma}}$ , as follows:

$$h_t^{\text{enc}}, c_t^{\text{enc}} = \text{LSTM}(x_t, h_{t-1}^{\text{enc}}, c_{t-1}^{\text{enc}}) \quad (1 \leq t \leq N), \quad (4.1)$$

$$z_{\text{mean}} = W_{\text{mean}}^{\text{enc}} \cdot h_N + b_{\text{mean}}^{\text{enc}}, \quad (4.2)$$

$$z_{\text{var}} = W_{\text{var}}^{\text{enc}} \cdot h_N + b_{\text{var}}^{\text{enc}}, \quad (4.3)$$

$$z_{\text{lang}} = z_{\text{mean}} + z_{\text{var}} \cdot \mathcal{N}(\mu, \sigma^2), \quad (4.4)$$

where  $h_t$  and  $c_t$  are the hidden and cell state of the LSTM at time step  $t$  respectively, and  $\mathcal{N}$  is a Gaussian distribution with mean  $\mu$  and standard deviation  $\sigma$ .  $h_0$  and  $c_0$  are set as zero vectors, while  $\mu$  and  $\sigma$  are 0 and 0.1 respectively.  $z_{\text{lang}}$  is the latent representation of a description. LSTM here, and in the following, is a peephole LSTM [152] following the implementation of Yamada et al. [185]. The language input is represented in one-hot encoded matrices, whose rows represent the sequence of input words and columns represent every word included in the vocabulary. In each row, only one cell is 1 and the rest are 0 which determines the word given to the model at that time step.

For PVAE-BERT, we replace the LSTM language encoder with the pre-trained BERT<sub>BASE</sub> model, and following the implementation by Devlin et al. [35], we

tokenise the descriptions accordingly with the subword-based tokeniser Word-Piece [184].

The language decoder LSTM generates a sequence by recursively expanding  $z_{\text{lang}}$ :

$$h_0^{\text{dec}}, c_0^{\text{dec}} = W^{\text{dec}} \cdot z_{\text{lang}} + b^{\text{dec}}, \quad (4.5)$$

$$h_t^{\text{dec}}, c_t^{\text{dec}} = \text{LSTM}(y_{t-1}, h_{t-1}^{\text{dec}}, c_{t-1}^{\text{dec}}) \quad (1 \leq t \leq N-1), \quad (4.6)$$

$$y_t = \text{soft}(W^{\text{out}} \cdot h_t^{\text{dec}} + b^{\text{out}}) \quad (1 \leq t \leq N-1), \quad (4.7)$$

where  $\text{soft}$  denotes the softmax activation function.  $y_0$  is the first symbol indicating the beginning of the sentence, hence the  $\langle \text{BOS} \rangle$  tag.

## 4.2.2 Action Variational Autoencoder

The action VAE accepts a sequence of joint angle values and visual features as input; it is responsible for reconstructing the joint angle values. Similar to the language VAE, it is composed of an encoder, a decoder and latent layers (in the bottleneck) where latent representations are extracted via sampling. The action encoder encodes a sequence of length  $M$ ,  $((j_1, v_1), (j_2, v_2), \dots, (j_M, v_M))$ , which includes concatenation of joint angles  $j$  and visual features  $v$ . Note that the visual features are extracted by the channel-separated CAE beforehand. The equations that define the action encoder are as follows<sup>3</sup>:

$$h_t^{\text{enc}}, c_t^{\text{enc}} = \text{LSTM}(v_t, j_t, h_{t-1}^{\text{enc}}, c_{t-1}^{\text{enc}}) \quad (1 \leq t \leq M), \quad (4.8)$$

$$z_{\text{mean}} = W_{\text{mean}}^{\text{enc}} \cdot h_M + b_{\text{mean}}^{\text{enc}}, \quad (4.9)$$

$$z_{\text{var}} = W_{\text{var}}^{\text{enc}} \cdot h_M + b_{\text{var}}^{\text{enc}}, \quad (4.10)$$

$$z_{\text{act}} = z_{\text{mean}} + z_{\text{var}} \cdot \mathcal{N}(\mu, \sigma^2), \quad (4.11)$$

where  $h_t$  and  $c_t$  are the hidden and cell state of the LSTM at time step  $t$  respectively, and  $\mathcal{N}$  is a Gaussian distribution with mean  $\mu$  and standard deviation  $\sigma$ .  $h_0$  and  $c_0$  are set as zero vectors, while  $\mu$  and  $\sigma$  are set as 0 and 0.1 respectively.  $z_{\text{act}}$  is the latent representation of a robot action.

The action decoder reconstructs the joint angles:

$$h_0^{\text{dec}}, c_0^{\text{dec}} = W^{\text{dec}} \cdot z_{\text{act}} + b^{\text{dec}}, \quad (4.12)$$

$$h_t^{\text{dec}}, c_t^{\text{dec}} = \text{LSTM}(v_t, \hat{j}_t, h_{t-1}^{\text{dec}}, c_{t-1}^{\text{dec}}) \quad (1 \leq t \leq M-1), \quad (4.13)$$

$$\hat{j}_{t+1} = \tanh(W^{\text{out}} \cdot h_t^{\text{dec}} + b^{\text{out}}) \quad (1 \leq t \leq M-1), \quad (4.14)$$

where  $\tanh$  denotes the hyperbolic tangent activation function and  $\hat{j}_1$  is equal to  $j_1$ , i.e. joint angle values at the initial time step.

---

<sup>3</sup>To ensure clarity, we use primarily the same symbols in the equations as those used in the language VAE equations.



### 4.2.3 Visual Feature Extraction

Following Yamada et al. [185], we utilise a convolutional autoencoder architecture to extract the visual features of the images. Different from the approach used by Yamada et al. [185], we change the number of input channels the model accepts from three to one and train an instance of CAE for each colour channel (red, green and blue) to recognise different colours more accurately, which we define as channel separation. Therefore, we call our visual perception model the channel-separated CAE (CS-CAE). The idea behind the channel-separated CAE is similar to depth-wise separable convolutions [31], where completely separating cross-channel convolutions from spatial convolutions leads to better results in image classification as the network parameters are used more efficiently. The CS-CAE accepts a colour channel of  $120 \times 160$  RGB images captured by the cameras in the eyes of NICO – referred to also as the egocentric view of the robot – at a time. As can be seen in detail in Table 4.1, it consists of a convolutional encoder, a fully connected bottleneck, which incorporates hidden representations, and a deconvolutional decoder. After training for each colour channel, we extract the visual features of each image for every channel from the middle layer in the bottleneck (FC 3). The visual features extracted from each channel are then concatenated to constitute the ultimate visual features  $v$ .

Table 4.1: Detailed Architecture of Channel-Separated Convolutional Autoencoder

Block	Layer	Out. Channels	Kernel Size	Stride	Padding	Activation
Encoder	Conv. 1	8	4x4	2	1	ReLU
	Conv. 2	16	4x4	2	1	ReLU
	Conv. 3	32	4x4	2	1	ReLU
	Conv. 4	64	8x8	5	2	ReLU
Bottleneck	FC 1	384	-	-	-	-
	FC 2	192	-	-	-	-
	FC 3	10	-	-	-	-
	FC 4	192	-	-	-	-
	FC 5	384	-	-	-	-
Decoder	Deconv. 1	32	8x8	5	2	ReLU
	Deconv. 2	16	4x4	2	1	ReLU
	Deconv. 3	8	4x4	2	1	ReLU
	Deconv. 4	1	4x4	2	1	Sigmoid

Channel separation increases the use of computational resources compared to the standard convolution approach because it essentially uses three separate models. Although these models are identical, they do not share weights. The number of model parameters is about three times that of the standard approach. Therefore, it requires roughly three times more computational power than the standard

approach. Nonetheless, channel separation excels at distinguishing the colours of objects as evidenced by our experiments.

#### 4.2.4 Sampling and Binding

Stochastic Gradient Variational Bayes-based sampling (SGVB) [90] enables one-to-many mapping between action and language. The two VAEs have identical random sampling procedures. After producing the latent variables,  $z_{\text{mean}}$  and  $z_{\text{var}}$ , via the fully connected layers, we utilise a normal distribution,  $\mathcal{N}(\mu, \sigma^2)$ , to derive random values,  $\epsilon$ , which are in turn used with  $z_{\text{mean}}$  and  $z_{\text{var}}$  to arrive at the latent representation  $z$ , which is also known as the reparameterisation trick [90]:

$$z = z_{\text{mean}} + z_{\text{var}} \cdot \epsilon, \quad (4.15)$$

where  $\epsilon$  is the approximation of  $\mathcal{N}(\mu, \sigma^2)$  with the mean,  $\mu$ , set to 0 and the variance,  $\sigma^2$ , set to 0.01.

As in the case of PRAE [185], to align the latent representations of robot actions and their descriptions, we use an extra loss term that brings the mean hidden features,  $z_{\text{mean}}$ , of the two VAEs closer to each other. This enables bidirectional translation between action and language, i.e. the network can transform actions into descriptions as well as descriptions into actions, after training without an explicit fusion of the two modalities. This loss term, i.e. binding loss, can be calculated as follows:

$$L_{\text{binding}} = \sum_i^B \psi(z_{\text{mean}_i}^{\text{lang}}, z_{\text{mean}_i}^{\text{act}}) + \sum_i^B \sum_{j \neq i} \max \left\{ 0, \Delta + \psi(z_{\text{mean}_i}^{\text{lang}}, z_{\text{mean}_j}^{\text{act}}) - \psi(z_{\text{mean}_j}^{\text{lang}}, z_{\text{mean}_i}^{\text{act}}) \right\}, \quad (4.16)$$

where  $B$  stands for the batch size and  $\psi$  is the Euclidean distance. The first term in the equation binds the paired instructions and actions, whereas the second term separates unpaired actions and descriptions. Hyperparameter  $\Delta$  is used to adjust the separation margin for the second term – the higher it is, the further apart the unpaired actions and descriptions are pushed in the latent space.

#### 4.2.5 Loss Function

The overall loss is calculated as the sum of the reconstruction, regularisation and binding losses. The binding loss is calculated for both VAEs jointly. In contrast, the reconstruction and regularisation losses are calculated independently for each VAE. Following PRAE [185], the reconstruction losses for the language VAE (cross entropy between input and output words) and action VAE (Euclidean distance



between original and generated joint values) are  $L_{\text{lang}}$  and  $L_{\text{act}}$  respectively:

$$L_{\text{lang}} = \frac{1}{N-1} \sum_{t=1}^{N-1} \left( - \sum_{i=0}^{V-1} x_{t+1}^{[i]} \log y_t^{[i]} \right), \quad (4.17)$$

$$L_{\text{act}} = \frac{1}{M-1} \sum_{t=1}^{M-1} \|j_{t+1} - \hat{j}_{t+1}\|_2^2, \quad (4.18)$$

where  $V$  is the vocabulary size,  $N$  is the number of words per description and  $M$  is the sequence length for an action trajectory. The regularisation loss is specific to variational autoencoders; it is defined as the Kullback-Leibler divergence for language  $D_{\text{KL}_{\text{lang}}}$  and action  $D_{\text{KL}_{\text{act}}}$ . Therefore, the overall loss function is as follows:

$$L_{\text{all}} = \alpha L_{\text{lang}} + \beta L_{\text{act}} + \gamma L_{\text{binding}} + \alpha D_{\text{KL}_{\text{lang}}} + \beta D_{\text{KL}_{\text{act}}}, \quad (4.19)$$

where  $\alpha$ ,  $\beta$  and  $\gamma$  are weighting factors for different terms in the loss function. In our experiments,  $\alpha$  and  $\beta$  are set to 1, while  $\gamma$  is set to 2 in order to bind the two modalities effectively.

### 4.2.6 Transformer-Based Language Encoder

For the model to understand unconstrained language input from non-expert human users, we replace the LSTM for the language encoder with a pre-trained BERT<sub>BASE</sub> language model [35] – see Figure 4.2. According to Devlin et al. [35], BERT is pre-trained with the BooksCorpus, which involves 800 million words, and English Wikipedia, which involves 2.5 billion words. With the introduction of BERT as the language encoder, we assume that PVAE-BERT can interpret action descriptions correctly in our scenario. However, since language models like BERT are pre-trained exclusively on textual data from the internet, they are not specialised for object manipulation environments like ours. Therefore, the embedding of an instruction like ‘push the blue object’ may not differ from the embedding of another such as ‘push the red object’ significantly. For this reason, we fine-tune the pre-trained BERT<sub>BASE</sub>, i.e. all of BERT’s parameters are updated, during the end-to-end training of PVAE-BERT so that it can separate similar instructions from each other, which is critical to our scenario.

### 4.2.7 Training Details

To train PVAE and PVAE-BERT, we first extract visual features using our CS-CAE. The visual features are used to condition the actions depending on the cube arrangement, i.e. the execution of a description depends also on the position of the target cube. For both PVAE and PVAE-BERT, the action encoder and action decoder are each a two-layer LSTM with a hidden size of 100, while the language decoder is a single-layer LSTM with the same hidden size. In contrast, the language encoder of PVAE-BERT is the pre-trained BERT<sub>BASE</sub> model with 12 layers, each with 12 self-attention heads and a hidden size of 768, whereas the language encoder

of PVAE is a one-layer LSTM with a hidden size of 100. Both PVAE and PVAE-BERT are trained end-to-end with both the language and action VAEs together. PVAE and PVAE-BERT are trained for 20,000 and 40,000 iterations respectively, with the gradient descent algorithm and Adam optimiser [89]. We take the learning rate as  $10^{-4}$  with a batch size of 100 pairs of language and action sequences after a few trials with different learning rates and batch sizes. Due to having approximately 110M parameters, compared with PVAE’s approximately 465K parameters, an iteration of PVAE-BERT training takes about 1.4 times longer than an iteration of PVAE training. Therefore, it takes about 2.8 times longer to train PVAE-BERT in total.

### 4.3 Experiments and Results

We evaluate the performance of our PVAE and its variant using BERT, namely PVAE-BERT, with multiple experiments. First, we compare the original PVAE with PRAE [185] in terms of action-to-language translation by conducting experiments of varying object colour options to display the superiority of variational autoencoders over regular autoencoders and the advantage of using the channel separation technique in visual feature extraction. Different object colour possibilities correspond to a different corpus and overall dataset size; the more object colour options there are, the larger both the vocabulary and the overall dataset become. Therefore, with these experiments, we also test the scalability of both approaches. In order to show the impact of channel separation on the action-to-language translation performance, we train our architecture with visual features provided by a regular CAE (no channel separation) as implemented by Yamada et al. [185]. Overall, these experiments are **Experiment 1a** (with 3 cube colour alternatives: red, green, blue) and **Experiment 1b** (with 6 cube colour alternatives: red, green, blue, yellow, cyan, violet) – see Table 4.3.

Moreover, in **Experiment 2**, we train PVAE-BERT on the dataset with 6 colour alternatives (red, green, blue, yellow, cyan, violet) to compare it with the standard PVAE by conducting action-to-language, language-to-action and language-to-language evaluation experiments. Unlike machine translation which requires translating from one natural language to another, in our language-to-language translation experiments, the model should produce the ground-truth language description when given the same description or one of its variations (e.g. different word order) as input. Experiment 2 uses the pre-trained BERT as the language encoder which is then fine-tuned with the rest of the model during training.

In Experiments 1a, 1b and 2, two cubes of different colours are placed on a table at which the robot is seated to interact with them. The words (vocabulary) that constitute the descriptions are given in Table 4.2. We introduce a more diverse vocabulary by adding an alternative word for each word in the original vocabulary. As descriptions are composed of 3 words with two alternatives per word, we arrive at 8 variations for each description of a given meaning. Table 4.2 does not include

Table 4.2: Vocabulary with Original and Alternative Words

Component	Original	Alternative
Verb	push	move-up
	pull	move-down
	slide	move-sideways
Colour	red	scarlet
	green	harlequin
	blue	azure
	yellow	blonde
	cyan	greenish-blue
	violet	purple
Speed	slowly	unhurriedly
	fast	quickly

nouns because we use a predefined grammar, which does not involve a noun, and the same-sized cubes for these experiments.

For each cube arrangement, the colours of the two cubes always differ to avoid ambiguities in the language description. Actions, which are transcribed in capitals, are composed of any of the three action types PUSH, PULL, SLIDE, two positions LEFT, RIGHT and two speed settings SLOWLY, FAST, resulting in 12 possible actions (3 action types  $\times$  2 positions  $\times$  2 speeds), e.g. PUSH-LEFT-SLOWLY means pushing the left object slowly. Every sentence is composed of three words (excluding the <BOS/EOS> tags which denote *beginning of sentence* or *end of sentence*) with the first word indicating the action, the second the cube colour and the last the speed at which the action is performed (e.g. ‘push green slowly’). Therefore, without the alternative words, there are 18 possible sentences (3 action verbs  $\times$  3 colours  $\times$  2 adverbs) for Experiment 1a, whereas for Experiment 1b and 2, the number of sentences is 36 as 6 cube colours are used in both experiments. As a result, our dataset consists of 6 cube arrangements (3 colour alternatives and the colours of the two cubes on the table never match) for Experiment 1a, 12 cube arrangements for Experiments 1b and 2 (3 secondary colours are used in addition to 3 primary colours, and secondary and primary colours are mutually exclusive),  $18 \times 8 = 144$  possible sentences for Experiment 1a,  $36 \times 8 = 288$  possible sentences for Experiments 1b and 2 with alternative vocabulary (see Table 4.2) – the factor of 8 because of eight alternatives per sentence. We have 72 patterns (action-description-arrangement combinations) for Experiment 1a (12 actions with six cube arrangements each) and 144 patterns for Experiments 1b and 2.

As in Section 3.3, we choose the patterns rigorously to ensure that the combinations of action, description and cube arrangements used in the test set are

excluded from the training set, while the training set includes all possible combinations of action, description and cube arrangements that are not in the test set. For Experiment 1a, 54 patterns are used for training while the remaining 18 for testing, whereas for Experiments 1b and 2, 108 patterns are for training, 36 for testing. Each pattern is collected six times in the simulation environment with random variations on the action execution resulting in different joint trajectories. We also use 4-fold cross-validation to provide more reliable results (see Table 4.3) for Experiment 1.

**Experiment 1c** tests for different shapes, other than cubes: we perform the same actions on toy objects, which are a car, duck, cup, glass, house and lego brick. For testing the shape processing capability of the model, all objects are of the same colour, namely yellow. Analogous to the other experiments, two objects of different shapes are placed on the table. We keep the actions as they are but replace the colours with object names in the descriptions. Before we extract the visual features from the new images, we train both the regular CAE and the channel-separated CAE with them. Similar to Experiments 1a and 1b, we experiment with three methods: PRAE with standard CAE, PVAE with standard CAE and PVAE with channel-separated CAE.

We use NICO in a virtual environment created with Blender<sup>4</sup> for our experiments (see Figure 4.1). NICO is a humanoid robot that has a height of approximately one metre and a weight of about 20 kg. The left arm of NICO is used to interact with the objects while utilising 5 joints. Actions are realised using the inverse kinematics solver provided by the simulation environment: for each action, first, the starting point and endpoint are adjusted manually, then, the Gaussian deviation is applied around the starting point and endpoint to generate the variations of the action, ensuring that there is a slight difference in the overall trajectory. NICO has a camera in each of its eyes, which is used to extract egocentric visual images.

### 4.3.1 Experiment 1: Action-to-Language Translation with Different Colours and Shapes

We use the same actions as those used by Yamada et al. [185], such as PUSH-RIGHT-SLOWLY. We use three colour options for the cubes as in the PRAE paper [185] for Experiment 1a, but six colours for Experiment 1b. In addition, we extend the descriptions in the PRAE paper [185] by adding an alternative for each word in the original vocabulary. Hence, the vocabulary size of 9 is extended to 17 for Experiment 1a and the vocabulary size of 11 is extended to 23 for Experiment 1b – note that we do not add an alternative for <BOS/EOS> tags. Since every sentence consists of three words, we extend the number of sentences by a factor of eight ( $2^3 = 8$ ).

After training PVAE and PRAE on the same training set, we test them for action-to-language translation. We consider only those produced descriptions in

---

<sup>4</sup><https://www.blender.org/>

Table 4.3: Action-to-Language Translation Accuracies at Sentence Level

Method	Experiment 1a (3 colours)		Experiment 1b (6 colours)		Experiment 1c (6 shapes)	
	Training	Test	Training	Test	Training	Test
PRAE + reg. CAE	33.33 ± 1.31%	33.56 ± 3.03%	33.64 ± 1.13%	33.33 ± 0.98%	68.36 ± 2.12%	65.28 ± 2.45%
PVAE + reg. CAE	66.66 ± 1.31%	65.28 ± 6.05%	69.60 ± 0.46%	61.57 ± 2.01%	80.71 ± 1.41%	73.15 ± 1.87%
PVAE + CS-CAE	100.00 ± 0.0%	90.28 ± 4.61%	100.0 ± 0.0%	100.0 ± 0.0%	95.99 ± 3.74%	92.13 ± 2.83%

which all three words and the <EOS> tag are correctly predicted as correct. The produced descriptions with one or more incorrect words are considered false translations. As each description has seven more alternatives, predicting any of the eight description alternatives is considered correct.

For Experiment 1a, our model is able to translate approximately 90% of the patterns in the test set, while PRAE could translate only one third of the patterns, as can be seen in Table 4.3. We can thus say that our model outperforms PRAE in one-to-many mapping. We also test the impact of channel separation on the translation accuracy by training our model with visual features extracted with the regular CAE as described in Yamada et al.’s approach [185]. It is clearly indicated in Table 4.3 that using variational autoencoders instead of standard ones increases the accuracy significantly. Using PVAE with CS-CAE improves the results further, indicating the superiority of channel separation in our tabletop scenario. Therefore, our approach with variational autoencoders and a channel-separated CAE is superior to both PRAE and PVAE with regular visual feature extraction.

In Experiment 1b, in order to test the limits of our PVAE and the impact of more data with a larger corpus, we add three more colour options for the cubes: yellow, cyan and violet. These secondary colours are combined amongst themselves for the arrangements in addition to the colour combinations used in the first experiment, i.e. a cube of a primary colour and a cube of a secondary colour do not co-occur. Therefore, this experiment has 12 arrangements. Moreover, the vocabulary size is extended to 23 from 17 in Experiment 1b (two alternative words for each colour – see Table 4.2). As in Experiment 1a, each sentence has eight alternative ways to be described.

We train both PVAE and PRAE on the extended dataset from scratch and test both architectures. As shown in Table 4.3 (Experiment 1b), PVAE succeeds in performing 100% by translating every pattern from action to description cor-

Table 4.4: Sentence Translation Accuracies for PVAE and PVAE-BERT

Translation Direction	PVAE	PVAE-BERT
	Test Acc. (T - F)	Test Acc. (T - F)
Action→Language	100.00% (216 - 0)	97.22% (210 - 6)
Language→Language	100.00% (216 - 0)	80.56% (174 - 42)

rectly, even for the test set. In contrast, PRAE performs poorly in this setting and manages to translate only one third of the descriptions correctly in the test set. Compared with the accuracy values reached in the first experiment with less data and a smaller corpus, the extension of the dataset helps PVAE to perform better in translation, whereas PRAE is not able to take advantage of more data. Similar to Experiment 1a, we also test the influence of channel separation on translation accuracy by training PVAE with visual features provided by a regular CAE. In this setting, PVAE only achieves around 61% of accuracy on the test set. This highlights once again the importance of channel separation in visual feature extraction for our setup. While the improvement by using our PVAE over PRAE is significant, further improvement is made by utilising the channel-separated CAE.

In addition, as the results show in the last column of Table 4.3 (Experiment 1c), our PVAE with channel separation in visual feature extraction outperforms the other methods even when the manipulated objects have different shapes. Although there is a slight drop in action-language translation performance, it is clear that PVAE with the channel-separated CAE is able to handle differently shaped objects. The PRAE model performs considerably better than it does in the experiments with cubes of different colours. Nevertheless, our variational autoencoder approach without channel separation improves the translation accuracy by approximately 8%. The channel separation in visual feature extraction improves the results even more similar to Experiment 1a and Experiment 1b, which shows the robustness of the channel-separated CAE when processing different objects.

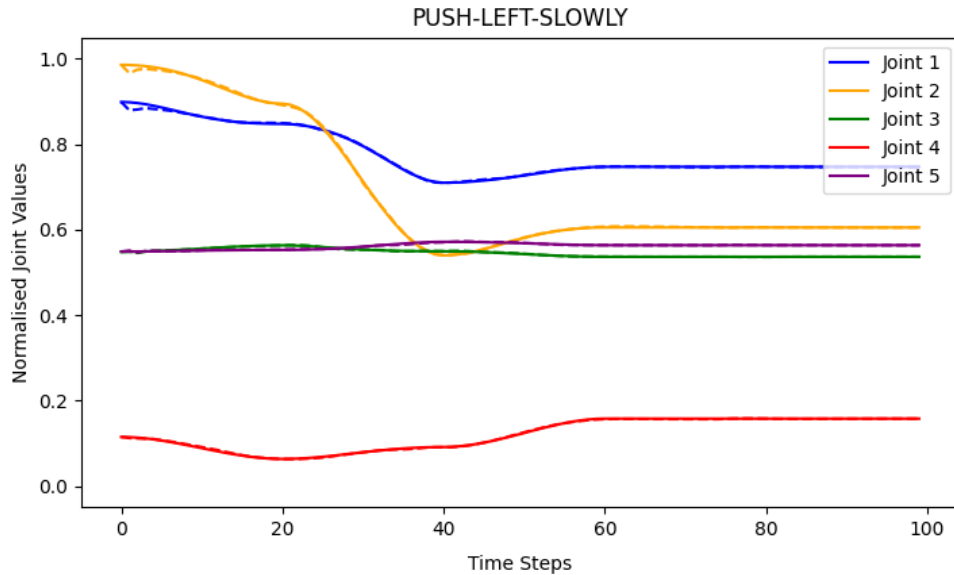


Figure 4.3: Joint angle trajectories for language-to-action translation by PVAE-BERT for the PUSH-LEFT-SLOWLY action. Solid lines show the ground truth, while the dashed lines, which are often covered by the solid lines, show the predicted joint angle values. In both plots, the X axis represents the time steps.

### 4.3.2 Experiment 2: Bidirectional Translations with BERT

In this experiment, we test the performance of PVAE-BERT on action-to-language, language-to-action and language-to-language translation. We use the same dataset as in Experiment 1b for a fair comparison with the original PVAE (LSTM language encoder). We thus use the same descriptions, which are constructed by using a verb, colour and speed from the vocabulary given in Table 4.2 as well as the <BOS/EOS> tags in the same order. Both PVAE and PVAE-BERT utilise CS-CAE-extracted visual features.

As shown in Table 4.4, when translating from action to language, PVAE-BERT achieves approximately 97% accuracy, failing to translate only six of the descriptions, comparable with the original architecture – the original PVAE correctly translates all 216 descriptions. The false translations are all due to incorrect translation of cube colours, e.g. the predicted description is ‘slide blue slowly’ instead of the ground truth ‘slide red slowly’. We hypothesise that the slight drop in the performance is due to the relatively small size of the dataset for almost 110 million parameters trained in the case of BERT. Nevertheless, these results show that fine-tuning BERT during training leads to almost perfect action-to-language translation in our scenario.

As can be seen in Figures 4.3 to 4.5, both PVAE and PVAE-BERT perform decently in language-to-action translation and produce joint angle values that are in line with and very similar to the original descriptions. In Figure 4.5, we can see that the joint trajectories output by PVAE-BERT are more accurate than those

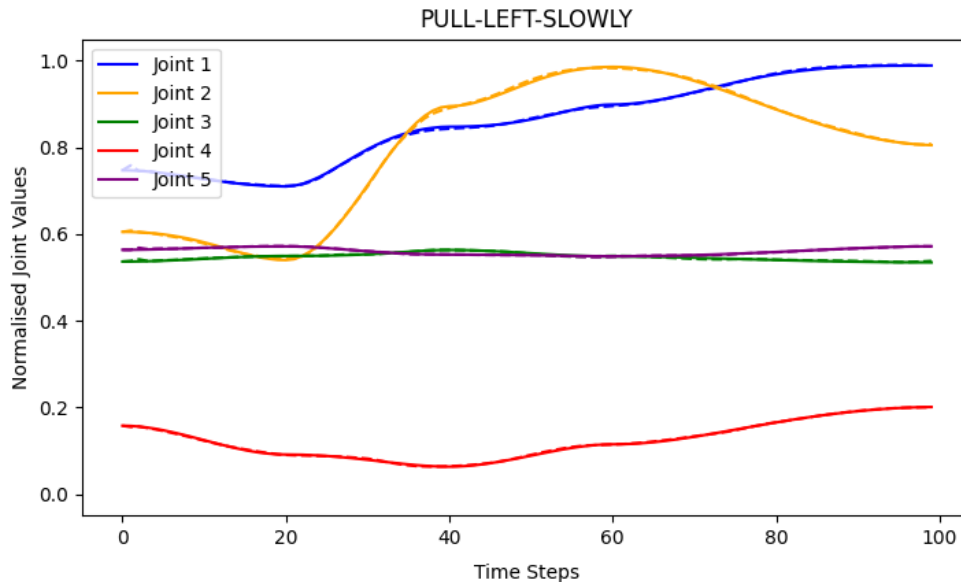


Figure 4.4: Joint angle trajectories for language-to-action translation by PVAE-BERT for the PULL-LEFT-SLOWLY action. Solid lines show the ground truth, while the dashed lines, which are often covered by the solid lines, show the predicted joint angle values. In both plots, the X axis represents the time steps.



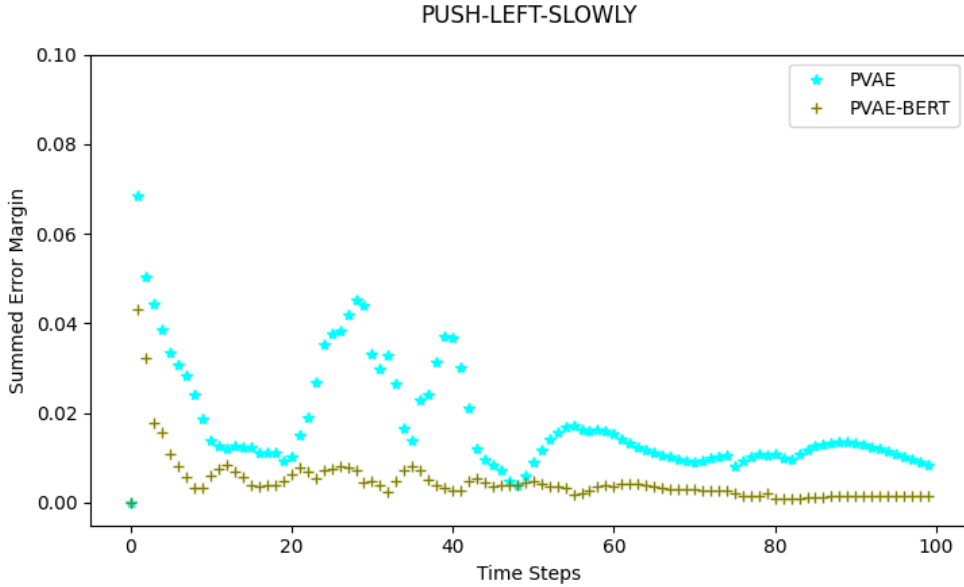


Figure 4.5: The total error margin of the five joint values produced by PVAE and PVAE-BERT for language-to-action translation per time step on the PUSH-LEFT-SLOWLY action. The X axis represents the time steps, while the Y axis shows the summed error margin.

produced by PVAE. We hypothesise that the error margins are negligible and both models succeed in language-to-action translation. Since we did not realise the actions with the generated joint values in the simulation, we do not report the language-to-action translation accuracies in Table 4.4. However, we calculated the mean squared errors (MSE) for both PVAE and PVAE-BERT, which were both very close to zero. Therefore, it is fair to say that both architectures recognise language and translate it to action successfully.

Language-to-language translation, however, suffers a bigger performance drop when BERT is used as the language encoder; PVAE-BERT reconstructs around 80% of the descriptions correctly (see Table 4.4). We hypothesise that this is partly due to having an asymmetric language autoencoder with a BERT encoder and an LSTM decoder. The BERT<sub>BASE</sub> language encoder constitutes the overwhelming majority of parameters in the PVAE-BERT model, which renders the language VAE heavily skewed to the encoder half. This may affect the performance of the language decoder when translating back to the description from the hidden code produced mainly by BERT as the decoder’s parameters constitute less than 1% of the parameters of the language VAE. This hypothesis is further supported by the original architecture, which has a symmetric language VAE, achieving 100% of accuracy on the same task.

Nevertheless, our findings show that the PVAE-BERT model achieves stable language-to-language translation performance even when the given descriptions do not comply with the fixed grammar and are full commands such as ‘push the yellow cube slowly’ or have a different word order such as ‘quickly push



Table 4.5: Variations of Descriptions for One Example and PVAE-BERT Language-to-Language Sentence Translation Accuracies

Var.	Type	Example	Acc.
1	Standard	‘push green slowly’	80.56%
2	Changed Word Order	‘slowly push green’	80.56%
3	Full Command	‘push the green cube slowly’	81.02%
4	‘please’+Full Command	‘please push the green cube slowly’	81.94%
5	Full Command+‘please’	‘push the green cube slowly please’	81.48%
6	Changed Word Order+ Full Command+‘please’	‘slowly push the green cube please’	81.48%
7	Polite Request	‘could you please push the green cube slowly?’	79.63%

yellow’. To turn predefined descriptions into full commands, we add the words ‘the’ and ‘cube’ to the descriptions and we also experiment with adding the word ‘please’ and changing the word order as can be seen in the examples given in Table 4.5. Although it is not explicitly stated in the table for space reasons, we

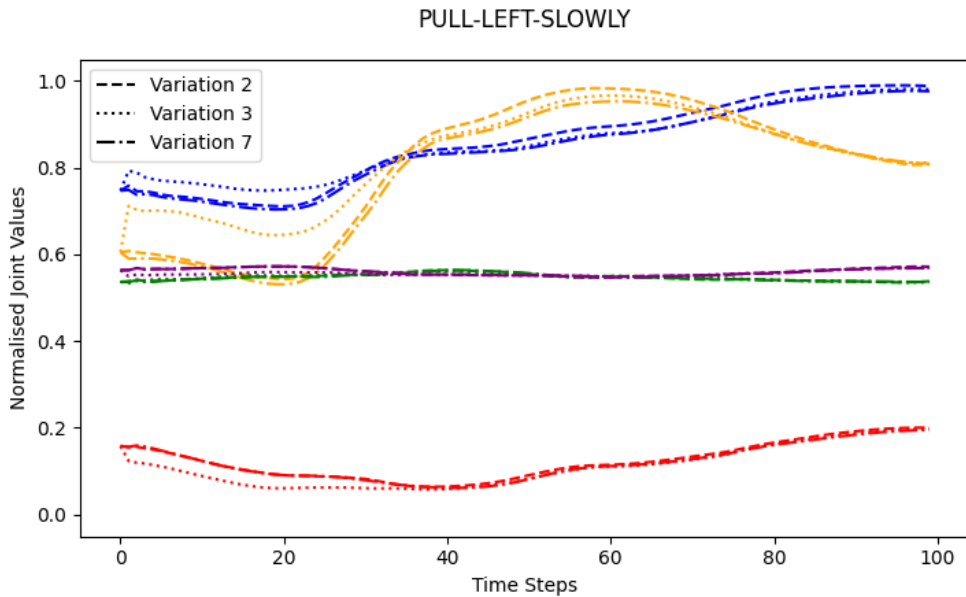


Figure 4.6: The joint values produced by PVAE-BERT given three variations (Table 4.5) of the same command for PULL-LEFT-SLOWLY – notice how the joint trajectories overlap most of the time. The X axis represents the time steps, while the Y axis shows the normalised joint values.

alternate between the main elements of the descriptions as in the other experiments following the vocabulary; for example, ‘push’ can be replaced by ‘move-up’ and ‘quickly’ can be replaced by ‘fast’. Moreover, we achieve consistent language-to-action translation performance with PVAE-BERT when we test it with different description types shown in the table as can be seen in Figure 4.6. As PVAE-BERT performs consistently even with descriptions not following the predefined grammar, we can see that adopting a language model to the architecture is promising towards acquiring natural language understanding skills.

### 4.3.3 Principal Component Analysis (PCA) on Hidden Representations

We also conduct a principal component analysis (PCA) on the hidden features extracted from PVAE-BERT. Figure 4.7 shows the latent representations of language in Plot (a) and of action in Plot (b). The PCA on the representations of language shows that the model learns the compositionality of language: the X-axis (principal component PC 1) distinguishes the descriptions in the speed component (adverb), the Y-axis (PC 3) distinguishes colour, and the Z-axis (PC 6) distinguishes the action type (verb)<sup>5</sup>. Plot (b) shows that the PCA representations of actions are semantically similar since their arrangement coincides with those in Plot (a).

Our method learns actions according to their paired descriptions: it learns the colour of the object (an element of descriptions) which is interacted with. However, it does not learn its position (an element of actions). We inspected the representations along all major principal components but could not find any direction along which the position was meaningfully distinguished. For example, in Plot (b), some of the filled red circles (corresponding to description ‘push red slowly’) are paired with the action PUSH-LEFT-SLOWLY while the others with PUSH-RIGHT-SLOWLY. As actions are learnt according to their paired descriptions, hence semantically, the filled red circles are grouped together even though the red cube may be on the right or left. In contrast, an action can be represented far from another identical action: e.g. the representations of ‘pull red slowly’ (filled red circles in Figure 4.7) are separated from those of ‘pull yellow slowly’ (filled yellow circles) along PC 3 even when they both denote the action PULL-LEFT-SLOWLY. These results indicate that the binding loss has transferred semantically driven ordering from the language to the action representations.

Even though our agent receives language instructions containing the colour but not the position information, the agent is still capable of acting based on the target object position (see Figures 4.3 to 4.6). Thus, the retrieval of the position information must be undertaken by the action decoder: it reads the images to obtain the position of the object that is of the colour given in the instruction. It is therefore not surprising that the PCA does not reveal any object position encodings in the bottleneck.

<sup>5</sup>The percentages of variance explained by PC 2 up to and including PC 6 were comparable; thus, we opted for PC 3 and PC 6 as they optimally distinguished object colour and action type.

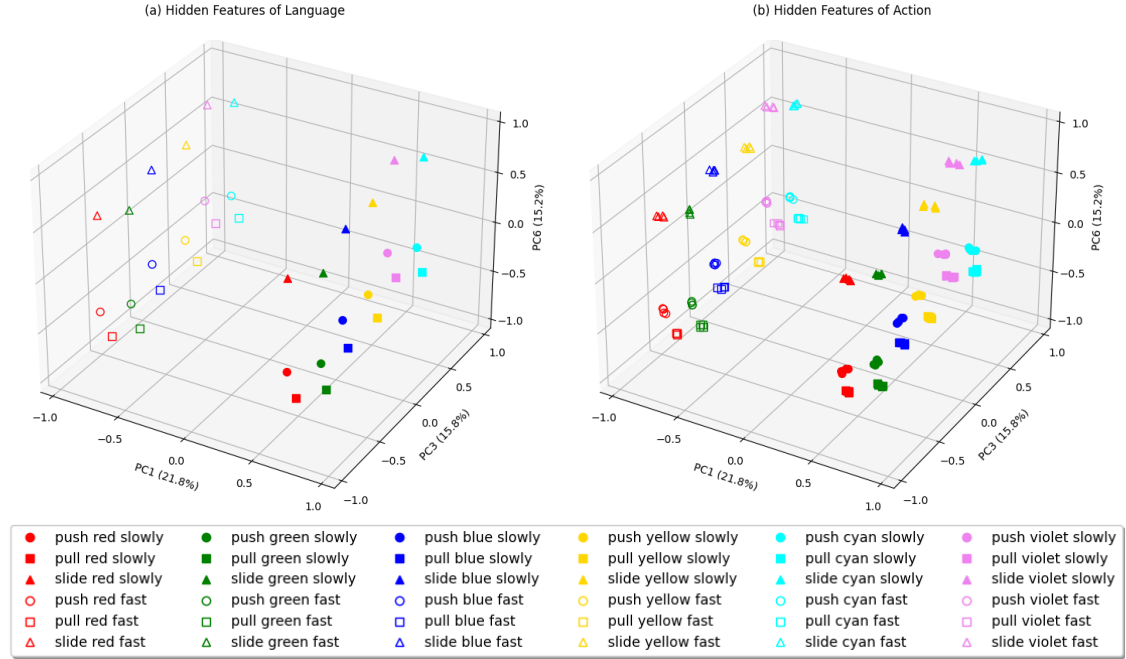


Figure 4.7: Hidden features of language (a) and hidden features of action (b). The PCA was performed jointly on the hidden features of 36 descriptions and the hidden features of 144 actions. For (b), each unique action (12 in total) occurs 12 times as there are 12 possible cube arrangements; therefore, 144 points are shown. For both (a) and (b), we label the points according to descriptions, i.e. for (b), actions are also labelled according to their paired descriptions. As can be seen from the legend, different shapes, colours and fillings indicate the verb (action type), object colour and adverb (speed) respectively.

## 4.4 Discussion

Experiments 1a and 1b show that our variational autoencoder approach with a channel-separated CAE visual feature extraction (‘PVAE + CS-CAE’) performs better than the standard autoencoder approach PRAE in the one-to-many translation of robot actions into language descriptions. Our approach is superior both in the case of three colour alternatives per cube and in the case of six colour alternatives per cube by a large margin. The additional experiment with six different objects highlights the robustness of our approach against the variation in object types. We demonstrate that a Bayesian inference-based method like variational autoencoders can scale up with more data for generalisation, whereas standard autoencoders cannot capitalise on a larger dataset, since the proposed PVAE model achieves better accuracy when the dataset and the corpus are extended with three extra colours or six different objects. Additionally, standard autoencoders are fairly limited in coping with the diversification of language as they do not have the capacity to learn the mapping between an action and its multiple descriptions. In contrast, variational autoencoders yield remarkably better results in one-to-many

translation between actions and descriptions, because stochastic generation (random normal distribution) within the latent feature extraction allows latent representations to slightly vary, which leads to VAEs learning multiple descriptions rather than a particular description for each action.

A closer look into action-to-language translation accuracies achieved by PRAE for Experiments 1a and 1b shows that having more variety in the data (i.e. more colour options for cubes) does not help the standard autoencoder approach to learn one-to-many binding between action and language. Both in the first case with three colour alternatives and in the second case with six colour alternatives, PRAE manages to translate only around one third of the samples from actions to descriptions correctly. In contrast, the accuracies achieved by our proposed PVAE for both datasets prove that the variational autoencoder approach can benefit from more data as the test accuracy for the ‘PVAE + CS-CAE’ goes up by approximately 10% to 100% when three more colour options are added to the dataset.

Furthermore, training PVAE with the visual features extracted by the standard CAE demonstrates that training and extracting features from each RGB channel separately mitigates the colour distinction issue for cubes when the visual input, like in our setup, includes objects covering a relatively small portion of the visual field. The ‘PVAE + regular CAE’ variant performs significantly worse than our ‘PVAE + CS-CAE’ approach. This also demonstrates the importance of the visual modality for the overall performance of the approach. Our analysis on the incorrectly translated descriptions shows that a large amount of all errors committed by the ‘PVAE + regular CAE’ were caused due to cube colour distinction failures such as translating ‘slide red fast’ as ‘slide green fast’, which proves CS-CAE’s superiority over the standard CAE in visual feature extraction in our scenario. Moreover, using CS-CAE for visual feature extraction rather than the standard CAE results in better action-to-language translation accuracy even when the objects are of various shapes. This indicates that CS-CAE not only works well with cubes of different colours but also with objects of different shapes. We emphasise the superiority of channel separation in our scenario, which has been tested and proven in a simulation environment. For real-world scenarios with different lighting conditions, it is advisable to take into account also the channel interaction [170] to have a more robust visual feature extraction.

Experiment 2 indicates the potential of utilising a pre-trained language model like BERT for the interpretation of language descriptions. This extension produces comparable results to the original PVAE with the LSTM language encoder in language-to-action and action-to-language translations. The drop in language-to-language performance to 80% is most probably caused by the asymmetric language VAE of the PVAE-BERT model that consists of a feedforward BERT encoder with attention mechanisms, which reads the entire input sequence in parallel, and of a recurrent LSTM decoder, which produces the output sequentially. A previous study on a text classification task also shows that LSTM models outperform BERT on a relatively small corpus because, with its large number of parameters, BERT tends to overfit when the dataset size is small [44]. Furthermore, we have also tested PVAE-BERT, which was trained on predefined descriptions, with full sentence de-

descriptions – e.g. ‘push the green cube slowly’ for ‘push green slowly’ – and with variations of the descriptions that have a different word order. We have confirmed that PVAE-BERT achieves the same performance in language-to-action and language-to-language translations. The pre-trained BERT allows the model to understand unconstrained natural language commands that do not conform to the defined grammar.

The PCA conducted on the hidden features of PVAE-BERT shows that our method can learn language and robot actions compositionally and semantically. Although it is not explicitly given, we have also confirmed that both PVAE and PVAE-BERT can reconstruct joint values with almost perfect accuracy when we analysed the action-to-action translation results. Together with the language-to-language performance, the action-to-action capability of both variants of our architecture demonstrates that the two variational autoencoders (language and action) in our approach retain their reconstructive nature.

## 4.5 Conclusion

In this chapter, we have reported the findings of the previous chapter and its extension with several experiments. We have shown that variational autoencoders outperform standard autoencoders in terms of one-to-many translation of robot actions to descriptions. Furthermore, the superiority of our channel-separated visual feature extraction has been proven with an extra experiment that involves different types of objects. In addition, using PVAE with a BERT model pre-trained on large text corpora, instead of the LSTM encoder trained on our small predefined grammar, unveils promising scaling-up opportunities for the proposed approach, and it offers the possibility to map unconstrained natural language descriptions with actions.

In the next chapter, we will present the PGAE architecture which overcomes the need to adjust the model architecture configuration according to the desired translation direction during inference. PGAE utilises a flexible multimodal fusion technique to explicitly bind the language and action streams in a more biologically plausible way. It also uses signal word prefixes as part of language input to direct the agent in the desired translation. Furthermore, it is capable of recognising and imitating the actions of a mirrored agent via robot demonstrations.



# Learning Flexible Translation Between Action and Language through Explicit Multimodal Fusion

---

Handling various robot action-language translation tasks flexibly is an essential requirement for natural interaction between a robot and a human. Previous approaches (Chapters 3 and 4) require a change in the configuration of the model architecture per task during inference, which undermines the premise of multi-task learning. In this chapter, we propose the Paired Gated Autoencoders (PGAE) architecture for flexible translation between robot actions and language descriptions in a tabletop object manipulation scenario. We train our model in an end-to-end fashion by pairing each action with appropriate descriptions and adding a signal in the language input informing about the translation direction. During inference, our model can flexibly translate from action to language and vice versa according to the given language signal. Moreover, our model can recognise and imitate the actions of a second agent by utilising robot demonstrations. The experiment results highlight the flexible bidirectional translation capabilities of our approach as well as the ability to generalise to the actions of the opposite-sitting agent<sup>1</sup>.

## 5.1 Introduction

Learning language involves multiple modalities such as audio, vision and proprioception. For example, a colour word refers to a visual concept; sensing the weight of an object is related to the concept of force; the concept of position such as left and right can be learnt with proprioception and vision. More modalities can be enumerated that help with learning language but the essential component of language learning pertains to embodiment, i.e. having a body and interacting in

---

<sup>1</sup>The code belonging to this chapter is available at <https://github.com/oo222bs/PGAE>.

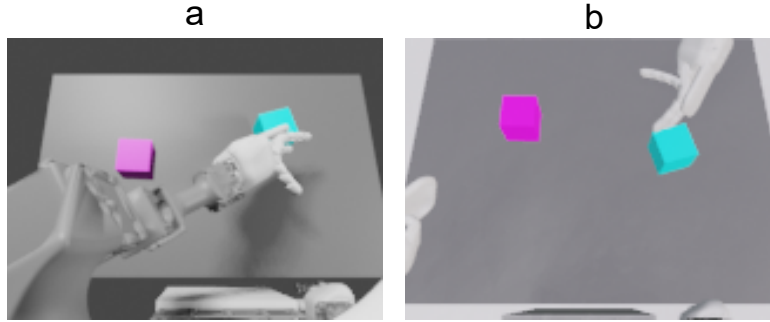


Figure 5.1: Our object manipulation scenario: a) an example action (‘push cyan slowly’) performed by NICO, b) the same action performed by the opposite-sitting agent as seen by NICO.

the environment [19]. The embodied language learning or language grounding has recently been a topic of interest at the crossroads between natural language processing and robotics [1, 19, 107, 161]. Inspired by the early language development in children where interactions in the environment are paired with language, language grounding approaches have made strides in representation learning, abstraction forming, sequence-to-sequence learning and bidirectional action-language translations. However, these approaches are not designed to endow a robot with the autonomy to understand and choose the appropriate action to carry out during an interaction with a human. They are either designed to carry out a single task<sup>2</sup> such as recognising an instruction and executing it [61, 107, 158, 161] or they can handle multiple tasks but they require the task mode in advance to know what is expected of them [128, 136, 185]. In contrast, a truly autonomous agent must be able to decide whether to produce language or execute an action according to the verbal instruction provided by its human partner. Therefore, end-to-end multimodal and multi-task models which do not require adaptation to new tasks by the experimenter are desired.

In this chapter, we address the problem of flexible bidirectional translation between robot actions and language. We define flexibility as translating between the two modalities without having to reconfigure the model for a specific task during inference. According to our scenario (Figure 5.1), we expect our agent to flexibly translate from action to language and vice versa – given textual descriptions, joint angle values and visual features, our agent must either manipulate an object or describe the object manipulation act carried out by itself or a second agent depending on the situation. To this end, we introduce the PGAE architecture for flexible translation between robot actions and language, realised by our humanoid robot NICO in the simulation environment. PGAE includes an attention mechanism in its bottleneck, which allows the model to directly exchange information between the action and language modalities. The attention mechanism, which is adapted

<sup>2</sup>Note that the term task in this chapter refers to different directions of action-language translation.



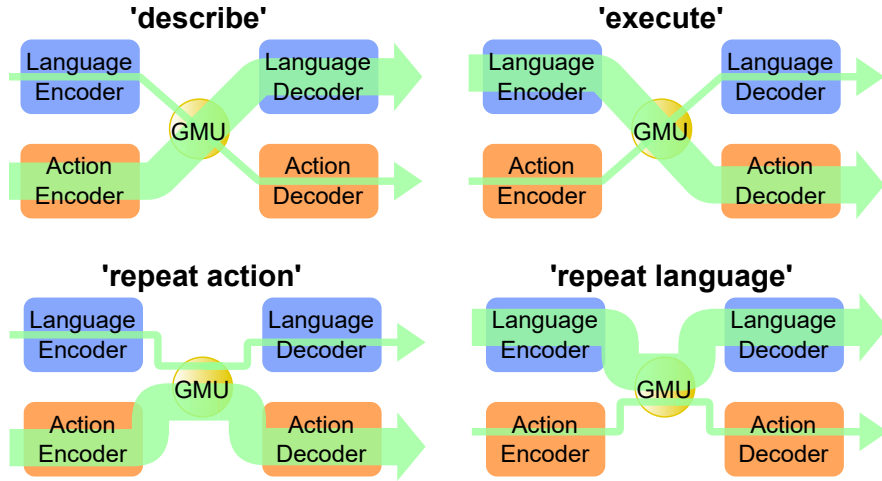


Figure 5.2: The schematic architecture with respect to the four different tasks: ‘describe’, ‘execute’, ‘repeat action’ and ‘repeat language’. The PGAE architecture consists of encoders, decoders and a GMU bottleneck. Each panel shows the architecture with the information flow (indicated by the green arrows) per task. The thick arrows denote the main information flow in a given task.

from the gated multimodal units (GMU) [13], acts as a filter to pick information between the modalities across all dimensions. Additionally, we signal the task by prepending a phrase to the language input and ensure that our model recognises the task and is trained accordingly. Thus, during inference, our model is able to perform each of the translation tasks (Figure 5.2) according to user input without having to configure the model in advance. Furthermore, we test a realistic setup in which the NICO robot can describe and repeat the actions of the opposite-sitting second agent. Our experiment results show that PGAE performs competitively in terms of translations between language and action with the previous approach PVAE that implicitly binds the two modalities and, in turn, could use only some parts of the network, which is set a priori according to the given translation task.

Our contributions to the field, with the introduction of PGAE, can be summarised as:

1. introducing an end-to-end neural network (NN) architecture that can flexibly handle multiple action-language translation tasks during inference, consistent with the training conditions,
2. enabling the robot to recognise and imitate both the self-performed actions and the actions of an opposite-sitting agent.

The remainder of Chapter 5 is structured as follows: In the next section, we describe the PGAE architecture in detail, including the signals used during training. The subsequent section involves the experiments and their results in all translation tasks. Finally, we conclude the chapter with a summary and hints about the next chapter.

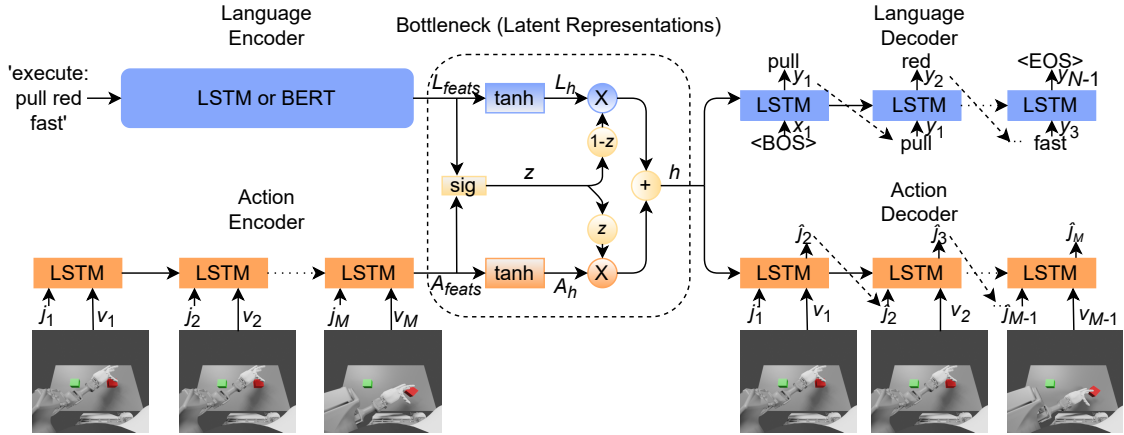


Figure 5.3: The architecture of the PGAE model. The language encoder is either an LSTM or the BERT model. The action encoder and both decoders are LSTMs – we show unfolded versions of the LSTMs.  $j$  and  $v$  denote joint angle values and visual features respectively.  $y$  is the language output which is produced one word at a time. The bottleneck, where the two streams are connected, is based on the GMU network;  $z$  is the gating vector, whilst  $h$  is the shared representation vector.

## 5.2 Proposed Method: Paired Gated Autoencoders (PGAE)

Similar to the models introduced in [Chapters 3 and 4](#) (PVAE and PVAE-BERT), PGAE is a bidirectional translation model between robot actions and language descriptions. As can be seen in [Figure 5.3](#), PGAE consists of two autoencoders, namely language and action. It is intended to associate simple robot actions like pushing a cube on the table with their corresponding language descriptions. PGAE accepts as input language descriptions, visual features extracted from images and joint angle values. PGAE outputs language descriptions and joint angle values conditioned on visual features. Moreover, PGAE is trained end-to-end with a signal prepended to the language input indicating the expected output of the training iteration. Five different signals are randomly chosen during training at each iteration: ‘describe’, ‘execute’, ‘repeat action’, ‘repeat both’ and ‘repeat language’. These signals direct the agent in the desired translation direction without reconfiguring the model during inference.

### 5.2.1 Language Autoencoder

The language autoencoder accepts as input one-hot encoded words of a description (or the whole description at once when BERT is used as the language encoder) and produces a description by outputting a word at each time step. The language AE has an encoder, a decoder and hidden layers (in the bottleneck) that contribute to the common hidden representations. The language encoder embeds a description

of length  $N + 1$ ,  $(x_1, x_2, \dots, x_{N+1})$ , including the signal, into the final state,  $f_{N+1}$ , as follows:

$$h_t^{\text{enc}}, c_t^{\text{enc}} = \text{LSTM}(x_t, h_{t-1}^{\text{enc}}, c_{t-1}^{\text{enc}}) \quad (1 \leq t \leq N + 1), \quad (5.1)$$

$$f_{N+1}^{\text{enc}} = [h_{N+1}^{\text{enc}}; c_{N+1}^{\text{enc}}], \quad (5.2)$$

where  $h_t^{\text{enc}}$  and  $c_t^{\text{enc}}$  are the hidden and cell state of the LSTM at time step  $t$  respectively.  $h_0^{\text{enc}}$  and  $c_0^{\text{enc}}$  are set as zero vectors, whereas  $x_1$  is the signal word. The square brackets  $[.;.]$  denote the concatenation operation. The LSTMs we use here and the action encoder and both decoders are peephole LSTMs [152], following the PRAE [185] and PVAE architectures (Chapter 3). The language encoder LSTM can also be replaced by a pre-trained language model to recognise unconstrained user instructions. Specifically, we use the pre-trained BERT<sub>BASE</sub> model [35] as the language encoder. This variation of the model is called PGAE-BERT.

The language decoder autoregressively generates the descriptions word by word by expanding the shared latent representation vector  $h$ :

$$h_0^{\text{dec}}, c_0^{\text{dec}} = W^{\text{dec}} \cdot h + b^{\text{dec}}, \quad (5.3)$$

$$h_t^{\text{dec}}, c_t^{\text{dec}} = \text{LSTM}(y_{t-1}, h_{t-1}^{\text{dec}}, c_{t-1}^{\text{dec}}) \quad (1 \leq t \leq N - 1), \quad (5.4)$$

$$y_t = \text{soft}(W^{\text{out}} \cdot h_t^{\text{dec}} + b^{\text{out}}) \quad (1 \leq t \leq N - 1), \quad (5.5)$$

where  $\text{soft}$  represents the softmax activation function.  $y_0$  is the vector for the symbol indicating the beginning of the sentence, the <BOS> tag.

### 5.2.2 Action Autoencoder

The action autoencoder accepts a sequence of joint angle values and visual features as input and generates the appropriate joint angle values. It consists of an encoder, a decoder and latent layers (in the bottleneck) that contribute to the common latent representations. The action encoder encodes a sequence of length  $M$ ,  $((j_1, v_1), (j_2, v_2), \dots, (j_M, v_M))$  that is the combination of joint angles  $j$  and visual features  $v$ . The visual features are extracted by the channel-separated convolutional autoencoder (CS-CAE) in advance. The action encoder can be defined as<sup>3</sup>:

$$h_t^{\text{enc}}, c_t^{\text{enc}} = \text{LSTM}(v_t, j_t, h_{t-1}^{\text{enc}}, c_{t-1}^{\text{enc}}) \quad (1 \leq t \leq M), \quad (5.6)$$

$$f_M^{\text{enc}} = [h_M^{\text{enc}}; c_M^{\text{enc}}], \quad (5.7)$$

where  $h_t^{\text{enc}}$  and  $c_t^{\text{enc}}$  are the hidden and cell state of the LSTM at time step  $t$ .  $h_0^{\text{enc}}$  and  $c_0^{\text{enc}}$  are set as zero vectors.  $f_M^{\text{enc}}$  is the final state of the action encoder.

The action decoder generates the joint angles at each time step by recursively expanding the shared latent representation vector  $h$ :

$$h_0^{\text{dec}}, c_0^{\text{dec}} = W^{\text{dec}} \cdot h + b^{\text{dec}}, \quad (5.8)$$

$$h_t^{\text{dec}}, c_t^{\text{dec}} = \text{LSTM}(v_t, \hat{j}_t, h_{t-1}^{\text{dec}}, c_{t-1}^{\text{dec}}) \quad (1 \leq t \leq M - 1), \quad (5.9)$$

$$\hat{j}_{t+1} = \tanh(W^{\text{out}} \cdot h_t^{\text{dec}} + b^{\text{out}}) \quad (1 \leq t \leq M - 1), \quad (5.10)$$

<sup>3</sup>Certain symbols in the equations are the same as those utilised for the language autoencoder.

where  $\tanh$  is the hyperbolic tangent activation function and  $\hat{j}_1$  is equal to  $j_1$ , i.e. joint angle values at the initial time step. Visual features,  $v$ , are used as in teacher forcing, i.e. they are not necessarily the result of the action output produced by the model but the features extracted based on the ground-truth trajectories.

### 5.2.3 Gated Multimodal Unit (GMU) Bottleneck

The language and action streams connect at the bottleneck, which is situated between the encoders and decoders of the model. We use a Gated Multimodal Unit (GMU) to fuse the language and action (joints, images) modalities. Thanks to its learned gating mechanism, the GMU allows our model to flexibly learn multiple tasks according to the given command such as ‘describe’ or ‘execute’. This way, our approach works in different translation directions using the whole model during inference without having to put the model in a specific mode. Our bottleneck can be defined with the following equations:

$$L_{\text{feats}} = f_{N+1}, A_{\text{feats}} = f_M, \quad (5.11)$$

$$L_h = \tanh(W^L \cdot L_{\text{feats}} + b^L), \quad (5.12)$$

$$A_h = \tanh(W^A \cdot A_{\text{feats}} + b^A), \quad (5.13)$$

$$z = \sigma(W^z \cdot [A_{\text{feats}}; L_{\text{feats}}] + b^z), \quad (5.14)$$

$$h = z \odot A_h + (1 - z) \odot L_h, \quad (5.15)$$

where  $\sigma$  denotes the sigmoid activation function, whilst  $\tanh$  stands for the hyperbolic tangent activation function and  $\odot$  is the Hadamard product.  $h$  represents the shared hidden representation vector and is used as input to both language and action decoders.  $W$  and  $b$  represent learnable weights and biases for the respective streams, while their superscripts  $L$ ,  $A$  and  $z$  stand for language, action and merged representations respectively.

### 5.2.4 Task Signals

Five different signals are used during training to guide the model in the desired translation direction. Four of those signals can be used during inference. According to the given signal, the input and output of the model change. The five signals are as follows.

- **Describe** tells the model to describe the given action sequence, i.e. action-to-language translation. With this signal, the model accepts as input the sequence of visual features and joint angle values for the action as well as the  $\langle \text{EOS} \rangle$  tag for language. The model is then trained to output the correct description and the static joint angle values corresponding to the final time step of the action sequence.
- **Execute** signals the model to execute the given language description, i.e. language-to-action translation. With this signal, the model expects to be fed

with the whole description sentence, and joint angle values and visual features corresponding to the first time step of the action sequence. The model is then expected to output the joint angle values of the action sequence from the action decoder and  $\langle \text{EOS} \rangle$  from the language decoder.

- **Repeat Action** is the signal for reconstructing the sequence of joint angle values. PGAE expects as input the sequence of joint angle values and visual features for the action encoder and the  $\langle \text{EOS} \rangle$  tag in addition to the signal for the language encoder. The action decoder reconstructs the joint values and the language decoder outputs the  $\langle \text{EOS} \rangle$  tag.
- **Repeat Both** demands the paired language and action input to be present. With this signal, PGAE is trained similarly to PVAE end-to-end. The language encoder accepts the full description in addition to the phrase ‘repeat both’, while the action encoder accepts the corresponding action sequence (joint values and visual features). The language decoder and the action decoder output the full description and joint angle values correspondingly. This signal is intended to be used only during training since it is implausible to expect the robot to repeat an action and its description simultaneously.
- **Repeat Language** is used for reconstructing the full description. The full description and the first time step of the action sequence are fed as input to the encoders. As output, the language decoder reconstructs the description and the action decoder outputs the joint angle values of the first time step.

### 5.2.5 Visual Feature Extraction

Following our previous approach PVAE [136], we employ the channel-separated convolutional autoencoder architecture (CS-CAE) to extract the visual features from images captured by the NICO robot. Instead of processing all three channels together, we train an instance of the CAE for each colour channel (red, green and blue), i.e. channel separation. The channel separation technique has been shown to distinguish between the colours of the objects more accurately in previous chapters (PVAE and PVAE-BERT). Each channel of  $120 \times 160$  RGB images fed into CS-CAE at a time. CS-CAE consists of a convolutional encoder, a fully connected bottleneck and a deconvolutional decoder. After training for the RGB channels separately, the channel-specific visual features are extracted from the bottleneck and then concatenated. The resulting visual features  $v$  are used as input to PGAE. For more details of CS-CAE including its detailed architecture, readers may refer to [Section 4.2.3](#) of the previous chapter.

### 5.2.6 Loss Function

The overall loss is calculated by adding up the reconstruction losses, i.e. language loss and action loss. Similar to the PVAE approach, the language loss,  $L_{\text{lang}}$ , is defined as the cross entropy loss between input and output words, whereas the

action loss,  $L_{\text{act}}$ , is defined as the mean squared error (MSE) between original and predicted joint values:

$$L_{\text{lang}} = \frac{1}{N-1} \sum_{t=1}^{N-1} \left( - \sum_{i=0}^{V-1} w^{[i]} x_{t+1}^{[i]} \log y_t^{[i]} \right), \quad (5.16)$$

$$L_{\text{act}} = \frac{1}{M-1} \sum_{t=1}^{M-1} \|j_{t+1} - \hat{j}_{t+1}\|_2^2, \quad (5.17)$$

where  $V$  is the vocabulary size,  $N$  is the number of words per description,  $M$  is the sequence length for an action trajectory and  $w$  is the weight vector used to counter the imbalance in the frequency of words. The overall loss is the sum of the language and action loss:

$$L_{\text{all}} = \alpha L_{\text{lang}} + \beta L_{\text{act}}, \quad (5.18)$$

where  $\alpha$  and  $\beta$  are weighting factors for language and action terms in the loss function. In our experiments, we set  $\alpha$  and  $\beta$  to 1. In contrast to our previous PVAE and PVAE-BERT models (Chapters 3 and 4), PGAE does not require a binding loss since the two modalities are explicitly bound via the GMU.

### 5.2.7 Training Details

To train PGAE and PGAE-BERT, we first extract visual features using our channel-separated CAE. The visual features are used to condition the actions depending on the cube arrangement, i.e. the action execution according to a given description is dependent on the position of the target cube. Both PGAE and PGAE-BERT are trained end-to-end. PGAE and PGAE-BERT are trained for 6,000 epochs with the gradient descent algorithm and the Adam optimiser [89]. In our experiments,  $h$  has 50 dimensions,  $x$  has 28 dimensions,  $j$  has 5 dimensions,  $N$  is equal to 5 and  $M$  is 50 for fast and 100 for slow actions. We take the learning rate as  $10^{-5}$  with a batch size of 6 samples after determining them as optimal hyperparameters. After a few trials, we decided to freeze the weights of BERT during training as fine-tuning BERT reduces the performance of our model. As BERT has millions of parameters, fine-tuning leads to overfitting.

## 5.3 Experiments and Results

We train and test our model on the Nico2Blocks dataset [136], introduced in Chapter 3, involving 864 samples of sequences of images, joint values and textual descriptions. The dataset consists of simple manipulations of two cubes of different colours on the table by the humanoid NICO robot. The NICO robot is a child-size humanoid robot with a camera in each of its two eyes. The dataset was created using the Blender software<sup>4</sup>. According to our scenario, using its left arm, NICO

<sup>4</sup><https://www.blender.org/>

manipulates one of the two cubes on the table for each sample utilising the inverse kinematics solver provided on Blender. Each sample includes a sequence of first-person view images and joint angle values from NICO’s left arm, accompanied by a textual description of the action. In total, the dataset includes 12 distinct actions, 6 cube colours, 288 descriptions and 144 patterns (action-description-cube arrangement combinations). We randomly vary the 144 patterns six times slightly in terms of action execution in simulation. Out of 864 samples, we exclude 216 samples, involving every unique description and action type, and use them as the test set. By carefully selecting the test samples, we ensure that the combinations of descriptions, action types and cube arrangements in the test set are not seen during training. For more details on the dataset, readers may consult [Section 3.3](#).

In addition to the previous data, we introduce a second agent that does the same actions from the opposite side of the table. Then, we include the resulting images from the perspective of NICO. We use the visual features extracted from these images as additional input and randomly select between them and the visual features extracted from those images, where NICO performs the actions. These cases are shown in [Table 5.2](#), with the ‘-opposite’ suffix for describing or repeating the actions of the second agent, and with the ‘-self’ suffix for describing or repeating the actions of NICO. Consequently, PGAE-self and PGAE-opposite are the same model (with the same weights) trained on the extended dataset – the former is tested with self-agent (NICO’s own) actions, while the latter with the second-agent actions. PGAE and PGAE-BERT in [Table 5.1](#), on the other hand, refer to the models trained on the same dataset as PVAE introduced in [Section 3.3](#), excluding the second-agent actions. We test the models on action-to-language, language-to-action, action-to-action and language-to-language translations as shown in both tables.

### 5.3.1 Action-to-Language Translation

PGAE and its variants use the ‘describe’ signal as input to the language encoder to translate from action to language. The first result columns of [Tables 5.1](#) and [5.2](#) report the accuracy of predicted test descriptions for different approaches. For a generated description to be accepted as correct, all of its words must match the ground truth description according to our predefined grammar, i.e. action-colour-speed-<EOS>. Moreover, the second halves of the tables show the normalised root-mean-square error (nRMSE) between the generated and ground truth joint values. We calculate nRMSE values by dividing the square root of the MSE (which is used as the action loss  $L_{act}$ ) by the observed range of joint values. Accordingly, the results in [Table 5.1](#) show that PGAE is competitive with the earlier approach, PVAE, that has to be set in the specific configuration, which includes using only the action encoder and language decoder while bypassing the language encoder and action decoder by avoiding feeding any language input and not outputting the final joint values. Therefore, PVAE cannot output joint values when it is configured to be used for action-to-language translation. Both PGAE and PGAE-BERT, however, recognise the task from the signal in the language input and generate



Table 5.1: Translation results on the test set of the Nico2Blocks dataset without the second agent. The top half of the table shows the description accuracies per task, while the joint angle value errors are given in the bottom half. Grey background denotes the main output of a task (e.g. description for the action-to-language translation). PVAE cannot produce the auxiliary output of a task (N.A.).

	<b>Describe</b> <b>Act.→Lang.</b>	<b>Execute</b> <b>Lang.→Act.</b>	<b>Repeat Lang.</b> <b>Lang.→Lang.</b>	<b>Repeat Act.</b> <b>Act.→Act.</b>
<b>Approach</b>	Description Accuracies			
PVAE	100%	N.A.	100%	N.A.
PGAE	93.05%	100%	96.30%	100%
PGAE-BERT	94.91%	100%	99.07%	100%
<b>Approach</b>	Joint Angle Value Errors (nRMSE)			
PVAE	N.A.	0.89%	N.A.	0.90%
PGAE	0.23%	0.44%	0.37%	0.44%
PGAE-BERT	0.21%	0.44%	0.33%	0.42%

Table 5.2: Translation results on the test set of the Nico2Blocks dataset involving the second agent. The top half of the table shows the description accuracies per task, while the joint angle value errors are given in the bottom half. Grey background denotes the main output of a task (e.g. joint values for the language-to-action translation). As the opposite-sitting agent data is irrelevant for ‘repeat language’ and ‘execute’ tasks, PGAE-self and PGAE-opposite share those results.

	<b>Describe</b> <b>Act.→Lang.</b>	<b>Execute</b> <b>Lang.→Act.</b>	<b>Repeat Lang.</b> <b>Lang.→Lang.</b>	<b>Repeat Act.</b> <b>Act.→Act.</b>
<b>Test Set</b>	Description Accuracies			
PGAE-self	80.56%	100%	93.98%	100%
PGAE-opposite	65.28%			100%
<b>Test Set</b>	Joint Angle Value Errors (nRMSE)			
PGAE-self	0.58%	0.79%	0.73%	0.89%
PGAE-opposite	2.40%			0.80%

both the description and the joint values for all different tasks. Both PGAE and PGAE-BERT achieve near-perfect joint value prediction. Moreover, PGAE-BERT performs slightly better than PGAE in terms of description accuracy.

In the setting where we train the model on both self- and opposite-agent actions and demand the model to describe the action performed by the opposite agent (Table 5.2), our model achieves 65% accuracy (PGAE-opposite). It achieves 80% accuracy when describing NICO’s own actions (PGAE-self). Moreover, the joint



value error increases slightly to over 2% for the opposite-agent actions, whereas it is comparable with the original approaches for PGAE-self.

### 5.3.2 Language-to-Action Translation

PGAE and its variants use the ‘execute’ signal prepended to the description in order to translate from language to action. The description accuracy (whether <EOS> is outputted by the language decoder) and nRMSE between predicted and ground-truth joint values for language-to-action translation are given in Column ‘Lang.→Act.’ of both [Table 5.1](#) and [Table 5.2](#). All of the approaches are able to generate near-perfect joint values (less than 1% nRMSE). Our previously introduced model PVAE, on the other hand, is not trained to generate descriptions (<EOS> in this case) when configured to execute actions based on the language input. Besides, training PGAE with the demonstrations from the opposite-sitting agent does not significantly affect the action-to-language performance (0.79% nRMSE for PGAE-self/PGAE-opposite).

### 5.3.3 Language-to-Language and Action-to-Action Translations

According to [Table 5.1](#), PGAE and PGAE-BERT are competitive with PVAE in terms of the description accuracy for language-to-language translation and slightly better in terms of the joint value prediction for the action-to-action translation. PVAE does not have the capacity to output joint values for language-to-language and descriptions for action-to-action translations, whereas PGAE and PGAE-BERT almost perfectly output the initial time-step joint values for language-to-language translation and achieve perfect description accuracy in action-to-action translation. As can be seen in [Table 5.2](#), training PGAE with the additional opposite-sitting agent demonstrations slightly increases the joint value error in action-to-action translation for both the actions executed by NICO (PGAE-self) and by the second agent (PGAE-opposite). Furthermore, there is a slight decrease in description accuracy regarding language-to-language translation when the second agent is involved in the training set.

## 5.4 Discussion

The results in different action-language translations, including those with an opposite-sitting agent, represent the capacity of the PGAE architecture to model bidirectional translation between robot actions and language descriptions while retaining a training-and-test-time-consistent configuration thanks to its explicit multimodal fusion mechanism. By using signals prepended to the language input, we overcome the limitation of setting the model in a specific configuration during deployment, which is necessary for the previous PVAE architecture. With the sec-

ondary agent experiments, we prove that PGAE can describe and mimic its own and opposite-agent actions to a considerable extent.

We also leverage a pre-trained language model, BERT, as a language encoder instead of our regular LSTM: PGAE-BERT. PGAE-BERT slightly outperforms the regular PGAE variant in both action-to-language and language-to-language translations. However, its main advantage over PGAE is that it has the potential to recognise unconstrained natural language by leveraging a pre-trained language model.

When we introduce the second agent in our dataset, PGAE’s action-to-language performance declines moderately – more than 10% for the self-actions and almost 30% for the opposite-agent actions. This drop in the action-to-language accuracy is expected as introducing the second-agent actions makes the problem more challenging, e.g. pulling an object by the second agent might be interpreted as pushing the object by NICO itself. Despite the performance drop, describing the actions of an opposite-sitting agent is an extra capability demonstrated by our approach.

The ultimate advantage of using PGAE over PVAE on bidirectional translation between action and language is that PGAE can output both language descriptions and joint angle values regardless of the translation direction, whereas PVAE can only output language when it is configured in action-to-language translation and action when it is configured in language-to-action translation. This highlights the superiority of integrating the task signals into the language input and having a common hidden representation vector over the artificial use of a loss term (i.e. binding loss) to align two separate modality-specific streams.

## 5.5 Conclusion

In this chapter, we have introduced an end-to-end NN approach that can flexibly perform translation between robot actions and language descriptions in multiple directions, some of which involve both first-person actions and opposite-sitting agent actions. By integrating the task signal in the language input, our approach can recognise the given task and output the suitable descriptions and joint values during inference. Our approach, PGAE, exhibits competitive performance in all four translation tasks while having a consistent configuration across learning and inference. With the additional demonstrations from a second agent, our model can recognise and imitate not only its own actions but also the actions of the second agent despite the inherently challenging nature of the task. To our knowledge, this skill set was not modelled by previous approaches. In summary, PGAE can perform various translation tasks robustly without any change in the use of the architecture between learning and test time, which the previous approaches lacked, through its attention-based explicit multimodal fusion mechanism and the insertion of the task signal to the language input.

In the next chapter, we will introduce the Paired Transformed Autoencoders (PTAE) model that can efficiently learn action-language translations by employing a superior multimodal fusion mechanism. Owing to its Crossmodal Transformer-

based bottleneck, PTAE can learn language-action associations with scarcely labelled data and it behaves naturally when it encounters conflicting input through different modalities.



---

# Efficient and Neurocognitively Plausible Translation via Crossmodal Attention

---

Human infant learning happens during exploration of the environment, by interaction with objects, and by listening to and repeating utterances casually, which is analogous to unsupervised learning. Only occasionally, a learning infant would receive a matching verbal description of an action it performs, which is similar to supervised learning. Such a learning mechanism can be mimicked with deep learning. In this chapter, we model this weakly supervised learning paradigm using our PGAE model from [Chapter 5](#), which combines an action and a language autoencoder. After observing a performance drop when reducing the proportion of supervised training, we introduce the Paired Transformed Autoencoders (PTAE) model, using Transformer-based crossmodal attention. PTAE achieves significantly higher accuracies in language-to-action and action-to-language translations, particularly in realistic but difficult cases when only a limited number of supervised training samples are available. We also test whether the trained model behaves realistically with conflicting multimodal input. In accordance with the concept of incongruence in psychology, conflicting input deteriorates the model output. Conflicting action input has a more severe impact than conflicting language input, and more conflicting features lead to larger interference. PTAE can be trained on mostly unlabelled data where labelled data is scarce, and it behaves plausibly when tested with incongruent input<sup>1</sup>.

## 6.1 Introduction

Embodiment, i.e. action-taking in the environment, is considered essential for language learning [19]. Recently, language grounding with robotic object manipulation has received considerable attention from the research community. Most approaches

---

<sup>1</sup>The code belonging to this chapter is available at <https://github.com/oo222bs/PTAE>.

proposed in this domain cover robotic action execution based on linguistic input [61, 107, 158, 161], i.e. language-to-action translation. Others cover language production based on the actions done on objects [43, 69], i.e. action-to-language translation. However, only few approaches [1, 11, 128, 136, 185] handle both directions, enabling not just action execution according to given instructions but also the description of those actions, i.e. bidirectional translation. Moreover, as infants learn, the actions that they are performing are not permanently labelled by matching words from their caretakers, hence, supervised learning with labels must be considered rare. Instead, infants rather explore the objects around them and listen to utterances, which may not frequently relate to their actions, hence, unsupervised learning without matching labels is abundant. Nevertheless, most language grounding approaches do not make use of unsupervised learning except those that use some unsupervised loss terms [1, 136, 185], while LLMs [23, 35, 142] introduced for various unimodal downstream language tasks rely on unsupervised learning for pretraining objectives.

In order to reduce this dependence on labelled data during training, we introduce a new training procedure where we limit the amount of training data used for supervised learning. More precisely, we only use a certain portion of training samples for crossmodal action-to-language and language-to-action translations while training unimodally on the rest of the training samples. As crossmodal translation requires each sample modality to be labelled with the other modality (e.g. an action sequence must be paired with a corresponding language description), we artificially simulate the realistic conditions where there is a large amount of unlabelled (unimodal) data but a much smaller amount of labelled (crossmodal) data.

Another aspect of human language learning is that it takes place in an environment while making use of different modalities such as vision and proprioception. Concepts such as weight, softness and size cannot be grounded without being in the environment and interacting with objects. Language learning approaches that use multiple modalities and take action in an environment into account are preferable to those that use a unimodal approach to process large amounts of text. A recent study [25] in language teaching concludes that learning is enhanced when the language learner uses language to produce meaningful outputs. Hence we strive to devise embodied multimodal models that tackle language grounding. To this end, our robotic object manipulation dataset is generated from a simulation setup as seen in [Figure 6.1](#). We use our humanoid child-size robot NICO to perform various actions on cubes on a table and label those actions with language descriptions. We introduce further details of our setup in [Section 6.3](#).

Different from other approaches, our previous PGAE model ([Chapter 5](#)) can bidirectionally translate between language and action, which enables an agent not only to execute actions according to given instructions but also to recognise and verbalise its own actions or actions executed by another agent. As the desired translation task is communicated to the network through an additional signal word in the language input, PGAE can flexibly translate between and within modalities during inference. However, when trained under limited supervision conditions,

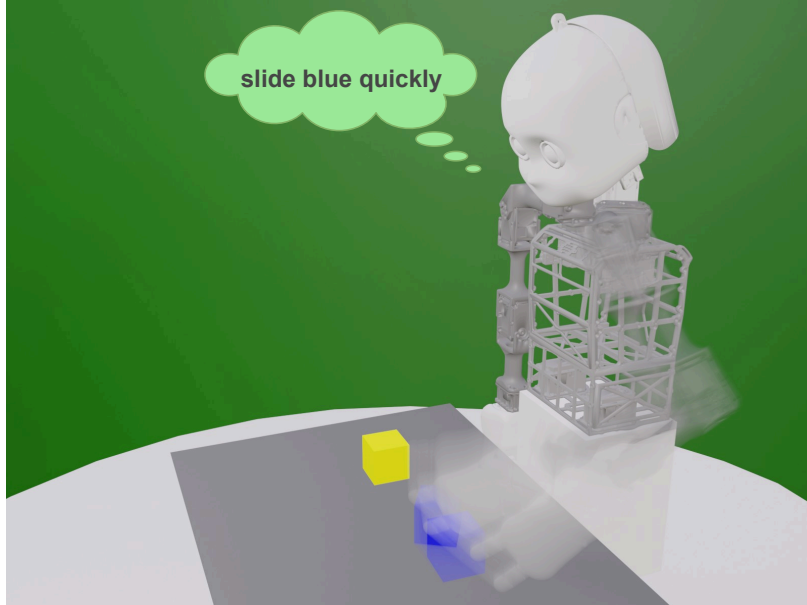


Figure 6.1: Our tabletop object manipulation scenario in the simulation environment: the NICO robot is moving the blue cube on the table. The performed action is labelled as “slide blue quickly”. Our approach can translate from language to action and vice versa; i.e. we perform actions that are described in language and also describe the given actions using language.

PGAE performs poorly on the action-to-language translation task, under two conditions: firstly, we experiment with reducing the number of supervised training *iterations* while using the whole data set for supervised training. Secondly, we experiment with reducing the number of training *samples* used with the supervised signals. In both instances, even though the first is more trivial than the second, the action-to-language performance of PGAE suffers as the proportion of supervision is lowered.

In this chapter, we introduce a novel model, PTAE, to overcome this hurdle. Inspired by the successful application of the Crossmodal Transformer in vision-language navigation by the Hierarchical Cross-Modal Agent (HCM) architecture [75], PTAE replaces the gated multimodal fusion mechanism of PGAE and optionally the LSTM-based encoders with a Crossmodal Transformer. Thanks to its more efficient and sequence-retaining crossmodal attention mechanism, PTAE achieves superior performance even when an overwhelming majority of training iterations (e.g. 98 or 99%) consist of unsupervised learning. When most training samples are used for unsupervised learning, PTAE maintains its perfect action-to-language performance up to 80% of training samples learnt unimodally and performs relatively well for the 90% case (over 80% sentence accuracy). Even in cases where only 1 or 2% of the training samples are used in a supervised fashion, which is analogous to realistic few-shot learning settings, PTAE describes actions well over chance level with up to a 50% success rate. Our results hint that PTAE



precludes the need for large amounts of expensive labelled data, which is required for supervised learning, as the new architecture with the Crossmodal Transformer as the multimodality fusion technique significantly outperforms PGAE (Chapter 5) under the limited supervision training conditions.

Furthermore, inspired by the concept of incongruence in psychology and to test the robustness of the trained model against noise, for each task, we introduce an extra input that is contradictory to the expected output of the model. For example, for language-to-action translation, we introduce extra conflicting action input showing an action different from the expected action of the model. The intertwined processing of language and action input in the Crossmodal Transformer resembles the tight interconnection between language and sensorimotor processes observed in the human brain [62, 172]. Embodied accounts of human language comprehension assume that linguistic information induces mental simulations of relevant sensorimotor experiences. As a direct consequence of embodied language processing, conflicts between linguistic input and sensorimotor processes have been shown to result in bidirectional impairments of language comprehension on the one hand and perceptual judgements and motor responses on the other hand [12, 51, 84, 115] although the strength of these behavioural effects has recently been debated [182]. In our PTAE model, we found asymmetry in terms of the impact of the action and language modalities on the performance of the model. Regardless of the output modality, introducing extra contradictory action input affects the model performance much more than introducing it in the language modality.

Our contributions in this chapter can be summarised as follows:

1. we introduce PTAE that handles realistic learning conditions that mainly include unsupervised/unpaired language and action experiences while requiring minimal use of labelled data, which is expensive to collect,
2. we show plausible behaviour of the model when testing it with psychology-inspired contradictory information.

The rest of Chapter 6 is structured as follows: in the next section, we define the PTAE architecture in detail. Section 6.3 introduces our experiments and their results. Section 6.4 discusses these results, while Section 6.5 concludes the chapter with a summary and pointers on the following chapters.

## 6.2 Proposed Method: Paired Transformed Autoencoders (PTAE)

The PTAE model has an encoder-decoder architecture that is capable of bidirectional translation between robot actions and language descriptions. It consists of a Crossmodal Transformer that is the backbone and multimodality fusion mechanism of the architecture and LSTM-based decoders that output language and joint values respectively. As input, PTAE accepts language descriptions of actions including the task signal, which defines the translation direction, as well as a sequence of

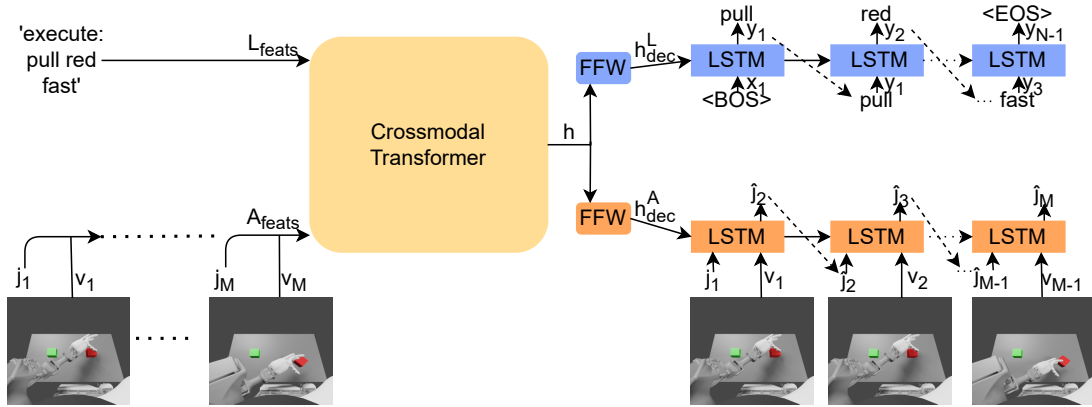


Figure 6.2: The architecture of the PTAE model. The inputs are a language description (incl. a task signal) and a sequence of visual features (extracted using the channel-separated convolutional autoencoder) and joint values, while the outputs are a description and a sequence of joint values. The language encoder can be an LSTM, the  $BERT_{BASE}$  model or the descriptions can be directly passed to the transformer word by word. The action encoder can be an LSTM or the action sequence can be passed directly to the Transformer. Both decoders are LSTMs – we show unfolded versions of the LSTMs. Forming the bottleneck, the Crossmodal Transformer connects the two streams.  $h$  is the shared representation vector.

the concatenation of multivariate joint values and visual features. According to the task signal, PTAE outputs joint values required for executing a particular action or outputs language descriptions of an action.

Different from most of the aforementioned approaches, our model is bidirectional: it can not only produce actions according to given language descriptions but also recognise actions and produce their descriptions. As our model is based on an autoencoder-like architecture, it can be trained in a mostly unsupervised way by asking the model to reproduce the given language or proprioception input. Moreover, our approach is flexible during inference since it does not need to be reconfigured for the translation task; due to the inclusion of the task signal in the language input, our PTAE can reliably execute the desired task on the go, whether it is a translation from language to action or vice versa. This is an essential step towards an autonomous agent that can interact within the environment and communicate with humans.

As shown in Figure 6.2, PTAE comprises a Crossmodal Transformer, which accepts multimodal input (i.e. language, proprioception and vision), and language and action decoders that output language descriptions and joint values respectively. The language and action input can optionally be preprocessed by LSTM-based encoders, as in the case of PGAE<sup>2</sup>. However, after some initial trials with both cases, in this chapter, we do not use any extra encoding layers before the

<sup>2</sup>Readers may consult Chapter 5 for detailed definitions of LSTM-based language and action encoders.

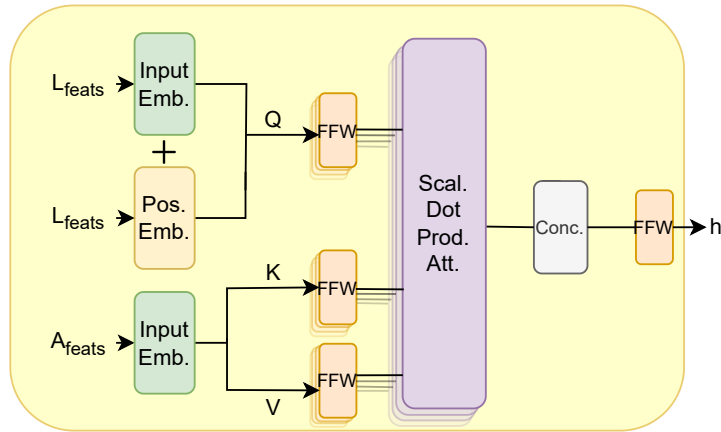


Figure 6.3: The architecture of the Crossmodal Transformer. The language features are embedded and used as the query vector ( $Q$ ), while the embedded action features are used as the key ( $K$ ) and value ( $V$ ) vectors. The positional embedding is applied only to the language features. The multi-head attention (MHA) involves the  $Q$ -,  $K$ - and  $V$ -specific feedforward (FFW) and scaled dot product attention layers following the original Transformer architecture. The multiple heads are then concatenated and fed to the final FFW, which outputs the common hidden representation vector  $h$ .

Crossmodal Transformer for the sake of simplicity and model size as we do not see any significant change in the performance.

## 6.2.1 Crossmodal Transformer

The Crossmodal Transformer (CMT) replaces the Gated Multimodal Unit (GMU) [13] in our previous PGAE model (Chapter 5) and can be employed in effect as language and action encoders. The simplified architecture of the CMT can be seen in Figure 6.3. The functionality of the CMT is to extract the common latent representations of paired language and action sequences. Following the HCM architecture [75], we use the language modality as queries ( $Q$  vectors) and the action modality (concatenated visual features and joint values) as keys ( $K$  vectors) and values ( $V$  vectors). The language descriptions are represented as one-hot encoded vectors, while action input is composed of joint values of NICO’s left arm and the visual features from images recorded by the camera in NICO’s eye. As in Chapters 3 to 5, we use a channel-separated convolutional autoencoder (CS-CAE) to extract visual features from images. The CMT encodes the common

latent representations as follows:

$$Q = \text{ReLU}(W^{\text{token}} \cdot x_t + b^{\text{token}}) + \text{PE}(x_t) \quad (1 \leq t \leq N + 1), \quad (6.1)$$

$$K, V = \text{ReLU}(W^{\text{act}} \cdot [v_t; j_t] + b^{\text{act}}) \quad (1 \leq t \leq M), \quad (6.2)$$

$$A_t = \text{MHA}(Q, K, V) \quad (1 \leq t \leq N + 1), \quad (6.3)$$

$$h_t = \text{PWFF}(A_t) \quad (1 \leq t \leq N + 1), \quad (6.4)$$

$$h = \text{AvgPool}(h_t) \quad (1 \leq t \leq N + 1), \quad (6.5)$$

where  $x$ ,  $v$  and  $j$  are linguistic, visual and proprioceptive inputs respectively – note that when no language or action encoder is used,  $x$  corresponds to  $L_{\text{feats}}$ , while the concatenation of visual features and joint values  $[v_t; j_t]$  corresponds to  $A_{\text{feats}}$  in Figure 6.3. ReLU is the rectified linear unit activation function, while PE, MHA and PWFF are the positional encodings, multi-head attention layer and the position-wise feedforward layer as used in the original Transformer paper [173]. As the Transformer architecture does not include any recurrence, we employ a fixed sinusoidal function-based PE layer on the language features to include the position information.  $A_t$  is the crossmodal attention vector for time step  $t$ , whereas  $h_t$  is the hidden vector for time step  $t$ . AvgPool is the average pooling applied on the time axis to the sequential hidden vector to arrive at the common latent representation vector  $h$ . For our experiments, we employ a single-layer CMT with 4 parallel attention heads.

### 6.2.2 Language Decoder

We use an LSTM as the language decoder to autoregressively generate the descriptions word by word by expanding the common latent representation vector  $h$  produced by the CMT:

$$h_0^{\text{dec}}, c_0^{\text{dec}} = W^{\text{dec}} \cdot h + b^{\text{dec}}, \quad (6.6)$$

$$h_t^{\text{dec}}, c_t^{\text{dec}} = \text{LSTM}(y_{t-1}, h_{t-1}^{\text{dec}}, c_{t-1}^{\text{dec}}) \quad (1 \leq t \leq N - 1), \quad (6.7)$$

$$y_t = \text{soft}(W^{\text{out}} \cdot h_t^{\text{dec}} + b^{\text{out}}) \quad (1 \leq t \leq N - 1), \quad (6.8)$$

where soft represents the softmax activation function.  $y_0$  is the vector for the symbol indicating the beginning of the sentence, the <BOS> tag.

### 6.2.3 Action Decoder

Similarly, an LSTM is employed as the action decoder to output joint angle values at each time step via the common representation vector  $h$ :

$$h_0^{\text{dec}}, c_0^{\text{dec}} = W^{\text{dec}} \cdot h + b^{\text{dec}}, \quad (6.9)$$

$$h_t^{\text{dec}}, c_t^{\text{dec}} = \text{LSTM}(v_t, \hat{j}_t, h_{t-1}^{\text{dec}}, c_{t-1}^{\text{dec}}) \quad (1 \leq t \leq M - 1), \quad (6.10)$$

$$\hat{j}_{t+1} = \tanh(W^{\text{out}} \cdot h_t^{\text{dec}} + b^{\text{out}}) \quad (1 \leq t \leq M - 1), \quad (6.11)$$

where the predicted joint values for time step  $t$  are represented by  $\hat{j}_t$  and  $\tanh$  is the hyperbolic tangent activation function. We take  $\hat{j}_1$  as  $j_1$ , i.e. ground-truth joint angle values corresponding to the initial position of the arm. The visual features used as input  $v$  are extracted from the ground-truth images and used similarly to teacher forcing, whereas the joint angle values  $\hat{j}_t$  are learnt autoregressively.

## 6.2.4 Visual Feature Extraction

Following the PGAE pipeline (Chapter 5), we use CS-CAE to extract visual features from first-person images from the eye cameras of NICO recorded in the simulation. We utilise channel separation when extracting visual features: an instance of the CAE is trained for each RGB colour channel. In Chapter 3, we demonstrate that CS-CAE distinguishes object colours more accurately than the regular CAE without channel separation.

As before, we feed each instance of CS-CAE with the corresponding channel of RGB images of size  $120 \times 160$ . CS-CAE consists of a convolutional encoder, a fully connected bottleneck, and a deconvolutional decoder. Each RGB channel is trained separately, after which we extract the channel-specific visual features from the bottleneck and concatenate them to arrive at composite visual features. These visual features make up  $v$  which is used as visual input to PTAE. For further details on the visual feature extraction process, readers may refer to Section 3.3.

## 6.2.5 Loss Function

As with PGAE in Chapter 5, we use two loss functions to calculate the deviation from the ground-truth language descriptions and joint values. The language loss,  $L_{\text{lang}}$ , is calculated as the cross entropy between input and output words, while the action loss,  $L_{\text{act}}$ , is the mean squared error (MSE) between original and predicted joint values:

$$L_{\text{lang}} = \frac{1}{N-1} \sum_{t=1}^{N-1} \left( - \sum_{i=0}^{V-1} x_{t+1}^{[i]} \log y_t^{[i]} \right), \quad (6.12)$$

$$L_{\text{act}} = \frac{1}{M-1} \sum_{t=1}^{M-1} \|j_{t+1} - \hat{j}_{t+1}\|_2^2, \quad (6.13)$$

where  $V$  is the vocabulary size,  $N$  is the number of words per description, and  $M$  is the sequence length for action trajectories. The total loss is then the sum of the language and action losses:

$$L_{\text{all}} = \alpha L_{\text{lang}} + \beta L_{\text{act}}, \quad (6.14)$$

where  $\alpha$  and  $\beta$  are weighting factors for language and action terms in the loss function. In our experiments, we take both  $\alpha$  and  $\beta$  as 1.0. We use the identical loss functions as PGAE except for the weight vector used in the language loss to counter the imbalance in the frequency of words, after seeing that it is unnecessary for PTAE.

### 6.2.6 Training Details

Visual features are extracted in advance by CS-CAE before training PTAE and PGAE. Visual features are necessary to execute actions according to language instructions since cube arrangements are decisive in manipulating the left or right object, i.e. determining whether to manipulate the left or right cube depends on the position of the target cube. After extracting visual features, both PGAE and PTAE are trained end-to-end with all three modalities. After initial experiments, PGAE is trained for 6,000 epochs, while PTAE is trained for 2,500 epochs using the gradient descent algorithm and Adam optimiser [89]. For PTAE, we decided that  $h$  has 256 dimensions following [75], whereas the same vector has 50 dimensions in PGAE.  $x$  has 28 dimensions,  $j$  has 5 dimensions,  $N$  is equal to 5, while  $M$  is fixed to 50 for fast and 100 for slow actions. For both PGAE and PTAE, we take the learning rate as  $10^{-5}$  with a batch size of 6 samples after determining them as optimal hyperparameters. PTAE has approximately 1.5M parameters compared to PGAE with a little over 657K parameters.

## 6.3 Experiments and Results

We use the same dataset [136] as in the previous chapter, except that in this chapter we exclude experiments with another agent from the opposite side of the table. The dataset encompasses 864 samples of sequences of images and joint values alongside their textual descriptions. It consists of robot actions on two cubes of different colours on the table by the NICO robot, generated using inverse kinematics and created in the simulation environment using Blender software<sup>3</sup>. The NICO robot has a camera in each eye, which is used to record a sequence of egocentric images. According to the scenario, NICO manipulates one of the two cubes on the table with its left arm at a time. Accordingly, we use and record 5 joints of the left arm during object manipulation. In total, the dataset includes 12 distinct actions<sup>4</sup>, 6 cube colours, 288 descriptions<sup>5</sup>, and 144 patterns<sup>6</sup> (action & cube arrangement combinations). The 144 patterns are randomly varied six times in terms of action execution in simulation. As a result, we arrive at a dataset of 864 samples in total. Out of 864 samples, 216 samples that involve every unique description and action type are excluded and used as the test set. The remaining 648 samples make up the training set. The vocabulary consists of the following words divided into 3 categories:

- 6 action words (3 original/3 alternative): ‘push/move-up’, ‘pull/move-down’, ‘slide/move-sideways’;

<sup>3</sup><https://www.blender.org/>

<sup>4</sup>We distinguish the actions according to the action type (PUSH, PULL or SLIDE), the target object position (LEFT or RIGHT) and the speed setting (SLOW or FAST).

<sup>5</sup>Since the vocabulary includes 6 action words, 12 colour words and 4 speed words, we arrive at 288 unique descriptions.

<sup>6</sup>The dataset involves 12 unique actions and 12 unique cube arrangements (e.g. blue and yellow cubes placed on the table from left to right); hence, their combinations yield 144 patterns.

- 12 colour words (6 original/6 alternative): ‘red/scarlet’, ‘green/harlequin’, ‘blue/azure’, ‘yellow/blonde’, ‘cyan/greenish-blue’, ‘violet/purple’;
- 4 speed words (2 original/2 alternative): ‘slowly/unhurriedly’, ‘fast/quickly’.

The sentences consist of a word from each category: therefore, our textual descriptions are 3-word sentences. For more details on the dataset, readers may consult [Section 3.3](#). PGAE and PTAE are trained on this dataset and their performances are tested in terms of action-to-language and language-to-action translations under various amounts of supervision.

### 6.3.1 Task Signals

We use four signals to train PTAE. According to the given signal, the input and output of the model change. The signals are:

- **Describe** indicates action-to-language translation,
- **Execute** indicates language-to-action translation,
- **Repeat Action** indicates action-to-action translation,
- **Repeat Language** indicates language-to-language translation.

According to the latter two ‘repeat’ signals, the network uses mainly unimodal information. The ‘describe’ and ‘execute’ signals, on the other hand, involve cross-modal translation from one modality to the other. The unimodal signals are used in the unsupervised learning of an autoencoder, whereas the crossmodal signals are used in supervised learning, where coordinated action values and language labels must be available. In the case of PGAE training, an additional ‘repeat both’ signal is also used, which also requires coordinated labels, and leads to a slightly better performance [135]. For PTAE, however, this was found unnecessary. More details of the task signals can be found in [Section 5.2.4](#).

### 6.3.2 Reduction of Supervised Training

We restrict the amount of supervision by increasing the ratio of unsupervised learning iterations, i.e. training with the unimodal ‘repeat’ signals, in the overall training iterations. Thereby, the ratio of supervised learning iterations, i.e. training with the crossmodal signals, decreases. The resulting training paradigm is analogous to developmental language learning, where an infant is exposed only to a limited amount of supervision. We train both PTAE and PGAE with varying ratios of unimodal/total training iterations. For another set of experiments, we restrict the amount of supervision by limiting the proportion of training samples used for crossmodal translation tasks. We test the performance of both models with varying degrees of unsupervised training under different schemes (limiting the percentage of *iterations* or *samples*) on the crossmodal translation tasks. Here, we investigate



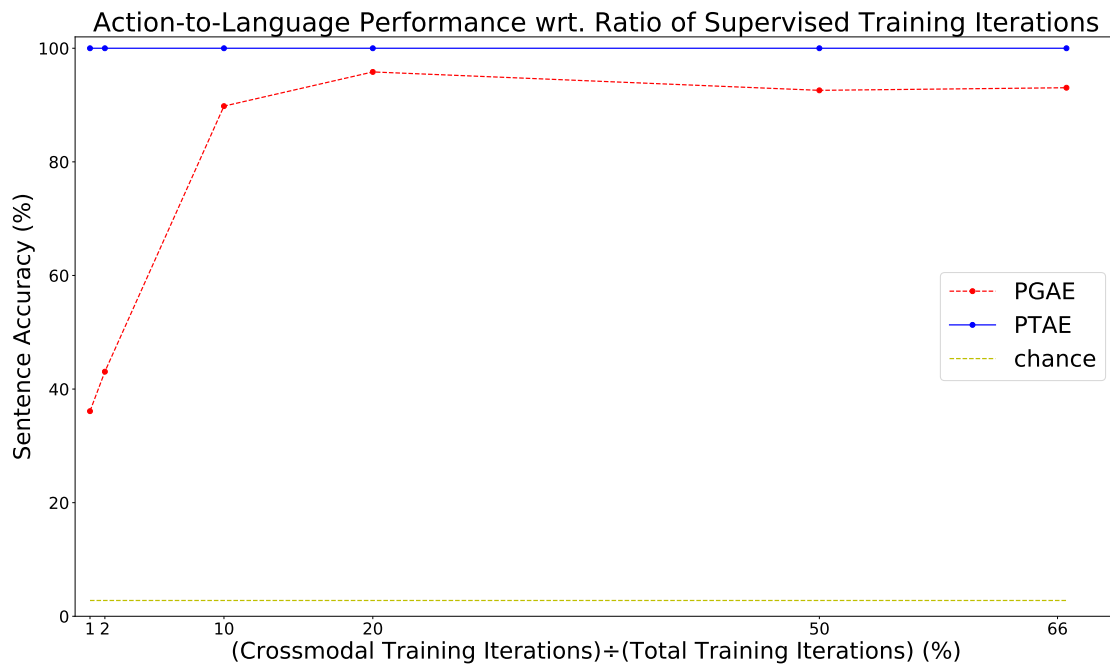


Figure 6.4: Sentence accuracy for action-to-language translation on the test set with respect to supervised training iterations. Supervised training refers to the crossmodal translation cases ‘describe’ and ‘execute’. The two crossmodal signals receive the same number of iterations between them out of the supervised iterations. We report the results for 1%, 2%, 10%, 20%, 50% and 66.6% (the regular training case) crossmodal (supervised) iterations. These percentages correspond to the fraction of supervised training iterations for PGAE and PTAE. Note that the 100% case is not shown here, since the models need unsupervised iterations (unimodal repeat signals) to be able to perform the ‘repeat language’ and ‘repeat action’ tasks.

action-to-language and language-to-action translations because they are the more important and difficult tasks. For the ‘repeat’ tasks, the results match those in the previous chapter; therefore, readers may refer to [Section 5.3.3](#).

**Reducing Supervised Training Iterations** Figure 6.4 shows the results of PGAE and PTAE on action-to-language translation with different percentages of training *iterations* used in a supervised fashion. Both PGAE and PTAE with different training regimes based on different proportions of supervised training iterations achieve accuracies higher than the chance level (2.78%), which we calculate based on our grammar (action, colour, speed):  $1 \div (3 \times 6 \times 2)$ . The action-to-language translation performance of PGAE falls when the ratio of crossmodal (i.e. supervised) training iterations is low, particularly when 10% or a smaller proportion of the iterations are supervised. Even though the description accuracy slightly increases to over 95% when supervised training amounts to only 20% of all training iterations (it may partially be due to overfitting), it sharply drops to well be-

low 50% when the rate is decreased to 2%. PGAE is able to describe 36% of the test samples when only 1% of the training iterations are used to learn crossmodal translations between action and language. In contrast, PTAE maintains its perfect description accuracy even when it has only been trained with 1% supervised training iterations. While there is a detrimental impact of reduced supervision, i.e. the limitation on the percentage of crossmodal training iterations, on the action-to-language translation performance of PGAE, the Transformer-based PTAE is not affected by the same phenomenon. For space reasons, we do not report language-to-action results with respect to the different percentages of supervised iterations, but we observed a similar trend comparable with Figure 6.4.

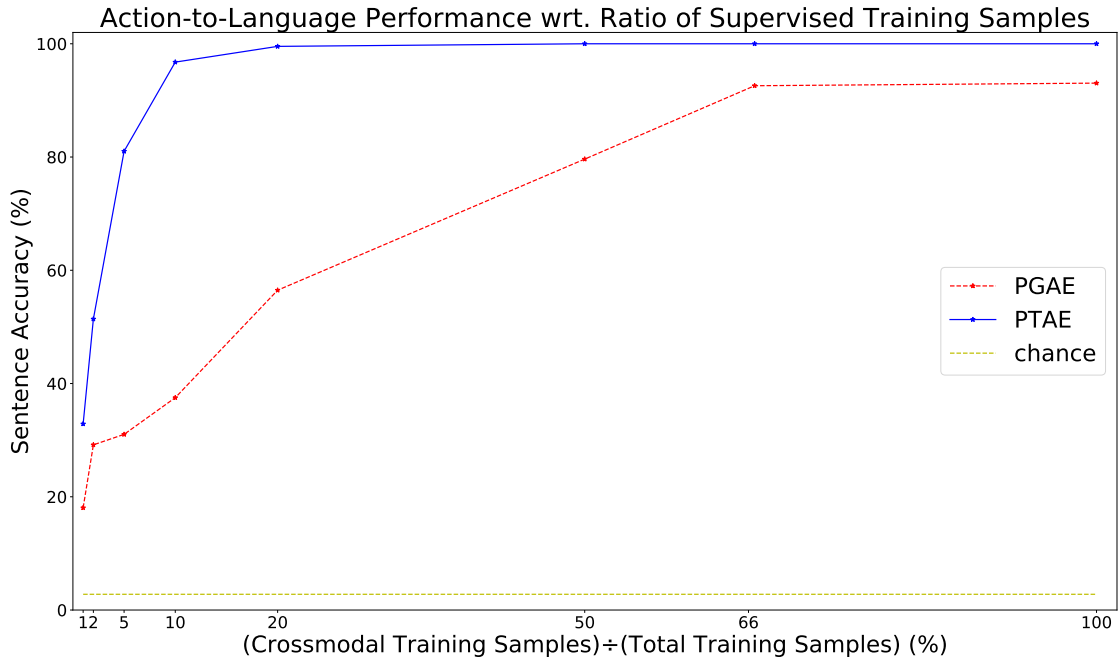


Figure 6.5: Sentence accuracy for action-to-language translation on the test set with respect to supervised training samples. Supervised training refers to the crossmodal translation cases ‘describe’ and ‘execute’. We limit the number of training samples for supervised tasks. We report the results for the 1%, 2%, 5%, 10%, 20%, 50% and 66.6% cases as well as the 100% regular training case. These percentages correspond to the fraction of training samples used exclusively for the supervised training for PGAE and PTAE, i.e. both ‘execute’ and ‘describe’ signals are trained with only a limited number of samples corresponding to the percentages.

**Reducing Supervised Training Samples** In order to further investigate the performance of PTAE with limited supervision, we introduce a more challenging training regime. We limit the number of training *samples* shown to supervised signals, ‘describe’ and ‘execute’, and show the rest of the training samples only on ‘repeat action’ and ‘repeat language’ modes. We train both PGAE and PTAE with varying percentages of supervised training samples. Figure 6.5 displays the

results. In all cases with different proportions of supervised training samples, both PGAE and PTAE outperform the chance level. While maintaining perfect sentence accuracy down to 20% supervised training and keeping up its performance for 10% supervised training for the ‘describe’ signal, the performance of PTAE drops sharply when the ratio of training samples used for crossmodal signals is 2% and below. Nevertheless, PTAE beats PGAE in each case when trained on different percentages of supervised training samples. The performance of PGAE suffers even when 50% of the training samples are used for supervised signals; it drops below 80% – PTAE retains 100% for the same case. It takes more than 90% of the training samples to be exclusively used in the unsupervised signals for the performance of PTAE to decrease meaningfully (from 100% to 81%), while this ratio is much lower for PGAE as its performance already drops significantly at 50%. Even for 1% supervised training samples, which amount to only 7 training samples, PTAE manages to translate one-third of the test samples from action to sentences.

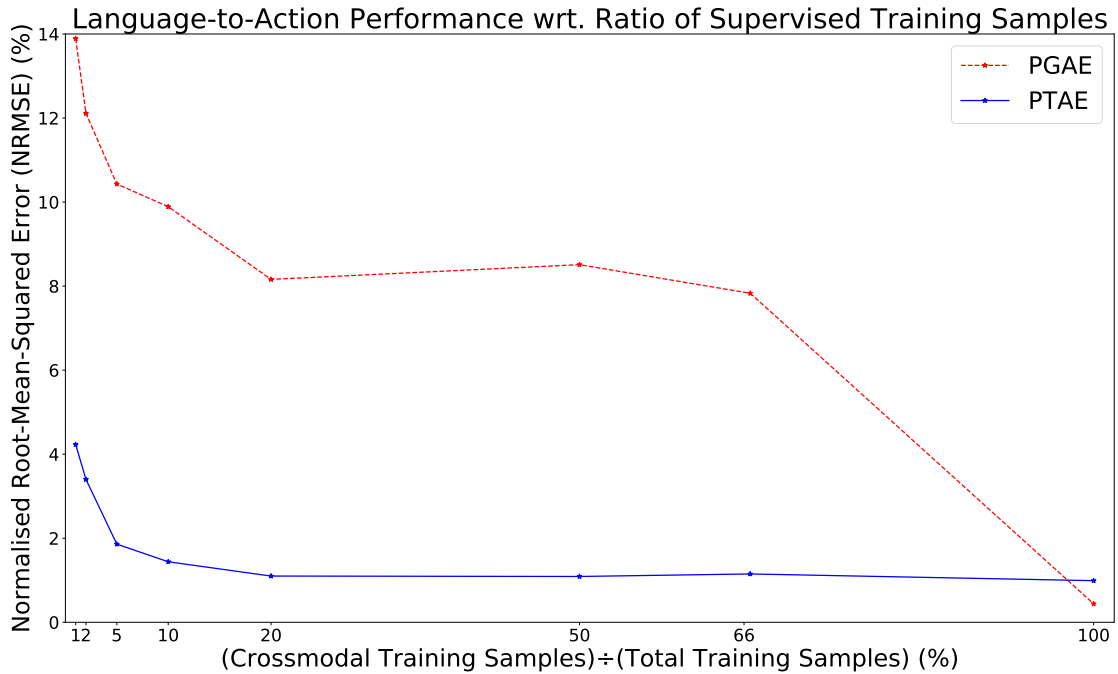


Figure 6.6: Joint value prediction error in language-to-action translation on the test set with respect to supervised training samples. Supervised training refers to the crossmodal translation cases ‘describe’ and ‘execute’. We limit the number of training samples for supervised tasks. We report the results for the 1%, 2%, 5%, 10%, 20%, 50% and 66.6% cases as well as the 100% regular training case. These percentages correspond to the fraction of training samples used exclusively for the supervised training for PGAE and PTAE. ‘execute’ and ‘describe’ translations are shown the same limited number of samples.

Language-to-action translation results with respect to different percentages of supervised training samples for PGAE and PTAE are shown in Figure 6.6. We show the deviation of the produced joint values from the original ones in terms of the

normalised root-mean-squared error (NRMSE), which we obtain by normalising the root-mean-squared error (RMSE) between the predicted and ground-truth values by the range of joint values – the lower percentages indicate better prediction (0% NRMSE meaning predicted values are identical with ground-truth values), whereas the higher percentages indicate worse prediction (100% NRMSE meaning the RMSE between predicted and ground-truth values is equal to the range of possible values). We can see a similar trend as in action-to-language translation apart from the regular case (100%) when PGAE has a lower error than PTAE, which is probably due to the fact that PGAE is trained for more than two times the number of iterations of PTAE since it takes longer for the training loss of PGAE to reach a global minimum. In all other cases, limiting the ratio of training samples to be used in the supervised modes impacts the language-to-action performance of PGAE heavily: the NRMSE rises from less than 0.5% to almost 8% when the percentage of supervised samples is reduced to two thirds of the training samples. The error rate increases further as the number of training samples used in the crossmodal training modes decreases. The NRMSE for PTAE is also inversely proportional to the ratio of supervised training samples. However, the impact of limiting the number of training samples for supervised modes on PTAE is much lower than on PGAE. When the percentage of supervised training samples is reduced to 1%, the deviation from the ground-truth joint values is only a little more than 4% for PTAE, whereas the same statistic for PGAE is almost 14%.

### 6.3.3 Exposure to Conflicting Input Modalities

We also investigate the impact of contradictory extra input on the performance of PTAE; we use PTAE-regular trained with 33% unsupervised training iterations and no contradictory input. We test the robustness of our approach to varying numbers of conflicts (up to 3) in the extra input. The definitions of the added conflict per task signal are given below.

- **Describe:** we add a conflicting description to the language input (conflict in language).
- **Execute:** we use a conflicting sequence of vision and proprioception input (conflict in action).
- **Repeat Action:** we add a conflicting description to the language input (conflict in language).
- **Repeat Language:** we use a conflicting sequence of vision and proprioception input (conflict in action).

The conflicts are introduced using the following scheme.

- **Conflict in the extra language input:** one, two or all of the action, colour and speed words that constitute a description do not match with those of the ground-truth paired description of the action. For instance, for the input

action paired with the description “push red slowly”, a description like “push green slowly” (one conflict present, namely colour), or “pull green fast” (all three conflicts present: action, colour and speed) is given to the model as conflicting extra language input.

- **Conflict in the extra action input:** one, two or all of the action-type, position and speed aspects which form distinct actions do not match with the language description. We choose one of those action trajectories that are not paired with the given language input. The conflict(s) can be in the action type (e.g. pushing instead of pulling), the position of the manipulated object (e.g. the left cube being pulled instead of the right), or the speed of the action (e.g. the cube being pulled fast instead of slowly).

The results of this experiment are given in Figures 6.7 and 6.8. In the case of the ‘describe’ and ‘repeat action’ signals, the action supplies the relevant input whereas the language is the conflicting distractor. Here, we observe only a slight decrease in performance. In the case of action-to-language translation (‘describe’) the sentence accuracy goes down from 100% to 95% when there are three conflicting input elements (action type, colour, speed) – see the left bar chart in Figure 6.7. Action-to-action (‘repeat action’) translation manages to retain its performance as the error in joint values only slightly increases from 1.03% to 1.09% for the case with 3 conflicts – see the bottom left bar chart.

In the case of ‘execute’ and ‘repeat language’ signals, the language supplies the relevant input while the action is the conflicting distractor. Here, we observe a big performance drop. Language-to-action translation (‘execute’) suffers heavily

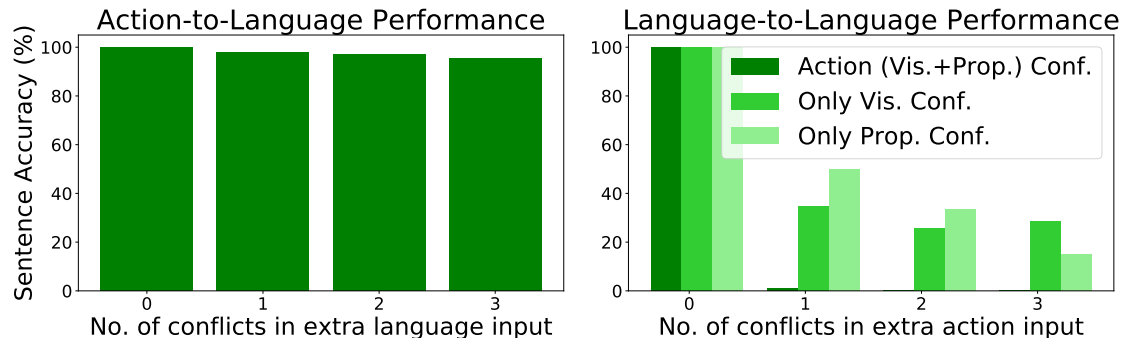


Figure 6.7: Model performance on action-to-language (left panel) and language-to-language (right panel) on the test set with respect to the number of conflicts introduced in the extra input. The predicted sentence accuracy per number of conflicts is visualised for both translations. For action-to-language, the extra conflicting inputs are added in the language input, whereas for language-to-language, they are added in the action input. When the conflict is introduced in the action modality, we also test having the conflict only in the vision or only in the proprioception submodalities – in this case, the other submodality has the matching input.

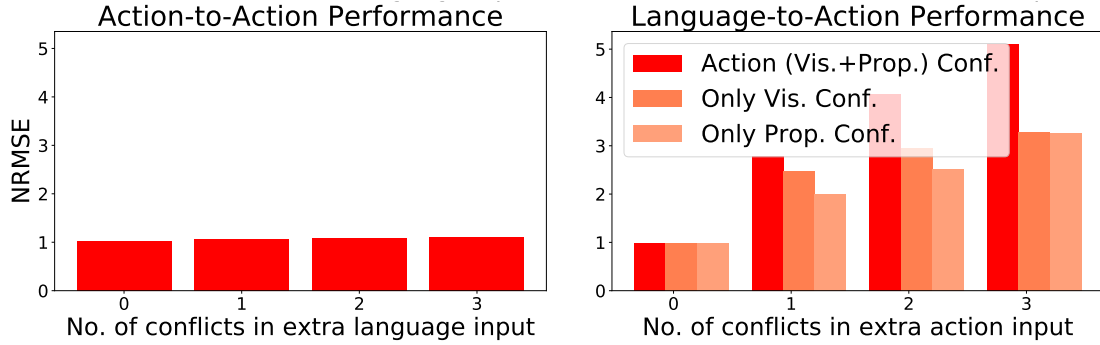


Figure 6.8: Model performance on action-to-action (left panel) and language-to-action (right panel) on the test set with respect to the number of conflicts introduced in the extra input. The NRMSE for predicted joint values is visualised for both translations. For action-to-action, the extra conflicting inputs are added to the language input, whereas for language-to-action, they are added to the action input. When the conflict is introduced in the action modality, we also test having the conflict only in the vision or only in the proprioception submodalities – in this case, the other submodality has the matching input.

as the deviation of the predicted joint values from the ground-truth joint values increases from 0.99% to 4.95% (the right bar chart in Figure 6.8). In the language-to-language translation case (‘repeat language’), PTAE loses its ability to repeat the given language description when one or more conflicting elements (action type, position, speed) are introduced with the extra input: the sentence accuracy decreases from 100% to 0% (the right bar chart in Figure 6.7).

The discrepancy in the results demonstrates the asymmetric impact of conflicts in the two modalities, namely, when language input is introduced as a contradictory element, the performance drops slightly, whereas when the contradictory input is introduced in the action stream, the model is affected heavily and performs poorly. The primary output modality has no significant impact on the result; for example, we can see that both ‘describe’ and ‘repeat language’ output language at large, but they are affected very differently by the conflicting input. To test whether the bigger impact of conflicting action input is due to the involvement of two modalities in action (vision and proprioception), we also tried introducing the conflict either only in vision or only in proprioception (the relatively brighter bars in the right charts in both Figure 6.7 and Figure 6.8). In either case, the performance is still substantially negatively affected, although the drop in performance is naturally not as severe as introducing the conflict in both modalities.

## 6.4 Discussion

The experimental results on action-to-language and language-to-action translations show the superior performance and efficiency of our novel PTAE model under limited supervision. Limiting the percentage of supervised crossmodal iterations

during training has no adverse effect on PTAE as it maintains its perfect sentence accuracy when translating from action to language. In contrast, the action-to-language translation accuracy for the previous PGAE model drops substantially when only a tiny proportion of the training iterations are supervised. When we challenge both models more by limiting the number of training samples for the supervised crossmodal ‘execute’ and ‘describe’ signals, we see a similar pattern: when half or less than half of the training samples are used for supervised signals, action-to-language sentence accuracy for PGAE decreases directly proportional to the ratio of supervised samples. PTAE, on the other hand, retains its action-to-language performance up until when an overwhelming majority of training samples are used in a supervised fashion. Even after being trained with 2% supervised training, which amounts to only 13 samples out of 648, PTAE is able to describe more than half of the action sequences correctly. These results are to some extent comparable with those achieved by the Child’s View for Contrastive Learning (CVCL) model [174], a generic contrastive language encoder- and action encoder-based network trained on a visuolinguistic infant-experience dataset when its linear classifier-fitted version is trained on a limited amount of supervised samples. All in all, PTAE shows superior action-to-language performance than PGAE for varied levels of limited supervision.

The adverse effect of limiting the number of supervised training samples on the language-to-action performance can already be seen for PGAE even when only one-third of the samples are excluded as the error rate between predicted and ground-truth joint values rises significantly. It continues to increase gradually after reducing the level of supervision further. On the contrary, PTAE is robust against limited supervision with respect to the ratio of crossmodal training samples until the supervised percentage is brought down heavily. Achieving similar error rates on the range from one fifth of training samples to all of them being trained in a supervised fashion also shows that for PTAE the learning of language-to-action translation reaches a plateau, where added labels do not provide additional useful information. After reducing the supervised ratio further, it can be seen that the error rate gradually increases, albeit only just over 4% for PTAE, when only 7 samples are used for the supervised signals. Overall, these results indicate the clear superiority of Transformer-based multimodal fusion over a simpler attention mechanism by the GMU in terms of performance and efficiency. Although it is relatively larger than PGAE, PTAE is trained much faster and reaches a global optimum in less than half of the training iterations of PGAE. It is clear from these results that scaled dot-product attention, which forms the backbone of the CMT, can work with a low proportion of supervision during training, whereas gated attention, which is used by GMU, requires a much larger supervised proportion to learn the crossmodal mapping between action and language. The CMT utilises a relatively long set of matrix operations over all time steps (temporal information is kept until the extraction of the representation vector), while the GMU relies on simpler equations over the mean input features that no longer bear a temporal dimension.

When introducing a conflicting modality input during testing, we observed an



asymmetry in that a conflicting action input leads to a larger disturbance than a conflicting language input. One possible reason is that the CMT architecture is asymmetric; as input, we are using the action input as two input vectors ( $K$  and  $V$ : keys and values), whereas the language is used as one input vector ( $Q$ : queries). This setting was chosen because the opposite setup (with action as queries) was found less performant. Our setup can be interpreted as language-conditioned action attention. A computationally more expensive architecture could combine both asymmetric setups, as has been done for learning vision and language representations [104].

Another possible reason for the larger impact of a conflicting action could be that the action input combines two submodalities, vision and proprioception, and hence involves more information than the language input. However, limiting the conflict to one of the submodalities did not completely remove the asymmetry as introducing the conflict only in one action submodality (vision or proprioception) still had a stronger effect on the model performance than a conflicting language input. Unlike language, vision contains the complete information to perform a task. Consider the example “pull red slowly” for language-to-action translation. Here, the language lacks any information about whether the object is on the left or right side, so the agent can only execute this correctly by taking the visual input into account during action execution. In contrast, in the opposite direction (action-to-language translation) and action repetition, the visual input contains all the necessary information.

## 6.5 Conclusion

In this chapter, we have introduced a paired Transformer-based autoencoder, PTAE, which we train largely unsupervised with additional but reduced supervision. PTAE achieves significantly better action-to-language and language-to-action translation performance under limited supervision conditions compared to the previous GMU-based model PGAE. Furthermore, we have tested the robustness of our new approach against contradictory extra input. In line with the concept of incongruence in psychology, these experiments show that conflict deteriorates the output of our model and more conflicting features lead to higher interference. We have also found an asymmetry between the action and language modalities in terms of their conflicting impact: the action modality has significantly more influence over the model performance regardless of the main output modality.

Our novel bidirectional embodied language learning model is flexible in performing multiple tasks and is efficient and robust against the dearth of labelled data. Hence, it is a step towards an autonomous agent that can communicate with humans while performing various tasks in the real world. In [Chapter 8](#), we substitute the language encoder and decoder of PTAE with a pre-trained Transformer-based LLM to tackle more diverse natural language descriptions, resulting from more actions and objects on the table.

Hitherto, the approaches introduced in this thesis (PVAE, PVAE-BERT,

PGAE and PTAE) use teacher trajectories to learn action-language mappings in a supervised fashion. They are unable to observe and act in the environment due to their unsuitable architectures that require observing the entire sequence before outputting language or action. To address this issue, in [Chapter 7](#), we will expand our approach with RL to reduce the need for expert-defined action trajectories. Specifically, we will introduce the Crossmodal Bidirectional Transformer (XBiT) model aimed at tackling generalisation to the action space and continuous object positions by implementing a two-stage SL pre-training and RL fine-tuning learning paradigm. After regular SL training on labelled supervised trajectories, XBiT employs a policy-gradient-based online RL fine-tuning method, using sparse rewards as direct feedback from the environment. Experiments show that RL fine-tuning enhances performance in language-to-action translation by exploring more dexterous object manipulation with diversified action trajectories while maintaining the perfect action-to-language performance.



---

# Asymmetrical Combination of Learning Paradigms for Superior Action Execution

---

Modern human-robot interaction requires a robot to possess sufficient language and physical skills. Supervised learning (SL) is a powerful paradigm for acquiring a combination of these skills using aligned language-action data. However, training data for robot actions in continuous high-dimensional environments is often limited in quality or quantity. On the other hand, reinforcement learning (RL) is well suited for action learning but impractical for learning language-action associations. To overcome these shortcomings, we introduce the Crossmodal Bidirectional Transformer (XBiT) model which combines SL pre-training and RL fine-tuning in robotic object manipulation. It first uses SL on labelled samples to learn action-language mappings and coarse motions. Next, via RL, it explores the environment and uses reward feedback to improve dexterity and execute fine-grained actions. The multimodal XBiT architecture integrates language, vision and proprioception, and it can perform actions instructed by language as well as describe them in language. Our tabletop experiments show that with limited labelled data, RL fine-tuning results in fast and significant improvements in action execution compared with SL-only training. More specifically, the improvement by RL is limited in object selection but significant in action precision. We conclude that language-enabled robots benefit from versatile neural architectures, combining multiple learning paradigms<sup>1</sup>.

## 7.1 Introduction

The human brain employs complementary learning mechanisms as its different regions specialise in distinct types of learning. According to Doya [39, 40], the basal ganglia are responsible for selecting actions by assessing candidate actions through reward-based RL, whereas the cerebellum controls muscles via error-based SL. On

---

<sup>1</sup>The code belonging to this chapter is available at <https://github.com/oo222bs/XBiT>.

the other hand, the outermost cerebral cortex is tasked with modelling a concise representation of the sensory state, guided by the statistical properties of the input, analogous to unsupervised learning. Inspired by these biological phenomena, comparable learning mechanisms can be implemented in robotics. RL is able to produce goal-directed robotic action sequences from random exploration, but interacting with the robot by language is non-trivial to realise. Supervised models, meanwhile, have recently made substantial progress in grounding language in robotic action, but their training requires large amounts of action sequence data, which is not always available. In addition, unsupervised learning can be used to efficiently learn state representations from unlabelled data. However, it falls short when fusing multiple modalities, which is integral to full-fledged grounded robot learning. In the human brain, the basal ganglia and the cerebellum complement each other in motor control and cognitive functions, and both have recurrent connections with the cerebral cortex. These three brain regions collaborate on everyday tasks; the cerebellum is utilised as an internal model of the environment, the cerebral cortex provides the state representations of the environment, and the basal ganglia enable action selection through evaluation of these environmental states [39, 40]. In a similar manner, to combine the merits of SL with RL for neuro-inspired robot learning, we introduce a combined supervised and RL training paradigm. Besides, we extract visual state representations with an auto-encoder-based vision encoder which is pre-trained on training demonstrations in an unsupervised fashion. As depicted in [Figure 7.1](#), we first train our model, XBiT, via SL with collected demonstrations. After this pre-training process, we continue to train XBiT on live demonstrations in simulation, using RL.

A cognitive robot that can interact with humans should be able to tackle robotic action-language translation bidirectionally, i.e. language-to-action and action-to-language translation. It is inadequate for a robot to merely perform actions according to given descriptions by humans. Effective communication is vital for a reliable autonomous agent designed for successful collaboration with humans [63]. To enable seamless communication between a human and a robot, besides possessing language understanding skills, the robot should be able to produce language, e.g. describe actions. Although it may seem appealing to use a separate model for each translation direction, a model that can both execute an action based on a given instruction and describe a given action in language is preferable to those models that can only do either of these tasks. Similar to multi-task learning approaches [78, 159, 175] that are trained on multiple robotic object manipulation tasks in parallel, learning one translation direction may help learn the other direction due to the multimodal nature of the robotics domain. As there are many ways to benefit from transfer learning [165], each action-language translation direction can be treated as a single task learnt in parallel with the other. This approach also aligns with the biological example of human learning, where prior knowledge is utilised when acquiring a new skill instead of learning each task from scratch independently.

As a multimodal approach that integrates language, vision and proprioception, XBiT is a bidirectional model that can execute actions according to instructions

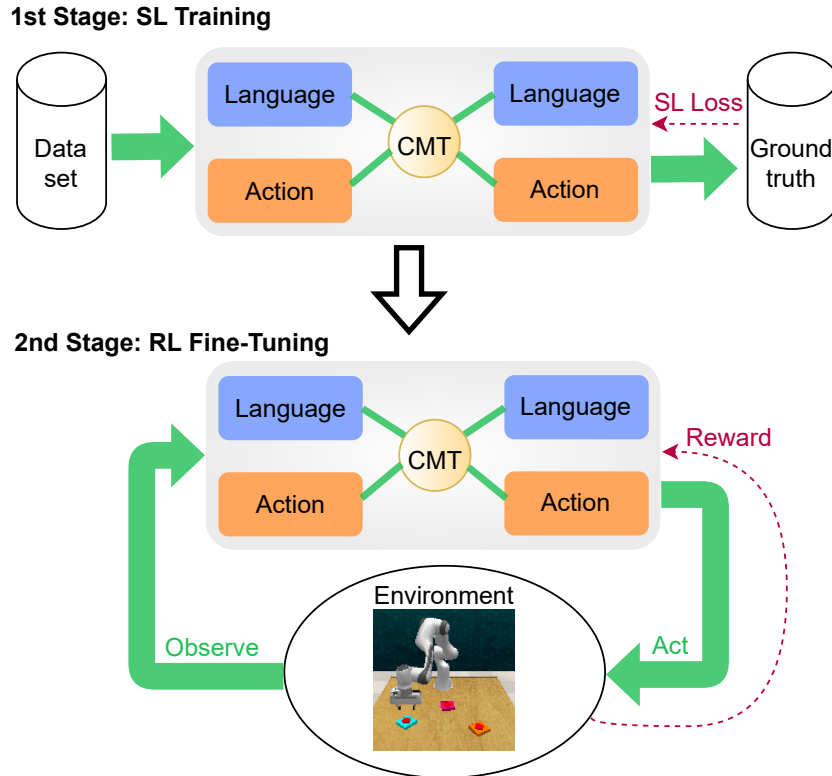


Figure 7.1: The two training stages of XBiT. XBiT consists of language and action streams integrated by the Crossmodal Transformer (CMT). First, we train XBiT on a dataset using supervised learning (SL) until convergence. Second, action performance improves via exploration in simulation using reinforcement learning (RL).

from humans with randomly initialised object positions as well as describe its own actions. Since SL-only training cannot generalise to the action space given a limited number of training trajectories, we employ RL to fine-tune XBiT. We train our model with a certain number of samples in a supervised fashion and then let a policy gradient-based RL algorithm (i.e. REINFORCE [181]) explore the action space further online and fine-tune on the pre-trained weights for only a mere fraction of supervised training epochs. As a result of pre-training and fine-tuning, XBiT can model bidirectional translation between dexterous object manipulation actions and language by combining a relatively simple crossmodal Transformers approach with RL. As visualised in Figure 7.2, to test our approach, we choose the PushButton, PushButtons, SlideBlockToTarget and PickUpCup tasks from the RL Bench learning environment [77], which uses the CoppeliaSim simulation environment [149]. Apart from the PushButton task, these tasks require the skill of language-defined target object selection among distractor objects. Moreover, all four tasks, including the PushButton task, are challenging for SL due to the continuous positioning of objects and the need for interpolation. We compare the results of the pure SL approach with the results of the RL fine-tuning. The results show

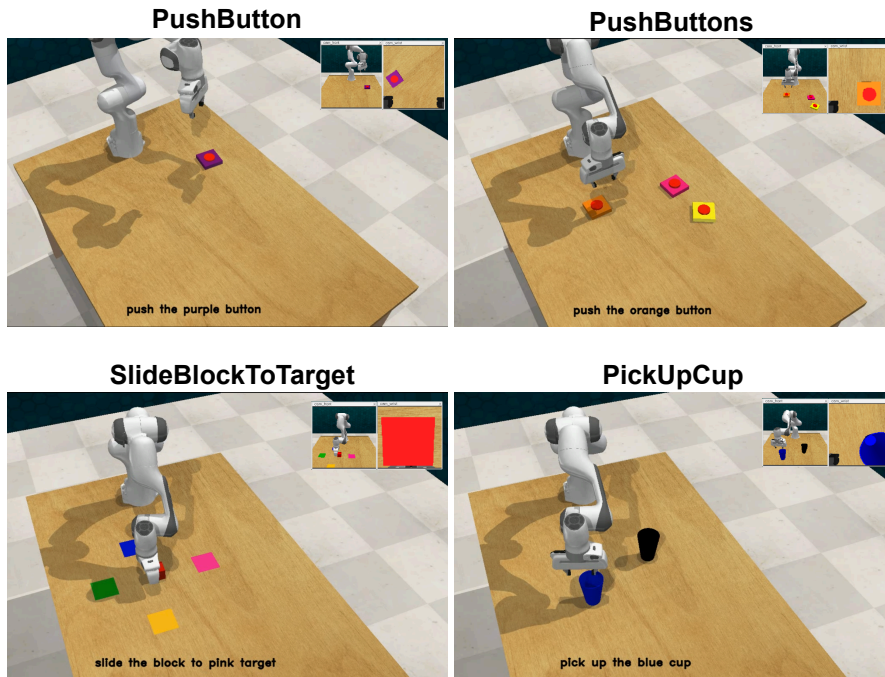


Figure 7.2: The scenario in simulation with action examples. The performed RL-Bench tasks are shown from left to right and top to bottom: PushButton, PushButtons, SlideBlockToTarget and PickUpCup. The Franka Panda robotic arm is employed to execute actions according to given language instructions. As can be seen from the insets, the front and wrist cameras are used for observation.

that fine-tuning with RL helps the model improve its action output, especially in terms of precision, with exploration to accomplish the given robotic manipulation task, whereas the purely supervised approach fails to generalise to different object positions in the action space and underperforms in terms of action execution accuracy. To the best of our knowledge, XBiT is the first model performing bidirectional robotic action-language translation by employing a two-stage SL and RL training paradigm. Therefore, the introduction of XBiT is valuable towards achieving general artificial intelligence. Our main contributions with XBiT are as follows:

1. introducing a novel flexible model architecture that can be trained with SL and RL,
2. using the limited available teacher trajectories to model the basics of action execution,
3. combining our approach with RL fine-tuning for closed-loop action execution in the environment,
4. showing that RL fine-tuning is more efficient and performant than further training in a supervised fashion.



The remainder of this chapter is structured as follows: in the next section, we briefly discuss related approaches in RL fine-tuning. In [Section 7.3](#), we detail our approach and its training scheme. [Section 7.4](#) shows our experiments and results, while [Section 7.5](#) discusses these results. Finally, we conclude the chapter with [Section 7.6](#).

## 7.2 Background: Fine-Tuning with Reinforcement Learning

Recently, the domain of robotic object manipulation has witnessed a great deal of progress [[41](#), [78](#), [79](#), [146](#), [160](#), [175](#)] with the advent of foundation models [[32](#), [35](#), [38](#), [140](#)] in language and vision. While these approaches perform well on generalisation, multi-task learning and robotic dexterity, they are large models primarily trained with supervised learning (SL), demanding vast amounts of labelled data. As data is not always available on a large scale, a complementary approach could be utilising reinforcement learning (RL). Specifically, RL can be used in the fine-tuning phase of learning [[129](#), [145](#)]. In the upcoming subsections, we will elaborate on RL fine-tuning with various examples, involving LLM training, RL pre-trained models and approaches pre-trained with imitation learning, i.e. behavioural cloning (BC).

Fine-tuning a pre-trained model with RL has been experimented with in different domains. For example, the video pre-training model VPT [[15](#)] learns to control a Minecraft agent by first pre-training on videos and then fine-tuning with RL in the environment. While the Minecraft agent trained from scratch with RL can barely do any meaningful actions, the RL-fine-tuned agent achieves human-level performance. Here, we review LLM fine-tuning, fine-tuning RL pre-trained methods and fine-tuning BC pre-trained methods.

### 7.2.1 Large Language Model (LLM) Fine-Tuning

Recent examples of combining SL with RL fine-tuning are LLMs. For example, ChatGPT [[129](#)] is pre-trained unsupervised on large amounts of text. In the fine-tuning phase, it first learns from labelled data in a supervised fashion. Afterwards, human testers rank model outputs and shape a reward model. At last, ChatGPT is fine-tuned by using the PPO algorithm [[156](#)] based on the rewards provided by the reward model. This fine-tuning process is coined as reinforcement learning from human feedback (RLHF). Similar to LLM fine-tuning, RLHF is also applied to text-to-image generation by DPOK [[46](#)], which outperforms SL fine-tuning in terms of generated image quality and alignment to text. RLHF optimises the model outputs so that they conform to human preferences. In contrast, XBiT seeks to avoid overfitting to the SL training data by generalising to the continuous action space.

## 7.2.2 Reinforcement Learning Pre-training

There exist approaches that use RL for both the pre-training and fine-tuning stages in the robotics field and beyond. For instance, Feng et al. [47] fine-tune a model-based RL algorithm (TD-MPC [60]), which is pre-trained with offline RL, with a relatively small amount of online data for visuomotor robot control. Likewise, in an analysis of online RL fine-tuning for offline RL pre-trained models, Luo et al. [105] conclude that a conservative online RL fine-tuning policy leads to stable and sample-efficient learning. Besides, RL fine-tuning can also adapt a robotic manipulation policy to variations in backgrounds, lighting conditions and object shapes [83].

Adeniji et al. [4] propose the LAMP (Language Reward Modulated Pretraining) approach that distinctively leverages a VLM for reward design in robotic object manipulation. In the pre-training stage, an off-policy model-based RL algorithm, MWM (Masked World Models) [157], learns a language-conditioned task-independent policy based on rewards generated by the VLM. Afterwards, this policy is fine-tuned on downstream tasks with sparse rewards from the environment.

## 7.2.3 Supervised Pre-training

PixL2R [52] is an example of two-stage SL-RL training for robotic object manipulation. Via SL, it first trains a network to align action trajectories with corresponding language commands. In the second phase, a separate RL policy is trained to act in the environment, receiving extrinsic rewards alongside intermediate rewards generated based on the priorly learnt associations between action and language. However, PixL2R does not constitute an RL fine-tuning approach as the RL policy learnt in the second phase is different from the model trained on action-language mappings in the first phase – the weights of the model learnt in the first phase are not modified and the model is merely used to partially guide the RL policy in the second phase.

JIRL (joint imitation-reinforcement learning) [36] framework gradually switches from BC to RL to boost performance while avoiding diverging excessively from the pre-trained policy, which may cause forgetting previously learnt skills. PIRL-Nav [145] employs imitation learning to bootstrap RL for navigation tasks. BC is used to pre-train the navigation policy on 77K human demonstrations, which is then further trained with the DD-PPO (Decentralised Distributed Proximal Policy Optimisation) [180] on-policy RL algorithm.

Similar to PIRLNav, our XBiT approach employs an RL fine-tuning paradigm. However, our approach is able to output not just actions but also language descriptions. To the best of our knowledge, XBiT is the first of its kind to exploit RL fine-tuning for an SL-pre-trained model in language-instructed robotic object manipulation.

## 7.3 Proposed Method: Crossmodal Bidirectional Transformer (XBiT)

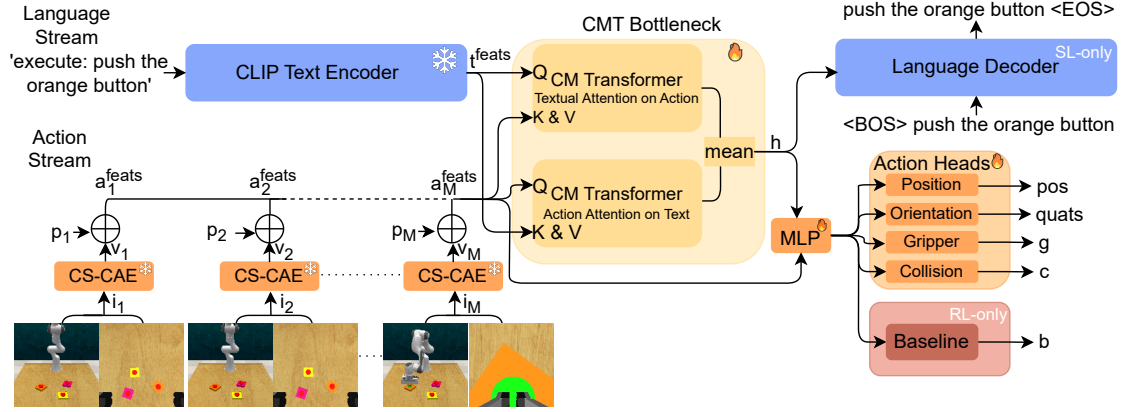



Figure 7.3: The XBiT architecture. The inputs are the signal word, i.e. ‘execute:’ or ‘describe:’, and a language description (only for the ‘execute’ signal) and a sequence of images and robotic arm poses, while the outputs are a description and the target robotic arm poses. A pre-trained CLIP text encoder embeds language input into text features  $t^{feats}$ . Visual features are extracted via the channel-separated convolutional autoencoder (CS-CAE) and concatenated ( $\oplus$ ) with the arm poses,  $p$ , to form the action input. Together, the action features,  $a^{feats}$ , are then passed to the CM Transformer (CMT) bottleneck, which fuses information from the two streams. The output of the CMT is the common representation vector  $h$ . The language decoder is a Transformer decoder, while the action decoders (heads) are individual feed-forward layers. The baseline unit is used to reduce variance during RL fine-tuning. Apart from the language decoder (only trained during SL) and the baseline unit (only trained during RL), the whole network is used during SL pre-training and RL fine-tuning, and the components marked with  are trained with SL and RL.

XBiT models bidirectional action-language translation (Fig 7.3). It consists of a CLIP text encoder, convolutional autoencoder-based vision encoder, Crossmodal Transformer (CMT), Transformer-based language decoder and fully connected (FC) action decoder layers. Following the PGAE and PTAE models [134, 135] (see Chapters 5 and 6), we use signal words as part of the language input to direct the model in the desired translation: action-to-language (‘describe’) and language-to-action (‘execute’) translations. XBiT performs these translations individually according to the given signal word, i.e. ‘describe’ or ‘execute’.

### 7.3.1 Language Encoder

The language encoder embeds the language input to extract the meaning of the instruction provided by the user. Following prior works [108, 159, 160], we use

the pre-trained CLIP Text Encoder [140] as our language encoder. The language input is fed to the network as text strings and then converted into a vector by the language encoder. Apart from the action instruction, the language input includes the signal word, which is either 'execute' or 'describe'. When 'describe' is chosen as the signal, the language input does not contain an action instruction. The CLIP text encoder embeddings are forwarded as language features,  $t_{\text{feats}}$ , to the CMT. Note that the choice of the CLIP text encoder as the language encoder is due to its pre-training with an image encoder aimed at pairing vision and language, which shows similarities with XBiT, but it can be replaced by any other pre-trained language encoder such as BERT [35], as demonstrated in our previous PVAE-BERT and PGAE approaches (see Chapters 4 and 5).

### 7.3.2 Action Encoder

The action input consists of two modalities: proprioception and vision. The proprioceptive observation,  $p$ , contains the end-effector poses, gripper information and collision avoidance signal. The visual observation consists of RGB images captured by two cameras in the environment: the front camera, which captures the view across the table, and the wrist camera, which captures the view over the gripper. For encoding the visual input, as it is done with our previous models, we first train a channel-separated convolutional autoencoder (CS-CAE) [136] in an unsupervised fashion on training images. We then freeze the CS-CAE and extract visual representations,  $v$ , from its middle layer during the training of XBiT. The visual representations from both cameras and the proprioceptive input are concatenated for each time step. This combined representation forms the action features,  $a_{\text{feats}}$ , which are fed to the CMT bottleneck.

### 7.3.3 Crossmodal Transformer Bottleneck

The Crossmodal Transformer (CMT) bottleneck fuses multiple modalities by applying crossmodal attention. It involves two CMT blocks in order to align language and action streams symmetrically, similar to the ViLBERT architecture [104]. In the top block (Figure 7.3, Bottleneck), language features are used as queries (Q), while action features serve as keys (K) and values (V). This kind of attention application can also be coined as *language-conditioned action attention*, as each language feature attends to all action features along the temporal dimension. It calculates how valuable the features of each pair of an image and proprioceptive input for a word in the language input are. In the bottom block (Figure 7.3), we do the opposite and use action features as Q, while textual features are fed to the CMT as K and V vectors. We call this *action attention on language*. In the end, the outputs of both CMT blocks are averaged over the batch dimension to produce the hidden representation vector  $h$ :

$$h = \mu (\text{CMT}^{\text{ta}}(t_{\text{feats}}, a_{\text{feats}}), \text{CMT}^{\text{at}}(a_{\text{feats}}, t_{\text{feats}})) \quad (7.1)$$

where  $\mu$  stands for mean averaging,  $\text{CMT}^{\text{ta}}$  and  $\text{CMT}^{\text{at}}$  stand for the respective CMT blocks,  $t_{\text{feats}}$  are the language embeddings, while  $a_{\text{feats}}$  are action features that involve visual features and proprioceptive input. For our experiments, we employ a single-layer CMT with 4 parallel attention heads for each block. Irshad et al. [75] detail the internal structure of a CMT block – see also Section 6.2.1 in the previous chapter.

### 7.3.4 Language Decoder

The language decoder outputs the description of a given action by using the common hidden representation  $h$ . We use a Transformer-based language decoder that learns to output the correct language description during training, where masking is employed to prevent looking ahead at the future tokens as in the original Transformer [173]. Due to its Transformer-based architecture, the language decoder outputs the descriptions word by word autoregressively during inference. Like each CMT block, the language decoder is a single-layer Transformer with 4 attention heads.

### 7.3.5 Action Decoder

The action output is the target end-effector position alongside the orientation, gripper condition and collision avoidance signal according to the current observation and language input. The common hidden representation  $h$  is projected with a multi-layer perceptron (MLP) to be used as input to the action decoder. For each type of action output, we use an FC layer as a head. The position head outputs the absolute position values,  $(\text{pos}_x, \text{pos}_y, \text{pos}_z)$ , in Cartesian space. The orientation head outputs the orientation values,  $(\text{quat}_x, \text{quat}_y, \text{quat}_z, \text{quat}_w)$ , in quaternion space. The gripper head outputs  $g$ , whether the gripper should be closed or open at the next step. Finally, the collision head provides the binary output of  $c$ , whether the robotic arm should avoid collision or not. We use the sigmoid activation function for the gripper and collision outputs.

Following PERACT [160], the practical execution of the action relies on a motion planner provided by the RL Bench environment. The motion planner tries to find an accurate path for the target action by using inverse kinematics or sample-based planning solutions. After the path is found, the robotic arm is moved to the desired pose.

### 7.3.6 Supervised Learning Loss Functions

During supervised training, we use two loss functions to calculate the deviation of the model output from the ground-truth language descriptions and end-effector poses. The language loss,  $L_{\text{lang}}$ , is defined as the cross entropy between target ( $x$ )

and output ( $y$ ) words:

$$L_{\text{lang}} = \frac{1}{N-1} \sum_{t=1}^{N-1} \left( - \sum_{i=0}^{V-1} x_{t+1}^{[i]} \cdot \log y_t^{[i]} \right), \quad (7.2)$$

where  $V$  is the vocabulary size and  $N$  is the number of words per description. The action loss,  $L_{\text{act}}$ , is a combination of the L1 loss on the gripper pose (including XYZ positions and orientation quaternions) and binary cross entropy loss on gripper open and collision outputs:

$$L_{\text{pos}} = \frac{1}{M-1} \sum_{t=1}^{M-1} \|\widehat{\text{pos}}_t - \text{pos}_t\|, \quad (7.3)$$

$$L_{\text{quat}} = \frac{1}{M-1} \sum_{t=1}^{M-1} \|\widehat{\text{quat}}_t - \text{quat}_t\|, \quad (7.4)$$

$$L_g = -\frac{1}{M-1} \sum_{t=1}^{M-1} (g_t \cdot \log(\hat{g}_t) + (1 - g_t) \cdot \log(1 - \hat{g}_t)), \quad (7.5)$$

$$L_c = -\frac{1}{M-1} \sum_{t=1}^{M-1} (c_t \cdot \log(\hat{c}_t) + (1 - c_t) \cdot \log(1 - \hat{c}_t)), \quad (7.6)$$

where  $M$  is the sequence length for action trajectories, while hats ( $\hat{\cdot}$ ) over variables denote model outputs. Following BC-Z [78], the weighting for the action output loss is 1 for positions, 0.1 for quaternions and 0.005 for gripper open and collision avoidance, which results in the following action loss:

$$L_{\text{act}} = L_{\text{pos}} + 0.1 \cdot L_{\text{quat}} + 0.005 \cdot (L_g + L_c). \quad (7.7)$$

The total loss is the sum of the language and action losses:

$$L_{\text{all}} = \alpha L_{\text{lang}} + \beta L_{\text{act}}, \quad (7.8)$$

where  $\alpha$  and  $\beta$  are weighting factors. In our experiments, considering the relative difficulty of producing language and action, we take  $\alpha$  as 1 and  $\beta$  as 10, which yields comparable loss values in magnitude for the modalities. Setting both  $L_{\text{lang}}$  and  $L_{\text{act}}$  to 1 resulted in the degradation of the action performance. Conversely, setting  $\beta$  to 100 led to poor language performance.

### 7.3.7 Reinforcement Learning Fine-Tuning

We employ the policy gradient-based REINFORCE algorithm [181] to update the weights of our model during fine-tuning. REINFORCE trains an actor, for which we use the XBiT architecture, and a baseline, for which we add a separate FC layer. The actor acts based on the current observation and language instruction and receives sparse rewards from the environment. The baseline tries to model the

rewards by estimating the future rewards based on the current observation and the action produced by the actor.

As a policy gradient method, REINFORCE directly searches for the optimal policy. It uses gradient ascent to update the policy weights in order to maximise the expected return. For each episode,  $E = \{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\}$ , encountered following the policy,  $\pi_\theta$ , where  $s$ ,  $a$  and  $r$  are states, actions and rewards respectively, REINFORCE tries to maximise its discounted return,  $G_k \leftarrow \sum_{t=k+1}^T \gamma^{t-k-1} r_t$ , by optimising the objective function,  $J_\theta = E_\theta \left[ \sum_{t=0}^T G_t \right]$ . The weights of the policy are updated as follows with the learning rate  $\alpha$  at the end of every episode:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \quad (7.9)$$

$$\text{where } \nabla_\theta J(\theta) = \sum_{t=1}^{T-1} \nabla_\theta \log(\pi_\theta(a_t|s_t)) G_t. \quad (7.10)$$

As our approach works in continuous domains, during training, we let the algorithm explore on top of action values,  $a$ , predicted by XBiT by employing a Gaussian distribution with a fixed standard deviation:  $\mathcal{N}(a, 0.01)$ . To reduce variance [120], we subtract the baseline prediction,  $b$ , from the discounted return,  $G$ , before we calculate the objective function,  $J_\theta$ .

### 7.3.8 Training Details

We train XBiT end-to-end with three modalities in both action-to-language and language-to-action directions. Considering the relative difficulty of the two translation directions, we randomly choose in which direction to train XBiT with a ratio of 4:1 in favour of the language-to-action translation. We freeze the CS-CAE and CLIP Text Encoder layers and train the rest of the network. In the initial SL phase, XBiT is trained for 20,000 epochs via gradient descent using the Adam optimiser [89]. Following Irshad et al. [75], we set  $h$  with 256 dimensions. The language vector,  $x$ , has 24 dimensions, whereas the proprioception vector,  $p$ , has 9 dimensions. The longest sequence is 39 time steps, while the shortest is 14 in the dataset collected in advance to pre-train XBiT in a supervised manner. We take the learning rate as  $10^{-5}$  with a batch size of 8 samples after determining them as optimal hyperparameters. In total, XBiT has approximately 67M parameters with only about 3M of them trained while the rest are frozen.

After supervised training, we further train the model (except the language decoder) with the REINFORCE algorithm. This training is conducted online in a simulation environment with sparse rewards. The baseline layer is initialised with all zeros. At the end of each episode, our agent receives a reward of 1.0 only if it completes the action successfully. Otherwise, it receives zero rewards. We set the discount factor  $\gamma$  to 0.9, which empirically led to robust learning.



Table 7.1: Dataset Splits Per Task

Task	Training Set	Validation Set	Test Set
PushButton	1,000	25	25
PushButtons	10,000	100	150
SlideBlockToTarget	5,000	100	150
PickUpCup	10,000	100	150

## 7.4 Experiments and Results

We perform the PushButton, PushButtons, SlideBlockToTarget and PickUpCup tasks from RLbench [77]. The first two tasks include either one button (PushButton) or three differently coloured buttons (PushButtons) situated on the table, and the Franka robotic arm is used to press the buttons. The buttons have a red top plate, which turns green after being pushed by the gripper. The PushButton task involves one of 18 buttons, each of which is of a different colour, placed on the table per episode. The PushButtons task involves three of 6 differently coloured buttons placed on the table per episode. The language instruction specifies the target button by naming its colour. The buttons are randomly placed in continuous positions on the table. The SlideBlockToTarget task involves a red block which needs to be moved to the correctly coloured target area on the table for the successful completion of an episode. The four differently coloured patches (blue, green, pink and yellow) are positioned randomly on four sides of the arbitrarily placed red block. The PickUpCup task requires one of the two coloured cups placed on the table to be grasped and lifted based on the language instruction. Each cup is of one of 8 colours and the two cups cannot be of the same colour in an episode. We generated a separate dataset for each task, involving successfully completed episodes exclusively. Table 7.1 lists the dataset splits for the four tasks. The training set sizes have been decided in terms of the varying difficulties of the four tasks. Accordingly, the simplest task of PushButton has the lowest number of training episodes, whereas the more challenging PushButtons and PickUpCup tasks include the most training samples. Our dataset is substantially smaller compared with the approaches that need hundreds of thousands to over a million training episodes [21, 22, 146, 175]. The sequences of  $128 \times 128$ -pixel images make up most of the dataset sizes, which sum up to 29.4 GB, and generating the data in simulation took approximately 106 hours. SL pre-training alone on each dataset takes a few days up to two weeks on 8 Nvidia Quadro RTX 6000 GPUs, depending on the dataset size. Therefore, a further upscaling of training data would be expensive concerning data generation and training time.

We train our model in both language-to-action and action-to-language translations using the ‘execute’ and ‘describe’ signals respectively. The signal is prepended to the language input, which determines the direction of the action-language trans-

lation. For each training iteration, XBiT is trained in one of the translation directions, with approximately 80% of the iterations in the more challenging language-to-action direction. After training XBiT in both translation directions through SL on training data, we fine-tune it only in the language-to-action direction via direct rewards from the environment using the REINFORCE algorithm for 1,000 episodes. In the following subsections, we report XBiT’s performance for action-to-language and language-to-action translations before and after RL fine-tuning.

### 7.4.1 Action-to-Language Translation

For XBiT to translate from action to language, we pass the ‘describe’ signal as a single word in the language input, excluding the ground-truth language description, together with the sequence of images and proprioception. For the PushButton and PushButtons tasks, the language output is considered correct when it describes the correct button based on its colour, e.g. “push the blue button <EOS>”, where <EOS> denotes the end-of-sentence tag. For the SlideButtonToTarget task, the language output must correctly denote the colour of the area the block is pushed towards, e.g. “slide the block to pink target <EOS>”. Finally, the PickUpCup language output must name the colour of the cup being lifted, e.g. “pick up the black cup <EOS>”. Apart from the colour word, all language descriptions are identical, following the “push the *colour* button”, “slide the block to *colour* target” and “pick up the *colour* cup” templates. We use sentence accuracy as the performance metric for action-to-language translation and only accept a language output as accurate when all output words match with the ground truth.

For all training checkpoints starting from 1K epochs until the final checkpoint, we achieve 100% action-to-language translation accuracy on the test sets of all four tasks, as XBiT describes all of the 475 episodes correctly. For the PushButton task, this may seem trivial as there is only a single button on the table, but for the other tasks, the model has to distinguish the target from the distractors (the target button from the other two buttons on PushButtons, the target area from the other three areas on SlideBlockToTarget and the target cup from the other cup on PickUpCup). Furthermore, for each checkpoint, we observe no performance drop after the RL fine-tuning stage which trains the model only in the language-to-action direction, as the fine-tuned model perfectly describes all actions in the test sets.

### 7.4.2 Language-to-Action Translation

We test XBiT after supervised training and RL fine-tuning on language-to-action translation in the simulation environment. During testing, we give a language instruction such as “push the orange button” with the ‘execute’ signal prefix and the current observation including two camera views and proprioceptive input. XBiT then outputs the next target pose accordingly, which triggers the motion planner to move the gripper to the desired location. The action is deemed correct only if the gripper successfully completes the task according to the task-specific success

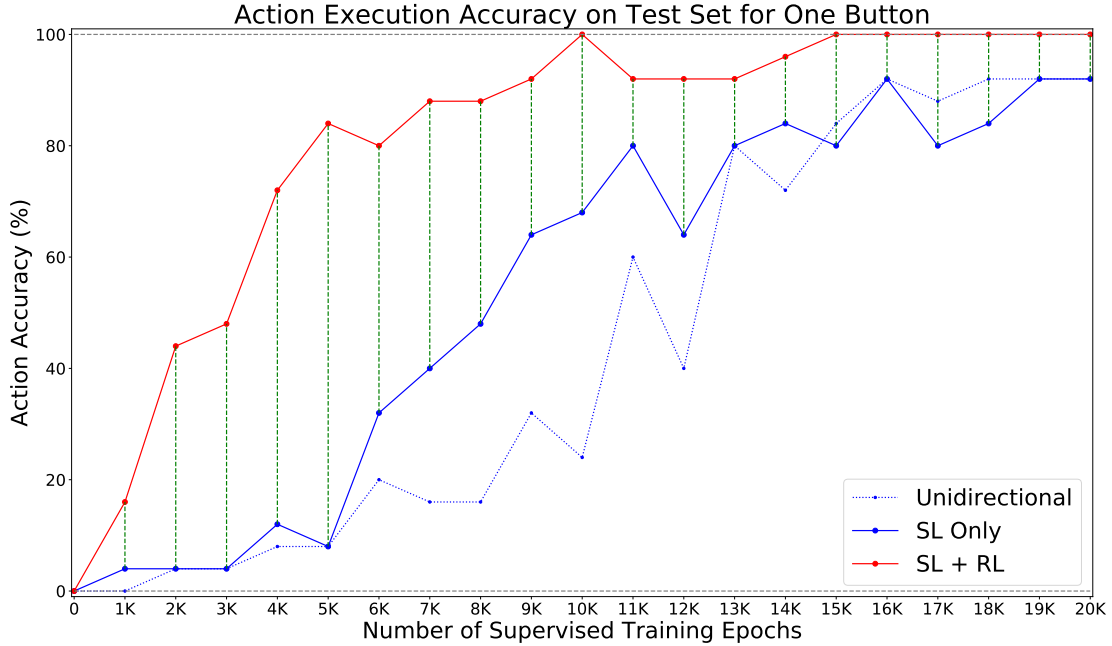


Figure 7.4: Action execution performance for the PushButton task (involving a single button) across different training epochs. The unidirectional XBiT is trained only in the language-to-action direction, with SL only.

metric such as pressing the target button causing its top plate to turn green or lifting the target cup up to a certain height.

Figure 7.4 shows the action accuracies of XBiT on the test set of the PushButton task across every 1000th training epoch before and after RL fine-tuning. As an ablation study, we first experiment with a unidirectional version of XBiT, trained only to do language-to-action translation, and compare it against the proposed bidirectional XBiT. Both models are trained using SL without RL fine-tuning. The blue thin dashed curve of the unidirectional model displays significantly slower learning than the solid blue curve of the bidirectional model. This indicates that bidirectionality does not impede but rather facilitates action learning.

When comparing RL fine-tuned XBiT with the SL-only XBiT in Figure 7.4, we can see that RL fine-tuning improves the action performance significantly and even leads to a perfect action completion for all checkpoints from 15K epochs onwards, including the 10K epoch checkpoint. With SL alone, XBiT does not achieve perfect precision, thus not reaching its full potential. Moreover, bootstrapping with RL does not lead to any successful episodes (see 0th checkpoint) – this can be regarded as an ablation experiment where XBiT is trained with RL only. However, on top of SL pre-training, RL fine-tuning boosts accuracies consistently across different training epochs, with the largest improvements in the intermediate SL stages between 2K and 10K epochs and the highest performance boost of over 75% for the 5K checkpoint. In order to determine if we can reach perfect performance with RL fine-tuning early on, without longer SL training, we also test the performance of

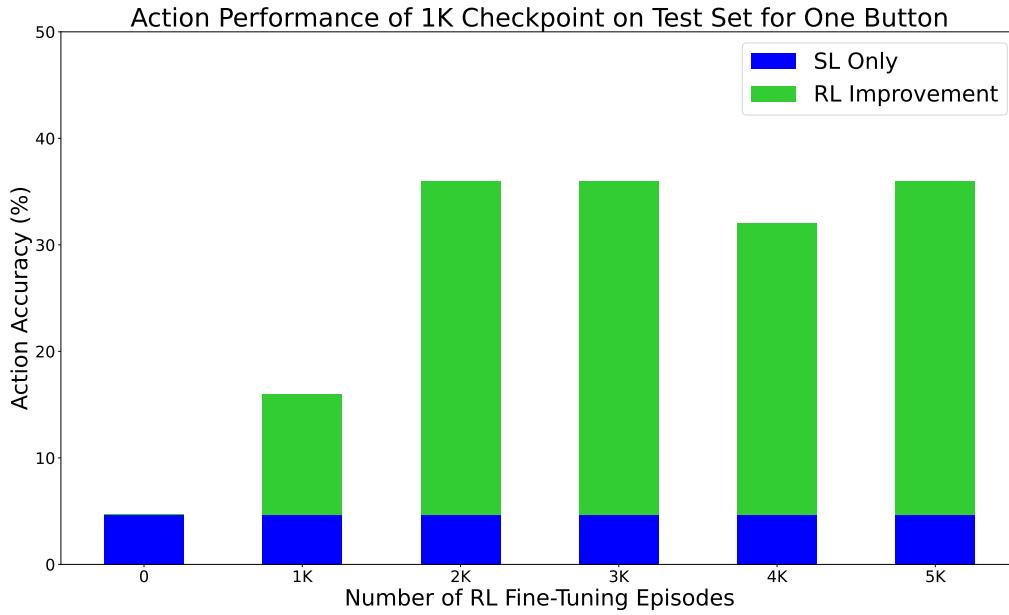


Figure 7.5: Action execution performance on the PushButton task after longer RL training for the 1K-epoch-SL-trained model.

1K epoch checkpoint after up to 5,000 RL episodes (Figure 7.5). The results show that while the performance improves at first, the resulting model cannot reach a high success rate, as the accuracies saturate around the one-third level.

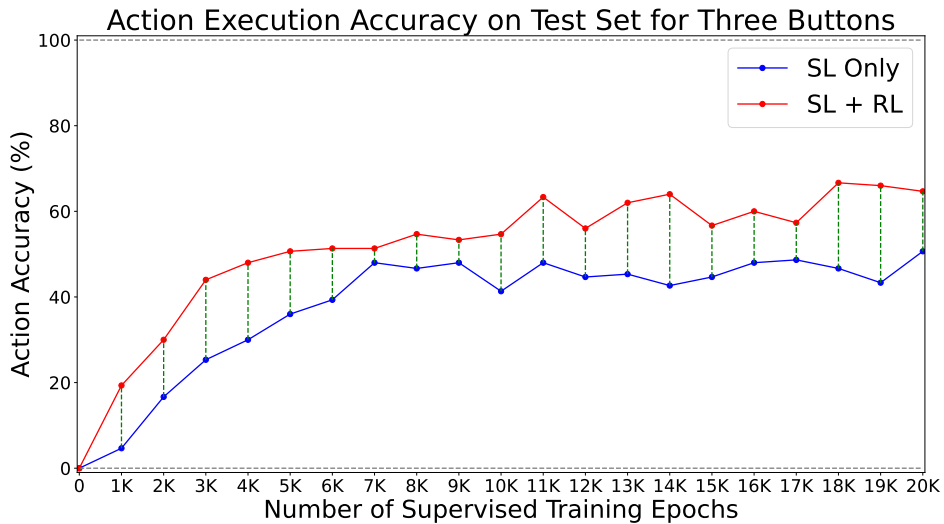


Figure 7.6: Action execution performance on the PushButtons task (involving three buttons) across different training epochs.

Figure 7.6 shows the action execution performance in the three-button setting (PushButtons task). As can be seen, RL fine-tuning consistently improves the action accuracies of XBiT starting from earlier checkpoints until the last checkpoint.

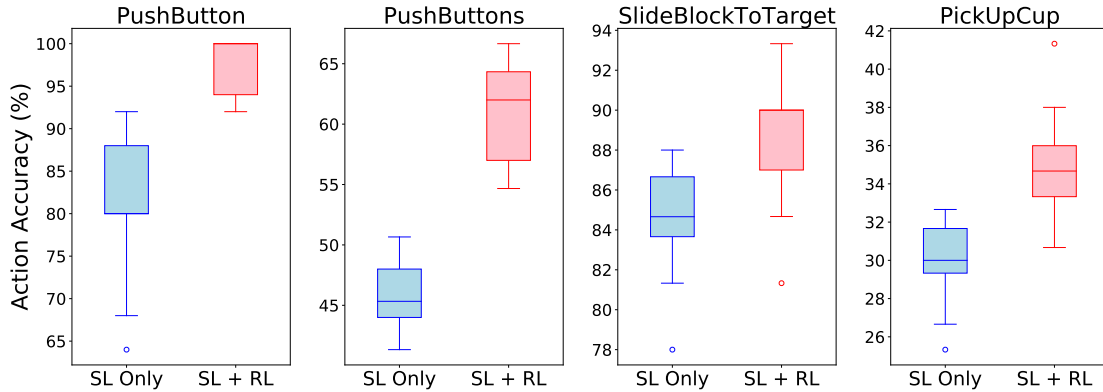


Figure 7.7: SL Only and SL+RL action execution accuracies averaged between 10K to 20K epoch checkpoints with intervals of 1,000 checkpoints for all of the four tasks.

We can see a performance boost of up to over 20% after RL fine-tuning. The best-performing purely-supervised set of weights can barely reach the 50% mark, whereas after fine-tuning our approach tops action execution accuracy with 66%.

Figure 7.7 shows the distributions of action execution performance with and without RL fine-tuning on all of the four tasks. We average starting from the 10K epoch checkpoint until the end of the SL training (20K epochs) for every 1000th epoch checkpoint. The left panel in the figure displays the model performance in the one-button setting (PushButton task). The boxes indicate the clear advantage of RL fine-tuning over purely supervised training with a significant improvement of 20% in the median values, consistent with Figure 7.4. Moreover, the RL fine-tuned checkpoint results are more narrowly distributed, while the supervised-only checkpoints have a visibly higher variance as the results range between 65% and 90%.

The left middle panel in Figure 7.7 displays the model performance on the PushButtons task. As can be seen, the median accuracy achieved by a purely supervised set of weights (45%) rises by 17% after fine-tuning (62%). Furthermore, the worst performance after RL fine-tuning is still better than the best-performing SL checkpoint ( $\sim 54\%$  vs  $\sim 50\%$ ).

The right middle panel in Figure 7.7 illustrates action execution performance on the SlideBlockToTarget task. While supervised training alone achieves decent performance with an average success rate of slightly over 84%, RL fine-tuning over the supervised model yields a significant performance improvement of 4%. The best-performing SL-only checkpoint reaches an action accuracy of 88%, whereas the highest accuracy after RL fine-tuning is over 93%. Moreover, the RL fine-tuned model attains a better result than its supervised-only counterpart in terms of the minimum success rate ( $\sim 81\%$  vs  $78\%$ ).

In Figure 7.7, the rightmost box plot shows the action accuracy results for the PickupCup task. Although the overall performance is the lowest amongst all four tasks, the SL pre-trained and RL fine-tuned model significantly outperforms the

Table 7.2: Average Target Selection and Action Precision Accuracies

Training	PushButtons		SlideBlockToTarget		PickUpCup	
	Selection (%)	Precision (%)	Selection (%)	Precision (%)	Selection (%)	Precision (%)
SL Only	67.82 ± 3.60	67.66 ± 4.15	98.67 ± 0.85	85.68 ± 2.68	92.00 ± 1.21	32.54 ± 2.36
SL + RL	72.48 ± 2.93	84.22 ± 4.91	98.06 ± 1.85	90.04 ± 2.26	90.85 ± 1.31	38.43 ± 3.22
RL Improvement	<b>4.66</b> ↑	<b>16.56</b> ↑	-0.60 ↓	<b>4.36</b> ↑	-1.15 ↓	<b>5.89</b> ↑

SL-only model on this task as well. The results show that RL fine-tuning can help XBiT attain an over 40% success rate on this challenging task as the supervision alone struggles to lift the target cup in one-third of the test episodes at best. All in all, the average performance from 10K to 20K checkpoints on the PickUpCup task is elevated by approximately 5% with RL fine-tuning.

The Pushbuttons, SlideBlockToTarget and PickUpCup tasks involve two challenges: (i) *selection* of the correct target and (ii) *precision* of controlling the gripper to successfully execute the action. To measure the individual contributions of RL to mastering these two challenges, we make the following definitions. The *selection* accuracy is the percentage of episodes where the gripper has moved closer to the target than to any of the distractors<sup>2</sup>, irrespective of successful action completion. The *execution* accuracy is the percentage of successfully completed episodes. Hence, we define *precision* accuracy, which is the number of successful executions given correct selection, as:

$$precision = \frac{execution}{selection} \cdot 100. \quad (7.11)$$

The precision accuracy measures the dexterity of the gripper manipulating the selected object. We quantify these performance measures of XBiT from 10K up to 20K checkpoints for the three tasks. The results in Table 7.2 show that the average button *selection* accuracy increases by around 4% during RL fine-tuning on the PushButtons task. The real benefit of RL, however, reveals itself in the action *precision* once a button is correctly selected: RL fine-tuning improves *precision* accuracy by more than 16% on average, from 67% to 84%. For the SlideBlockToTarget task, the selection accuracy does not improve any further, since it is at an almost perfect performance after the supervised training. The precision, on the other hand, rises to above 90% after RL fine-tuning with an increase of more than 4% over the SL-only model. The results on the PickUpCup task indicate a similar pattern: there is a slight decrease in the selection rate, whereas the precision accuracy is enhanced by almost 6%.

## 7.5 Discussion

In our experiments, we have investigated how the strengths of SL and RL can be combined to overcome the weaknesses of both learning paradigms. RL is a

<sup>2</sup>In the SlideBlockToTarget task, *selection* is considered correct when the block is moved towards the target area rather than any of the other three regions.

powerful way to explore the action space and learn various tasks based on direct feedback from the environment. It is especially suitable for dynamic environments (e.g. a rescue robot scenario [126]) and tasks that require active learning (e.g. humanoid walking [97, 121]), where labelled data for SL is unavailable. However, REINFORCE fails to learn from scratch based on only sparse rewards in our robotic manipulation scenario. Therefore, SL is necessary to learn from teacher trajectories initially. Nevertheless, SL alone cannot unlock the full potential of end-to-end models and causes overfitting to training samples, thus requiring further learning with RL to compensate for the scarcity of available data.

Our results on language-to-action translation show that when data is in short supply, SL is limited in generalising to different object positions in a continuous space and object combinations. Online RL fine-tuning in simulation with sparse rewards as direct feedback brings about better performance, achieving perfection in some cases. In the one-button scenario, the SL pre-trained and RL fine-tuned XBiT outperforms the solely SL-trained model by a great margin, showing the potential of our approach. The three-button experiments also show results in favour of RL fine-tuning over pure supervision. However, compared to the one-button experiments, the boost in performance is lower. The reason for this may have to do with the difficulty of the task. In the PushButtons task, the model must first identify the target button amongst the distractor buttons (the *selection* problem) and then accurately press it (the *precision* problem). As evidenced by the results in Table 7.2, when the SL pre-trained model chooses the wrong button, RL fine-tuning has limited ability to rectify the trajectory and move the gripper towards the correct button (less than 5% increase in *selection*). In contrast, for those cases when the model selects the correct button, RL boosts the action execution performance significantly (more than 16% increase in *precision*). An alternative way to overcome the problem of selecting the correct button could be to implement a modular approach, where a colour detection module identifies button colours before proceeding with low-level action execution. On the task requiring a block to be moved to the target area (SlideBlockToTarget), the performance boost by RL fine-tuning is limited to 4%. Here, the SL-only model achieves good results by completing the majority of test episodes successfully, leaving little room for RL fine-tuning to improve – the improvement in precision is still noteworthy as it results in the successful completion of about one-third of the remaining episodes (see the precision accuracies for SlideBlockToTarget in Table 7.2). The almost perfect selection rate on the SlideBlockToTarget task cannot be improved further using RL. On the PickUpCup task, the overall accuracy of our model is noticeably lower than the other tasks. On this task, we observed a particular phenomenon causing low precision rates despite high selection accuracies: as the target cup can be grasped along its rim, the variation in the grasp location of the cup causes the model to try to grasp the cup often in the centre of the rim, leading to failed grasp attempts, hence unsuccessful episodes. Nevertheless, RL fine-tuning leads to a significantly better performance over pure supervision.

Unlike most approaches in robotic object manipulation, XBiT does not only execute actions based on language input, but it can also describe robotic actions



based on visual and proprioceptive observations. In our action-to-language translation experiments, XBiT displays perfect sentence accuracies across all checkpoints on all four tasks, even if it sometimes fails to perform the actions successfully. We also show that the bidirectional model learns action execution skills more quickly than its unidirectional variant, which is incapable of translating from action to language. More importantly, we report that XBiT retains its perfect action-to-language translation performance even after RL fine-tuning which is used to train the model only in the language-to-action direction, including the baseline unit and excluding the language decoder. Hence, fine-tuning in one translation direction does not negatively affect the performance in the other direction. RL fine-tuning on action does not lead to forgetting action-to-language translation despite both modes sharing the common CMT bottleneck.

We perform RL fine-tuning for 1,000 episodes only, which is a tiny fraction of SL training steps. Although it needs to interact with the environment, the RL fine-tuning stage is very efficient in terms of training iterations. A further attempt at improving model performance would be iteratively alternating between SL and RL training. We tried further training an RL fine-tuned checkpoint (20K on PushButtons) with SL but saw no further improvement in performance. We also tried RL fine-tuning for up to 5K episodes starting from earlier checkpoints, for example, the 1K checkpoint on the PushButton task; while we observed some improvement at first (from 1K to 2K episodes), the action accuracy eventually plateaued, reaching its upper limit (Figure 7.5). This might be the case because when the model is under-trained, it can only come close to accomplishing the task in fewer cases. In contrast, it should come close to accomplishing the task in most cases when it has been sufficiently trained with SL. RL fine-tuning perfects these actions and creates a model that is always able to press the button (see the 10K checkpoint and compare with earlier checkpoints in Figure 7.4.)

REINFORCE is a simple off-policy gradient-based RL algorithm that directly optimises the policy to maximise return. Its simplicity, efficiency and compatibility with our model were the main factors in choosing REINFORCE when fine-tuning XBiT. In terms of architecture, the baseline output unit can be compared with the critic unit of an actor-critic RL architecture. However, while SL teaches a policy, it does not provide any reward information, leaving the baseline network untrained. Nevertheless, since the baseline is not crucial in REINFORCE, the RL phase seamlessly continues to refine policy learning starting with a zero-valued baseline. In contrast, an untrained critic would disrupt the already trained policy when switching from SL to RL in actor-critic learning. This would render an actor-critic learning paradigm less suitable for our approach. To enable the early learning of the value function in the absence of rewards, inverse reinforcement learning [3], which learns a value function directly from expert trajectories, could be considered. Furthermore, our approach could be tested with more sophisticated RL algorithms such as DDPG (Deep Deterministic Policy Gradient) [98] and PPO [156]. Though not explored in this work, multi-task RL approaches [80, 176] could be considered instead of learning every new manipulation task in isolation.

## 7.6 Conclusion

In this chapter, we have introduced a novel bidirectional robotic object manipulation model that can be pre-trained in a supervised fashion in the first learning stage and fine-tuned with RL in the second stage. Our end-to-end model, XBiT, performs well in both action-to-language and language-to-action directions in a simulated 3D continuous tabletop scenario. Fine-tuning with RL boosts its language-to-action performance, improving the precision of actions by a significant margin on all four tasks while leading to relatively minor gains or no improvements in the selection performance on the three tasks requiring correct identification of the target. Interestingly, perfect performance in the action-to-language direction, for which XBiT is not fine-tuned in the second stage, is retained, indicating no negative interference caused by RL. In the future, our method can be applied to more object manipulation tasks, including real-world scenarios. Powerful LLMs such as GPT-3.5 [129], Palm2 [9], Llama [168] and Vicuna [30] can also be employed with XBiT to enable complex human-robot dialogue.

Until now, we have only tackled simple language related to robotic object manipulation with our bidirectional action-language translation approaches. We have trained our models to understand and output robotic task-oriented sentences. However, a humanoid companion robot must have the ability to comprehend and produce language in any given context, not restricted to the specific robotic task. Therefore, in the next chapter, we will introduce the CrossT5 architecture, designed for combining low-level robotic action skills with the advanced linguistic capabilities of LLMs. Specifically, CrossT5 integrates our bidirectional action-language model PTAE into the encoder-decoder-based T5 LLM via our crossmodal fusion method CMT. Experiments show that the combined model achieves competitive performance in both robotic action-language translation and NLP tasks.

# Capturing Strengths of Large Language Models for Bidirectional Action-Language Translation

---

Natural language processing (NLP) and computer vision tasks have recently seen large improvements through the rise of Transformer-based architectures. The powerful LLMs take advantage of the extensive textual datasets that are abundantly available online. However, robotic action and particularly bidirectional action-language translation tasks are less explored as they require more specific and labelled data. Therefore, this chapter aims to enable robotic action capabilities for a pre-trained LLM while maintaining maximum efficiency in terms of training time and dataset size. To achieve this, we partition a Transformer-based LLM and insert our multimodal PTAE (Chapter 6) architecture in the middle. Specifically, we split a pre-trained T5 LLM between its encoder and decoder parts in order to insert the Crossmodal Transformer (CMT) network of the PTAE bidirectional action-language model. We conduct experiments on a new dataset, consisting of unimodal language translation and crossmodal bidirectional action-language translation. The natural language capabilities of the original T5 are efficiently reestablished by training the CMT on a tiny fraction of the original training data of T5. Furthermore, the new model, named CrossT5, achieves high accuracy for the vision- and language-guided robotic-action tasks. By design, the CrossT5 agent acts robustly when tested with language commands not included in the dataset. The results demonstrate that this novel approach is successful in combining the advanced linguistic capabilities of LLMs with the low-level robotic control skills of vision-action models<sup>1</sup>.

---

<sup>1</sup>The code of this chapter is available at <https://github.com/samsonoko/CrossT5>.

## 8.1 Introduction

Recently, LLMs have gained popularity as versatile and powerful approaches to general-purpose language processing. Being merely trained on large text corpora, they are capable of a wide range of NLP tasks, including text generation, translation, classification and analysis as well as holding natural conversations, answering questions and generating action plans for robotic tasks. Popular implementations include PaLM 2 [9], LLaMA [168] and GPT-4 [130], which reaches a size of approximately 1 trillion parameters, as well as publicly available applications like ChatGPT [129], Gemini [167] and Bing Chat [117] and open-source models such as BLOOM [155] and T5 (Text-to-Text Transfer Transformer) [144].

LLMs are built based on the Transformer model [173], an encoder-decoder architecture incorporating self-attention [101]. At its core, the Transformer converts one sequence of tokens into another, making it ideal for handling natural language. However, it is not trivial to bring the LLM capabilities into the robotics domain that would allow humans to communicate with a robot. Through additional input processing, the Transformer architecture can also be adapted or extended for other modalities as well as crossmodal applications to translate between different modalities. Most popular multimodal implementations focus on augmenting LLMs with visual capabilities such as image captioning, object recognition and visual information extraction [27, 96, 198]. Besides, many implementations only deal with either image captioning (vision to language) or image generation (language to vision). Approaches that handle both directions rely on separate models trained for their respective tasks sharing parameters [192]. These often require large-scale image-text datasets and only cover tasks on single images rather than image sequences or features extracted from them [95].

LLMs hold great potential for HRI [17], but they require combining more modalities beyond vision and language. A robot perceives its physical environment, e.g. via image sequences, proprioception and tactile input, to carry out coordinated actions. Through language, a robot can understand human commands and describe information on its own, hence, a robot should be able to translate between language and action in both directions, showing the same crossmodal behaviour as expressed by humans. A common way to augment LLMs for robotic multimodality is via early fusion, where the robotic sensory input is converted into a textual format and is passed to the LLM as tokens of a uniform language prompt, as shown in Figure 8.1a. The outputs of the LLM are tokens which may not be optimal to encode robot actions. Due to the novel tokens, fine-tuning LLMs is highly recommended [21]. Nevertheless, high-quality large datasets for multiple robotic modalities are scarce [14]. Therefore, LLMs are often augmented with external tools [116], and for human-robot collaboration, it is advisable to combine a nearly modality-independent physical task model with a dedicated dialogue model [91].

One of the most important factors influencing the performance of a model is the choice of a dataset. For language-only tasks, such as machine translation

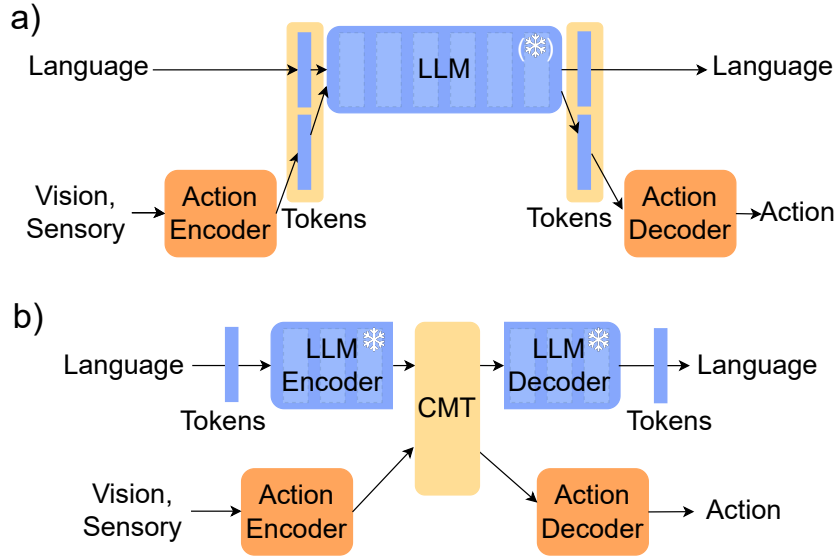


Figure 8.1: Multimodal integration architectures combining LLMs with robotic behaviour. a) Integration in many conventional architectures is via the input and output tokens of the LLM. b) Our concept uses the CMT for intra-LLM integration.

and text summarisation, there is an abundance of large-scale datasets<sup>2</sup>. For cross-modal vision and language processing on static images, datasets are also ready for use [10, 100]. However, for paired datasets of robotic action (including vision and proprioception modalities) and language, data is harder to collect and requires an expensive labelling effort [66, 175].

To achieve multimodal and crossmodal functionality on robotic action and natural language without the need for aligned large-scale datasets and expensive training, we propose a novel approach, shown in Figure 8.1b. We integrate a crossmodal architecture built for action-language tasks deeply into a pre-trained LLM. This allows us to use the pre-trained weights of the LLM while following the training procedure of the crossmodal architecture. The new model, which is called CrossT5, is trained on a combination of two different datasets, one for learning robotic action through the crossmodal architecture and another for retaining the original features of the LLM. This follows the assumption that the natural language capabilities of the LLM can deal with the addition of the new modality, and the resulting model can be used for both NLP and crossmodal action tasks.

We use an HRI setup involving the NICO robot [86] in the CoppeliaSim simulator [149], as visualised in Figure 8.2. The robot perceives its environment through cameras in both eyes and interacts with it by controlling its arms.

After conducting experiments with different dataset splits and loss calculations, we arrive at a final version of CrossT5 that satisfies our demands. The key advantages of our proposed approach are as follows:

<sup>2</sup><https://commoncrawl.org/>

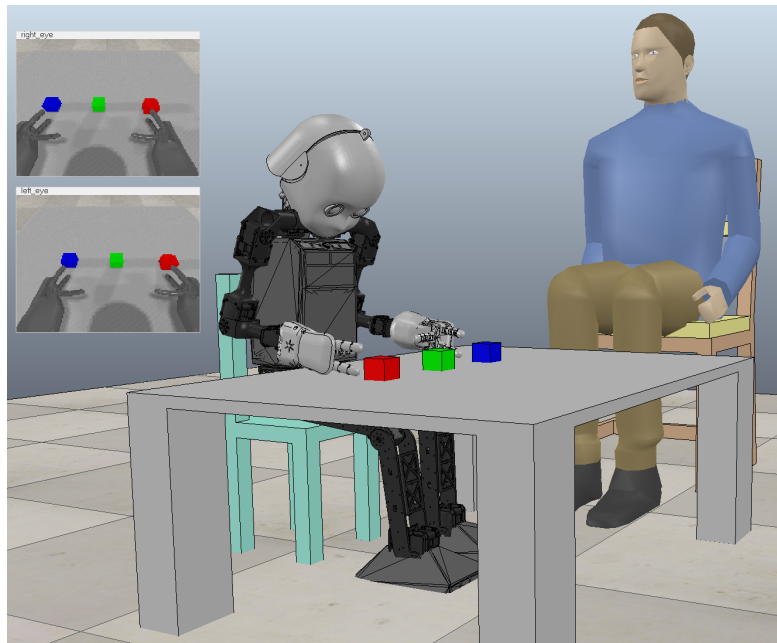


Figure 8.2: The NICO robot setup. NICO is supposed to converse with the human, execute commands and comment on the scene, which is captured by its left and right eye cameras (as shown in insets).

- since the natural language capabilities are inherited from the LLM, CrossT5 does not need a large-scale paired language-action dataset to train on, hence, the object manipulation dataset can be small in size;
- CrossT5 performs competitively with its pre-trained version before modification on natural language tasks by including a tiny portion of the natural language dataset to reestablish the LLM features during training;
- the CrossT5 architecture permits an easy exchange of the LLM, making it scalable for larger model variants or different LLMs;
- after a very efficient short training, CrossT5 adapts to the language encodings of the LLM and achieves high performance on both the natural language task and the action-language translation tasks;
- alongside competitive results for both the language-only task and the multi-modal PTAE tasks (Chapter 6), CrossT5 demonstrates high robustness for differently phrased language commands, showing successful adoption of the linguistic capabilities of the LLM.

The remainder of this chapter is organised as follows. In Section 8.2, we give background information on the different types of crossmodal fusion approaches and provide concrete examples. Next, Section 8.3 defines our proposed method CrossT5 in detail. In Section 8.4, we describe the various experiments conducted

with CrossT5 and their results, while [Section 8.5](#) discusses these results. Finally, we conclude the chapter with a summary and an outlook on the upcoming chapter in [Section 8.6](#).

## 8.2 Background: Types of Crossmodality

While there are numerous approaches in vision-language and action-language crossmodality, many of them focus on specific modalities. Accordingly, we divide them into four categories.

### 8.2.1 Leveraging LLMs for Vision-to-Language Tasks

Many multimodal approaches focus on adding image processing and understanding capabilities for LLMs [116], which are either frozen or fine-tuned as part of the new architecture. While these approaches are crossmodal, they are not capable of bidirectional translation. For instance, BLIP-2 [96] leverages the performance of pre-trained language models and image encoders to enable vision-to-language generation without expensive training. To achieve this, the image encoder and the language model are connected via a Querying Transformer (Q-Former), through which the vision and language Transformers share some of their parameters. For their experiments, they choose an OPT [193] model to serve as the decoder-only LLM variant, while a FLAN-T5 [33] variant serves as the encoder-decoder variant. Despite having fewer trainable parameters than its competitors, BLIP-2 achieves SOTA performance, even outperforming much larger models in zero-shot VQA.

Similarly, the Flamingo model [8] incorporates pre-trained and frozen LLMs and vision encoders. It accepts visual and textual data as input and can perform tasks such as captioning, VQA and visual dialogue. Similar to our CrossT5, it combines the two modalities using gated cross-attention layers, where the keys and values are obtained from the visual features and the queries from the text input. In few-shot learning, Flamingo sets a new SOTA on all of the 16 considered benchmarks and is often on par with models specifically fine-tuned for a respective task.

MiniGPT-4 [198] explores the advanced multimodal vision-to-language capabilities of GPT-4. It incorporates a frozen LLM, the Vicuna [30] model built upon LLaMA, which is connected to a frozen vision encoder through a linear projection layer. This projection layer is trained for aligning the visual features with the LLM, which are passed to the language model in a combined text prompt. MiniGPT-4 expresses a variety of capabilities similar to those of GPT-4, processing visual information through a correct alignment of features. Employing a similar method, PaLM-E [41] extends the concept to more input modalities, each with their respective encoder. After being embedded by the encoders, the input is fed to an LLM, which is a decoder-only PaLM [32]. PaLM-E performs well on VQA tasks and robotic manipulation planning. However, the performance on natural language generation tasks drops significantly when a smaller PaLM is used.



As an innovative framework, VisionLLM [177] treats images as a foreign language, enabling the LLM to comprehend and execute vision-centric tasks. Other approaches with more emphasis on action execution show that although they are still limited to the language output, LLMs can generate action plans [73] and interact with multiple sensory modalities [196] on a variety of different tasks in a virtual environment in a zero-shot fashion.

### 8.2.2 Action Execution from Multimodal Input

One category of approaches is not built for language production but instead tackles the execution of robotic action based on multimodal input [160], in most cases language, vision and proprioception. Many of these approaches also focus on multi-task generalisation [55, 78]. For example, VIMA [79] uses an encoder-decoder architecture to translate from visual and linguistic input to robotic action output. The model accepts combined prompts consisting of language commands and descriptions as well as visual information, using a pre-trained T5 as the language encoder. VIMA outperforms SOTA approaches on different robotic tasks, including zero-shot generalisation.

Tackling the problem of the lack of adaptation in action execution, the Language-Informed Latent Actions with Corrections (LILAC) framework [34] corrects robotic action through language commands in real time. Similarly, the ATLA model [147] demonstrates that language descriptions can help in adapting robotic policies to unseen tools. ATLA uses LLMs to generate these descriptions and obtain the respective feature representations. Another approach, InstructRL [102], utilises a unified multimodal encoder to encode both language and vision for robotic tasks in a virtual environment. We provide a more comprehensive review of the approaches in this category in Section 2.1.

### 8.2.3 Action-Centric Crossmodality

Action-centric crossmodal approaches are able to reason and describe actions and visual information, but they are not built for language-only tasks. For example, the RT-2 model [21] demonstrates robotic action-language crossmodality, leveraging a large-scale pre-trained LLM. Its architecture treats the robotic action as an extension of language, tokenising it on the input level and detokenising it for the robotic action output. Despite having a high computation cost, RT-2 outperforms earlier models such as its predecessor RT-1 [22] and generalises to unseen objects and backgrounds while maintaining a high accuracy. Trained on the Open X-Embodiment dataset, a consolidated dataset with 22 robotic embodiments, the RT-X models show a further improvement in success rate [175].

Another approach, the Mani-GPT [194] model, incorporates an LLM and a network for grasping objects. Its input consists of object labels from a vision module, the past dialogue history and human instructions. Based on the nature of the input, the model classifies it into one of several response categories and generates a corresponding output. A similar approach, SayCan [6], leverages the semantic

knowledge inherent in LLMs for the execution of low-level skills. The model incorporates the PaLM LLM and is connected to a robotic system that lets it navigate through and manipulate its environment.

Although these approaches demonstrate bidirectional crossmodality to a certain extent by being able to generate both language and action, their language output is limited to the context of the visual or robotic scene the model is confronted with as they do not involve pre-trained LLMs for general-purpose dialogues. We provide a more comprehensive review of the approaches in this category in [Section 2.3](#).

### 8.2.4 Full Crossmodality

A bidirectional approach to both language and visual robotic action tasks is our PTAE model ([Chapter 6](#)), which connects two separate input and output channels through the CMT to enable action-to-language, language-to-action and unimodal language and action skills. The action dataset that the model is trained on focuses on moving two coloured cubes positioned on a table in front of the robot. PTAE receives a textual description of the action as well as visual features and the robotic joint sequence. It achieves high performance on the specified tasks, even with a small number of crossmodal labelled training samples, given sufficient unimodal experience from unlabelled data ([Section 6.3](#)). However, it is not built for language tasks such as natural dialogue outside of its multimodal training data.

As another example of full crossmodality, the multimodal GATO agent [146] can perform many different tasks in various modalities, such as image captioning, acting as a chatbot and playing Atari games, all with a single pre-trained model. It is trained on a wide variety of data involving visual, language and action modalities. All input modalities are fed to the model in a batched and tokenised fashion. While GATO’s capabilities are numerous, it requires extensive training on a large number of datasets to achieve its crossmodal proficiency. In contrast, for CrossT5, we use a pre-trained LLM to obtain language proficiency without having to fine-tune it.

## 8.3 Proposed Method: CrossT5

Our proposed model integrates a pre-trained LLM with the crossmodal PTAE architecture. PTAE consists of two input encoders, one for language and another for action, and two respective decoders. The two input and output modalities are connected via the CMT in the middle. For action encoding and decoding, PTAE uses LSTMs. Since PTAE only features simple language processing, we extend it with an LLM to enhance its linguistic capabilities.

A few factors should be taken into consideration for deciding on the LLM. It has to be pre-trained and open source, as well as compact enough to be run and fine-tuned locally without major expense. Furthermore, its architecture needs to be compatible with the PTAE model to allow for a seamless integration of the two models. T5 [144] meets all of these requirements. Its largest variant is the most performant, but for our conceptual research, we utilise the T5-small variant with 60

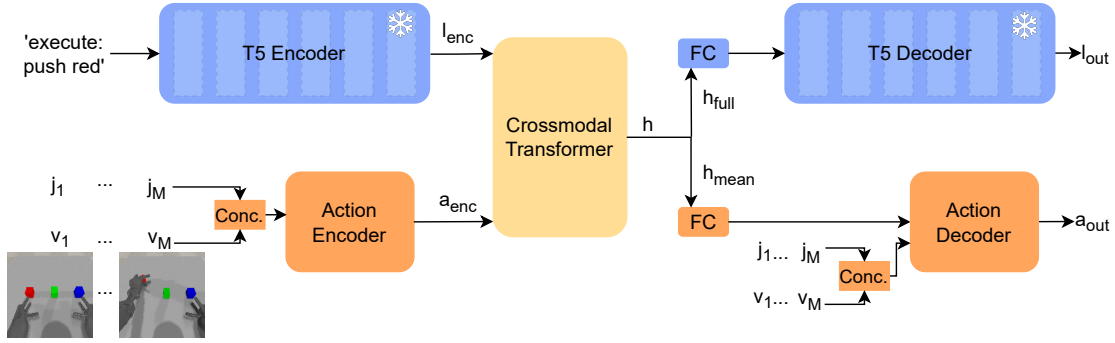


Figure 8.3: The CrossT5 architecture. The T5 encoder and decoder are integrated with the PTAE architecture via the Crossmodal Transformer (CMT).  $j$  vectors represent joint angle values of NICO’s arms, while  $v$  vectors are visual features extracted from images recorded by NICO’s eye cameras. Conc. denotes concatenation and FC is a fully connected layer.

million parameters. A fine-tuned variant of T5, FLAN-T5, could be suitable as well, but Wei et al. [178] found that fine-tuning tends to result in worse performance for smaller FLAN-T5 variants. Another model compact enough could be the smallest variant of BLOOM [155], featuring 560 million parameters. However, BLOOM has no encoders but a stack of 70 decoder blocks, hence there is no distinguished single point to integrate the PTAE. In contrast, T5 matches the encoder-decoder architecture of the PTAE model, making the gap between the encoder and decoder an obvious point of integration. This yields the new CrossT5 model.

As presented in Section 8.2, most other vision-language approaches instead make use of early fusion, combining the data at an initial part of the model. These models also implement only a single multimodal direction but are either not built for language-to-action capabilities (BLIP-2, MiniGPT-4) or action-to-language capabilities (RT-2, VIMA).

### 8.3.1 Model Architecture

CrossT5 follows the PTAE architecture (Section 6.2). It retains the CMT alongside the two encoders and two decoders, of which the action encoder and decoder remain unchanged from the PTAE, while the language encoder and decoder are from T5. The CMT takes the language encoding as *query*, while the action encoding is taken as *key* and *value*. After applying scaled dot product attention, it outputs the crossmodal hidden representation vector  $h$ , which is passed onto the two decoders. T5-small consists of 6 encoder and 6 decoder layers, which are split up in the middle to connect it with the CMT (Figure 8.3).

The hidden dimension of the CMT and therefore of the output vector  $h$  is set to 512 to match the encoding of the T5-small. In the PTAE model, the mean over the temporal dimension of the vector  $h$  is used as the input for the language and action decoders. Since the tokens of the T5 language encoding are presented as one sequence, this step is unnecessary for the T5 language decoder and only done

for the action decoder.

The T5 weights are frozen. In contrast, we train the CMT, the action encoder and the action decoder from scratch. The T5 decoder is still used for backpropagation of the language error during loss calculation, but its weights are not modified.

### 8.3.2 Crossmodal Language-Action & Natural Translation Dataset

We name our dataset **Crossmodal Language-Action and Natural Translation**, or *CLANT* for short. CLANT combines two separate datasets: an *action dataset* is used to augment the model with bidirectional action capabilities, while a *language dataset* is necessary to restore the capabilities of T5 since its encoder and decoder are separated by the insertion of CMT as the multimodal fusion method.

Table 8.1: Vocabulary with Original and Alternative Words

Component	Original	Alternative
Action	push	move-up
	pull	move-down
	slide-to-left	move-sideways-to-left
	slide-to-right	move-sideways-to-right
Colour	red	scarlet
	green	harlequin
	blue	azure
	yellow	blonde
	cyan	greenish-blue
	violet	purple

**Action Dataset** We train the crossmodal tasks of the architecture using the *NICO CoppeliaSim Dataset*. A variant of this dataset [136] was used for training PTAE – see Section 6.3. The dataset consists of a total of 1440 samples, out of which 360 samples (25%) are used for testing, while 1080 samples (75%) are used for training.

Each sample consists of a sequence of images and joint values and a description of an action carried out by the NICO robot [86]. NICO sits in front of a table where three coloured cubes are placed in three pre-configured positions. The entire scene is generated on Coppeliasim [149]. NICO has a camera in each eye, overseeing the table, its hands, arms and their shadows. In each sample, NICO moves one cube using either its right or left arm. During this action, the camera records, depending on the sample, a sequence of  $T = 40, 60$  or  $85$  images. A 30-dimensional feature

vector from each image is extracted using CS-CAE (Section 4.2.3), resulting in a matrix of  $30 \times T$  features for each action. The joint angle values of both arms are also recorded for each of the  $T$  time steps, with 5 joint values for each arm.

Each textual description consists of two words, one describing the desired action and the other describing the colour of the target cube. There are 6 different cube colours and 4 distinct actions, each of which has an alternative name. The action vocabulary consists of the words given in Table 8.1, which results in  $12 \times 8 = 96$  distinct textual commands in total.

The dataset includes one sample for each distinct action, so the alternative language descriptions do not increase the number of action sequences. As the vocabulary does not include object positions, language descriptions do not specify cube positions but cube colours.

Since there are always exactly three cubes on the table, with four possible actions, the total number of possible actions for a given cube configuration is  $3 \times 4 = 12$ . The number of distinct cube configurations is the number of subsets of 3 cubes from a set of 6 possible cube colours, which is 120. This makes a total of  $12 \times 120 = 1440$  samples in the dataset.

**Language Dataset** The language dataset used for training the natural language capabilities of CrossT5 is the *Tilde RAPID 2019 German to English* dataset [16]. It consists of more than a million sentence pairs in German and English taken from the press release database of the European Commission. Because the original T5 model was already trained on English-to-German translation, this dataset is well suited for training the new model. We focus on language translation because it can be evaluated more easily than text summary or information extraction and works well on shorter text samples.

For the new CLANT dataset, the first 1440 samples from RAPID 2019 are taken and added to the NICO CoppeliaSim dataset as an additional category. The contents of a CLANT dataset sample are visualised in Figure 8.4. As the content of RAPID 2019 is not ordered by any specification, taking the samples from the beginning does not cause the data to be restricted in vocabulary or versatility. 1440 translation samples would be an insufficient amount of data to train from scratch for a complex task like translation, but this is beyond the scope of this chapter. The CMT in the centre of CrossT5 only has to learn how to identically reproduce the encoding from the T5 encoder to the T5 decoder for the language translation to stay intact.

### 8.3.3 Training Setup

The new architecture should allow the user to specify whether one is performing actions or having a conversation. To implement this kind of control, we use different mode signals like those used for PGAE (Chapter 5), PTAE (Chapter 6) and XBiT (Chapter 7), so that the model can differentiate between the different operation modes. Following the same idea, the new signals now not only distinguish between

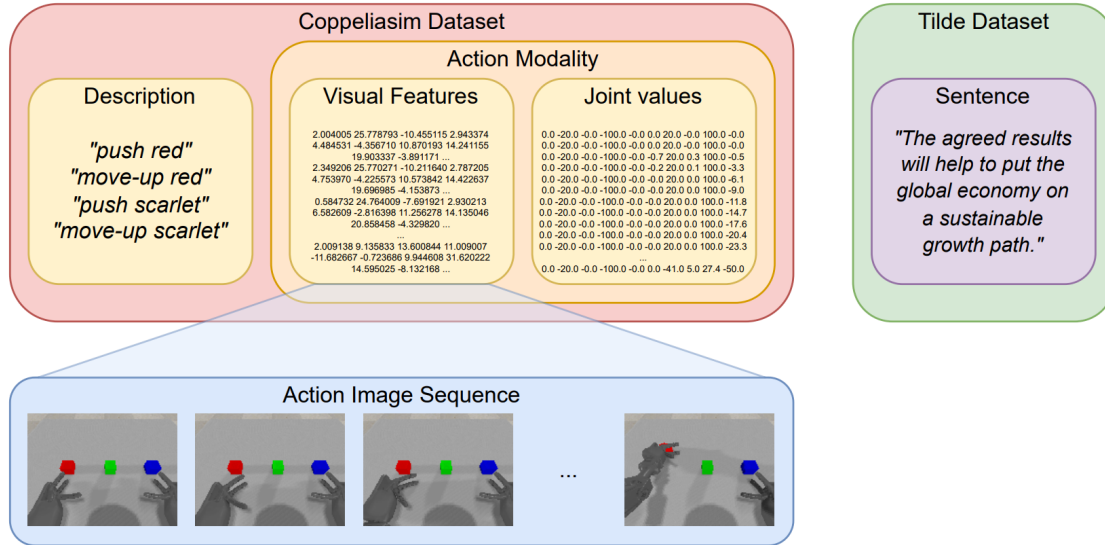


Figure 8.4: The setup of a CLANT dataset sample. The NICO Coppeliasim dataset provides a textual action description, joint values and the visual features extracted from the action image sequence before training, while the Tilde RAPID 2019 dataset provides a natural language translation sample. These two datasets are combined into a single dataset, from which, depending on the performed task, different parts are used or ignored on the input level.

the unimodal and crossmodal tasks as in the previous chapters but also the natural language translation task of the integrated T5 model. The signals are given below.

- **Translate** indicates the unimodal natural language translation. To retain the capabilities of T5, this mode trains the new model on natural language translation samples taken from the Tilde RAPID dataset.
- **Describe** indicates the crossmodal action-to-language translation. This mode trains the model to describe the perceived NICO action.
- **Execute** indicates the crossmodal language-to-action translation. This mode trains the model to generate the correct sequence of joint values corresponding to the NICO action description.
- **Repeat Action** indicates the unimodal action-to-action translation. This mode trains the model to repeat the sequences of joint values from the received NICO action.
- **Repeat Language** indicates the unimodal language-to-language translation. This mode trains the model to repeat the language description from the received NICO sample.

We refer to the last four signals as “PTAE signals” for the remainder of this chapter. During training, one of the signals is chosen at random to determine



the mode according to a varying probability distribution. In ‘translate’ mode, the model receives an English sentence sample from the Tilde RAPID 2019 dataset as its language input with the added prefix “*translate English to German:* ”. This prefix is needed for the pre-trained T5 model to differentiate the translation task from the rest of its skill set. In ‘translate’ mode, the action input is set as the repeated initial joint configuration of the robot concatenated with zeros for the visual features. The language target is set as the equivalent output that the original T5 model produces, while the action target is the initial joint for all time steps. This trains the robot to not move during a unimodal language task by design.

For the PTAE signals, the action is given to the model in the form of the joint value sequence of NICO’s arms concatenated to the corresponding matrix of visual features. If the mode signal is ‘execute’ or ‘repeat language’, the action input is instead provided as the repetition of the joint values and visual features at the first time step. In the same way, the language input is the two-word description of the action, except for the ‘describe’ or ‘repeat action’ mode, where the description is left out. In all cases, the desired mode signal term is appended as a prefix to the language input.

In the ‘execute’ and ‘repeat action’ modes, the action target is the correct sequence of joint values for the action, while the language target is an empty string. In the ‘describe’ mode, the action target is the repeated final time step joint configuration, while in the ‘repeat language’ mode, the action target is the repeated initial joint configuration. In both modes, the language target is the corresponding two-word description of the action.

The ‘translate’ mode indicates that the new model treats the input in the same way the pre-trained T5 would do. The *translate* task was chosen because the focus for keeping T5’s capabilities lies in its proficiency in English-to-German translation, which T5-small can perform best and which can be quantitatively measured with ease.

Technically, T5 supports a maximum encoding length of up to 512 tokens with positional encoding. By default, this number is capped at 20 tokens. Since the translation dataset includes sentences longer than 20 tokens, we set the maximum sequence length to 60 tokens.

During training, the T5 decoder uses *teacher forcing*: instead of generating one token per time step by using the previously generated tokens as input, the decoder uses the ground-truth token. The error is calculated between the predicted and the ground-truth tokens individually. Since this method does not rely on the prior  $t$  tokens for the token at  $t + 1$  to be predicted, the entire encoding can be processed in parallel with only one pass of the decoder. As the T5 language encodings can be up to 60 tokens in length and would normally require the same number of decoder iterations, this significantly optimises the training time. The LSTM action decoder inherited from PTAE also uses teacher forcing but does not run in parallel.



### 8.3.4 Loss Calculation

For the action loss, we keep Equation (6.13) of PTAE, as the action decoder and the general training process remain unchanged. This implementation takes the mean squared error (MSE) of the produced joint sequence and the target joint sequence. For the language loss, there are two alternatives, implemented at different points in the model architecture. We train CrossT5 with these two loss calculations to evaluate their usability for the modified architecture:

**h-vector Loss** In this calculation mode, the loss is defined as the MSE between the hidden vector  $h$  of the CMT and the target hidden vector  $\hat{h}$ :

$$L_{\text{lang}} = \frac{1}{N} \sum_{t=1}^N (h_t - \hat{h}_t)^2, \quad (8.1)$$

where  $t$  indexes the numerical values within an encoding of length  $N$ . This eliminates the necessity of running the T5 decoder thereby reducing computational costs. For the T5 translation, the target  $\hat{h}$  is the encoding generated by the T5 encoder, because for translation, the T5 encoder already produces an optimum encoding that generates a good result when passed to the decoder without change. The CMT only needs to learn not to modify the encoding. While this proves to be true and works well for the T5 translation tasks, the targets for the crossmodal tasks of the NICO CoppeliaSim dataset have to be given additional attention. Since not using the T5 decoder during training means only comparing numerical encodings against each other, we need T5 encodings that evidently produce correct sentences such as “*push blue*” or “*pull red*” in the decoder. The encodings of prompts such as “*translate English to English: NICO command*” reliably decode to “*NICO command*” in vast majority of the cases, allowing us to evaluate this training method. Therefore, during training, if the model receives an encoded action for “*push yellow*” in the ‘describe’ mode, the resulting output vector  $h$  is compared against the corresponding T5 encoding of “*translate English to English: push yellow*”. Given that these two encodings differ in token count, ranging from 2 to 10 tokens, which leads to complications in the loss calculation, language encodings of the PTAE signals are padded to a uniform length of 20 tokens. Unfortunately, this method is found to be inadequate, as demonstrated in Figure 8.5. The language performance for ‘describe’ and ‘repeat language’ is 0% and 1.36% respectively. Even for ‘execute’, where the model is trained to give an empty output, the score is just 0.56%. This outcome may be due to the use of padding tokens. The only PTAE signal that CrossT5 learns to some extent is ‘repeat action’, where the ground-truth language output is empty. Different from ‘execute’, the language input for ‘repeat action’ is “*repeat action:*” and the action input is a joint sequence. As a result, the T5 decoder does not receive a meaningful encoding and produces an empty output in 53.89% of the cases.

**T5 Decoder Loss** This method uses the loss calculation implemented in T5. All language output is decoded through T5 with teacher forcing, the loss calculated

as the cross-entropy loss between target  $x$  and prediction  $y$ :

$$L_{\text{lang}} = \frac{1}{M} \sum_{t=1}^M \left( - \sum_{i=1}^V x_t^{[i]} \log y_t^{[i]} \right), \quad (8.2)$$

where  $t$  indexes the tokens within a sequence of length  $M$ , and  $V$  is the vocabulary size. The error is backpropagated through the decoder into the CMT. This increases the accuracy for the PTAE signals dramatically, because calculating the loss at the actual output layer teaches the model to generate encodings directly corresponding to, for example, “*push blue*” and not “*translate English to English: push blue*”. Unfortunately, as displayed in [Figure 8.5](#) in the middle, this training method does not work well for English-to-German translation. When training only with the **T5 decoder loss** calculation, the translation performance drops to 5.86% on the test data. Even though the dataset used for training the ‘translate’ mode aligns with what the pre-trained T5 model can already do, for the same input prompt, the pre-trained model often generates an output completely different from the target. Although most dataset targets and the respective T5 predictions are identical in terms of information and grammatical correctness, the flexible syntax of the German language means that they can be completely different in their word order and sentence structure. For this reason, the cross-entropy loss calculates a large error for an acceptable translation. This interferes with the CMT which tries to learn to change the translation encodings in accordance with the dataset targets instead of learning to identically map them to the decoder.

To solve this problem, we propose a new way of calculating the loss. Since training the translation using the  $h$ -vector loss already proved to be sufficient, we keep it only for the ‘translate’ mode. In each training iteration, the current mode signal is checked in the loss function. If it is ‘translate’, the function calculates the MSE loss between the  $h$ -vector and the T5 encoder output. If it is any of the PTAE signals, the function instead runs the T5 decoder and calculates the cross-entropy loss of its output compared to the target of the NICO sample. This is possible because of the way the T5 model is integrated into the new architecture, making it optional during training. We name this dynamic way of switching the loss calculation **mixed loss**, which we formally define as follows:

$$L_{\text{lang}} = \begin{cases} \frac{1}{N} \sum_{t=1}^N (h_t - \hat{h}_t)^2 & \text{if signal='translate',} \\ \frac{1}{M} \sum_{t=1}^M \left( - \sum_{i=1}^V x_t^{[i]} \log y_t^{[i]} \right) & \text{otherwise.} \end{cases} \quad (8.3)$$

The total loss combines the corresponding language loss  $L_{\text{lang}}$  and the MSE-defined action loss  $L_{\text{act}}$ , taking a weighted average:

$$L_{\text{all}} = \alpha L_{\text{lang}} + \beta L_{\text{act}}, \quad (8.4)$$

where  $\alpha = \beta = 1$  for our experiments. The results shown in [Figure 8.5](#) on the right confirm the success of this method, as the mixed loss calculation results in good accuracy for all mode signals.

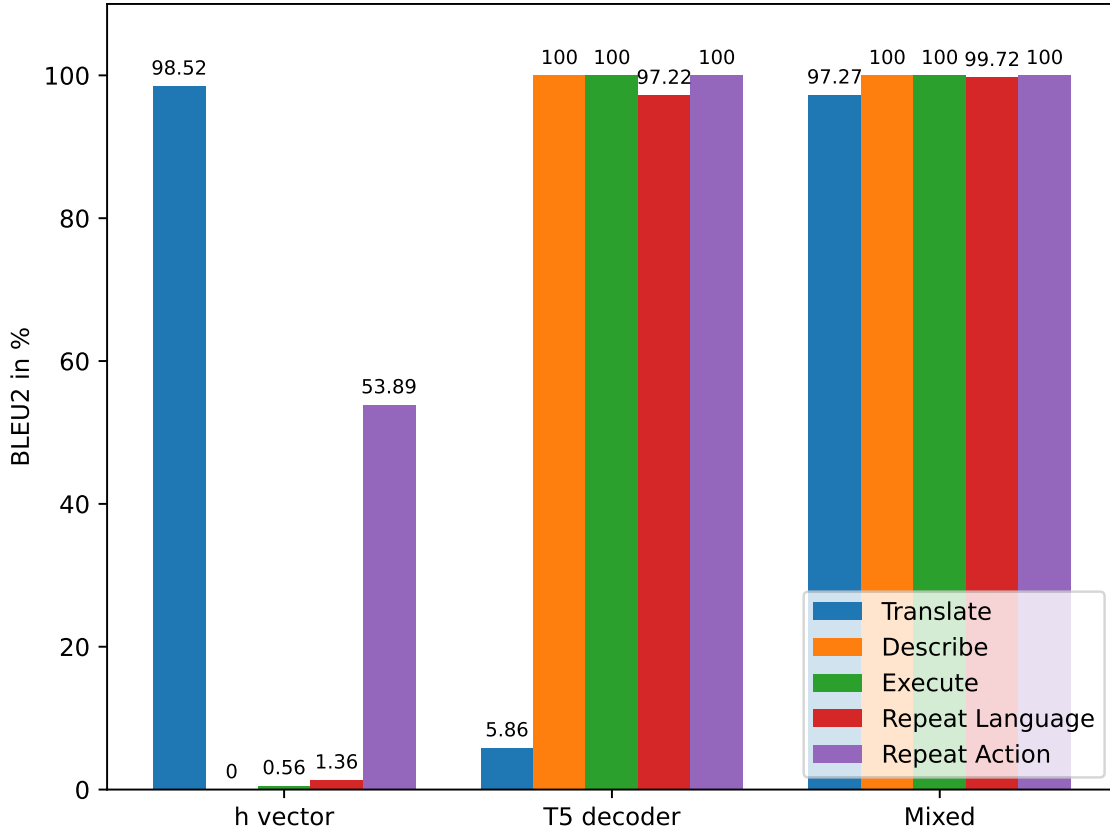


Figure 8.5: Language test performance for each signal in the three loss modes “*h*-vector”, “T5 decoder” and “Mixed”, trained for 10,000 epochs.

## 8.4 Experiments and Results

We train the CrossT5 model for up to 10,000 epochs as a trade-off between high performance and minimal training time. Training CrossT5 on a system equipped with an RTX 3080 Ti for 10,000 epochs takes approximately 6 hours.

Table 8.2: Language test performance (in percentages) with different shares of the ‘translate’ signal, trained for 5,000 epochs.

Signal	10%	25%	50%	75%	90%
Translate	57.48	71.33	91.48	92.67	92.04
Describe	96.94	99.17	99.17	93.33	94.44
Execute	100	100	100	100	100
Repeat Language	100	99.44	99.17	99.37	99.27
Repeat Action	100	100	100	100	100

To sufficiently attend to both the English-to-German translation and robotic

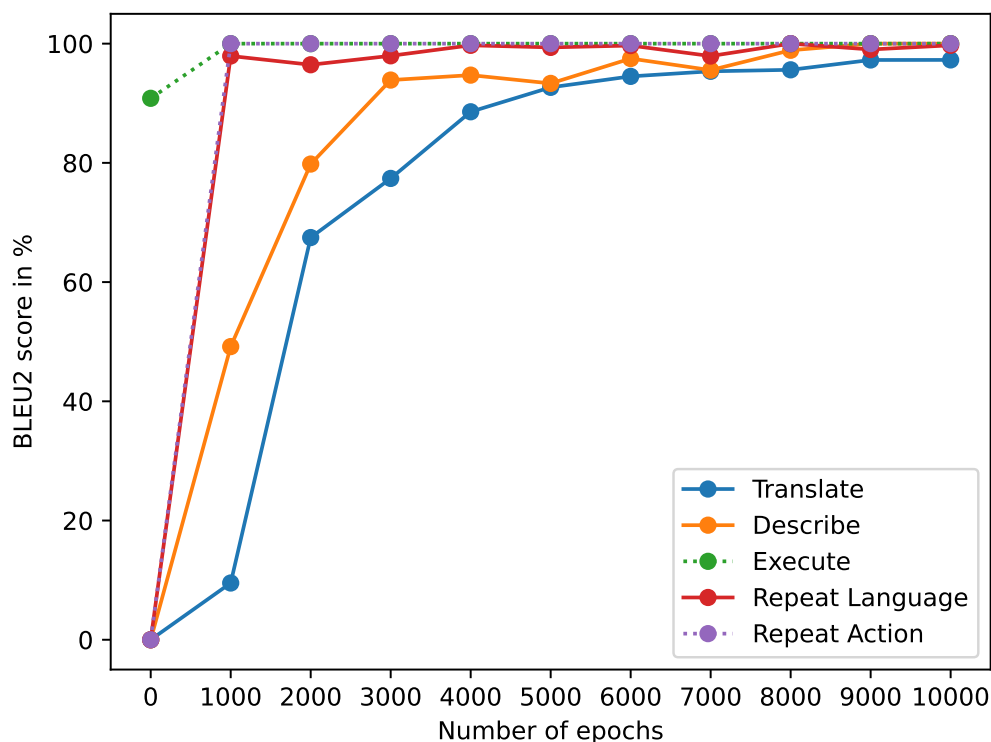


Figure 8.6: CrossT5 language test performance over training, calculated as the BLEU2 score of the language output compared against the target.

action-language translation, CrossT5 is trained with the ‘translate’ and PTAE signals for about 75% and 25% of the training iterations respectively. Amongst the PTAE signals, the training iterations are further split into one third for ‘describe’, another third for ‘execute’, one sixth for ‘repeat action’ and the remaining sixth for ‘repeat language’. However, the model also performs well with different splits. Both 50% for ‘translate’ and 50% for the PTAE signals and also 90% for ‘translate’ and 10% for the PTAE signals only affect the accuracy by 1-2% as can be seen in [Table 8.2](#). Moreover, the model performs well for all NICO signals, albeit with a slight performance drop in ‘describe’ at a 75% or higher share of the ‘translate’ signal.

Evaluating the translation performance is non-trivial. As explained in [Section 8.3.4](#), T5 and hence CrossT5 often give a grammatically and factually correct output that only differs from the dataset target in terms of syntax structure or choice of words. In addition, because we used T5-small, the T5 variant with only 60 million parameters, there are inaccuracies in English-to-German translations even by the stand-alone T5 model. When evaluating the translation performance based on the dataset targets, the calculated BLEU2 accuracy only reaches about 10% at 10,000 epochs. For the translation performance alone, this can be considered a low score. However, it is not representative of the spirit of our proposed architecture, because we train CrossT5 by identically reproducing the language encodings

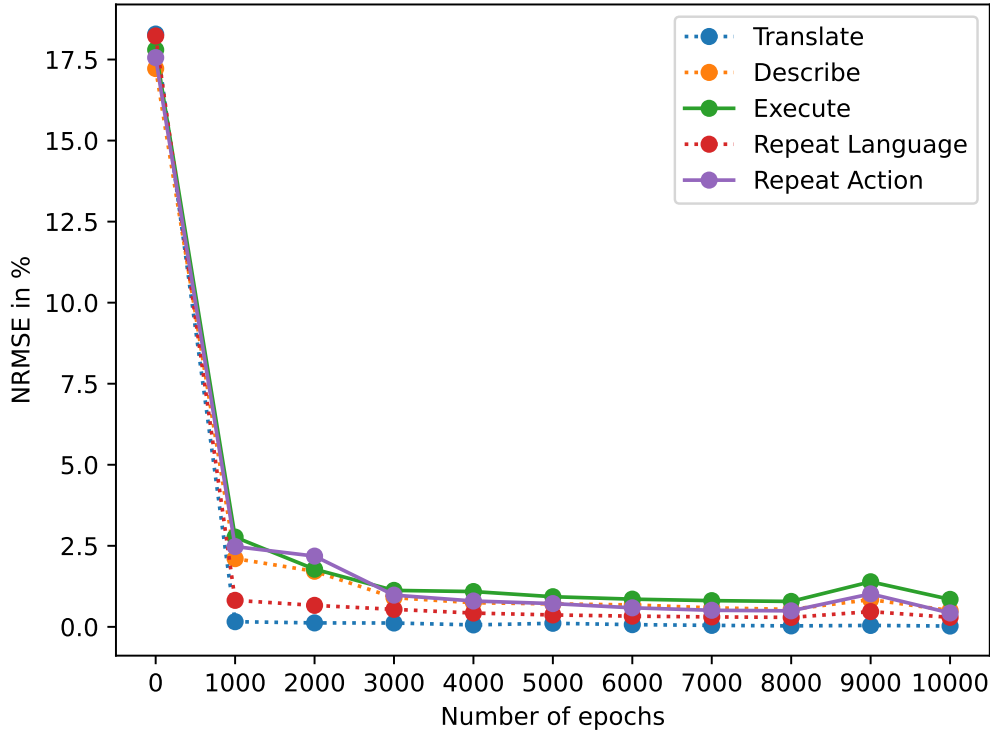


Figure 8.7: CrossT5 action test performance over training, calculated as the NRMSE between the action output and the target joint sequence.

in the CMT, thus, its translation performance can never exceed and most likely not fully reach the level of T5. In this context, the fairest way of measuring the translation performance of CrossT5 is to compare its output with the output of the stand-alone T5 model instead of the ground-truth output from the dataset. This gives us an estimate of how much of the original T5 performance is retained in the new model.

For the evaluation of the PTAE-signal performance, the output is compared with the actual dataset targets (ground-truth output), because CrossT5 is trained from scratch on these tasks. The language accuracy for the PTAE signals ‘describe’ and ‘repeat language’ is calculated as the BLEU2 score compared to the target sentences, whereas the action accuracy for ‘execute’ and ‘repeat action’ is the similarity to the ground-truth joint sequence, calculated using the NRMSE metric.

Figures 8.6 and 8.7 show the language and action results of CrossT5 over 10,000 epochs, evaluated on the test set. At 10,000 epochs, the model achieves a BLEU2 score of 97% for the ‘translate’ signal and between 99 and 100% for the PTAE signals. These results demonstrate that almost all of the initial language translation capabilities of T5 stay intact, while the added crossmodal skills also achieve high performance; most PTAE signals reach over 90% accuracy after only 3,000 epochs. Furthermore, as the CMT quickly learns that the language output for ‘execute’ and ‘repeat action’ should always be an empty string, these signals even reach 100%

accuracy after only 1,000 epochs. In terms of action accuracy, CrossT5 performs well with an NRMSE of 0.85% at maximum and an average of 0.4% at 10,000 epochs. The crossmodal PTAE signals ‘describe’ and ‘execute’ have noticeably always a higher error than the unimodal PTAE signals ‘repeat language’ and ‘repeat action’. Besides, the action performance for the ‘translate’ signal is significantly better than all PTAE signals, having an NRMSE of only 0.025%.

As the weights of the CMT are initialised randomly, the language performance at 0 epochs displayed in Figure 8.6 is at chance level. For the action performance in Figure 8.7, the NRMSE score of around 18% for the action output is worse than the average distance of any dataset joint sequence from the initial joint values (around 6%) and worse than the average distance of all joint sequences to each other (around 11%) in the dataset. For the language accuracy in Figure 8.6, the BLEU2 score of the output is 0% for all signals, except for the ‘execute’ signal, where it is about 90%. This is because, by chance, some random initialisation seeds cause the T5 decoder to always generate an empty output for a certain input, which is the target for ‘execute’ and ‘repeat action’. For instance, another run with a different seed instead resulted in an 80% accuracy for the ‘repeat action’ signal and 0% for the rest of the signals, including ‘execute’.

### 8.4.1 Action Execution Evaluation in Simulation

Calculating the NRMSE between the predicted sequences and ground-truth sequences is not sufficient for a thorough assessment of the action execution performance of CrossT5. Therefore, we conduct an additional practical evaluation in which the joint values generated by CrossT5 for the ‘execute’ signal are executed on the 3D CoppeliaSim simulator. For this evaluation, we maintain the setup of the dataset, consisting of NICO in front of a table with three cubes on it. For each generated action, the respective cube configuration is loaded in the simulator. To evaluate the success of actions scrupulously, three factors are taken into consideration:

1. whether the target cube is pushed beyond a minimum threshold in the right direction (**successful action**),
2. whether the target cube is pushed in the right direction for at least the dataset threshold (**perfect action**),
3. whether any distractor cube is contacted or accidentally moved during the action (**failed action**).

Based on these factors, an automatic score is calculated for each generated joint sequence in the test set for the ‘execute’ signal.

To determine the target cube displacement threshold for successful actions, the ground-truth joint sequences are also executed in simulation and used as a benchmark. To reach this threshold, the execution must be of at least the same quality as the ground-truth joint sequence from the corresponding dataset sample,

Table 8.3: Quality of the action execution (in percentages) for the checkpoints ranging from 1,000 to 10,000 epochs, given as the share of “successful” and “perfect” executions of all the evaluated test dataset samples.

Quality	Epochs									
	1,000	2,000	3,000	4,000	5,000	6,000	7,000	8,000	9,000	10,000
Successful	53.17	95.00	96.39	98.06	98.61	98.61	97.78	98.33	93.61	95.83
Perfect	34.17	75.56	75.00	79.17	77.22	81.67	83.33	82.50	59.44	81.11

meaning a movement of roughly the same magnitude, without touching any of the other cubes. Even if an executed action does not reach the threshold set by its ground truth, it can still be considered successful if the target cube is displaced in the right direction for at least the predetermined minimum threshold. This distinction is necessary as we observed that many CrossT5-generated movements were just below the dataset threshold, but were de facto still successful, i.e. no contact with other cubes and the target cube clearly moved in the right direction.

To emphasise this fact, we distinguish between **perfect** actions, indicating an execution quality equal to the ground truth from the dataset, and **successful** actions, indicating an execution quality that should still be considered successful, i.e. the correct cube is moved in the correct direction but not reaching the dataset threshold.

The checkpoints from 1,000 to 10,000 training epochs are evaluated on the test set, and the results can be seen in Table 8.3. As the results show, CrossT5 achieves high quality for its predicted joint sequences executed on the simulator. In the best case, it manages to execute 98.61% of the actions successfully, while 83.33% are of almost identical quality to the ground-truth actions.

### 8.4.2 Language Robustness

We conduct robustness experiments on CrossT5 to find out if the combination of a crossmodal architecture, designed for simple language input, with an LLM can boost robustness to variations in the language input without utilising a complex language-action dataset.

In the NICO CoppeliaSim dataset, each sample contains a language description of the respective action that serves both as an input command in the language-to-action translation (e.g. “execute: push red”) and as a target output in the action-to-language translation (‘describe’ signal). These language descriptions are constructed with a limited vocabulary, only containing words for the colour of the target cube and the intended push direction. Even though the dataset includes alternative words for both colour and direction (e.g. ‘scarlet’ instead of ‘red’ and ‘move-up’ instead of ‘push’), these variations are hard-coded and do not teach any semantics to the model.

In the new CrossT5 architecture, the language descriptions are encoded and processed through T5 but only the simple language commands like “push red”



Table 8.4: The 9 different robustness patterns exemplified for “push red”. *direction* and *colour* refer to the corresponding word from the dataset sample.

No.	Robustness Pattern	Example
1	‘please’ + <i>direction</i> + <i>colour</i>	‘please push red’
2	‘would you please’ + <i>direction</i> + <i>colour</i>	‘would you please push red?’
3	<i>direction</i> + ‘the’ + <i>colour</i> + ‘cube’	‘push the red cube’
4	<i>direction</i> + ‘the’ + <i>colour</i> + ‘cube now’	‘push the red cube now’
5	‘please’ + <i>direction</i> + ‘the’ + <i>colour</i> + ‘cube’	‘please push the red cube’
6	‘would you please’ + <i>direction</i> + ‘the’ + <i>colour</i> + ‘cube’	‘would you please push the red cube?’
7	<i>colour</i> + <i>direction</i>	‘red push’
8	‘the’ + <i>colour</i> + ‘cube’ + <i>direction</i>	‘the red cube push’
9	<i>alternative direction word</i> + <i>colour</i>	‘shove red’

continue to be used for the training. To evaluate whether the model can handle more complex language, 9 alternative command patterns are introduced, testing different aspects of more natural language such as varying vocabulary, word order, utilising adverbs and polite phrases. By applying these 9 command patterns, more complex language input is constructed from the simple descriptions used during training, as visualised in Table 8.4.

We reevaluate the 10,000th epoch checkpoint on these 9 robustness modes and afterwards test its practical action performance in the same way as in Section 8.4.1. As can be seen in Table 8.5, there is a small to medium performance drop in some of the robustness modes, which is expected for the more complex commands as in Patterns 2 and 6. However, the overall performance especially for the “successful” actions is retained in many cases, even exceeding the accuracy of the simple dataset commands in Patterns 7 and 8. Even for Pattern 6, which yields the lowest scores, the accuracy of 84.72% is still acceptable. It is also important to note that even for the failed action executions, in none of the cases did the model confuse actions or colours and move the target cube in the wrong direction or move the wrong cube.

Table 8.5: Quality of the action execution (in percentages) for the 10,000th epoch checkpoint, evaluated on the 9 different robustness modes.

Quality	Robustness Pattern								
	1	2	3	4	5	6	7	8	9
Successful	95.28	88.61	95.28	93.89	94.17	84.72	96.67	96.39	95.56
Perfect	79.17	68.06	76.67	75.83	74.44	64.17	80.00	78.89	78.61

Overall, CrossT5 demonstrates a good performance on more natural linguistic input, in many cases reaching or even exceeding the performance on the original dataset samples. The language robustness experiments cover only the language-to-action translation; as the language output has been specifically trained to align to the ground-truth descriptions from the dataset, the action-to-language and

language-to-language translations still produce the regular language outputs. For example, a command like “repeat language: would you please push red” will still output “push red” in most cases.

## 8.5 Discussion

With an almost perfect performance on the English-to-German translation task from the pre-trained T5 and the crossmodal language-action tasks from the NICO dataset, CrossT5 shows promising initial results. The experiments were only conducted with one combined dataset and one language model. Extending the model with larger T5 variants should be straightforward, but more adjustments may be necessary for different LLM architectures. While the bottleneck between the encoder and decoder of T5 is an obvious place to insert the PTAE via CMT, many LLMs have decoder-only architectures. Finding an optimal pair of layers for insertion of the CMT in such models may require experimental exploration. Alternative architectures, e.g. those that add small trainable matrices between a frozen network’s layers, as done for fine-tuning in multi-task learning [71, 151], could also be investigated for a multimodal enhancement of LLMs.

An advantage of integrating a pre-trained language model and only training the crossmodal architecture to keep its capabilities is that it saves a lot of computational resources during training. As the language model is kept frozen, our dataset is significantly smaller than those used for training SOTA LLMs. With only 1080 training and 360 test samples, CrossT5 is able to achieve the original T5’s translation performance. Moreover, for our model, the loss is calculated for the LLM tasks on the hidden representation vector  $h$ , which is generated by the CMT and does not have to use the language model decoder output. Thereby, the English-to-German translation does not require targets during training. Even for the integration of LLMs pre-trained on different NLP tasks, a small collection of suitable inference prompts (e.g. ‘translate’ signal) would suffice to reliably retain LLM capabilities during the training of our novel CrossT5 model.

The number of parameters for the CMT is about 2.6M (the action encoder and decoder have an additional 3M parameters together) as opposed to the 60M frozen parameters of T5-small, making the CMT’s parameter count approximately 22.8 times less. The language dataset used to retrain T5’s original abilities has a size of 131KB, as opposed to the 745GB of the C4 dataset the original T5 was trained on. This is about 5.7 million times less data. The training time of CrossT5 was 6 hours on an RTX 3080 Ti GPU. While there is no official information about the training time of T5, similar projects using the T5 architecture [154, 171] report an estimate of multiple days to over a week of training time on four 40GB A100 GPUs even for the smallest checkpoint. Despite such a contrast in the dataset size and computational resources, CrossT5 achieves over 97% of the performance of the original T5 after retraining.

In addition to its high performance for both the crossmodal PTAE tasks as well as the translation from T5, CrossT5 also displays an impressive performance when

tested with more complex language commands on the ‘execute’ signal. Even though the model has only been trained on the simple descriptions from the NICO CoppeliaSim dataset, such as “push red”, it is able to execute more natural commands like “Would you please push the red cube” in more than 84% of the cases. Other command variations, like switching the word order (“red push”) or using alternative descriptions (“shove red”) do not deteriorate the practical performance. This shows that CrossT5 not only retains the features of T5 but can further leverage its advanced linguistic capabilities for action-language tasks.

Due to the CLANT dataset’s restricted set of actions and predefined object positions, the CrossT5 model has not been tested on its ability to generalise to new scenes and actions. As our goal was to prove the feasibility of our approach with the limited resources at our disposal, testing CrossT5 on larger datasets with more diverse setups and continuous object positions is out of the scope of this thesis. As the architecture is easily scalable, it can be used with different and even larger variants of T5 and other LLMs by merely adjusting the hidden dimension size of the CMT. As a first experiment in that direction, we tested the model with FLAN-T5, which features the same size variants and architecture as T5 but an extended skill set, and the model achieved good performance.

## 8.6 Conclusion

Towards combining unimodal LLM conversation skills with a robot’s sense and motor control capabilities, we have proposed a model that robustly integrates a robotic model with an LLM, where both share internal representations within a crossmodal fusion network with mutual read and write access. The CMT, inserted between the T5 encoder and decoder, adapts well to T5’s internal representation while requiring minimal training data and effort. This leaves the performance of T5 intact.

Our experiments show that the new CrossT5 model achieves nearly 100% accuracy on crossmodal action-language tasks and retains the performance of its T5 LLM on a representative NLP task. Furthermore, it performs well when practically evaluated in terms of action execution in simulation and remains robust when confronted with more natural language input. Thus, our approach of dissecting a frozen LLM and integrating it with a crossmodal architecture fulfils its intended objective. The resulting model excels in bidirectional language-action translation tasks without compromising the skill set of the language model. By design, it also demands little computational resources for training and is robust against more natural language commands even without dedicated training. Hence, this procedure efficiently equips an LLM with crossmodal capabilities.

So far, we have only dealt with embodied language learning in simulation. Moreover, our models have processed and produced language in the textual format. In contrast, HRI scenarios in practice demand embodied agents verbally interacting with humans in the real world. For this reason, our final model ELMiRA, which we will introduce in the next chapter, is aimed at open-ended, free-flowing human-

robot dialogue involving robotic object manipulation and scene understanding skills. As a modular approach, ELMiRA leverages SOTA off-the-shelf foundation models for full-fledged human-robot conversation requiring speech recognition, visual reasoning, dialogue management and speech synthesis capabilities. The experiments demonstrate that ELMiRA is a promising HRI approach applicable to diverse robotic platforms and scenarios.



# Seamless Integration of Foundation Models for Full-Fledged Dialogue in Robotic Manipulation

---

LLMs have recently achieved significant success in deep learning. The remaining challenges in robotics and human-robot interaction (HRI) still need to be tackled; yet, pre-trained off-the-shelf LLMs with advanced language and reasoning capabilities can provide solutions to problems in the field. In this chapter, we conceptualise an open-ended HRI scenario involving a humanoid robot communicating with a human while performing robotic object manipulation tasks at a table. To this end, we combine pre-trained general models of speech recognition, vision-language, text-to-speech and open-world object detection with robot-specific models of visuospatial coordinate transfer and inverse kinematics, as well as a task-specific motion model. Our experiments reveal robust performance by the language model in accurately selecting the task mode and by the whole model in correctly executing actions during open-ended dialogue. Our innovative architecture enables a seamless integration of open-ended dialogue, scene description, open-world object detection and action execution. It is promising as a modular solution for diverse robotic platforms and HRI scenarios<sup>1</sup>.

## 9.1 Introduction

The recent advances made in deep learning have led to successful applications of artificial intelligence such as chatbots, visual object detection, image captioning and speech recognition. The advent of the Transformer architecture [173] has brought about the possibility of training ever larger models with enormous amounts of

---

<sup>1</sup>The project page with the code and an exemplary video for this chapter can be found at <https://knowledge technologyhh.github.io/ELMiRA>.

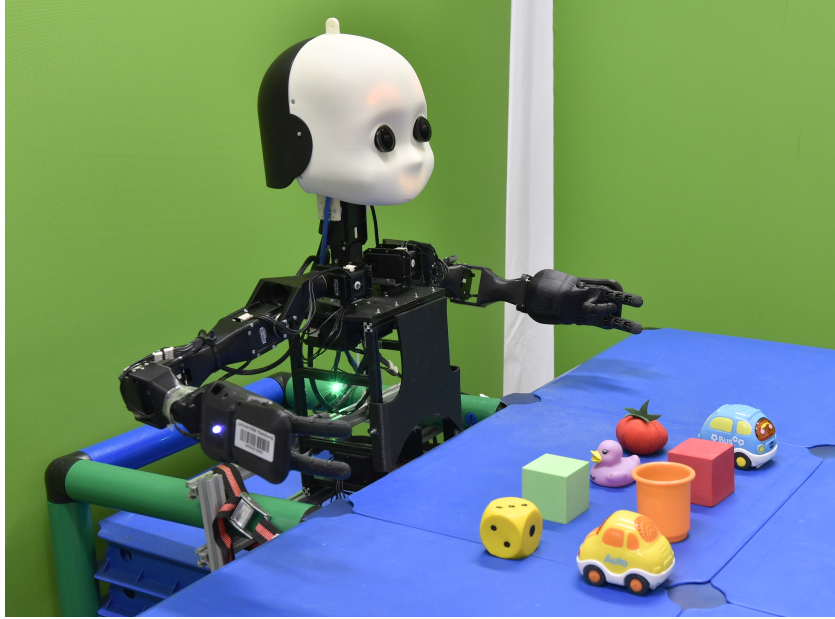


Figure 9.1: The NICO robot in our scenario. The scenario involves multiple toy objects being manipulated by NICO, which can describe the objects, based on user instructions. NICO can also have open-ended conversations with users.

data. Omnipresent LLMs, for example, have usually billions of parameters and are trained on trillions of text tokens. Although these large models perform well on tasks they are trained on, training them on such large datasets requires a considerable amount of computational power, which is not available to small labs or end users. Fortunately, the availability of pre-trained models renders training from scratch unnecessary. Pre-trained models can be either fine-tuned on a smaller dataset for a different domain or directly used as out-of-the-box solutions for numerous tasks.

In this chapter, we propose a modular approach, named ELMiRA (**E**mbodying **L**anguage **M**odels in **R**obot **A**ction), that integrates highly capable foundation models trained on big datasets for a conversational HRI scenario. Independent from the robotic platform, ELMiRA uses pre-trained models of automatic speech recognition (ASR), text-to-speech (TTS), vision-language (VLM) and object detection, which do not require further training. To adapt ELMiRA to our specific robotic setup, we employ a visuospatial coordinate transfer network and an inverse kinematics (IK) solver. Moreover, we devise a specific motion planner to perform several actions within our object manipulation scenario. By utilising multiple off-the-shelf models, our humanoid robot NICO [86] can manipulate objects on a table while conversing with humans in an open-ended fashion (Figure 9.1). This brings capabilities of pre-trained models such as conversing with a human, zero-shot open-vocabulary object detection and scene description into a robotic application.



## 9.2 Background: Leveraging LLMs for Robotics

Recently, many approaches have been proposed utilising LLMs or VLMs in the context of robotic manipulation [37, 74, 99, 147, 166]. SayCan [6] uses LLMs to split high-level instructions into executable actions, which are evaluated by a value function in terms of affordance. It chooses actions with a combined score of high common-sense relevance and affordance. Likewise, ViLa [72] employs GPT4-V [130] to decompose high-level instructions into low-level executable actions for object manipulation. Jointly processing vision and language via a VLM results in a task-focused understanding of the current scene based on the given instruction. ViLa outperforms SayCan in common-sense-related tasks and works with different modalities in input instructions, i.e. providing a goal image or a combined image-language goal. TidyBot [183] exploits common-sense knowledge inherent in LLMs for generalising to object types according to user preferences in a room-tidying task. It can generalise to objects regarding their attributes like colour or practical use. PIVOT [124] leverages VLMs for robotic control by casting robotic tasks as VQA problems. The scene images are annotated with possible movements for spatial control and fed to a VLM with an accompanying question querying for the best actions. The actions are iteratively optimised based on the VLM’s answers without requiring learning. This iterative process is repeated until the VLM outputs the best possible action. However, the experimental results show that ambiguities in depth information degrade PIVOT’s performance, indicating the need for an adaptive mechanism. Generally, the state-of-the-art approaches focus on action planning, while we capitalise on LLMs to facilitate open-ended multi-turn dialogue. Moreover, the zero-shot generalisation capabilities of ELMiRA allow us to detect and localise all objects as well as describe any given scene independent of the scenario.

## 9.3 Proposed Method: ELMiRA

To leverage state-of-the-art models in different domains, we devise the modular ELMiRA architecture, shown in Figure 9.2. It is composed of an ASR, a VLM, an object detector, a visuospatial coordinate transfer unit, a motion planner, an IK solver and a TTS model. We evaluate it in a tabletop object manipulation task-oriented HRI scenario where a user communicates verbally with the robot. The conversation can range from chitchat to task-related visual processing requiring action commands. The robot needs to know when to speak and when to execute an action according to the user instruction.

### 9.3.1 Vision-Language Model

The VLM assumes the role of a dialogue manager. It receives user input in textual format via the ASR model (Whisper [141]) and sends textual output to the TTS model (VITS [87]) to produce it as speech. We use GPT-4V as the VLM, while

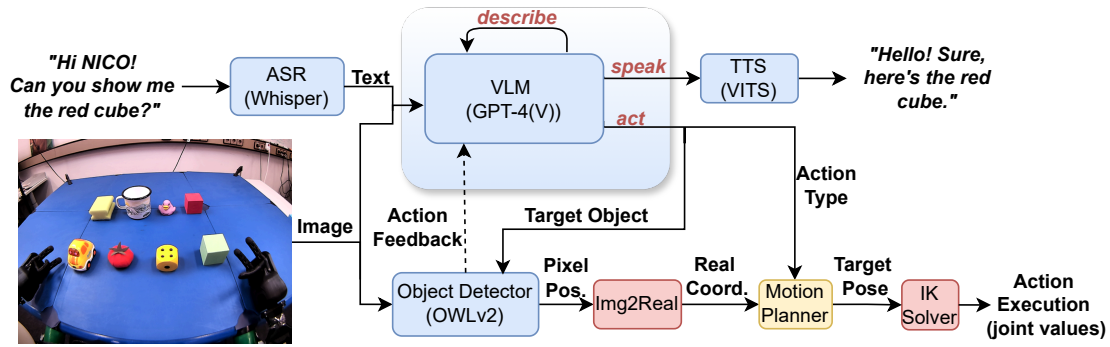


Figure 9.2: The ELMiRA architecture. It accepts as input human user speech and images from the robotic eye camera; it outputs speech and executes actions. Blue modules denote pre-trained off-the-shelf models, red modules denote robotic platform-specific models and the yellow module is task-specific. Akin to a conductor of an orchestra, the VLM decides whether to output speech (*speak* mode), describe the current scene (*describe* mode) or trigger a robot action (*act* mode).

also employing the GPT-4 LLM via the OpenAI API. We utilise both the text-only GPT-4 and the image-allowing GPT-4V since the GPT-4V is not yet available as a conversational assistant that can manage a dialogue. We therefore deploy GPT-4 as the chatbot that has conversation memory, while triggering a one-time response only GPT-4V instance when visual processing is required. Based on the user input and visual observation, GPT-4 chooses one of the three task modes: *speak*, *describe* and *act*.

- ***speak***: when the given user input is not related to the tabletop scenario, ELMiRA generates a textual output via GPT-4, which is passed to the TTS model to produce speech.
- ***describe***: when the user asks the robot to describe what it sees on the table, the current scene image from NICO’s perspective is fed to the GPT-4V with the user input. The GPT-4V then passes the description of the tabletop scene to GPT-4 which in turn triggers the TTS to produce speech.
- ***act***: when the user asks the robot to manipulate an object, the VLM triggers the action mode and infers the action type and target object. The target object name is passed to the object detector alongside the current image to localise the target object in pixel space. The pixel positions are then passed to the Visuospatial Coordinate Transfer model which locates the real-world coordinates of the object. These coordinates and the action type are fed to the motion planner to find the target arm pose including orientation. The IK solver uses the target pose to produce the joint angle values of the suitable arm for action execution.

### 9.3.2 Object Detection

We deploy the open-vocabulary object detection model OWLv2 [118] to detect the target object on the table. OWLv2 is triggered in the *act* mode after GPT-4 extracts the target object name from the user action command. It receives the current scene image through the robot eye camera as well as the target object name as input. In principle, OWLv2 can detect multiple instances of the same object but we choose the one with the highest score and extract the pixel coordinates of the bottom centre of the bounding box. In case the target object cannot be found on the table or is located in an unreachable position, the object detection module informs the VLM that the action cannot be executed.

### 9.3.3 Visuospatial Coordinate Transfer

The Img2Real module (Figure 9.2) transforms the pixel coordinates to 3D real-world coordinates. It uses a multi-layer perceptron (MLP) trained as an implicit energy-based model (EBM) [48] with an InfoNCE loss function and sampling-based derivative-free optimisation for inference. We train the implicit MLP in advance by distinguishing a set of uniformly distributed real-world coordinates from random counter-examples based on the corresponding points in image pixel space. When deployed, it finds the real-world coordinates for given pixel positions of the target object by iteratively resampling random candidates based on their predicted probabilities and adding Gaussian noise.

### 9.3.4 Motion Planner

Our motion planner decides whether to use the left or right arm of the robot based on the real-world coordinates of the target object and outputs the corresponding end-effector trajectory of the chosen arm for a given action type (i.e. *show*, *touch*, *push-forwards*, *push-leftwards* and *push-rightwards*). Each target pose in the trajectory consists of the Cartesian  $(x, y, z)$  position of the end-effector and its target orientation as a unit quaternion.

### 9.3.5 Inverse Kinematics Solver

The corresponding joint configurations needed to execute the robot arm’s trajectory are computed using EvoIK [49], an evolutionary inverse kinematics solver which aligns the forward kinematics of its population with the target pose by minimising the weighted sum of the Euclidean distance between the position vectors and the geodesic distance between the orientation quaternions. The trajectory is computed iteratively, using the previous joint angles as the initial centre of the population for the following step to accelerate the computations and guide the algorithm to find solutions which are close to each other.

Table 9.1: Mode detection success. We measure the mode selection performance of ELMiRA for each of the three predefined modes.

Act	Describe	Speak
86.67±4.71%	46.67±28.67%	100.0±0.0%

## 9.4 Experiments and Results

We conduct two sets of experiments with the NICO robot on our tabletop setup, aimed at evaluating the robustness of ELMiRA in two aspects: mode selection and action execution.

### 9.4.1 Mode Selection Experiments

We test the mode selection performance of our method by having three instances of a scripted human-robot conversation, where we check whether the VLM detects the correct mode intended by the user. The detection success rate for each mode is given in Table 9.1. In most cases (77% on average), ELMiRA understands the user’s intention and switches to the correct mode. However, frequently the VLM refuses to describe a scene in *describe* mode and goes into *speak* mode instead. In the few cases, when it refuses to trigger the *act* mode, it also chooses the *speak* mode.

Table 9.2: Action execution success. We measure the action execution performance of ELMiRA for pushing objects forwards, to the left and to the right. The push action is considered successful if the target object moves in the intended direction for at least the distance of 2 cm.

Forwards	Leftwards	Rightwards
79.17%	86.96%	72.0%

### 9.4.2 Action Execution Experiments

The action execution experiments involve three directions of push (*forwards*, *leftwards* and *rightwards*) on 8 different objects (a sponge, a die, a rubber duck, a toy tomato, a green cube, a toy car, a cup and a red cube). We use a minimum threshold of 2 cm displacement in the direction of the intended push action as an objective success metric. Table 9.2 shows the action execution results per push direction. Overall NICO executes the given action with an average accuracy of 79%. The *push-leftwards* action is the most successful, while *push-rightwards* is the least successful action. Figure 9.3 displays the distribution of target object displacements

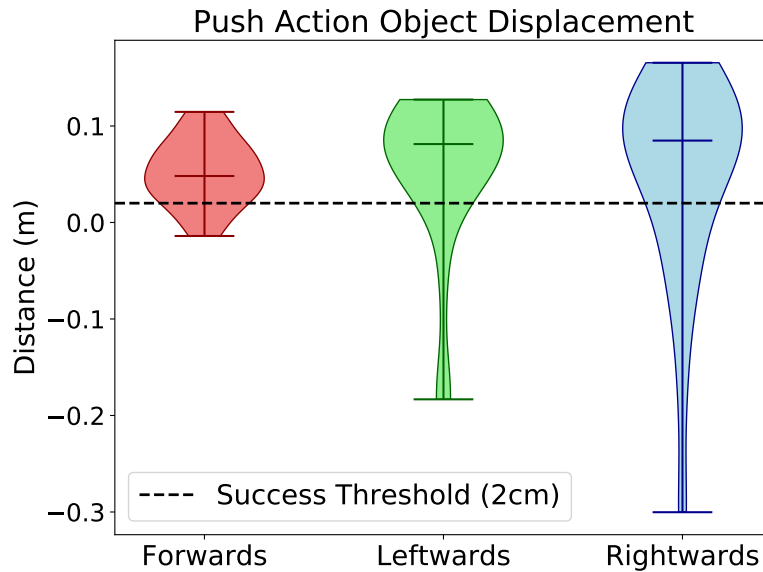


Figure 9.3: Object displacements per push direction. The distributions of displacement in metres are visualised in a distinct colour for each of the three push directions. The dashed line represents the success threshold of 0.02 m.

in the intended direction; samples over 2 cm (dashed line) are considered correct. NICO clearly pushes the target object in the right direction in most cases, with few exceptions in which the target is moved in the wrong direction due to mistakes in the IK solution. Apart from the push actions, we tested ELMiRA with show and touch actions, both of which were distinguishably and correctly executed.

## 9.5 Discussion

The advancement of the field by novel pre-trained models in various tasks has allowed us to design the modular ELMiRA architecture for a comprehensive HRI scenario. The publicly available ASR model Whisper makes it possible to convert user speech input into the textual format, which our VLM subsequently processes. The availability of GPT-4V as a VLM provides us with a dialogue manager capable of controlling the flow of the conversation by choosing the appropriate mode at each turn. The open-vocabulary object detector OWLv2 enables object type-independent localisation and generalisation to the action space. Lastly, the end-to-end text-to-speech model VITS facilitates the conversion of the textual format of the response by the VLM back into the speech format for an interactive verbal conversation with a human partner.

Apart from these out-of-the-box models, we use a few robotic platform- and task-specific modules, which need to be trained or manually designed. We have managed to keep the number of these modules minimal to make ELMiRA adaptable to diverse robotic platforms and scenarios. The backbone of our approach

does not require any training as it relies on pre-trained models.

ELMiRA also suffers from several limitations. Firstly, as we noticed during the mode selection experiments, GPT-4 occasionally hallucinates and refuses to describe the scene when asked to have a look at the table. We describe the capabilities of NICO and the scenario details in the initial prompt given to GPT-4. Nevertheless, it states that it has no vision access, preventing it from processing images when it should be able to do so. Although these mistakes can be rectified by further user intervention, insisting that the model should switch to the *describe* mode, we recommend improving the initial prompt to minimise instances of hallucination in the future. During the action execution experiments, we observed the second limitation of our approach: occasionally, the actions cannot be performed correctly, causing the target object to be moved in the wrong direction. This displacement happens due to the IK solver’s imprecise calculation of the joint angle trajectories. This type of error can be resolved by employing more recent IK solvers like CycleIK [58]. In addition, the VLM in our architecture can be used to detect the success of a performed action, as done by Du et al. [42], so that ELMiRA is aware of the success of its actions without requiring user feedback. Lastly, ELMiRA’s reaction and processing time may sometimes require longer response times, making it far from ideal for a real-time live demonstration. These waiting times can be minimised by better parallelisation of different modules.

## 9.6 Conclusion

We have proposed a novel modular architecture leveraging recent progress in large language modelling, open-vocabulary object detection and speech recognition. Specifically, we have used pre-trained general ASR, VLM, object detection and TTS models alongside small robotic platform-specific and task-specific modules for a dialogue-based tabletop object manipulation scenario. ELMiRA is a starting point for facilitating full-fledged dialogue and robotic action through the seamless integration of open-world object detection, scene description and general conversational skills. By drawing inspiration from neuroscience concepts [179] and leveraging increasingly capable foundation models, our modular approach can widely benefit HRI.

In this thesis, we have introduced seven distinct neural network models tackling bidirectional action-language translation with a focus on different aspects at hand per model while iteratively expanding the capabilities of modelling embodied language learning. The following and final chapter will conclude this doctoral thesis by summarising and discussing the novel approaches introduced during this research to address our objectives and the ensuing research question, listing our contributions and elaborating on future directions to arrive at fully autonomous embodied agents capable of cooperating and solving everyday tasks with humans through interactive verbal communication.

## Conclusion

---

In this doctoral thesis, we have explored various methods governing language learning for embodied agents within the framework of crossmodal neural network architectures. As a result, this research encompasses several key advancements in the domain of language-instructed robotic object manipulation, including improved language comprehension, training- and test-time-consistent action-language translation, efficient utilisation of labelled data, the ability to generalise to larger action spaces, a recipe for embodying LLMs with action taking and a modular approach for HRI scenarios.

Firstly, we have relaxed the limitations on language understanding in bidirectional action-language modelling. Secondly, we have enabled flexible inference-time translation, removing the need for expert configuration on the model. Thirdly, we have addressed the shortage of labelled data by exploiting a more efficient Transformer-based attention mechanism that can learn action-language mappings using predominantly unimodal data.

Moreover, we have tackled generalisation to the action space by initially pre-training on relatively small datasets and then fine-tuning with direct feedback from the simulation environment. This approach leads to improved action precision on multiple object manipulation tasks. Further, we have capitalised on the linguistic knowledge embedded in LLMs for a bidirectional action-language model by replacing its language components with a pre-trained LLM. This integration combines language-instructed action execution skills with language-only task-solving capabilities. Lastly, we have unleashed the full potential of powerful deep learning models from computer vision and NLP by developing a modular approach tackling open-ended free-flowing dialogue and closed-loop action execution in HRI scenarios.

In sum, we have developed various ANN model architectures, leveraging autoencoder-based networks (e.g. variational and convolutional autoencoders) and multimodal fusion mechanisms (e.g. binding loss, gated attention, Transformer-based crossmodal attention). We have trained these models by employing diverse learning paradigms, ranging from supervised learning to unsupervised learning to reinforcement learning. To evaluate our models, we have generated paired robotic action-language instruction datasets using different simulation environments such



as CoppeliaSim. Utilising these datasets and simulations, we have conducted comprehensive experiments on unimodal and crossmodal action-language translation, performing several tabletop object manipulation tasks such as moving cubes, pressing buttons and lifting cups. Last but not least, in order to evaluate our approaches consistently and accurately, we have used various measures such as sentence accuracy, joint angle value deviation, target object selection and action precision rates.

In the following sections, we will start by comparing our approaches with each other and elucidate their respective novelties. We will then elaborate on how this dissertation has addressed the all-important research question defined in the Introduction, highlighting the contributions made by this doctoral work and the new knowledge produced as a result. Following that, we will outline the limitations of this work and explain what it entails for future research and how future researchers can benefit from it. We will conclude this thesis with reflections on our overall contribution to the field, particularly in the pursuit of developing fully autonomous embodied language learning agents capable of serving as companions for humans in real-world settings.

## 10.1 Discussion

This thesis contributes to the domain of language grounding by introducing a sequence of novel multimodal NN architectures and their thorough analysis involving various experiments in robotic object manipulation settings. All of these NN architectures allow bidirectional robot action-language description translation; i.e. contrary to most approaches in the field, they allow an embodied agent to not only perform object manipulation actions according to given user instructions but also describe an executed action in natural language according to visual and proprioceptive observations. The last two models go beyond action-describing language input understanding and output production by integrating LLMs into their architectures and capitalising on the language skills embedded in LLMs via transfer learning. Alongside robotic object manipulation, these two models can perform language-only tasks such as machine translation and text generation while potentially comprehending unrestricted language input.

All NN architectures introduced within the scope of this dissertation work with multiple modalities. As input, they accept natural language instructions (as text or audio), sequences of images and proprioceptive observations (as joint angle values or end-effector poses). All architectures except the final architecture use the channel separation technique to extract visual features accurately representing object colours. Utilising various crossmodal fusion techniques, they combine these three input modalities to comprehend the task at hand, observe the scene visually and perceive the location and movement of robotic actuators. As a result of this crossmodal combination, they provide a unified representation of the task and the current scene. The unified crossmodal representation is then exploited to output the two modalities: language and proprioception. Employing different decoder

networks for each output modality, all our architectures produce natural language and joint angle values or end-effector poses. Therefore, excluding the final modular architecture, which utilises multiple off-the-shelf models in combination, they can all be treated as autoencoders.

While sharing many similarities functionally and architecturally, the models also exhibit many differences. The first architecture, PVAE, is designed to ease the problem of strict action-language mapping by accepting the use of synonymous words to describe a given action. Using a Bayesian-based sampling method, i.e. variational autoencoders, PVAE forms one-to-many associations between action and language. The second architecture, PVAE-BERT, builds upon this premise and further relaxes this restriction by employing a pre-trained language model, i.e. BERT. Replacing the previous LSTM-based language encoder with BERT leads to comprehension of almost unconstrained language instructions. However, both PVAE and PVAE-BERT require a binding loss for implicit alignment of the language and action stream representations. Thus, they must be configured at test time to conduct action-language translations in the desired direction. For example, when the task is to translate from action to language, only the action encoder and language decoder are used, removing the language encoder and action decoder from the architecture in practice. Consequently, the two models cannot handle arbitrary action-language translations without expert intervention during inference.

The third model, PGAE, addresses the problem of flexible action-language translation during inference. It replaces the implicit crossmodal binding mechanism of PVAE with a learnt gating mechanism, i.e. GMU, that explicitly connects the action and language streams. Provided by modality-specific features, the GMU network produces a joint multimodal representation vector. Besides the utilisation of GMU, PGAE introduces signal words that are prepended to the language input to guide the model in the desired translation direction; for instance, the ‘execute’ signal tells the model to perform language-to-action translation, whereas ‘describe’ asks the model to translate from action to language. Owing to GMU and signal words, PGAE can be deployed at test time, consistent with training, without requiring architectural reconfiguration based on the desired translation direction. For all of the translation tasks, GMU uses its entire architecture and outputs both language and action. Specifically, when the task is action-to-language translation, the language decoder outputs the description of the given action as expected, while the action decoder outputs the same joint angle values as the final time step of the action to keep the agent still. In contrast, when the ‘execute’ signal is included in the language input, PGAE outputs the sequence of joint angles required to perform the desired action while the language decoder remains silent. Therefore, PGAE is a valuable approach towards building autonomous agents that can naturally interact with humans.

The fourth architecture, PTAE, tackles the challenge of overreliance on supervised learning that necessitates the presence of large amounts of labelled data. PTAE replaces the GMU bottleneck with CMT which enhances the efficiency of associating robot actions with their corresponding language descriptions. The CMT bottleneck applies language-conditioned attention to the unified visual and pro-

prioceptive observation representations by using language as queries and action as keys and values. Relying on unsupervised learning by training mainly on repeating actions and language descriptions, PTAE requires considerably fewer supervised iterations and samples to achieve superior language-to-action and action-to-language translation performance. PTAE mimics the developmental language learning phenomenon where children usually learn language by interacting with objects in their surroundings and casually repeating what they hear from their parents with minimal supervision.

The fifth model, XBiT, brings another learning paradigm into play: reinforcement learning. It employs a two-stage learning strategy where the model is first trained on a pre-collected dataset in a supervised fashion and then fine-tuned via RL by observing and acting in the environment and receiving positive feedback when completing the desired action successfully. To have an SL- and RL-compatible architecture, XBiT replaces the sequential action output decoder with feedforward layers, which enable XBiT to behave in an observe-and-act loop when performing an action. Combining SL with RL helps XBiT master continuous dexterous robotic object manipulation, leading to superior performance in action precision. XBiT brings the language-conditioned robotic object manipulation modelling closer to the real-world application phase. In the future, XBiT can be deployed in a real-world HRI scenario to complement a free-flowing closed-loop dialogue.

With the sixth architecture, CrossT5, we redirect our attention to advancing the linguistic capabilities of bidirectional robotic action-language modelling. Inspired by the progress in language modelling, CrossT5 adopts the T5 LLM into our previous Transformer-based PTAE architecture in order to equip it with NLP abilities embedded in LLMs. The choice of an encoder-decoder LLM simplifies its integration into the PTAE architecture. Moreover, CrossT5 exploits a distinct multimodal integration mechanism that fuses different modalities in the middle layers of the model via the CMT network, in contrast to the mainstream multimodal integration of early fusion commonly followed by VLMs, which tokenises the inputs from different sources in a standardised manner, irrespective of modalities. The compatibility of T5 and PTAE architectures, leading to their seamless integration, means that CrossT5 needs only a sliver of the LLM dataset and computational power required to train LLMs for learning bidirectional robotic action-language description associations while retaining the natural language skills of T5. Not only does CrossT5 achieve competitive performance in English-to-German and bidirectional action-language translations, but it also displays robustness in action execution when it encounters differently phrased language commands owing to the language understanding ability of the pre-trained T5.

The seventh and final architecture, ELMiRA, targets real-world HRI scenarios involving verbal communication alongside user-guided robotic action execution. Unlike all our other architectures, ELMiRA is a modular approach chiefly composed of pre-trained models of speech recognition, object detection, speech synthesis and visual language. When constructing ELMiRA, we took account of the recent developments in deep learning brought about by increasingly capable foundation models. ELMiRA brings the success of language modelling to an HRI application

by embodying an LLM with an action-taking humanoid robot. Besides, all our prior approaches are trained and tested in simulation, whereas ELMiRA works in the real world. It accepts audio as input and produces speech, which makes the user experience more interactive and appealing. Since it leverages the advanced capabilities of off-the-shelf models like GPT-4 and OWLv2, ELMiRA requires minimal training only for its robotic platform-specific modules. Therefore, it can serve as a blueprint for diverse future HRI scenarios involving a humanoid robot capable of full-fledged dialogue, scene understanding, zero-shot object localisation and object manipulation. As our most powerful approach, ELMiRA is the culmination of all the architectures in this thesis, enabling free-flowing, open-ended human-robot conversation with closed-loop action execution on any given object according to human instructions.

## 10.2 Addressing the Research Question

While tackling the research question of **how to achieve more robust, free-form language learning, which is not restricted to a set of predefined descriptions, for a humanoid robot utilising neurocognitively plausible mechanisms**, we have reached several conclusions. These are organised according to their relation to the research objectives (see [Section 1.2](#)), which are provided in brackets, and followed by the respective contributions made.

**Substituting regular autoencoders with their Bayesian equivalents enables one-to-many action-to-language translation and scales well with extended data, while utilising pre-trained language models enhances language comprehension (*Objective 1*).** We have shown that Bayesian methods like VAEs are a better choice for both the association of a robot action with multiple synonymous language descriptions and more varied and extensive datasets due to their use of stochastic generation for hidden representations since stochastic generation in the latent feature space allows minor variations in hidden representations ([Chapter 3](#)). Moreover, we have successfully incorporated an earlier pre-trained language model, i.e. BERT, into one of our architectures as the language encoder. This adoption leads to a more advanced language understanding capacity as it allows the model to go beyond processing language inputs conforming to predefined grammar and enables the comprehension of differently phrased action execution commands ([Chapter 4](#)).

**Employing explicit multimodal fusion mechanisms and guiding the model in the desired translation direction results in flexible translation, while commonly modelling action-language mapping for different perspectives negatively affects performance (*Objective 2*).** We have integrated fully connected neural networks (i.e. GMU and CMT) as explicit multimodal fusion mechanisms into our architectures to have unified multimodal hidden representations and included the intended action-language translation direction in

the language input which equips a bidirectional action-language translation model with the flexible translation ability at test time, consistent with the training conditions, i.e. training- and test-time-consistent model, eliminating the necessity of an expert operator configuring the model during deployment (Chapters 5 and 6). We have also modelled self- and opposite-sitting-robot actions (first-person and counterpart actions) together with the PGAE model, which has revealed that although the robot can recognise and imitate the actions of the opposite-sitting robot to a degree, the overall model performance on action-to-language and language-to-action translations moderately declines (Chapter 5).

**Self-attention-based multimodal fusion outperforms gating-based fusion under limited supervision, while contradictory inputs degrade the model performance akin to human behaviour (*Objective 3*).** We have ascertained that a Transformer-based multimodal fusion network is more performant and efficient than a simpler gated network in bidirectional action-language translation, especially when supervised data is lacking by primarily exploiting unsupervised learning on unlabelled data as the CMT-based PTAE model outperforms its GMU-based alternative PGAE. The introduction of PTAE minimises the need for labelled data for action-language mapping as it primarily relies on unsupervised learning by repeating actions or language descriptions. PTAE has also exhibited biologically plausible behaviour similar to humans when exposed to contradictory action and language inputs, as contradictory multimodal inputs degrade the action and language performance in our psychology-inspired experiments (Chapter 6).

**Asymmetrically combining distinct learning paradigms permits generalisation to a continuous action space (*Objective 4*).** We have employed unsupervised, supervised and reinforcement learning in different stages of model training for the XBiT approach, which significantly improves the language-to-action model performance while preserving the perfect language production ability. The RL fine-tuning in simulation after SL pre-training on pre-collected data leads to a more dexterous and precise action execution in multiple robotic object manipulation tasks. Despite language-to-action-only fine-tuning, the model retains its ability to translate from action to language with 100% accuracy (Chapter 7).

**Intra-LLM integration precludes the need for a large-scale paired action-language dataset, while an orthogonal modular approach leveraging pre-trained models facilitates interactive, open-ended human-robot dialogue (*Objective 5*).** Our innovative novel intra-LLM integration approach, CrossT5, has proven to be an effective and inexpensive solution, on the one hand, for enabling action execution in the environment for pre-trained LLMs and, on the other hand, for carrying the language-only task-solving skill set of LLMs over to bidirectional action-language modelling. CrossT5 inherits natural language capabilities from the pre-trained LLM T5 for a versatile bidirectional action-language model with advanced language comprehension and production

skills, trained on a remarkably smaller combined machine translation and tabletop object manipulation dataset (Chapter 8). Furthermore, we have developed a modular HRI approach, ELMiRA, built upon pre-trained out-of-the-box foundation models. ELMiRA facilitates spontaneous, interactive, free-flowing, verbal human-robot conversations alongside closed-loop motor control in the real world. We have demonstrated that the VLM-based dialogue manager can control the flow of dialogue by switching between different output modes, while the minimal task- and robotic platform-specific modules allow the successful execution of robotic actions. ELMiRA is the most advanced of all our approaches as it works in the real world with the ability to engage in interactive, unrestricted, free-flowing conversation, describe scenes and perform actions without being limited to specific objects. It does not require further training and encompasses skills of zero-shot object detection, scene understanding, vision-language modelling and robotic action execution, applicable to different robotic platforms and HRI scenarios (Chapter 9).

### 10.3 Future Research

We believe that we have significantly contributed to the knowledge present in the domain of embodied language learning with this doctoral thesis. Nevertheless, future research is essential to achieving fully autonomous companion robots that can cooperate and solve everyday tasks with humans through advanced verbal communication and motor control skills.

Excluding ELMiRA, all of our bidirectional action-language translation models are trained and tested either on pre-collected simulation data or directly in simulation. In order to equip them with real-world action execution capabilities, future work can explore sim-to-real transfer [195] as well as training on real-world data and conducting experiments with robots in the real world. Additionally, it is advisable to diversify the manipulation tasks and extend the variety and quantity of objects. To surpass low-level granular motor control, robotic planning models like SayCan and PaLM-E can be employed. These models should also assist in tackling multi-task learning where a single set of weights is used to learn multiple object manipulation tasks.

Another aspect that future work can further analyse is the language comprehension capabilities of our models, especially CrossT5 and ELMiRA. Even though we tested these models with diverse language instructions for action execution regarding their robustness to different phrasings, we advise future researchers to collect large quantities of language instructions via crowdsourcing to investigate the viability of using pre-trained LLMs within model architectures to relate from language to motor control. As ELMiRA enables interactive verbal communication with humans, another option could be conducting HRI experiments with several human subjects to evaluate our modular approach further in terms of criteria such as likeability, satisfaction and awareness.

By utilising SL pre-training and RL fine-tuning, XBiT expands our previous models as it reduces the need for expert-defined action trajectories. Nevertheless,



we used a simple online RL algorithm, i.e. REINFORCE, for the fine-tuning stage. XBiT can be tested with more sophisticated RL algorithms, e.g. DDPG and PPO, to boost its action execution performance. Besides, inverse RL can be considered to directly learn value functions from expert trajectories for tasks for which rewards cannot be easily defined. Like CrossT5, the XBiT architecture can be modified to incorporate powerful LLMs such as GPT-4 and Llama 3 to enable complex natural language capabilities. Lastly, future work can explore how to embed XBiT into the ELMiRA framework for continuous human-robot dialogue with the possibility of learning closed-loop action execution for any given manipulation task.

With CrossT5, we transfer only the English-to-German translation skills of the small T5 variant to our bidirectional action-language modelling in practice. As T5-small is limited in NLP tasks, it should be straightforward to replace it with its larger variants or even its enhanced version FLAN-T5. As long as the LLM in question has an encoder-decoder architecture, it is trivial to switch LLMs in our intra-LLM multimodal fusion concept. However, opting for a more common decoder-only LLM with text generation capabilities that can be used for dialogue applications as in the case of ELMiRA may require preliminary exploration to find the exact layers between which to divide the LLM into two parts. Another avenue worth exploring is to add a few trainable layers between frozen LLM network layers as done for fine-tuning in multi-task learning.

Our modular HRI solution, ELMiRA, can also be further improved in terms of dialogue and action execution capabilities. For example, the current evolutionary solution, EvoIK, can be replaced with more recent IK solvers such as CycleIK. Moreover, improving the parallelisation of different modules can minimise the waiting times to tailor the approach for real-time human-robot interactions. Another interesting approach could be incorporating our previous learning models, which are more biologically plausible, into the ELMiRA concept. Last but not least, the models used in the different modules of the ELMiRA architecture can be replaced by more capable foundation models as they become available.

## 10.4 Final Remarks

In conclusion, the overall contribution of this thesis is advancing the language comprehension and production capabilities as well as improving motor control skills of bidirectional action-language models by using neurocognitively plausible mechanisms (e.g. multimodal fusion, channel separation in vision and crossmodal attention) and multiple learning paradigms (i.e. unsupervised learning, SL and RL), realised with a humanoid robot in simulation and real-world settings. We have shown that incorporating explicit multimodal fusion networks for action-language associations and including the desired translation direction in the language input enable flexible bidirectional translation between robot actions and language descriptions. In addition, utilising multiple learning paradigms asymmetrically, e.g. SL and RL, reduces the amount of labelled action sequences necessary and improves the action execution performance on multiple object manipulation tasks. Furthermore, pre-



trained models like open-vocabulary object detection, speech recognition, TTS and LLMs can be employed in a modular approach for diverse HRI scenarios involving free-flowing verbal communication between a robot and a human and language-instructed action execution in the real world. Therefore, we have paved the way towards developing an artificial intelligent agent capable of collaborating with humans on daily tasks via superior communication and sensorimotor capabilities.



---

# Nomenclature

---

<b>AE</b>	Autoencoder
<b>ANN</b>	Artificial Neural Network
<b>ASR</b>	Automatic Speech Recognition
<b>BC</b>	Behavioural Cloning
<b>BERT</b>	Bidirectional Encoder Representations from Transformers
<b>CAE</b>	Convolutional Autoencoder
<b>CLIP</b>	Contrastive Language-Image Pre-training
<b>CMT</b>	Crossmodal Transformer
<b>CNN</b>	Convolutional Neural Network
<b>CS-CAE</b>	Channel-Separated Convolutional Autoencoder
<b>DDPG</b>	Deep Deterministic Policy Gradient
<b>ELMiRA</b>	Embodying Language Models in Robot Action
<b>FFW</b>	Feedforward Layer
<b>GMU</b>	Gated Multimodal Unit
<b>GPT</b>	Generative Pre-trained Transformer
<b>HRI</b>	Human-Robot Interaction
<b>IK</b>	Inverse Kinematics
<b>LLM</b>	Large Language Model
<b>LSTM</b>	Long Short-Term Memory Network
<b>MHA</b>	Multi-Head Attention
<b>MLP</b>	Multilayer Perceptron
<b>MSE</b>	Mean Squared Error
<b>NICO</b>	Neuro-Inspired Companion
<b>NLP</b>	Natural Language Processing
<b>NN</b>	Neural Network
<b>NRMSE</b>	Normalised Root-Mean-Squared Error

<b>PGAE</b>	Paired Gated Autoencoders
<b>PPO</b>	Proximal Policy Optimisation
<b>PRAE</b>	Paired Recurrent Autoencoders
<b>PTAE</b>	Paired Transformed Autoencoders
<b>PVAE</b>	Paired Variational Autoencoders
<b>ReLU</b>	Rectified Linear Unit
<b>RL</b>	Reinforcement Learning
<b>RLHF</b>	Reinforcement Learning from Human Feedback
<b>RNN</b>	Recurrent Neural Network
<b>SL</b>	Supervised Learning
<b>SOTA</b>	State of the Art
<b>TTS</b>	Text to Speech
<b>VAE</b>	Variational Autoencoder
<b>VL</b>	Vision Language
<b>VLM</b>	Vision Language Model
<b>VQA</b>	Visual Question Answering
<b>XAI</b>	Explainable Artificial Intelligence
<b>XBiT</b>	Crossmodal Bidirectional Transformer

# Code Snippets

## PVAE & PVAE-BERT

### Model Definition

```

1 # Peephole LSTM used as encoders and decoders
2 class PeepholeLSTM(nn.Module):
3     def __init__(self, input_size, hidden_size, peephole=False, forget_bias=0.0):
4         super().__init__()
5         self.input_sz = input_size
6         self.hidden_size = hidden_size
7         self.peephole = peephole
8         self.W = nn.Parameter(torch.Tensor(input_size+hidden_size, hidden_size*4))
9         self.peep_i = nn.Parameter(torch.Tensor(hidden_size))
10        self.peep_f = nn.Parameter(torch.Tensor(hidden_size))
11        self.peep_o = nn.Parameter(torch.Tensor(hidden_size))
12        self.bias = nn.Parameter(torch.Tensor(hidden_size * 4))
13        self.forget_bias = forget_bias
14        self.init_weights()
15
16    def forward(self, x, sequence_len=None, init_states=None):
17        """Assumes x is of shape (sequence, batch, feature)"""
18        if sequence_len is None:
19            seq_sz, bs, _ = x.size()
20        else:
21            seq_sz = sequence_len.max()
22            _, bs, _ = x.size()
23        hidden_seq = []
24        if init_states is None:
25            c_t, h_t = (torch.zeros(bs, self.hidden_size).to(x.device), torch.
26                       zeros(bs, self.hidden_size).to(x.device))
27        else:
28            c_t, h_t = init_states
29        HS = self.hidden_size
30        for t in range(seq_sz):
31            x_t = x[t, :, :]
32            if sequence_len is not None:
33                if sequence_len.min() <= t+1:
34                    old_c_t = c_t.clone().detach()
35                    old_h_t = h_t.clone().detach()
36                # batch the computations into a single matrix multiplication
37                lstm_mat = torch.cat([x_t, h_t], dim=1)
38                if self.peephole:
39                    gates = lstm_mat @ self.W + self.bias
40                else:
                    gates = lstm_mat @ self.W + self.bias

```

```

41         g_t = torch.tanh(gates[:, HS * 2:HS * 3])
42     if self.peephole:
43         i_t, j_t, f_t, o_t = (
44             (gates[:, :HS]), # input
45             (gates[:, HS:HS * 2]), # new input
46             (gates[:, HS * 2:HS * 3]), # forget
47             (gates[:, HS * 3:]) # output
48         )
49     else:
50         i_t, f_t, o_t = (
51             torch.sigmoid(gates[:, :HS]), # input
52             torch.sigmoid(gates[:, HS:HS * 2]), # forget
53             torch.sigmoid(gates[:, HS * 3:]) # output
54         )
55     if self.peephole:
56         c_t = torch.sigmoid(f_t + self.forget_bias + c_t * self.peep_f) *
57             c_t + torch.sigmoid(i_t + c_t * self.peep_i) * torch.tanh(j_t)
58         h_t = torch.sigmoid(o_t + c_t * self.peep_o) * torch.tanh(c_t)
59     else:
60         c_t = f_t * c_t + i_t * g_t
61         h_t = o_t * torch.tanh(c_t)
62     out = h_t.clone()
63     if sequence_len is not None:
64         if sequence_len.min() <= t:
65             c_t = torch.where(torch.tensor(sequence_len).to(c_t.device) <=
66                 t, old_c_t.T, c_t.T).T
67             h_t = torch.where(torch.tensor(sequence_len).to(h_t.device) <=
68                 t, old_h_t.T, h_t.T).T
69             out = torch.where(torch.tensor(sequence_len).to(out.device) <=
70                 t, torch.zeros(out.shape).to(out.device).T, out.T).T
71         hidden_seq.append(out.unsqueeze(0))
72     hidden_seq = torch.cat(hidden_seq, dim=0)
73     return hidden_seq, (c_t, h_t)
74
75 # Binding loss (description, action)
76 def aligned_discriminative_loss(lang, act, margin=1.0):
77     batch_size = int(lang.shape[0]) # number of actions
78     # replicate the descriptions according to no. of actions
79     lang_tile = torch.tile(lang, (batch_size, 1))
80     # do the same for actions
81     act_tile = torch.tile(act, (1, batch_size)).view(batch_size ** 2, -1)
82     # Calculate the euclidean distance between paired descriptions and actions
83     pair_loss = torch.sqrt(torch.sum(torch.square(lang - act), axis=1))
84     all_pairs = torch.square(lang_tile - act_tile)
85     loss_array = torch.sqrt(torch.sum(all_pairs, axis=1)).view(batch_size,
86         batch_size)
87     # Make representation of an action be far from that of its unpaired
88     # description
89     lang_diff = torch.unsqueeze(pair_loss, axis=0) - loss_array + margin
90     act_diff = torch.unsqueeze(pair_loss, axis=1) - loss_array + margin
91     lang_diff = torch.maximum(lang_diff.to('cuda'), torch.zeros(lang_diff.size()).
92         to('cuda'))
93     act_diff = torch.maximum(act_diff.to('cuda'), torch.zeros(act_diff.size()).to(
94         'cuda'))
95     mask = 1.0 - torch.eye(batch_size)
96     lang_diff = lang_diff * mask.to('cuda')
97     act_diff = act_diff * mask.to('cuda')
98     # return the binding loss
99     return torch.mean(lang_diff) + torch.mean(act_diff) + torch.mean(pair_loss)
100
101 # Method for Losses
102 def loss(output, gt_description, gt_action, B_bin, net_conf):
103     [L_output, B_output, L_z_mean, VB_z_mean, L_z_log_var, VB_z_log_var] = output
104     # Calculate the loss
105     B_output = B_output * B_bin[1:]
106     # description loss
107     L_loss = torch.mean(-torch.sum(gt_description * torch.log(L_output), 2))
108     B_loss = torch.mean(torch.square(B_output - gt_action)) # action loss (MSE)

```

```

101 # binding loss
102 share_loss = aligned_discriminative_loss(L_z_mean, VB_z_mean, net_conf.delta)
103 # add the regularisation loss (KLD)
104 L_reg = L_z_log_var.exp()-L_z_log_var-1+L_z_mean.square()
105 L_reg = torch.mean(0.5 * torch.sum(L_reg, axis=-1))
106 VB_reg = VB_z_log_var.exp() - VB_z_log_var - 1 + VB_z_mean.square()
107 VB_reg = torch.mean(0.5 * torch.sum(VB_reg, axis=-1))
108 # total loss
109 loss = net_conf.L_weight*L_loss + net_conf.B_weight*B_loss + net_conf.S_weight
    *share_loss + net_conf.KL_weight*L_reg + net_conf.KL_weight*VB_reg
110 # return losses
111 return L_loss, B_loss, share_loss, loss, L_reg, VB_reg
112
113 # BERT language encoder
114 class BertEncoder(nn.Module):
115     def __init__(self, params):
116         super(BertEncoder, self).__init__()
117         self.params = params
118         self.tokeniser = BertTokenizer.from_pretrained('bert-base-uncased')
119         self.encoder = BertModel.from_pretrained('bert-base-uncased')
120
121     def forward(self, inp):
122         # Use the vocabulary to feed descriptions
123         file = open('../vocabulary.txt', 'r')
124         vocab = file.read().splitlines()
125         file.close()
126         t = inp[:, :, :].argmax(axis=-1)
127         descriptions = []
128         for i in range(inp.shape[1]):
129             sentence = ''
130             for k in range(1, inp.shape[0] - 1):
131                 sentence += vocab[t[k, i]] + ' '
132             descriptions.append(sentence[:-1])
133         encoded_input = self.tokeniser(descriptions, return_tensors='pt', padding=
            True)
134         output = self.encoder(**encoded_input.to(self.encoder.device))
135         v = self.mean_pooling(output, encoded_input['attention_mask'])
136         return v
137
138 # Paired Variational Autoencoders
139 class PVAE(nn.Module):
140     def __init__(self, params):
141         super(PVAE, self).__init__()
142         self.params = params # obtain the parameters
143         #Encoders
144         self.lang_encoder = Encoder(self.params, True)
145         self.action_encoder = Encoder(self.params, False)
146         # Bottleneck
147         self.lang_mean = nn.Linear(self.params.L_num_units*self.params.
            L_num_layers*2, self.params.S_dim)
148         self.lang_log_variance = nn.Linear(self.params.L_num_units*self.params.
            L_num_layers*2, self.params.S_dim)
149         self.action_mean = nn.Linear(self.params.VB_num_units*self.params.
            VB_num_layers*2, self.params.S_dim)
150         self.action_log_variance = nn.Linear(self.params.VB_num_units*self.params.
            VB_num_layers*2, self.params.S_dim)
151         # Initial states
152         self.initial_lang = nn.Linear(self.params.S_dim, self.params.L_num_units*
            self.params.L_num_layers*2)
153         self.initial_act = nn.Linear(self.params.S_dim, self.params.VB_num_units*
            self.params.VB_num_layers*2)
154         # Decoders
155         self.lang_decoder = Decoder(self.params, True)
156         self.action_decoder = Decoder(self.params, False)
157
158     def forward(self, inp):
159         # Encode language

```



```

160     encoded_lang = self.lang_encoder(inp['L_bw'], inp['L_len'].int().numpy(),
161                                     True)
162     # Encode action
163     # Prepare vision-joint angle input
164     VB_input = torch.cat([inp['V_bw'], inp['B_bw']], dim=2)
165     encoded_act = self.action_encoder(VB_input, inp['V_len'].int().numpy())
166     VB_input_f = inp['VB_fw'] #Initial input for action decoder
167     # Get means and variances
168     L_z_mean = self.lang_mean(encoded_lang)
169     L_z_log_var = self.lang_log_variance(encoded_lang)
170     VB_z_mean = self.action_mean(encoded_act)
171     VB_z_log_var = self.action_log_variance(encoded_act)
172     # Get z_lang and z_act using the reparameterisation trick
173     L_sampling = self.reparameterise(L_z_mean, L_z_log_var)
174     VB_sampling = self.reparameterise(VB_z_mean, VB_z_log_var)
175     # Get initial states for decoders
176     L_dec_init_state = self.initial_lang(L_sampling)
177     VB_dec_init_state = self.initial_act(VB_sampling)
178     # Reproduce the inputs
179     L_output = self.lang_decoder(inp['L_fw'][0], len(inp['L_fw']),
180                                 L_dec_init_state, True)
181     B_output = self.action_decoder(VB_input_f, len(inp['B_fw']),
182                                   VB_dec_init_state)
183     return L_output, B_output, L_z_mean, VB_z_mean, L_z_log_var, VB_z_log_var
184
185 def reparameterise(self, mean, log_var):
186     std = torch.exp(log_var*0.5)
187     epsilon = torch.normal(mean=0.0, std=0.1,
188                             size=(mean.size()[0], mean.size()[1])).to('cuda')
189     return mean + std * epsilon
190
191 # PVAE-BERT
192 class PVAEBERT(nn.Module):
193     def __init__(self, params):
194         super(PVAEBERT, self).__init__()
195         self.params = params # obtain the parameters
196         # Encoders
197         self.lang_encoder = BertEncoder(self.params)
198         self.action_encoder = Encoder(self.params, False)
199         ...
200         ...
201         ...

```

Listing B.1: PVAE & PVAE-BERT model definition classes and methods taken from <https://github.com/oo222bs/PVAE-BERT/blob/master/src/pvae.py>

## CS-CAE

```

1 # Convolutional Encoder
2 class ConvolutionalEncoder(nn.Module):
3     def __init__(self):
4         super(ConvolutionalEncoder, self).__init__()
5         self.first_conv = nn.Conv2d(in_channels=1, out_channels=8, kernel_size=(4,
6                                         4), stride=(2, 2), padding=1)
7         self.second_conv = nn.Conv2d(in_channels=8, out_channels=16, kernel_size
8                                         =(4, 4), stride=(2, 2), padding=1)
9         self.third_conv = nn.Conv2d(in_channels=16, out_channels=32, kernel_size
10                                        =(4, 4), stride=(2, 2), padding=1)
11         self.fourth_conv = nn.Conv2d(in_channels=32, out_channels=64, kernel_size
12                                        =(8, 8), stride=(5, 5), padding=2)
13         self.relu = nn.ReLU()
14
15     def forward(self, input_images):
16         first_conv = self.relu(self.first_conv(input_images.float()))

```

```

13     second_conv = self.relu(self.second_conv(first_conv))
14     third_conv = self.relu(self.third_conv(second_conv))
15     fourth_conv = self.relu(self.fourth_conv(third_conv))
16     return fourth_conv
17
18 # Fully connected layers (Bottleneck)
19 class Bottleneck(nn.Module):
20     def __init__(self):
21         super(Bottleneck, self).__init__()
22         self.first_fc = nn.Linear(in_features=3*4*64, out_features=384)
23         self.second_fc = nn.Linear(in_features=384, out_features=192)
24         self.third_fc = nn.Linear(in_features=192, out_features=10)
25         self.fourth_fc = nn.Linear(in_features=10, out_features=192)
26         self.fifth_fc = nn.Linear(in_features=192, out_features=384)
27
28     def forward(self, encoded_features):
29         flattened = torch.flatten(encoded_features, start_dim=1)
30         first_dense = self.first_fc(flattened)
31         second_dense = self.second_fc(first_dense)
32         third_dense = self.third_fc(second_dense)
33         fourth_dense = self.fourth_fc(third_dense)
34         fifth_dense = self.fifth_fc(fourth_dense)
35         return third_dense, fifth_dense
36
37 # Deconvolutional Decoder
38 class DeconvolutionalDecoder(nn.Module):
39     def __init__(self):
40         super(DeconvolutionalDecoder, self).__init__()
41         self.first_deconv = nn.ConvTranspose2d(in_channels=32, out_channels=32,
42         kernel_size=(8, 8), stride=(5, 5), padding=2, output_padding=1)
43         self.second_deconv = nn.ConvTranspose2d(in_channels=32, out_channels=16,
44         kernel_size=(4, 4), stride=(2, 2), padding=1)
45         self.third_deconv = nn.ConvTranspose2d(in_channels=16, out_channels=8,
46         kernel_size=(4, 4), stride=(2, 2), padding=1)
47         self.fourth_deconv = nn.ConvTranspose2d(in_channels=8, out_channels=1,
48         kernel_size=(4, 4), stride=(2, 2), padding=1)
49         self.relu = nn.ReLU()
50         self.sigmoid = nn.Sigmoid()
51
52     def forward(self, dense_features):
53         reshaped = torch.reshape(dense_features, (dense_features.size()[0], -1, 3, 4))
54         first_deconv = self.relu(self.first_deconv(reshaped))
55         second_deconv = self.relu(self.second_deconv(first_deconv))
56         third_deconv = self.relu(self.third_deconv(second_deconv))
57         output = self.sigmoid(self.fourth_deconv(third_deconv))
58         return output
59
60 # Convolutional Autoencoder (CAE)
61 class CAE(nn.Module):
62     def __init__(self):
63         super(CAE, self).__init__()
64         self.convolutional_encoder = ConvolutionalEncoder()
65         self.bottleneck = Bottleneck()
66         self.deconvolutional_decoder = DeconvolutionalDecoder()
67
68     def forward(self, input_images):
69         encoded_features = self.convolutional_encoder(input_images)
70         _, dense_out = self.bottleneck(encoded_features)
71         output_images = self.deconvolutional_decoder(dense_out)
72         return output_images
73
74     def extract_visual_features(self, input_image):
75         encoded_features = self.convolutional_encoder(input_image)
76         visual_features, _ = self.bottleneck(encoded_features)
77         return visual_features
78
79 # Train for every channel
80 def train():

```

```

77 train_for_channel('red')
78 train_for_channel('green')
79 train_for_channel('blue')

```

Listing B.2: CS-CAE model definition classes and methods taken from [https://github.com/oo222bs/PVAE-BERT/blob/master/src/channel\\_separated\\_cae.py](https://github.com/oo222bs/PVAE-BERT/blob/master/src/channel_separated_cae.py)

## PGAE

### Model Definition

```

1 def train_gmu_opp(model, batch, optimiser, epoch_loss, params):
2     optimiser.zero_grad() # free the optimizer from previous gradients
3     gt_description = batch['L_fw'][1:]
4     gt_action = batch['B_fw'][1:]
5     ran_sig = torch.randint(3, (1,))
6     opp = 0
7     if ran_sig == 0:
8         rep_sig = torch.randint(3, (1,))
9         if rep_sig == 0:
10            signal = 'repeat action'
11            opp = torch.randint(2, (1,))
12            gt_description = gt_description[-1].unsqueeze(0)
13        elif rep_sig == 1:
14            signal = 'repeat both'
15            opp = torch.randint(2, (1,))
16        else:
17            signal = 'repeat language'
18            gt_action = batch['B_fw'][0].repeat(len(gt_action), 1, 1)
19                * batch["B_bin"][1:]
20    elif ran_sig ==1:
21        signal = 'describe'
22        opp = torch.randint(2, (1,))
23        if opp == 0:
24            gt_action = batch['B_bw'][0].repeat(len(gt_action), 1, 1)
25                * batch["B_bin"][1:]
26        else:
27            gt_action = batch['B_fw'][0].repeat(len(gt_action), 1, 1)
28                * batch["B_bin"][1:]
29    else:
30        signal = 'execute'
31        gt_description = gt_description[-1].unsqueeze(0)
32    output = model(batch, signal, opp)
33    L_loss, B_loss, batch_loss = loss_gmu(output, gt_description, gt_action,
34        batch["B_bin"], signal, params) # compute loss
35    batch_loss.backward() # compute gradients
36    optimiser.step() # update weights
37    epoch_loss.append(batch_loss.item()) # record the batch loss
38    return L_loss, B_loss, batch_loss, signal # return the losses
39
40 # Gated Multimodal Unit (Arevalo et al., 2017)
41 class GatedMultimodalUnit(nn.Module):
42     def __init__(self, params, bert=False):
43         super(GatedMultimodalUnit, self).__init__()
44         self.params = params
45         if bert:
46             self.lang_h_linear = nn.Linear(768, self.params.hidden_dim)
47             self.z_linear = nn.Linear(self.params.VB_num_units
48                 * self.params.VB_num_layers * 2 + 768, self.params.hidden_dim)
49         else:
50             self.lang_h_linear = nn.Linear(self.params.L_num_units

```

```

51         * self.params.L_num_layers * 2, self.params.hidden_dim)
52         self.z_linear = nn.Linear(self.params.VB_num_units
53         * self.params.VB_num_layers * 2 + self.params.L_num_units
54         * self.params.L_num_layers * 2, self.params.hidden_dim)
55         self.act_h_linear = nn.Linear(self.params.VB_num_units
56         * self.params.VB_num_layers * 2, self.params.hidden_dim)
57         self.tanh = nn.Tanh()
58         self.sigmoid = nn.Sigmoid()
59
60     def forward(self, act_features, lang_features):
61         h_act = self.tanh(self.act_h_linear(act_features))
62         h_lang = self.tanh(self.lang_h_linear(lang_features))
63         z = self.sigmoid(self.z_linear(torch.cat([act_features, lang_features],
64         dim=-1)))
65         h = z * h_act + (1 - z) * h_lang
66         return h
67
68 class PGAE(nn.Module):
69     def __init__(self, params):
70         super(PGAE, self).__init__()
71         self.params = params
72         self.lang_encoder = Encoder(self.params, True)
73         self.action_encoder = Encoder(self.params, False)
74         self.hidden = GatedMultimodalUnit(self.params)
75         self.initial_lang = nn.Linear(self.params.hidden_dim, self.params.
76         L_num_units*self.params.L_num_layers*2)
77         self.initial_act = nn.Linear(self.params.hidden_dim, self.params.
78         VB_num_units*self.params.VB_num_layers*2)
79         self.lang_decoder = Decoder(self.params, True)
80         self.action_decoder = Decoder(self.params, False, False)
81
82     def forward(self, inp, signal):
83         if signal == 'repeat both':
84             ...
85         elif signal == 'repeat language':
86             ...
87         elif signal == 'repeat action':
88             ...
89         elif signal == 'describe':
90             signalrow = torch.zeros((1, inp['L_fw'].size()[1],
91             inp['L_fw'].size()[2]+5), requires_grad=True).to('cuda')
92             VB_input = torch.cat([inp['V_fw'], inp['B_fw']], dim=2)
93             signalrow[0, :, self.params.L_input_dim] = 1.0
94             l_fw_ndim = torch.cat((inp['L_fw'][-1].unsqueeze(0), torch.zeros(1,
95             inp['L_fw'].size()[1], 5).to('cuda')), axis=-1).to('cuda')
96             lang_in_length = inp['L_len'].int().numpy()
97             - (inp['L_len'].int().numpy() - 1)
98         else:
99             l_fw_ndim = torch.cat((inp['L_fw'], torch.zeros(inp['L_fw'].size()[0],
100             inp['L_fw'].size()[1], 5).to('cuda')), axis=-1).to('cuda')
101             signalrow = torch.zeros((1, l_fw_ndim.size()[1], l_fw_ndim.size()[2]),
102             requires_grad=True).to('cuda')
103             signalrow[0, :, self.params.L_input_dim+1] = 1.0
104             VB_input=torch.cat((inp['V_fw'][0].repeat(len(inp['V_fw'])), 1, 1),
105             inp['B_fw'][0].repeat(len(inp['B_fw'])), 1, 1)), dim=2)
106             * inp['B_bin'][:, :, 0].unsqueeze(-1).repeat(1, 1,
107             inp['V_fw'].size()[-1] + inp['B_fw'].size()[-1])
108             lang_in_length = inp['L_len'].int().numpy()
109             lang_inp = torch.cat((signalrow, l_fw_ndim), axis=0).to('cuda')
110             encoded_lang = self.lang_encoder(lang_inp, lang_in_length + 1, True)
111             encoded_act = self.action_encoder(VB_input, inp['V_len'].int().numpy())
112             z = self.hidden(encoded_act, encoded_lang)
113             L_dec_init_state = self.initial_lang(z)
114             VB_dec_init_state = self.initial_lang(z)
115             if signal == 'repeat both':
116                 ...
117             elif signal == 'describe':
118                 L_output = self.lang_decoder(inp['L_fw'][0], len(inp['L_fw']),
119                 L_dec_init_state, True)

```

```

112         VB_input_f = [inp["V_bw"][0].repeat(len(inp['V_fw']),1,1),
113                      inp["B_bw"][0, :, :]]
114         B_output = self.action_decoder(VB_input_f, len(inp['B_fw']),
115                                       VB_dec_init_state, teacher_forcing=True)
116     elif signal == 'repeat language':
117         ...
118     else:
119         VB_input_f = inp['VB_fw']
120         L_output = self.lang_decoder(inp['L_fw'][0],2,L_dec_init_state,True)
121         B_output = self.action_decoder(VB_input_f, len(inp['B_fw']),
122                                       VB_dec_init_state, teacher_forcing=True)
123     return L_output, B_output

```

Listing B.3: Examples of PGAE model definition classes and methods taken from <https://github.com/oo222bs/PGAE/blob/main/src/pgae.py>

## PTAE

### Model Definition

```

1 class PTAE(nn.Module):
2     def __init__(self, params, lang_enc_type='LSTM', act_enc_type='LSTM',
3               app_length=True):
4         super(PTAE, self).__init__()
5         from crossmodal_transformer import Visual_Ling_Attn as CMTransformer
6         self.params = params
7         self.lang_enc_type = lang_enc_type
8         self.act_enc_type = act_enc_type
9         if self.lang_enc_type == 'LSTM':
10            self.lang_encoder = Encoder(self.params, True,
11                                      lstm_type='bidirectional')
12        elif self.lang_enc_type == 'BERT':
13            self.lang_encoder = LanguageModel(self.params, 'bert-base')
14        elif self.lang_enc_type == 'WordEmbedding':
15            self.word_embedder = Embedder(self.params.L_input_dim + 5,
16                                         emb_dim=params.L_num_units)
17        if self.act_enc_type == 'LSTM':
18            self.action_encoder = Encoder(self.params, False,
19                                        lstm_type='bidirectional')
20        self.hidden = CMTransformer(self.params)
21        self.initial_lang = nn.Linear(self.params.hidden_dim,
22                                     self.params.L_num_units*self.params.L_num_layers*2)
23        self.initial_act = nn.Linear(self.params.hidden_dim,
24                                     self.params.VB_num_units*self.params.VB_num_layers*2)
25        self.lang_decoder = Decoder(self.params, True, lstm_type='regular',
26                                   appriori_length=app_length)
27        self.action_decoder = Decoder(self.params, False, lstm_type='regular',
28                                    appriori_length=app_length)
29    def forward(self, inp, signal, appriori_len=True):
30        if signal == 'repeat language':
31            ...
32        elif signal == 'repeat action':
33            ...
34        elif signal == 'describe':
35            signalrow = torch.zeros((1, inp['L_fw'].size()[1],
36                                   inp['L_fw'].size()[2]+5), requires_grad=True).to('cuda')
37            VB_input = torch.cat([inp['V_fw'], inp['B_fw']], dim=2)
38            signalrow[0, :, self.params.L_input_dim] = 1.0
39            if appriori_len:
40                l_fw_ndim = torch.cat((inp['L_fw'][-1].unsqueeze(0), torch.zeros
41                                     (1, inp['L_fw'].size()[1], 5).to('cuda')), axis=-1).to('cuda')
42            else:

```

```

41         l_fw_eos = torch.zeros((1, inp['L_fw'].shape[1],
42                               inp['L_fw'].shape[2]+5))
43         l_fw_eos[:, :, 1] = 1
44         l_fw_ndim = l_fw_eos.to('cuda')
45         lang_in_length = inp['L_len'].int().numpy()
46                         - (inp['L_len'].int().numpy() - 1)
47     else:
48         l_fw_ndim = torch.cat((inp['L_fw'], torch.zeros(inp['L_fw'].size()[0],
49               inp['L_fw'].size()[1], 5).to('cuda')), axis=-1).to('cuda')
49         signalrow = torch.zeros((1,l_fw_ndim.size()[1],l_fw_ndim.size()[2]),
50               requires_grad=True).to('cuda')
51         signalrow[0, :, self.params.L_input_dim+1] = 1.0
52         VB_input=torch.cat((inp['V_fw'][0].repeat(len(inp['V_fw'])), 1, 1),
53               inp['B_fw'][0].repeat(len(inp['B_fw']), 1, 1)), dim=2)
54         * inp['B_bin'][:, :, 0].unsqueeze(-1).repeat(1, 1,
55               inp['V_fw'].size()[-1] + inp['B_fw'].size()[-1])
56         lang_in_length = inp['L_len'].int().numpy()
57         lang_inp = torch.cat((signalrow, l_fw_ndim), axis=0).to('cuda')
58         if self.lang_enc_type == 'LSTM':
59             encoded_lang = self.lang_encoder(lang_inp, lang_in_length + 1, True)
60         elif self.lang_enc_type == 'BERT':
61             encoded_lang = self.lang_encoder(lang_inp)
62         elif self.lang_enc_type == 'WordEmbedding':
63             encoded_lang = self.word_embedder(lang_inp.argmax(axis=-1)).permute
64                 (1,0,2)
65         else:
66             encoded_lang = lang_inp.permute(1,0,2).float()
67         if self.act_enc_type == 'LSTM':
68             encoded_act = self.action_encoder(VB_input,inp['V_len'].int().numpy())
69         else:
70             encoded_act = VB_input.permute(1,0,2).float()
71         h = self.hidden(encoded_lang, encoded_act, None, None).mean(1)
72         L_dec_init_state = self.initial_lang(h)
73         VB_dec_init_state = self.initial_act(h)
74         if signal == 'describe':
75             VB_input_f = [inp["V_bw"][0].repeat(len(inp['V_fw']), 1, 1),
76                   inp["B_bw"][0, :, :]]
77             if appriori_len:
78                 L_output = self.lang_decoder(inp['L_fw'][0], len(inp['L_fw']),
79                       L_dec_init_state, True)
80                 B_output = self.action_decoder(VB_input_f, len(inp['B_fw']),
81                       VB_dec_init_state)
82             else:
83                 L_output = self.lang_decoder(inp['L_fw'][0], None,
84                       L_dec_init_state, True)
85                 B_output = self.action_decoder(VB_input_f, None, VB_dec_init_state)
86         elif signal == 'repeat language':
87             ...
88         else:
89             VB_input_f = inp['VB_fw']
90             if appriori_len:
91                 L_output = self.lang_decoder(inp['L_fw'][0], 2, L_dec_init_state,
92                       True)
93                 B_output = self.action_decoder(VB_input_f, len(inp['B_fw']),
94                       VB_dec_init_state)
95             else:
96                 L_output = self.lang_decoder(inp['L_fw'][0], None,
97                       L_dec_init_state, True)
98                 B_output = self.action_decoder(VB_input_f, None, VB_dec_init_state)
99         return L_output, B_output

```

Listing B.4: Examples of PTAE model definition classes and methods taken from <https://github.com/oo222bs/PTAE/blob/main/src/ptae.py>

## Training

```

1 def main():
2     # get the network configuration (parameters like number of layers and units)
3     parameters = PTAEConfig()
4     parameters.set_conf("../train/ptae_conf.txt")
5     # get the training configuration
6     # (batch size, initialisation, num_of_epochs number, saving and loading
7     # directory)
8     train_conf = TrainConfig()
9     train_conf.set_conf("../train/train_conf.txt")
10    seed = train_conf.seed
11    batch_size = train_conf.batch_size
12    num_of_epochs = train_conf.num_of_epochs
13    learning_rate = train_conf.learning_rate
14    save_dir = train_conf.save_dir
15    # Create a model instance
16    model = PTAE(parameters, lang_enc_type='None', act_enc_type='None').to(device)
17    # Initialise the optimiser
18    optimiser = torch.optim.Adam(model.parameters(), lr=learning_rate)
19    scheduler = MultiStepLR(optimiser, milestones=[10000], gamma=0.5)
20    # Inspect the model with tensorboard
21    model_name = "ptae_limited_supervised_data_98pc"
22    model.train() # tell the model that it's training time
23    # Load the dataset
24    training_data = PairedNico2BlocksDataset(train_conf)
25    test_data = PairedNico2BlocksDataset(train_conf, True)
26    # Load the training and testing sets with DataLoader
27    train_dataloader=DataLoader(training_data, batch_size=batch_size, shuffle=True)
28    test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
29    step = 0
30    super_perc = 2
31    super_batches = np.random.choice(range(len(train_dataloader)),
32    math.ceil(len(train_dataloader)*super_perc/100), replace=False)
33    # Training
34    for epoch in range(num_of_epochs):
35        epoch_loss, epoch_loss_describe, epoch_loss_execute =
36            ([[] for i in range(3)])
37        iter_within_epoch = 0
38        for input in train_dataloader:
39            # Generate random index for description alternatives
40            sentence_idx = np.random.randint(8)
41            L_fw_feed = input["L_fw"][5*sentence_idx:5+5*sentence_idx, :, :]
42            input["L_fw"] = L_fw_feed.to(device)
43            if iter_within_epoch in super_batches:
44                supervised_sig = torch.randint(2, (1,))
45                if supervised_sig == 0:
46                    signal = 'describe'
47                else:
48                    signal = 'execute'
49            else:
50                rep_sig = torch.randint(2, (1,))
51                if rep_sig == 0:
52                    signal = 'repeat action'
53                else:
54                    signal = 'repeat language'
55            #Train and print the losses
56            l, b, t, signal = train_limited_data(model, input, optimiser,
57            epoch_loss, parameters, signal, vis_out=False)
58            print("step:{} total:{}, language:{}, behavior:{}, signal:{}".format(
59            step, t, l, b, signal))
60            step = step + 1
61            if signal == "describe":
62                epoch_loss_describe.append(epoch_loss[-1])
63            elif signal == "execute":
64                epoch_loss_execute.append(epoch_loss[-1])
65            iter_within_epoch = iter_within_epoch + 1

```



```

63     scheduler.step()
64     # Flush and close the summary writer of Tensorboard
65     writer.flush()
66     writer.close()

```

Listing B.5: Snippet from PTAE model training script taken from [https://github.com/oo222bs/PTAE/blob/main/src/main\\_ptae.py](https://github.com/oo222bs/PTAE/blob/main/src/main_ptae.py)

## XBiT

### Model Definition

```

1 class XBiT(nn.Module):
2     def __init__(self, params, lang_enc_type, lang_onehot_inp=False, coll_out=False):
3         super(XBiT, self).__init__()
4         from crossmodal_transformer import Visual_Ling_Attn as CMTransformer
5         from crossmodal_transformer import Linguistic_Vis_Attn as
            CMTransformerMirrored
6         from crossmodal_transformer import LanguageDecoderLayer as LangDecoder
7         self.params = params
8         self.lang_enc_type = lang_enc_type
9         self.lang_onehot_inp = lang_onehot_inp
10        self.vis_feat_extractor_red = CAE(self.params.B_max_length)
11        self.vis_feat_extractor_green = CAE(self.params.B_max_length)
12        self.vis_feat_extractor_blue = CAE(self.params.B_max_length)
13        ...
14        ...
15        ...
16        if self.lang_enc_type == 'BERT':
17            self.lang_encoder = LanguageModel('bert-base', self.lang_onehot_inp)
18        else:
19            self.lang_encoder = LanguageModel('clip', self.lang_onehot_inp)
20        self.hidden = CMTransformer(self.params)
21        self.hidden_mirror = CMTransformerMirrored(self.params)
22        self.lang_decoder = LangDecoder(self.params)
23        self.init_act_decoder = nn.Linear(self.params.hidden_dim * 2,
24                                         int(self.params.hidden_dim / 2))
25        self.second_act_decoder = nn.Linear(int(self.params.hidden_dim / 2),
26                                           int(self.params.hidden_dim / 4))
27        self.xyz_coord = nn.Linear(int(self.params.hidden_dim / 4), 3)
28        self.quats = nn.Linear(int(self.params.hidden_dim / 4), 4)
29        self.grip_open = nn.Linear(int(self.params.hidden_dim / 4), 1)
30        if coll_out:
31            self.ignore_coll = nn.Linear(int(self.params.hidden_dim / 4), 1)
32        self.dropout = nn.Dropout(p=self.params.T_dropout)
33        self.relu = nn.ReLU()
34        self.sigmoid = nn.Sigmoid()
35        self.layer_norm = nn.LayerNorm(self.params.hidden_dim)
36        def forward(self, inp, signal):
37            if signal == 'describe':
38                ... # visual_features
39                V_input = torch.cat([vs_ftrs_red.transpose(0, 1),
40                                   vs_ftrs_green.transpose(0, 1), vs_ftrs_blue.transpose(0, 1),
41                                   vs_ftrs_wrist_red.transpose(0, 1), vs_ftrs_wrist_green.transpose
42                                   (0, 1), vs_ftrs_wrist_blue.transpose(0, 1)], dim=2)
43                VB_input = torch.cat([V_input, inp['B_fw']], dim=2) # action input
44                if self.lang_onehot_inp:
45                    signalrow = torch.zeros((1, inp['L_oh_fw'].size()[1],
46                                             inp['L_oh_fw'].size()[2] + 5), requires_grad=True).to('cuda')
47                    signalrow[0, :, self.params.L_input_dim] = 1.0
48                    l_fw_eos = torch.zeros((1, inp['L_oh_fw'].shape[1],
49                                           inp['L_oh_fw'].shape[2] + 5))

```

```

49         l_fw_eos[:, :, 5] = 1
50         l_fw_ndim = l_fw_eos.to('cuda')
51         lang_inp = torch.cat((signalrow, l_fw_ndim), axis=0).to('cuda')
52     else:
53         lang_inp = [signal + ' : ' for i in range(len(inp['L_fw']))]
54     else:
55         ... # visual_features
56         V_input = torch.cat([vs_ftrs_red[0], vs_ftrs_green[0],
57                             vs_ftrs_blue[0], vs_ftrs_wrist_red[0], vs_ftrs_wrist_green[0],
58                             vs_ftrs_wrist_blue[0]], dim=-1)
59         VB_input = torch.cat((V_input, inp['B_fw'][0]), dim=-1).unsqueeze(0)
60         if self.lang_onehot_inp:
61             l_fw_ndim = torch.cat((inp['L_fw'], torch.zeros(inp['L_fw'].size()
62                 [0], inp['L_fw'].size()[1], 5).to('cuda')), axis=-1)
63             signalrow = torch.zeros((1, l_fw_ndim.size()[1], l_fw_ndim.size()
64                 [2]), requires_grad=True).to('cuda')
65             signalrow[0, :, self.params.L_input_dim + 1] = 1.0
66             lang_inp = torch.cat((signalrow, l_fw_ndim), axis=0).to('cuda')
67         else:
68             lang_inp = [signal + ' : ' + desc for desc in inp['L_fw']]
69         encoded_lang = self.lang_encoder(lang_inp)
70         encoded_act = VB_input.permute(1, 0, 2).float()
71         h = self.hidden(encoded_lang, encoded_act, None, None)
72         h_mirror = self.hidden_mirror(encoded_act, encoded_lang, None, None).mean
73             (1).unsqueeze(1)
74         h = torch.mean(torch.stack((h, h_mirror)), dim=0)
75         mask_enc_att = None
76         if signal == 'describe':
77             B_dec_inp = inp["B_bw"][0]
78             ... # visual_features from the last time step
79             h_linear = self.relu(self.init_act_decoder(torch.cat((VB_input_f,
80                 h.repeat(1, VB_input_f.shape[0], 1).permute(1, 0, 2)), axis=-1)))
81             act_dec_inp = self.relu(self.second_act_decoder(h_linear))
82             L_tar = torch.cat((inp['L_oh_fw'][:-1],
83                 torch.zeros(inp['L_oh_fw'].size()[0] - 1,
84                     inp['L_oh_fw'].size()[1], 2).to('cuda')), axis=-1).to('cuda')
85             mask_self_att_lang = torch.triu(torch.ones(h.shape[0], self.params.
86                 T_num_heads, L_tar.shape[0], L_tar.shape[0]).bool(), diagonal=1)
87             L_output = self.lang_decoder(L_tar.permute(1, 0, 2), h,
88                 mask_self_att_lang, mask_enc_att)
89             xyz_coord = self.xyz_coord(act_dec_inp)
90             quats = self.quats(act_dec_inp)
91             grip_open = self.sigmoid(self.grip_open(act_dec_inp))
92             B_output = torch.cat((xyz_coord, quats, grip_open), -1)
93             # if we have an ignore collision output
94             if self.coll_out:
95                 collision = self.sigmoid(self.ignore_coll(act_dec_inp))
96                 B_output = torch.cat((B_output, collision), -1)
97         else:
98             ... # visual_features
99             h_linear = self.relu(self.init_act_decoder(torch.cat((VB_input_f,
100                 h.repeat(1, VB_input_f.shape[0], 1).permute(1, 0, 2)), axis=-1)))
101             act_dec_inp = self.relu(self.second_act_decoder(h_linear))
102             L_tar = torch.cat((inp['L_oh_fw'][0].unsqueeze(0), torch.zeros(1,
103                 inp['L_oh_fw'].size()[1], 2).to('cuda')), axis=-1).to('cuda')
104             mask_self_att_lang = torch.triu(torch.ones(h.shape[0], self.params.
105                 T_num_heads, L_tar.shape[0], L_tar.shape[0]).bool(), diagonal=1)
106             L_output = self.lang_decoder(L_tar.permute(1, 0, 2), h,
107                 mask_self_att_lang, mask_enc_att)
108             ...
109             ...
110             ...
111             B_output = torch.cat((xyz_coord, quats, grip_open), -1)
112             # if we have an ignore collision output
113             if self.coll_out:
114                 collision = self.sigmoid(self.ignore_coll(act_dec_inp))
115                 B_output = torch.cat((B_output, collision), -1)
116         return L_output, B_output

```

Listing B.6: Examples of XBiT model definition classes and methods taken from <https://github.com/oo222bs/XBiT/blob/main/src/xbit.py>

## CMT

```

1 class PositionWiseFeedForward(nn.Module):
2     def __init__(self, d_model=512, d_ff=2048, dropout=0.1):
3         super(PositionWiseFeedForward, self).__init__()
4         self.fc1 = nn.Linear(d_model, d_ff)
5         self.fc2 = nn.Linear(d_ff, d_model)
6         self.dropout = nn.Dropout(p=dropout)
7         self.dropout_2 = nn.Dropout(p=dropout)
8         self.layer_norm = nn.LayerNorm(d_model)
9
10    def forward(self, input):
11        pwff = self.fc2(self.dropout_2(F.relu(self.fc1(input))))
12        pwff = self.dropout(pwff)
13        out = self.layer_norm(input + pwff)
14        return out
15
16 class ScaledDotProductAttention(nn.Module):
17    def __init__(self, d_model, d_k, d_v, h):
18        """
19        :param d_model: Output dimensionality of the model
20        :param d_k: Dimensionality of queries and keys
21        :param d_v: Dimensionality of values
22        :param h: Number of heads
23        """
24        super(ScaledDotProductAttention, self).__init__()
25        self.fc_q = nn.Linear(d_model, h * d_k)
26        self.fc_k = nn.Linear(d_model, h * d_k)
27        self.fc_v = nn.Linear(d_model, h * d_v)
28        self.fc_o = nn.Linear(h * d_v, d_model)
29        self.d_model = d_model
30        self.d_k = d_k
31        self.d_v = d_v
32        self.h = h
33        self.init_weights(gain=1.0)
34
35    def init_weights(self, gain=1.0):
36        nn.init.xavier_normal_(self.fc_q.weight, gain=gain)
37        nn.init.xavier_normal_(self.fc_k.weight, gain=gain)
38        nn.init.xavier_normal_(self.fc_v.weight, gain=gain)
39        nn.init.xavier_normal_(self.fc_o.weight, gain=gain)
40        nn.init.constant_(self.fc_q.bias, 0)
41        nn.init.constant_(self.fc_k.bias, 0)
42        nn.init.constant_(self.fc_v.bias, 0)
43        nn.init.constant_(self.fc_o.bias, 0)
44
45    def forward(self, queries, keys, values, attention_mask=None,
46               attention_weights=None):
47        b_s, nq = queries.shape[:2]
48        nk = keys.shape[1]
49        # (b_s, h, nq, d_k)
50        q = (self.fc_q(queries).view(b_s, nq, self.h, self.d_k).permute(0,2,1,3))
51        # (b_s, h, d_k, nk)
52        k = (self.fc_k(keys).view(b_s, nk, self.h, self.d_k).permute(0,2,3,1))
53        # (b_s, h, nk, d_v)
54        v = (self.fc_v(values).view(b_s, nk, self.h, self.d_v).permute(0,2,1,3))
55        att = torch.matmul(q, k) / np.sqrt(self.d_k) # (b_s, h, nq, nk)
56        if attention_weights is not None:
57            att = att * attention_weights

```

```

57     if attention_mask is not None:
58         att = att.masked_fill(attention_mask, -np.inf)
59     att = torch.softmax(att, -1)
60     if attention_mask is not None:
61         att = att.masked_fill(attention_mask, 0)
62     # (b_s, nq, h*d_v)
63     out = (torch.matmul(att, v).permute(0, 2, 1, 3).contiguous().view(b_s, nq,
64         self.h * self.d_v))
65     out = self.fc_o(out) # (b_s, nq, d_model)
66     return out
67 class MultiHeadAttention(nn.Module):
68     def __init__(self, d_model, d_k, d_v, h, dropout=0.1):
69         super(MultiHeadAttention, self).__init__()
70         self.attention = ScaledDotProductAttention(d_model=d_model, d_k=d_k,
71             d_v=d_v, h=h)
72         self.dropout = nn.Dropout(p=dropout)
73         self.layer_norm = nn.LayerNorm(d_model)
74
75     def forward(self, queries, keys, values, attention_mask=None,
76         attention_weights=None):
77         att = self.attention(queries, keys, values, attention_mask,
78             attention_weights)
79         att = self.dropout(att)
80         return self.layer_norm(queries + att)
81
82 class InterModuleAttnLayer(nn.Module):
83     def __init__(self, d_model=512, d_k=64, d_v=64, h=8, d_ff=2048, dropout=0.1,
84         pooler=False):
85         super(InterModuleAttnLayer, self).__init__()
86         self.enc_att = MultiHeadAttention(d_model, d_k, d_v, h, dropout)
87         self.pwff = PositionWiseFeedForward(d_model, d_ff, dropout)
88
89     def with_pos_embed(self, tensor, pos: Optional[Tensor]):
90         return tensor if pos is None else tensor + pos
91
92     def forward(self, input_1, input_2, mask_self_att, mask_enc_att,
93         pos_embed=None):
94         enc_att = self.enc_att(input_1, input_2, input_2, mask_enc_att)
95         ff = self.pwff(enc_att)
96         return ff
97 # Base Class for Crossmodal Transformer - Hidden layer dimension size is 256, 1-
98 # layer 4-head Transformer Encoder, number of dimensions per q,k,v is 64, ff
99 # dimensionality is 1024
100 class Visual_Ling_Attn(nn.Module):
101     def __init__(self, params):
102         super(Visual_Ling_Attn, self).__init__()
103         self.params = params
104         self.d_model = self.params.hidden_dim
105         self.d_att = int(self.d_model / self.params.T_num_heads)
106         self.layers = nn.ModuleList([InterModuleAttnLayer(self.d_model,
107             self.d_att, self.d_att, self.params.T_num_heads, self.params.T_ff_dim,
108             self.params.T_dropout) for _ in range(self.params.T_num_layers)])
109         self.vis_fc = nn.Linear(self.params.VB_input_dim, self.d_model)
110         self.ins_fc = nn.Linear(512, self.d_model)
111         self.dropout = nn.Dropout(p=self.params.T_dropou )
112         self.layer_norm = nn.LayerNorm(self.d_model)
113
114     def forward(self, input, input_2, self_att_mask, enc_att_mask):
115         out = F.relu(self.vis_fc(input_2))
116         out = self.dropout(out)
117         out = self.layer_norm(out)
118         input = F.relu(self.ins_fc(input))
119         input = self.dropout(input)
120         input = self.layer_norm(input)
121         # Apply positional encoding to Q input
122         dev = input.get_device()

```

```

121     pe = sinusoid_encoding_table(input.shape[1], input.shape[2])
122     pe = pe.expand(input.shape[0], pe.shape[0], pe.shape[1]).to(dev)
123     input = input + pe
124     for l in self.layers:
125         out = l(input, out, self_att_mask, enc_att_mask)
126     return out

```

Listing B.7: Definition of CMT taken from [https://github.com/oo222bs/XBiT/blob/main/src/crossmodal\\_transformer.py](https://github.com/oo222bs/XBiT/blob/main/src/crossmodal_transformer.py)

## RL Fine-Tuning

```

1 def create_agent(cfg: DictConfig):
2     # get the network configuration (parameters such as number of layers and units)
3     paramaters = XBiTConfig()
4     lang_onehot = False
5     coll_out = True
6     train = True           # set to True if RL fine-tuning is desired
7     # set to True if baseline layer is desired for RL fine-tuning (default True)
8     baseline = True
9     rl_alg = 'REINFORCE' # choose the RL fine-tuning algorithm
10    device = 'cuda' if torch.cuda.is_available() else 'cpu'
11    print('Using {} device'.format(device))
12    model = XBiT(paramaters, lang_enc_type='clip', lang_onehot_inp=lang_onehot,
13               coll_out=coll_out, baseline=baseline, rl_alg=rl_alg).to(device)
14    # Load the trained model
15    checkpoint = torch.load(save_dir+model_name+'.tar') # get the checkpoint
16    if train:
17        model.load_state_dict(checkpoint['model_state_dict'], strict=False)
18        optimiser = torch.optim.Adam(model.parameters(), lr=1e-6)
19        model.set_optimiser(optimiser)
20        model.set_save_dir(save_dir)
21        model.set_model_name(model_name)
22        model.train()
23    else:
24        model.load_state_dict(checkpoint['model_state_dict'], strict=False)
25        model.eval()
26    return model
27
28 def update_with_reinforce(self, log_probs, reward, baseline_preds=None,
29                          gamma=0.90, negative_upd=True):
30     if baseline_preds != None:
31         baseline_preds = torch.cat(baseline_preds, dim=-1).squeeze(0).squeeze(0)
32     returns = [] # compute return for each step
33     policy_loss = [] # compute policy loss for each step
34     R = 0
35     rewards = [0.0 for i in range(len(log_probs)-1)]
36     if reward == 0.0:
37         rewards.append(0.0)
38     else:
39         rewards.append(1.0)
40     # compute returns
41     for r in rewards[::-1]:
42         R = r + gamma * R
43         returns.insert(0, R)
44     returns = torch.tensor(returns).to('cuda') # convert to tensor
45     if baseline_preds !=None:
46         adjusted_returns = returns-baseline_preds.detach()
47     # compute policy loss at each step
48     if baseline_preds !=None:
49         for log_prob, R in zip(log_probs, adjusted_returns):
50             # don't update when adjusted return is negative
51             if negative_upd == False and R <= 0.0:
52                 policy_loss.append(torch.zeros(1,1).to('cuda'))

```

```
53         else:# the minus sign cause we perform gradient ascent
54             policy_loss.append(-(log_prob.sum(dim=-1) * R))
55     else:
56         for log_prob, R in zip(log_probs, returns):
57             # the minus sign cause we perform gradient ascent
58             policy_loss.append(-(log_prob.sum(dim=-1) * R))
59     if baseline_preds != None:
60         baseline_loss = torch.nn.functional.mse_loss(baseline_preds, returns)
61     self.optimiser.zero_grad()
62     policy_loss = torch.cat(policy_loss).mean()
63     if baseline_preds !=None:
64         overall_loss = policy_loss + baseline_loss
65         overall_loss.backward()
66     else:
67         policy_loss = policy_loss
68         policy_loss.backward()
69     self.optimiser.step()
70     if baseline_preds!=None:
71         return rewards[-1], baseline_loss
72     else:
73         return rewards[-1]
74
75 class BaselineNetwork(nn.Module):
76     def __init__(self, input_size, output_size):
77         super().__init__()
78         self.input_layer = nn.Linear(input_size, int(input_size/2))
79         self.relu = nn.ReLU()
80         self.output_layer = nn.Linear(int(input_size/2), output_size)
81         torch.nn.init.orthogonal_(self.input_layer.weight, np.sqrt(2))
82         torch.nn.init.constant_(self.input_layer.bias, 0.0)
83         torch.nn.init.orthogonal_(self.output_layer.weight)
84         torch.nn.init.constant_(self.output_layer.bias, 0.0)
85
86     def forward(self, h_t):
87         m_t = self.relu(self.input_layer(h_t.detach()))
88         b_t = self.output_layer(m_t)
89         return b_t
```

Listing B.8: Snippet from XBiT RL fine-tuning phase taken from [https://github.com/oo222bs/XBiT/blob/main/agents/xbit/launch\\_utils.py](https://github.com/oo222bs/XBiT/blob/main/agents/xbit/launch_utils.py) and [https://github.com/oo222bs/XBiT/blob/main/agents/xbit/xbit\\_agent.py](https://github.com/oo222bs/XBiT/blob/main/agents/xbit/xbit_agent.py)

---

# Publications Originating from this Thesis

---

## Journal Articles

- Frank Röder, **Ozan Özdemir**, Phuong D. H. Nguyen, Stefan Wermter, and Manfred Epe. The Embodied Crossmodal Self Forms Language and Interaction: A Computational Cognitive Review. *Frontiers in Psychology*, 12:716671. doi: 10.3389/fpsyg.2021.716671, 2021.
- Jae Hee Lee, Yuan Yao, **Ozan Özdemir**, Mengdi Li, Cornelius Weber, Zhiyuan Liu, and Stefan Wermter. Spatial Relation Learning in Complementary Scenarios with Deep Neural Networks. *Frontiers in Neurorobotics*, 16:844753. doi: 10.3389/fnbot.2022.844753, 2022.
- **Ozan Özdemir**, Matthias Kerzel, Cornelius Weber, Jae Hee Lee, and Stefan Wermter. Language-Model-Based Paired Variational Autoencoders for Robotic Language Learning. In *IEEE Transactions on Cognitive and Developmental Systems*, vol. 15, no. 4, pp. 1812-1824, Dec. 2023. doi: 10.1109/TCDS.2022.3204452, 2023.
- **Ozan Özdemir**, Matthias Kerzel, Cornelius Weber, Jae Hee Lee, M. Burhan Hafez, Patrick Bruns, and Stefan Wermter. Learning Bidirectional Action-Language Translation with Limited Supervision and Testing with Incongruent Input. *Applied Artificial Intelligence*, 37(1). <https://doi.org/10.1080/08839514.2023.2179167>, 2023.
- Anton Caesar, **Ozan Özdemir**, Cornelius Weber, and Stefan Wermter. Enabling Action Crossmodality for a Pretrained Large Language Model. *Natural Language Processing Journal*, 7:100072. <https://doi.org/10.1016/j.nlp.2024.100072>, 2024.



## Conference Papers

- **Ozan Özdemir**, Matthias Kerzel, and Stefan Wermter. Embodied Language Learning with Paired Variational Autoencoders. *2021 IEEE International Conference on Development and Learning (ICDL), Beijing, China, 2021*, pp. 1-6, doi: 10.1109/ICDL49984.2021.9515668, 2021.
- **Ozan Özdemir**, Matthias Kerzel, Cornelius Weber, Jae Hee Lee, and Stefan Wermter. Learning Flexible Translation Between Robot Actions and Language Descriptions. In: *Elias Pimenidis, Plamen Angelov, Chrisina Jayne, Antonios Papaleonidas, Mehmet Aydin (Eds.) Artificial Neural Networks and Machine Learning - ICANN 2022. Lecture Notes in Computer Science, vol 13530, pp. 246-257. Springer, Cham. [https://doi.org/10.1007/978-3-031-15931-2\\_21](https://doi.org/10.1007/978-3-031-15931-2_21), 2022.*
- Connor Gäde, **Ozan Özdemir**, Cornelius Weber, and Stefan Wermter. Embodying Language Models in Robot Action. In *Proceedings of the 32nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2024)*, pages 625–630. Ciaco - *ibdoc.com*, Oct 2024.

---

# Acknowledgements

---

As this enriching four-year journey comes to an end, I would like to express my heartfelt gratitude to all those who made this research possible. First and foremost, I want to thank my supervisor, Professor Wermter, for his excellent guidance and unwavering support throughout my doctoral study. My special thanks go to Dr Weber for his dedicated supervision and the insightful discussions on my research and the C4 project, alongside the shared experience of supervising numerous bachelor's and master's students.

I am profoundly thankful to Dr Lee for his collaboration on many papers and the valuable advice he ceaselessly offered. Dr Kerzel deserves particular acknowledgement for supporting me with my initial research ideas, assisting dataset generation and collaborating on many of my publications. I also wish to thank Katja Kösters for providing administrative support and efficiently managing all sorts of paperwork during my doctoral study. My gratitude extends to our former engineer, Erik Strahl, for his technical assistance with the GPU servers and work PCs.

I also want to thank Anton Caesar and Connor Gäde for their primary contributions to the approaches presented in [Chapter 8](#) and [Chapter 9](#) respectively. Additionally, I thank Connor Gäde for his efforts in applying my research ideas to our humanoid robot NICO in real-world settings and Philipp Allgeuer for his contribution to the software used for [Chapter 9](#). I owe a debt of gratitude to Kyra Ahrens for her help with the German translation of the Abstract and for many pleasant conversations, work related and beyond, over the years.

I gratefully acknowledge the financial support from the German Research Foundation (DFG) under project CML (TRR 169).

Finally, I am deeply grateful to my girlfriend, Maren, and my parents, Esin and Hasan, for their emotional support throughout my doctoral journey. Your unconditional love and encouragement mean the world to me.



# Bibliography

- [1] Josh Abramson, Arun Ahuja, Arthur Brussee, Federico Carnevale, Mary Cassin, Stephen Clark, Andrew Dudzik, Petko Georgiev, Aurelia Guy, Tim Harley, Felix Hill, Alden Hung, Zachary Kenton, Jessica Landon, Timothy P. Lillicrap, Kory W. Mathewson, Alistair Muldal, Adam Santoro, Nikolay Savinov, Vikrant Varma, Greg Wayne, Nathaniel Wong, Chen Yan, and Rui Zhu. Imitating interactive intelligence. *arXiv preprint arXiv:2012.05672*, 2020.
- [2] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.
- [3] Stephen Adams, Tyler Cody, and Peter A Beling. A survey of inverse reinforcement learning. *Artificial Intelligence Review*, 55(6):4307–4346, 2022.
- [4] Ademi Adeniji, Amber Xie, Carmelo Sferrazza, Younggyo Seo, Stephen James, and Pieter Abbeel. Language reward modulation for pretraining reinforcement learning. *arXiv preprint arXiv:2308.12270*, 2023.
- [5] Christopher Agia, Toki Migimatsu, Jiajun Wu, and Jeannette Bohg. Stap: Sequencing task-agnostic policies. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7951–7958, 2023.
- [6] Michael Ahn, Anthony Brohan, Noah Brown, et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, pages 287–318. PMLR, 2022.
- [7] Ahmed Akakzia, Cédric Colas, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud. Grounding Language to Autonomously-Acquired Skills via Goal Generation. In *International Conference on Learning Representations (ICLR)*, Virtual (formerly Vienna, Austria), 2021.
- [8] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.
- [9] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

- [10] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual Question Answering. In *2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile*, pages 2425–2433. IEEE Computer Society, 2015.
- [11] Alexandre Antunes, Alban Lafflaquiere, Tetsuya Ogata, and Angelo Cangelosi. A bi-directional multiple timescales LSTM model for grounding of actions and verbs. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2614–2621, 2019.
- [12] Pia Aravena, Esteban Hurtado, Rodrigo Riveros, Juan Felipe Cardona, Facundo Manes, and Agustín Ibáñez. Applauding with closed hands: neural signature of action-sentence compatibility effects. *PloS ONE*, 5(7):e11751, 2010.
- [13] John Arevalo, Thamar Solorio, Manuel Montes-y Gómez, and Fabio A González. Gated multimodal networks. *Neural Computing and Applications*, 32(14):10209–10228, 2020.
- [14] Muhammad Awais, Muzammal Naseer, Salman Khan, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, and Fahad Shahbaz Khan. Foundational Models Defining a New Era in Vision: A Survey and Outlook. *arXiv preprint arXiv:2307.13721*, 2023.
- [15] Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (VPT): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022.
- [16] Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. Findings of the 2019 conference on machine translation (wmt19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, Florence, Italy, August 2019. Association for Computational Linguistics.
- [17] Erik Billing, Julia Rosén, and Maurice Lamb. Language Models for Human-Robot Interaction. In *Companion of the 2023 ACM/IEEE International Conference on Human-Robot Interaction, HRI '23*, page 905–906, New York, NY, USA, 2023. Association for Computing Machinery.
- [18] Zhenshan Bing, Alexander Koch, Xiangtong Yao, Kai Huang, and Alois Knoll. Meta-reinforcement learning via language instructions. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5985–5991. IEEE, 2023.

- 
- [19] Yonatan Bisk, Ari Holtzman, Jesse Thomason, Jacob Andreas, Yoshua Bengio, Joyce Chai, Mirella Lapata, Angeliki Lazaridou, Jonathan May, Aleksandr Nisnevich, Nicolas Pinto, and Joseph Turian. Experience grounds language. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 8718–8735. Association for Computational Linguistics, November 2020.
- [20] Jessica Borja-Diaz, Oier Mees, Gabriel Kalweit, Lukas Hermann, Joschka Boedecker, and Wolfram Burgard. Affordance learning from play for sample-efficient policy learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6372–6378. IEEE, 2022.
- [21] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.
- [22] Anthony Brohan, Noah Brown, Justice Carbajal, et al. RT-1: Robotics Transformer for Real-World Control at Scale. In *Proceedings of Robotics: Science and Systems (RSS)*, Daegu, Republic of Korea, July 2023.
- [23] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.
- [24] Anton Caesar, Ozan Özdemir, Cornelius Weber, and Stefan Wermter. Enabling action crossmodality for a pretrained large language model. *Natural Language Processing Journal*, 7:100072, 2024.
- [25] Laia Canals and Yishay Mor. Towards a signature pedagogy for technology-enhanced task-based language teaching: Defining its design principles. *RECALL*, 35(1):4–18, 2023.
- [26] Joyce Y. Chai, Qiaozi Gao, Lanbo She, Shaohua Yang, Sari Saba-Sadiya, and Guangyue Xu. Language to action: Towards interactive task learning with physical agents. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2–9. International Joint Conferences on Artificial Intelligence Organization, 7 2018.
- [27] Guo Chen, Yin-Dong Zheng, Jiahao Wang, Jilan Xu, Yifei Huang, Junting Pan, Yi Wang, Yali Wang, Yu Qiao, Tong Lu, and Limin Wang. VideoLLM: Modeling Video Sequence with Large Language Models. *arXiv preprint arXiv:2305.13292*, 2023.
- [28] Xi Chen, Josip Djolonga, Piotr Padlewski, Basil Mustafa, Soravit Changpinyo, Jialin Wu, Carlos Riquelme Ruiz, Sebastian Goodman, Xiao Wang,

- Yi Tay, et al. PaLI-X: On scaling up a multilingual vision and language model. *arXiv preprint arXiv:2305.18565*, 2023.
- [29] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.
- [30] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing GPT-4 with 90%\* ChatGPT quality, March 2023.
- [31] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.
- [32] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.
- [33] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.
- [34] Yuchen Cui, Siddharth Karamcheti, Raj Palleti, Nidhya Shivakumar, Percy Liang, and Dorsa Sadigh. No, to the Right: Online Language Corrections for Robotic Manipulation via Shared Autonomy. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction, HRI '23*. ACM, March 2023.
- [35] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.
- [36] Sheelabhadra Dey, Sumedh Pendurkar, Guni Sharon, and Josiah P Hanna. A joint imitation-reinforcement learning framework for reduced baseline regret. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3485–3491. IEEE, 2021.
- [37] Norman Di Palo, Arunkumar Byravan, Leonard Hasenclever, Markus Wulfmeier, Nicolas Heess, and Martin Riedmiller. Towards a unified agent



- with foundation models. In *Workshop on Reincarnating Reinforcement Learning at ICLR 2023*, 2023.
- [38] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2020.
- [39] Kenji Doya. What are the computations of the cerebellum, the basal ganglia and the cerebral cortex? *Neural Networks*, 12(7-8):961–974, 1999.
- [40] Kenji Doya. Complementary roles of basal ganglia and cerebellum in learning and motor control. *Current Opinion in Neurobiology*, 10(6):732–739, 2000.
- [41] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, et al. PaLM-E: An embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.
- [42] Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. In Sarath Chandar, Razvan Pascanu, Hanie Sedghi, and Doina Precup, editors, *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pages 120–136. PMLR, 22–25 Aug 2023.
- [43] Aaron Eisermann, Jae Hee Lee, Cornelius Weber, and Stefan Wermter. Generalization in multimodal language learning from simulation. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2021)*, Jul 2021.
- [44] Aysu Ezen-Can. A comparison of LSTM and BERT for small corpus. *arXiv preprint arXiv:2009.05451*, 2020.
- [45] Otto Fabius and Joost R Van Amersfoort. Variational recurrent auto-encoders. *arXiv preprint arXiv:1412.6581*, 2014.
- [46] Ying Fan, Olivia Watkins, Yuqing Du, Hao Liu, Moonkyung Ryu, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, Kangwook Lee, and Kimin Lee. Reinforcement learning for fine-tuning text-to-image diffusion models. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- [47] Yunhai Feng, Nicklas Hansen, Ziyang Xiong, Chandramouli Rajagopalan, and Xiaolong Wang. Finetuning offline world models in the real world. In *Conference on Robot Learning (CoRL)*, pages 425–445. PMLR, 2023.

- [48] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning (CoRL)*, pages 158–168. PMLR, 2022.
- [49] Connor Gäde, Jan-Gerrit Habekost, and Stefan Wermter. Domain adaptation as auxiliary task for sim-to-real transfer in vision-based neuro-robotic control. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2024.
- [50] Connor Gäde, Ozan Özdemir, Cornelius Weber, and Stefan Wermter. Embodying language models in robot action. In *Proceedings of the 32nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2024)*, pages 625–630. Ciaco - i6doc.com, Oct 2024.
- [51] Arthur M Glenberg and Michael P Kaschak. Grounding language in action. *Psychonomic Bulletin & Review*, 9(3):558–565, 2002.
- [52] Praseon Goyal, Scott Niekum, and Raymond Mooney. Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards. In Jens Kober, Fabio Ramos, and Claire Tomlin, editors, *Proceedings of the 2020 Conference on Robot Learning (CoRL)*, volume 155 of *Proceedings of Machine Learning Research*, pages 485–497. PMLR, 16–18 Nov 2021.
- [53] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. The “something something” video database for learning and evaluating visual common sense. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 5842–5850, 2017.
- [54] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18995–19012, 2022.
- [55] Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. Instruction-driven history-aware policies for robotic manipulations. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning (CoRL)*, volume 205 of *Proceedings of Machine Learning Research*, pages 175–187. PMLR, 14–18 Dec 2023.
- [56] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation

- and reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning (CoRL)*, volume 100 of *Proceedings of Machine Learning Research*, pages 1025–1037. PMLR, 30 Oct–01 Nov 2020.
- [57] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870. PMLR, 2018.
- [58] Jan-Gerrit Habekost, Connor Gäde, Philipp Allgeuer, and Stefan Wermter. Inverse kinematics for neuro-robotic grasping with humanoid embodied agents. *arXiv preprint arXiv:2404.08825*, 2024.
- [59] Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.
- [60] Nicklas A Hansen, Hao Su, and Xiaolong Wang. Temporal difference learning for model predictive control. In *International Conference on Machine Learning (ICML)*, pages 8387–8406. PMLR, 2022.
- [61] Jun Hatori, Yuta Kikuchi, Sosuke Kobayashi, Kuniyuki Takahashi, Yuta Tsuboi, Yuya Unno, Wilson Ko, and Jethro Tan. Interactively picking real-world objects with unconstrained spoken language instructions. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3774–3781. IEEE, 2018.
- [62] Olaf Hauk, Ingrid Johnsrude, and Friedemann Pulvermüller. Somatotopic representation of action words in human motor and premotor cortex. *Neuron*, 41(2):301–307, 2004.
- [63] Hongmei He, John Gray, Angelo Cangelosi, Qinggang Meng, T. Martin McGinnity, and Jörn Mehnen. The challenges and opportunities of human-centered AI for trustworthy robots and autonomous systems. *IEEE Transactions on Cognitive and Developmental Systems*, 14(4):1398–1412, 2022.
- [64] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, Oct 2017.
- [65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.
- [66] Stefan Heinrich, Matthias Kerzel, Erik Strahl, and Stefan Wermter. Embodied multimodal interaction in language learning: the EMIL data collection. In *Proceedings of the ICDL-EpiRob Workshop on Active Vision, Attention, and Learning (ICDL-Epirob 2018 AVAL)*, page 2p, 2018.

- [67] Stefan Heinrich, Cornelius Weber, Stefan Wermter, Ruobing Xie, Yankai Lin, and Zhiyuan Liu. Crossmodal language grounding, learning, and teaching. In *CoCo@ NIPS*, 2016.
- [68] Stefan Heinrich and Stefan Wermter. Interactive natural language acquisition in a multi-modal recurrent neural architecture. *Connection Science*, 30(1):99–133, 2018.
- [69] Stefan Heinrich, Yuan Yao, Tobias Hinz, Zhiyuan Liu, Thomas Hummel, Matthias Kerzel, Cornelius Weber, and Stefan Wermter. Crossmodal language grounding in an embodied neurocognitive model. *Frontiers in Neuro-robotics*, 14:52, 2020.
- [70] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [71] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR)*, 2022.
- [72] Yingdong Hu, Fanqi Lin, Tong Zhang, Li Yi, and Yang Gao. Look before you leap: Unveiling the power of GPT-4V in robotic vision-language planning. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024.
- [73] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 9118–9147. PMLR, 17–23 Jul 2022.
- [74] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning (CoRL)*, pages 1769–1782. PMLR, 2023.
- [75] Muhammad Zubair Irshad, Chih-Yao Ma, and Zsolt Kira. Hierarchical cross-modal agent for robotics vision-and-language navigation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13238–13246, 2021.
- [76] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver IO: A general architecture for structured

- inputs & outputs. In *International Conference on Learning Representations (ICLR)*, 2021.
- [77] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RL-Bench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.
- [78] 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 (CoRL)*, pages 991–1002. PMLR, 2022.
- [79] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. VIMA: Robot manipulation with multimodal prompts. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 14975–15022. PMLR, 23–29 Jul 2023.
- [80] Chenying Jin, Xiang Feng, and Huiqun Yu. A brain-inspired incremental multitask reinforcement learning approach. *IEEE Transactions on Cognitive and Developmental Systems*, 16(3):1147–1160, 2024.
- [81] Chuhao Jin, Wenhui Tan, Jiange Yang, Bei Liu, Ruihua Song, Limin Wang, and Jianlong Fu. Alphablock: Embodied finetuning for vision-language reasoning in robot manipulation. *arXiv preprint arXiv:2305.18898*, 2023.
- [82] Vidur Joshi, Matthew Peters, and Mark Hopkins. Extending a parser to distant domains using a few dozen partially annotated examples. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1190–1199, Melbourne, Australia, July 2018. Association for Computational Linguistics.
- [83] Ryan Julian, Benjamin Swanson, Gaurav Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman. Never stop learning: The effectiveness of finetuning in robotic reinforcement learning. In *Conference on Robot Learning (CoRL)*, pages 2120–2136. PMLR, 2021.
- [84] Michael P Kaschak, Carol J Madden, David J Therriault, Richard H Yaxley, Mark Aveyard, Adrienne A Blanchard, and Rolf A Zwaan. Perception of motion affects language processing. *Cognition*, 94(3):B79–B89, 2005.
- [85] Matthias Kerzel, Theresa Pekarek-Rosin, Erik Strahl, Stefan Heinrich, and Stefan Wermter. Teaching NICO how to grasp: an empirical study on cross-modal social interaction as a key factor for robots learning from humans. *Frontiers in Neurorobotics*, 14:28, 2020.

- [86] Matthias Kerzel, Erik Strahl, Sven Magg, Nicolás Navarro-Guerrero, Stefan Heinrich, and Stefan Wermter. NICO—Neuro-Inspired COmpanion: A developmental humanoid robot platform for multimodal interaction. In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 113–120. IEEE, 2017.
- [87] Jaehyeon Kim, Jungil Kong, and Juhee Son. Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech. In *International Conference on Machine Learning (ICML)*, pages 5530–5540. PMLR, 2021.
- [88] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.
- [89] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, May 7-9, 2015*.
- [90] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of International Conference on Learning Representations (ICLR), Banff, AB, Canada, April 14-16, 2014*.
- [91] Niko Kleer, Maurice Rekrut, Julian Wolter, Tim Schwartz, and Michael Feld. A Multimodal Teach-in Approach to the Pick-and-Place Problem in Human-Robot Collaboration. In *Companion of the 2023 ACM/IEEE International Conference on Human-Robot Interaction, HRI '23*, page 81–85, New York, NY, USA, 2023. Association for Computing Machinery.
- [92] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems*, 2, 1989.
- [93] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.
- [94] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [95] Suhyeon Lee, Won Jun Kim, and Jong Chul Ye. LLM Itself Can Read and Generate CXR Images. *arXiv preprint arXiv:2305.11490*, 2023.



- 
- [96] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of the 40th International Conference on Machine Learning, ICML'23*. JMLR.org, 2023.
- [97] Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. Reinforcement learning for robust parameterized locomotion control of bipedal robots. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2811–2817. IEEE, 2021.
- [98] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.
- [99] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: from natural language instructions to feasible plans. *Autonomous Robots*, Nov 2023.
- [100] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Computer Vision—ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.
- [101] Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A Structured Self-Attentive Sentence Embedding. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.
- [102] Hao Liu, Lisa Lee, Kimin Lee, and Pieter Abbeel. Instruction-Following Agents with Multimodal Transformer. *arXiv preprint arXiv:2210.13431*, 2023.
- [103] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- [104] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. ViLBERT: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.



- [105] Yicheng Luo, Jackie Kay, Edward Grefenstette, and Marc Peter Deisenroth. Finetuning from offline reinforcement learning: Challenges, trade-offs and practical solutions. *arXiv preprint arXiv:2303.17396*, 2023.
- [106] 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 (CoRL)*, pages 1113–1132. PMLR, 2020.
- [107] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. In Dylan A. Shell, Marc Toussaint, and M. Ani Hsieh, editors, *Robotics: Science and System XVII*, 2021.
- [108] Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters*, 2023.
- [109] Yecheng Jason Ma, William Liang, Vaidehi Som, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. Liv: Language-image representations and rewards for robotic control. *arXiv preprint arXiv:2306.00958*, 2023.
- [110] Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2022.
- [111] Joseph Marino. Predictive coding, variational autoencoders, and biological connections. *Neural Computation*, 34(1):1–44, 2021.
- [112] Oier Mees, Jessica Borja-Diaz, and Wolfram Burgard. Grounding language with visual affordances over unstructured data. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11576–11582. IEEE, 2023.
- [113] Oier Mees, Lukas Hermann, and Wolfram Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and Automation Letters*, 7(4):11205–11212, 2022.
- [114] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.
- [115] Lotte Meteyard, Bahador Bahrami, and Gabriella Vigliocco. Motion detection and motion verbs: Language affects low-level visual perception. *Psychological Science*, 18(11):1007–1013, 2007. PMID: 17958716.
- [116] Grégoire Mialon, Roberto Dessi, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Roziere, Timo Schick, Jane

- Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented Language Models: a Survey. *Transactions on Machine Learning Research (TMLR)*, 2023.
- [117] Microsoft. Introducing Microsoft 365 Copilot — your copilot for work. 2023.
- [118] Matthias Minderer, Alexey Gritsenko, and Neil Houlsby. Scaling open-vocabulary object detection. *Advances in Neural Information Processing Systems*, 36, 2024.
- [119] Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen, Xiao Wang, Xiaohua Zhai, Thomas Kipf, and Neil Houlsby. Simple open-vocabulary object detection with vision transformers. *European Conference on Computer Vision (ECCV)*, 2022.
- [120] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. *Advances in Neural Information Processing Systems*, 27, 2014.
- [121] Jun Morimoto, Gordon Cheng, Christopher G Atkeson, and Garth Zeglin. A simple reinforcement learning algorithm for biped walking. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 3, pages 3030–3035. IEEE, 2004.
- [122] Suraj Nair, Eric Mitchell, Kevin Chen, Silvio Savarese, Chelsea Finn, et al. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning (CoRL)*, pages 1303–1315. PMLR, 2022.
- [123] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. In *Conference on Robot Learning (CoRL)*, pages 892–909. PMLR, 2023.
- [124] Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, et al. PIVOT: Iterative visual prompting elicits actionable knowledge for VLMs. *arXiv preprint arXiv:2402.07872*, 2024.
- [125] Hwei Geok Ng, Paul Anton, Marc Brügger, Nikhil Churamani, Erik Fließwasser, Thomas Hummel, Julius Mayer, Waleed Mustafa, Thi Linh Chi Nguyen, Quan Nguyen, et al. Hey robot, why don't you talk to me? In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 728–731. IEEE, 2017.
- [126] Farzad Niroui, Kaicheng Zhang, Zendai Kashino, and Goldie Nejat. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. *IEEE Robotics and Automation Letters*, 4(2):610–617, 2019.

- [127] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. <https://octo-models.github.io>, 2023.
- [128] Tetsuya Ogata, Masamitsu Murase, Jun Tani, Kazunori Komatani, and Hiroshi G. Okuno. Two-way translation of compound sentences and arm motions by recurrent neural networks. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1858–1863, 2007.
- [129] OpenAI. Introducing ChatGPT. 2022.
- [130] OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.
- [131] OpenAI. GPT-4V(ision) System Card. 2023.
- [132] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research (TMLR)*, 2024.
- [133] Ozan Özdemir, Matthias Kerzel, Cornelius Weber, Jae Hee Lee, and Stefan Wermter. Language-model-based paired variational autoencoders for robotic language learning. *IEEE Transactions on Cognitive and Developmental Systems (TCDS)*, 15(4):1812–1824, 2023.
- [134] Ozan Özdemir, Matthias Kerzel, Cornelius Weber, Jae Hee Lee, Muhammad Burhan Hafez, Patrick Bruns, and Stefan Wermter. Learning bidirectional action-language translation with limited supervision and testing with incongruent input. *Applied Artificial Intelligence*, 37(1):2179167, 2023.
- [135] Ozan Özdemir, Matthias Kerzel, Cornelius Weber, Jae Hee Lee, and Stefan Wermter. Learning flexible translation between robot actions and language descriptions. In Elias Pimenidis, Plamen Angelov, Chrisina Jayne, Antonios Papaleonidas, and Mehmet Aydin, editors, *Artificial Neural Networks and Machine Learning – ICANN 2022*, pages 246–257, Cham, 2022. Springer Nature Switzerland.
- [136] Ozan Özdemir, Matthias Kerzel, and Stefan Wermter. Embodied language learning with paired variational autoencoders. In *2021 IEEE International Conference on Development and Learning (ICDL)*, pages 1–6, Aug 2021.

- 
- [137] Ozan Özdemir, Cornelius Weber, Jae Hee Lee, and Stefan Wermter. Improving action precision of a bidirectional transformer via reinforcement learning. Submitted to *IEEE Transactions on Cognitive and Developmental Systems (TCDS)*, 2024.
- [138] Chris Paxton, Yonatan Bisk, Jesse Thomason, Arunkumar Byravan, and Dieter Foxl. Prospection: Interpretable plans from language by predicting the future. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6942–6948. IEEE, 2019.
- [139] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
- [140] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, pages 8748–8763. PMLR, 2021.
- [141] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning (ICML)*, pages 28492–28518. PMLR, 2023.
- [142] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.
- [143] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.
- [144] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(1):1–67, January 2020.
- [145] Ram Ramrakhya, Dhruv Batra, Erik Wijmans, and Abhishek Das. PIRLNav: Pretraining with imitation and RL finetuning for ObjectNav. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17896–17906, 2023.
- [146] Scott Reed, Konrad Zolna, Emilio Parisotto, et al. A generalist agent. *Transactions on Machine Learning Research (TMLR)*, 2022.

- [147] Allen Z Ren, Bharat Govil, Tsung-Yen Yang, Karthik R Narasimhan, and Anirudha Majumdar. Leveraging language for accelerated learning of tool manipulation. In *Conference on Robot Learning (CoRL)*, pages 1531–1541. PMLR, 2023.
- [148] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*, 28, 2015.
- [149] 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](http://www.coppeliarobotics.com).
- [150] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.
- [151] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive Neural Networks. *arXiv preprint arXiv:1606.04671*, 2022.
- [152] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proceedings of Interspeech 2014*, pages 338–342, 2014.
- [153] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.
- [154] Gabriele Sarti and Malvina Nissim. IT5: Text-to-text pretraining for Italian language understanding and generation. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 9422–9433, Torino, Italy, May 2024. ELRA and ICCL.
- [155] BigScience Workshop: Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. *arXiv preprint arXiv:2211.05100*, 2023.
- [156] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

- 
- [157] Younggyo Seo, Danijar Hafner, Hao Liu, Fangchen Liu, Stephen James, Kimin Lee, and Pieter Abbeel. Masked world models for visual control. In *Conference on Robot Learning (CoRL)*, pages 1332–1344. PMLR, 2023.
- [158] Lin Shao, Toki Migimatsu, Qiang Zhang, Karen Yang, and Jeannette Bohg. Concept2Robot: Learning manipulation concepts from instructions and human demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2020.
- [159] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. CLIPort: What and where pathways for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, pages 894–906. PMLR, 2022.
- [160] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-Actor: A multi-task transformer for robotic manipulation. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, pages 785–799. PMLR, 2023.
- [161] Mohit Shridhar, Dixant Mittal, and David Hsu. INGRESS: Interactive visual grounding of referring expressions. *The International Journal of Robotics Research*, 39(2-3):217–232, 2020.
- [162] Andrew Silva, Nina Moorman, William Silva, Zulfiqar Zaidi, Nakul Gopalan, and Matthew Gombolay. LanCon-Learn: Learning with language to enable generalization in multi-task manipulation. *IEEE Robotics and Automation Letters*, 7(2):1635–1642, 2021.
- [163] Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning (ICML)*, pages 9767–9779. PMLR, 2021.
- [164] Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Brianna Zitkovich, Fei Xia, Chelsea Finn, et al. Open-world object manipulation using pre-trained vision-language models. *arXiv preprint arXiv:2303.00905*, 2023.
- [165] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4–7, 2018, Proceedings, Part III 27*, pages 270–279. Springer, 2018.
- [166] Chao Tang, Dehao Huang, Wenqi Ge, Weiyu Liu, and Hong Zhang. GraspGPT: Leveraging semantic knowledge from a large language model for task-oriented grasping. *IEEE Robotics and Automation Letters*, 2023.
- [167] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.



- [168] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [169] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [170] Du Tran, Heng Wang, Matt Feiszli, and Lorenzo Torresani. Video classification with channel-separated convolutional networks. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5551–5560, 2019.
- [171] Matej Ulčar and Marko Robnik-Šikonja. Sequence-to-sequence pretraining for a less-resourced Slovenian language. *Frontiers in Artificial Intelligence*, 6, 2023.
- [172] Michiel van Elk, Hein T van Schie, Rolf A Zwaan, and Harold Bekkering. The functional role of motor activation in language processing: Motor cortical oscillations support lexical-semantic retrieval. *Neuroimage*, 50(2):665–677, 2010.
- [173] 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.
- [174] Wai Keen Vong, Wentao Wang, A Emin Orhan, and Brenden M Lake. Grounded language acquisition through the eyes and ears of a single child. *Science*, 383(6682):504–511, 2024.
- [175] Quan Vuong, Sergey Levine, Homer Rich Walke, Karl Pertsch, Anikait Singh, Ria Doshi, Charles Xu, Jianlan Luo, Liam Tan, Dhruv Shah, et al. Open X-Embodiment: Robotic learning datasets and RT-X models. In *2nd Workshop on Language and Robot Learning: Language as Grounding*, 2023.
- [176] Lei Wang, Yunzhou Zhang, Delong Zhu, Sonya Coleman, and Dermot Kerr. Supervised meta-reinforcement learning with trajectory optimization for manipulation tasks. *IEEE Transactions on Cognitive and Developmental Systems*, 16(2):681–691, 2024.
- [177] Wenhai Wang, Zhe Chen, Xiaokang Chen, Jiannan Wu, Xizhou Zhu, Gang Zeng, Ping Luo, Tong Lu, Jie Zhou, Yu Qiao, and Jifeng Dai. VisionLLM: Large language model is also an open-ended decoder for vision-centric tasks. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 61501–61513. Curran Associates, Inc., 2023.



- 
- [178] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned Language Models are Zero-Shot Learners. In *International Conference on Learning Representations (ICLR), Virtual Event, April 25-29, 2022*, 2022.
- [179] Stefan Wermter, Günther Palm, and Mark I. Elshaw. *Biomimetic Neural Learning for Intelligent Robots. Intelligent Systems, Cognitive Robotics and Neuroscience*. Springer, Jul 2005.
- [180] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *International Conference on Learning Representations (ICLR)*, 2019.
- [181] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.
- [182] Alice Winter, Carolin Dudschig, Jeff Miller, Rolf Ulrich, and Barbara Kaup. The action-sentence compatibility effect (ACE): Meta-analysis of a benchmark finding for embodiment. *Acta Psychologica*, 230:103712, 2022.
- [183] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. TidyBot: Personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, 2023.
- [184] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [185] Tatsuro Yamada, Hiroyuki Matsunaga, and Tetsuya Ogata. Paired recurrent autoencoders for bidirectional translation between robot actions and linguistic descriptions. *IEEE Robotics and Automation Letters*, 3(4):3441–3448, 2018.
- [186] Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan, Chris Tar, Yun-hsuan Sung, Brian Strope, and Ray Kurzweil. Multilingual universal sentence encoder for semantic retrieval. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 87–94, Online, July 2020. Association for Computational Linguistics.
- [187] Xiangtong Yao, Zhenshan Bing, Genghang Zhuang, Kejia Chen, Hongkuan Zhou, Kai Huang, and Alois Knoll. Learning from symmetry: Meta-reinforcement learning with symmetrical behaviors and language instructions. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5574–5581. IEEE, 2023.

- [188] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, pages 1094–1100. PMLR, 2020.
- [189] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2528–2535. IEEE, 2010.
- [190] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. In Jens Kober, Fabio Ramos, and Claire J. Tomlin, editors, *4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA*, volume 155 of *Proceedings of Machine Learning Research*, pages 726–747. PMLR, 2020.
- [191] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11975–11986, 2023.
- [192] Han Zhang, Weichong Yin, Yewei Fang, Lanxin Li, Boqiang Duan, Zhihua Wu, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. ERNIE-ViLG: Unified Generative Pre-training for Bidirectional Vision-Language Generation. *arXiv preprint arXiv:2112.15283*, 2021.
- [193] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: Open Pre-trained Transformer Language Models. *arXiv preprint arXiv:2205.01068*, 2022.
- [194] Zhe Zhang, Wei Chai, and Jiankun Wang. Mani-GPT: A Generative Model for Interactive Robotic Manipulation. *Procedia Computer Science*, 226:149–156, 2023. Proceedings of International Conference on Biomimetic Intelligence and Robotics.
- [195] Wenshuai Zhao, Jorge Peña Queraltá, 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.
- [196] Xufeng Zhao, Mengdi Li, Cornelius Weber, Burhan Hafez, and Stefan Wermter. Chat with the Environment: Interactive Multimodal Perception

- Using Large Language Models. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3590–3596, Oct 2023.
- [197] Hongkuan Zhou, Zhenshan Bing, Xiangtong Yao, Xiaojie Su, Chenguang Yang, Kai Huang, and Alois Knoll. Language-conditioned imitation learning with base skill priors under unstructured data. *arXiv preprint arXiv:2305.19075*, 2023.
- [198] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. MiniGPT-4: Enhancing vision-language understanding with advanced large language models. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.



# Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Sofern im Zuge der Erstellung der vorliegenden Dissertationsschrift generative Künstliche Intelligenz (gKI) basierte elektronische Hilfsmittel verwendet wurden, versichere ich, dass meine eigene Leistung im Vordergrund stand und dass eine vollständige Dokumentation aller verwendeten Hilfsmittel gemäß der Guten wissenschaftlichen Praxis vorliegt. Ich trage die Verantwortung für eventuell durch die gKI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate.

Hamburg, 09.12.2024  
Ort, Datum



Unterschrift



# Erklärung zur Veröffentlichung

Ich erkläre mein Einverständnis mit der Einstellung dieser Dissertation in den Bestand der Bibliothek.

Hamburg, 09.12.2024  
Ort, Datum



Unterschrift



