

Spatial Reasoning and Planning for Deep Embodied Agents



Shu Ishida
Christ Church

Supervised by Dr. João F. Henriques

Visual Geometry Group
Department of Engineering Science
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Hilary 2024

Abstract

Humans can perform complex tasks with long-term objectives by planning, reasoning, and forecasting outcomes of actions. For embodied agents (e.g. robots) to achieve similar capabilities, they must gain knowledge of the environment transferable to novel scenarios with a limited budget of additional trial and error. Learning-based approaches, such as deep reinforcement learning, can discover and take advantage of inherent regularities and characteristics of the application domain from data, and continuously improve their performances, however at a cost of large amounts of training data. This thesis explores the development of data-driven techniques for spatial reasoning and planning tasks, focusing on enhancing learning efficiency, interpretability, and transferability across novel scenarios.

Four key contributions are made. Firstly, CALVIN, a differential planner that learns interpretable models of the world for long-term planning. It successfully navigated partially observable 3D environments, such as mazes and indoor rooms, by learning the rewards (goals and obstacles) and state transitions (robot dynamics) from expert demonstrations.

Secondly, SOAP, a reinforcement learning algorithm that discovers macro-actions (options) unsupervised for long-horizon tasks. Options segment a task into subtasks and enable consistent execution of the subtask. SOAP showed robust performances on history-conditional corridor tasks as well as classical benchmarks such as Atari.

Thirdly, LangProp, a code optimisation framework using Large Language Models to solve embodied agent problems that require reasoning by treating code as learnable policies. The framework successfully generated interpretable code with comparable or superior performance to human-written experts in the CARLA autonomous driving benchmark.

Finally, Voggite, an embodied agent with a vision-to-action transformer backend that solves complex tasks in Minecraft. It achieved third place in the MineRL BASALT Competition by identifying action triggers to segment tasks into multiple stages.

These advancements provide new avenues for applications of learning-based methods in complex spatial reasoning and planning challenges.

Keywords — machine learning, neural networks, deep reinforcement learning, imitation learning, hierarchical reinforcement learning, policy optimisation, robotics, autonomous driving, embodied agents, option discovery, skill learning, navigation, planning, computer vision, large language models, multi-modal foundation models.

Declaration

This thesis is submitted to the Department of Engineering Science, University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy. I declare that this thesis is entirely my own work and, except where stated, describes my own research.

Shu Ishida

Christ Church

Acknowledgements

Words cannot fully express my sincere gratitude towards my supervisor, Dr. João F. Henriques, for his support and guidance throughout my journey as a DPhil student. He has never hesitated to share with me his wisdom and insight on a wide variety of disciplines, including very helpful advice on programming, debugging, visualisation, and mathematical techniques. I have always admired his exceptional intelligence, hands-on skills, deep conceptual understanding, and attention to technical details, as well as his humility and approachability. He has always believed in me, even in times when I lost confidence in myself. I am immensely grateful for João's unwavering trust and encouragement, and for this fortunate opportunity of working with him throughout my DPhil research.

I would like to thank the other Principal Investigators at the Visual Geometry Group (VGG) — Prof. Andrew Zisserman, Prof. Andrea Vedaldi, Dr. Christian Ruppert, and Dr. Iro Laina — for making the lab a vibrant place for research. I would also like to thank all my fellow and former research students in the lab who helped make the lab a welcoming and friendly environment. A special thanks to the following people: Max Bain, who introduced me to my supervisor João and encouraged me to join VGG; Oliver Groth, Gül Varol, and João for organising the Big Picture Debates online during the COVID-19 lockdown when I was joining the lab, which helped me virtually meet people and have meaningful discussions with them during those challenging times; Shangzhe (Elliot) Wu, Luke Melas-Kyriazi, Paul Engstler, Ragav Sachdeva, Vladimir Iashin along with many others who have regularly organised lab socials, lunches and dinners for us.

This work has been made possible by the Autonomous Intelligent Machines and Systems Centre for Doctoral Training programme (AIMS CDT), funded by EPSRC (Engineering and Physical Research Council). I am extremely grateful for having Mrs. Wendy Poole as our Centre Administrator of AIMS CDT, who has dedicated so much thought and effort into running the course smoothly, and providing us with the best experience that the CDT could offer. I deeply respect Wendy for her planning and organisational skills, and her prompt and accurate responses to enquiries. I have made valuable friends in the AIMS CDT throughout the years, thanks to the annual meetings and regular social events across cohorts. My thanks also go to Ivan Kiskin, Yuki Asano, and Robert McCraith for kindly offering their knowledge, wisdom and guidance as experienced

members/alumni of the CDT.

I would like to thank Stuart Golodetz for supervising me for one of my AIMS CDT mini-projects in collaboration with Five AI. This was the first research collaboration I undertook, and it was a valuable experience working with him and learning from his insight and good coding practices. I would also like to thank the organisers of the MineRL BASALT Competition at NeurIPS 2022, in particular Anssi Kanervisto, who was incredibly helpful and patient with all my technical enquiries during the competition.

I thank all the people at Wayve Technologies, where I conducted a part of my research as an intern. In particular, I would like to thank Anthony Hu, who has been an amazing mentor and supervisor during my internship. I have learnt so much both from his research and team communication skills, as well as his kindness and care. He offered me substantial guidance when I was setting up my simulation and experiments, and have always provided me with academic and emotional support. I also thank Gianluca Corrado for his valuable and helpful advice as my manager, and George Fedoseev, Hudson Yeo, Lloyd Russell, and Corina Gurau for offering their insight, support and company during my internship in the world modelling team.

I would also like to thank my supervisors and colleagues at Microsoft Research Cambridge, where I spent the last few months of my DPhil degree as a research science intern in the Game Intelligence team. My special thanks go to Sergio Valcarcel Macua, Raluca Georgescu, Abdelhak Lemkhenter, and Tabish Rashid, who have spent a considerable amount of time mentoring me throughout the project, as well as to Katja Hofmann, Sam Devlin, Dave Bignell, Tim Pearce, Yuhan Cao, Shanzheng Tan, and Linda Yilin Wen. Everyone was always happy to help me with any issues with the code and infrastructure, and I really appreciated their inclusiveness in their activities and their generosity with their time. I would also like to express my immense gratitude towards my fellow forty interns, all of whom have made my internship experience special. Everyone has been very friendly, sociable and inclusive, and created a wholesome welcoming community. Special mention to Marko Tot, Chentian Jiang, Daniel Clark, Lucia Adams, and Dmitrii Usynin.

This research would not have been possible without the generous support by the Ezoë Memorial Recruit Foundation. I would like to express my sincere gratitude to the foundation for partially funding my undergraduate studies and fully funding my CDT and DPhil studies via the Recruit academic scholarship.

Over the course of my studies at Oxford, I had the privilege of being taught and advised by brilliant researchers, lecturers and supervisors. I would especially like to thank my undergraduate college tutor and postgraduate college advisor Prof. Malcolm

McCulloch, my Master's thesis supervisor Prof. Nick Hawes, my thesis mentor Marc Rigter, and lectures on machine learning and optimisation Prof. Philip Torr and Dr. M. Pawan Kumar for their valuable teaching and insights, and for their kind and generous support of my research and career. I would also like to thank my thesis examiners, Prof. Ioannis Havoutis and Dr. Markus Wulfmeier, for their thorough and insightful feedback.

Through my extra-curricular activities, I had the privilege of working with research students with great leadership and skills. My thanks go to Jonas Beuchert and Augustinas Malinauskas for teaming up at the Oxford Hackathon, Lynn Hirose and Naho Tomiki for collaborating on the University of Tokyo Project Sprint, Mark Finean, Charlie Street, and Ricardo Cannizzaro for leading Team ORIon for the RoboCup competitions, and Timothy Seabrook, Ding Shin Huang, Chuxuan (Jessie) Jiang, and Siobhan Mackenzie Hall for leading initiatives at the Oxford Artificial Intelligence Society, as well as to all the team and committee members who made the initiatives possible. I would also like to thank Prof. Paul Newman, Prof. Nick Hawes, Prof. Ioannis Havoutis, Dr. Lars Kunze and Dr. Bruno Lacerda for their support for the RoboCup competition, as well as Dr. Natalia Efremova for her mentorship and insights for one of the OxAI Labs research projects.

I was fortunate to make many dear friends and meet countless kindhearted and wonderful people during my studies, for which I am immensely thankful. These people have helped shape who I am today, and I continue to learn from their philosophies. While I would not be able to express my gratitude in this limited space for everyone who deserves them, I would like to mention the following friends who have deeply impacted me and supported me.

My flatmates: Andreas Schmid, Richard Csaky, and Danilo Jr Dela Cruz — my time in Oxford would not have been the same without the company of my amazing flatmates. I have learnt a lot from them, both in terms of work and life. I fondly remember our co-working days and board game nights, the bike trips with Andreas, the road trip with Richard, and many geeky and philosophical evening chats with Danilo.

My Christ Church friends and Japan Society friends — having your company, enjoying meals, events, and celebrations together, and being part of the wonderful community were the things that made my life at Oxford special, and I cannot thank you all enough.

Kengo Shibata, Hiroto Takahashi, Fabrice Wunderlich, Zihan (Amanda) Zhu, Aya Fujita, Catherine Choi — some of my DPhil student friends who have been with me throughout most of my journey and have helped create wonderful memories at Oxford. I cherish all the deep conversations we have had and the time we have spent together.

Bingqing Liu — my study partner throughout my undergraduate degree. Thank you

for always providing inspiration and motivation, taking the initiative to organise study sessions, and proactively finding events and competitions that we could participate in.

Nader Raafat, Megan Craig — my friends who have supported me since my undergraduate days, during the lockdown and through thick and thin, and have continued to keep in touch with me despite the long distances. You have always been there for me, which means a fortune.

Ximei Liu — thank you for always believing in me and encouraging me. Your enthusiasm and courage are only outshone by your kindness and care, and I am blessed to have you in my life and share the adventures together.

Last but not least, I would like to thank my mother, father, and grandparents for their unconditional love, and for always looking out for me. Knowing that they will be there for me and that I can count on their moral support has given me courage and strength, and is the reason why I can keep pursuing my goals and dreams.

Contents

Contents	viii
List of Figures	xii
List of Tables	xvii
List of Abbreviations	xix
1 Introduction	1
1.1 Motivation	1
1.2 Research objectives	3
1.2.1 Learn a generalisable planner	3
1.2.2 Discover reusable skills	4
1.2.3 Solve POMDP environments with memory-augmented policies	5
1.2.4 Explain the behaviour of experts and agents	5
1.2.5 Train embodied agents to perform complex tasks	6
1.3 Main contributions	6
1.4 Outline	9
2 Background on planning and data-driven decision making	10
2.1 Planning algorithms	10
2.1.1 Markov Decision Process	11
2.1.2 Partially Observable Markov Decision Process	11
2.1.3 Classical approaches to planning	12
2.2 Reinforcement Learning	13
2.2.1 On-policy and off-policy	13
2.2.2 Value-based methods	14
2.2.3 Policy gradient methods	17
2.2.4 Actor-Critic methods	18
2.2.5 Hierarchical Reinforcement Learning	20
2.2.6 Transfer and generalisation in Reinforcement Learning	22
2.2.7 Model-based methods	23
2.3 Other learning methods	26
2.3.1 Imitation Learning	26
2.3.2 Value Iteration Networks	27
2.3.3 Fine-tuning and Curriculum Learning	27
2.4 Embodied agents	28
2.4.1 Visual navigation	28
2.4.2 Decision making in autonomous driving	29
2.4.3 LLMs for automating compositional tasks	31

2.4.4	Task execution in virtual worlds	31
3	Towards real-world navigation with deep differentiable planners	35
3.1	Introduction.....	36
3.2	Background	38
3.2.1	Value Iteration	38
3.2.2	Value Iteration Network	39
3.2.3	Applications of Value Iteration Networks	40
3.2.4	Limitations of Value Iteration Networks	41
3.3	Collision Avoidance Long-term Value Iteration Network.....	43
3.3.1	Augmented navigation state-action space	43
3.3.2	Embodied navigation in 3D environments	47
3.4	Experiment setup.....	51
3.4.1	Evaluation benchmarks.....	51
3.4.2	Baselines and hyper-parameter choices	52
3.4.3	Expert trajectory generation	53
3.4.4	Inference time rollouts	53
3.5	Experiments	55
3.5.1	Fully observable 2D grid environment	55
3.5.2	Partially observable 2D grid environment.....	57
3.5.3	Embodied navigation with orientation.....	59
3.5.4	Synthetically-rendered 3D environments.....	60
3.5.5	Indoor images from a real-world robot	64
3.5.6	Implementation	65
3.6	Conclusion.....	65
4	Option discovery via Expectation Maximisation and Policy Gradients	66
4.1	Introduction.....	67
4.2	Background	70
4.2.1	Option-Critic architecture.....	70
4.2.2	Double Actor-Critic	72
4.2.3	Expectation Maximisation algorithm	73
4.2.4	Forward-backward algorithm	74
4.3	Option assignment formulation.....	76
4.3.1	Option policy and sub-policy	77
4.3.2	Evaluating the probability of latents.....	77
4.4	Proximal Policy Optimisation via Expectation Maximisation	79
4.4.1	Expected return maximisation objective with options	79
4.4.2	PPO objective with Generalised Advantage Estimation	81
4.5	Sequential Option Advantage Propagation	82
4.5.1	Policy Gradient objective with options	83
4.5.2	Analytic back-propagation of the policy gradient.....	85
4.5.3	Learning objective for option-specific policies and values	86
4.6	Experiments	87

4.6.1	Option learning in corridor environments	89
4.6.2	Stability of the algorithms on CartPole, LunarLander, Atari, and MuJoCo environments	93
4.7	Conclusion	94
5	A code optimisation framework using Large Language Models	99
5.1	Introduction	100
5.2	Background	102
5.2.1	LLMs for code generation	102
5.2.2	LLMs for automated task completion	103
5.3	The LangProp Framework	104
5.3.1	Model definition	104
5.3.2	Model forward pass definition	106
5.3.3	Prompt template engine	109
5.3.4	Trainer forward-backward definition	110
5.3.5	Training paradigm	111
5.4	General domain experiments	113
5.4.1	Generalised Sudoku	113
5.4.2	CartPole	113
5.5	Driving in CARLA	114
5.5.1	Data collection	115
5.5.2	Training the LangProp agent	117
5.5.3	Benchmark	121
5.5.4	Experiments	122
5.6	Implementation	126
5.7	Conclusion	126
6	Embodied task execution in a virtual world	127
6.1	Introduction	128
6.2	Background	130
6.2.1	OpenAI VPT	130
6.2.2	MineRL BASALT Competition 2022	131
6.3	Segmenting stages of task execution in Minecraft	131
6.3.1	Motivation	131
6.3.2	Segmenting by states via Invariant Information Clustering	133
6.3.3	Segmenting by actions via task-specific heuristics	137
6.4	Submission to the MineRL BASALT Competition	139
6.4.1	Voggite	139
6.4.2	Implementation	140
6.4.3	Submission and results	140
6.5	Conclusion	141
7	Conclusion	142
7.1	Discussion	142
7.2	Limitations and future work	147

7.2.1	CALVIN	147
7.2.2	SOAP.....	148
7.2.3	LangProp.....	149
7.2.4	Voggite	151
A	CALVIN embodied navigation with comparisons	154
A.1	Rollout of CALVIN	154
A.2	Rollout of VIN	155
A.3	Rollout of GPPN	155
B	Benchmarking option-based Reinforcement Learning agent	159
C	LangProp code generation and prompt examples	160
C.1	Notes on training with LangProp	160
C.1.1	Choosing the priority discount factor	160
C.1.2	Specifying the policy	161
C.2	LangProp prompt definitions	162
C.2.1	Policy setup prompt examples	162
C.2.2	Policy update prompt example	165
C.2.3	Policy definition for the LangProp driving agent in CARLA	169
C.3	Code generation examples	172
C.3.1	Solutions for Sudoku	172
C.3.2	Solutions for CartPole	175
C.3.3	Driving code generated by LangProp.....	177
D	Clustering the Minecraft inventories	184
	Bibliography	187

List of Figures

3.1	(1st column) Input images seen during a run of CALVIN on AVD (Section 3.5.5). This embodied neural network has learned to efficiently explore and navigate unseen indoor environments, to seek objects of a given class (highlighted in the last image). (2nd-3rd columns) Predicted rewards and values (resp.), for each spatial location (higher for brighter values). The unknown optimal trajectory is dashed, while the robot’s trajectory is solid.	36
3.2	(left) A 2D maze, with the target in yellow. (middle) Values produced by the VIN for each 2D state (actions are taken towards the highest value). Higher values are brighter. The correct trajectory is dashed, and the current one is solid. The agent (orange circle) is stuck due to the local maximum below it. (right) Same values for CALVIN. There are no spurious maxima, and the values of walls are correctly considered low (dark).	42
3.3	Example rollout of CALVIN after 21 steps (left column), 43 steps (middle column) and 65 steps (right column). CALVIN successfully terminated at 65 steps. (top row) Input visualisation: unexplored cells are dark, and the discovered target is yellow. The correct trajectory is dashed, and the current one is solid. The orange circle shows the position of the agent. (bottom row) Predicted values (higher values are brighter). Explored cells have low values, while unexplored cells and the discovered target are assigned high values.	58
3.4	CALVIN’s learnt rewards and values on partially observable 2D mazes with embodied navigation (Section 3.5.3). (left) Input visualisation: unexplored cells are dark, the target is yellow (just found by the agent), and a black arrow shows the agent’s position and orientation. (middle) Close-up of predicted rewards (higher values are brighter) inside the white rectangle of the left panel. The 3D state space (position/orientation) is shown, with rewards for the 8 orientations in a radial pattern within each cell (position). Explored cells have low rewards, with the highest reward at the target. (right) Close-up of predicted values. They are higher facing the direction of the target. Obstacles (black border) have low values.	60
3.5	Example results on MiniWorld (Section 3.5.4). Left to right: input images, predicted rewards and values. The format is as in Figure 3.1. Notice the high reward on unexplored regions, replaced with a single peak around the target when it is seen (last row).	62
3.6	Transition model learnt from MiniWorld trajectories for the <i>move forward</i> action at each discretised orientation, at 45° intervals. Higher values are brighter (yellow for a probability of 1), and lower values are darker (purple for a probability of 0).	63
4.1	An HMM for sequential data \mathbf{X} of length T , given latent variables \mathbf{Z} .	74

4.2	Probabilistic graphical models showing the relationships between options z , actions a and states s at time step t . b_t in the standard options framework denotes a boolean variable that initiates the switching of options when activated. This work adopts a more general formulation compared to the options framework, as defined in Equation (4.15).	77
4.3	An HMM showing the relationships between options z , actions a and states s . The dotted arrows indicate that the same pattern repeats where the intermediate time steps are abbreviated.	78
4.4	A corridor environment. The above example has a length $L = 20$. The agent represented as a green circle starts at the left end of the corridor, and moves towards the right. When it reaches the right end, the agent can either take an up action or a down action. This will either take the agent to a yellow cell or a grey cell. The yellow cell gives a reward of 1, while the grey cell gives a reward of -1 . All other cells give a reward of 0. The location of a rewarding yellow cell and the penalising grey cell are determined by the colour of the starting cell (either "blue" or "red"), as shown, and this is randomised, each with 50% probability. The agent only has access to the colour of the current cell as observation. For simplicity of implementation, the agent's action space is {"up", "down"}, and apart from the fork at the right end, taking either of the actions at each time step will move the agent one cell to the right. The images shown are taken from rollouts of the SOAP agent after training for $100k$ steps. The agent successfully navigated to the rewarding cell in both cases.	89
4.5	Training curves of RL agents showing the episodic rewards obtained in the corridor environment with varying corridor lengths. The mean (solid line) and the min-max range (coloured shadow) for 5 seeds per algorithm are shown.	95
4.6	Training curves of RL agents showing the episodic rewards obtained in the CartPole-v1 and LunarLander-v2 environments. The mean (solid line) and the min-max range (coloured shadow) for 5 seeds per algorithm are shown.	96
4.7	Training curves of RL agents showing the episodic rewards obtained in the Atari environments. The mean (solid line) and the min-max range (coloured shadow) for 3 seeds per algorithm are shown. [Spans multiple pages]	96
4.8	Training curves of RL agents showing the episodic rewards obtained in the MuJoCo environments. The mean (solid line) and the min-max range (coloured shadow) for 3 seeds per algorithm are shown. Note that the PPOEM algorithm failed mid-way in some cases due to training instabilities.	98
5.1	An overview of the LangProp framework, which consists of a LangProp model, an LLM optimiser, and a LangProp trainer. During training, the LLM generates and updates the policy scripts which are evaluated against a training objective. Policies with higher performances are selected for updates, and the best policy is used for inference.	104

5.2	The policy evaluation and update mechanism. The performances of the policies are monitored and aggregated over time by a policy tracker as <i>priorities</i> , used to rerank the policies.	108
5.3	The total number of <i>environment</i> steps required to learn CartPole-v1 (10 seeds per method) in comparison to an RL method, PPO. Most seeds converged to an optimal solution within 10 LangProp updates.	114
5.4	An overview of the CARLA agents for data collection and evaluation. All agents are implemented as a standardised <code>DataAgent</code> class, with one or multiple <code>AgentBrain</code> that takes in the preprocessed observations and outputs a control action. These inputs and outputs are processed, recorded and saved by the <code>DataAgent</code> so that they can be used for data collection or for online training.	115
5.5	An overview of the LangProp agent training pipeline. The LangProp model is updated on a dataset that includes both offline expert data as well as online LangProp data annotated with expert actions, similar to DAgger. The agent is given negative rewards upon infraction.	117
5.6	Training curves for the different training methods of the LangProp agent. The training scores are evaluated on 1000 samples from the offline training dataset and/or online replay buffer, and the validation scores are evaluated on 1000 samples from the offline validation dataset. Updates are performed every 1000 frames of agent driving, and upon infractions in the RL setting. The score is in the range of $[-10, 1]$ due to exception penalties. The axis is limited to $[-1, 1]$ in the plots.	124
6.1	Applying IIC to a video sequence of expert demonstration for the Obtain Diamond task. The example shown is for lookahead $L = 1$. The video sequence is subsampled for every 10 frames and shown from left to right, top to bottom. The coloured strips for every tile of observation correspond to clusters found by applying IIC to the observations. There are 30 clusters in this case.	136
6.2	Trigger actions defined for the Voggite agent to switch to a different policy during task execution. For the “find a cave task”, no triggers are defined and the VPT policy is fine-tuned as normal (with improvements to training techniques as outlined in Section 6.4.3). For “make a waterfall” task, the agent changes its behaviour to climbing down a mountain after the bucket has been used. For “create an animal pen” and “build a house” tasks, the policy distribution is shifted to move around less and commit to building structures after taking the first “use” action.	138

6.3	Diagram of the Voggite training pipeline. VPT embeddings are pre-computed for the expert demonstrations and stored as a permutable dataset, removing the sequential constraint of forward-passing through the VPT transformer. A policy head is trained and fine-tuned on each task in the MineRL BASALT Competition, given the VPT embeddings and expert actions labels. The categorical losses are reweighted inversely to the frequency of the expert actions' occurrences.	140
A.1	Example rollout of embodied CALVIN after 30 steps (left column), 60 steps (middle column) and 90 steps (right column). CALVIN successfully terminated at 91 steps. (first row) Input visualisation: unexplored cells are dark, and the discovered target is yellow. The correct trajectory is dashed, and the current one is solid. The orange triangle shows the position and the orientation of the agent. (second row) Predicted rewards (higher values are brighter). The 3D state-space (position/orientation) is shown, with rewards for the 8 orientations in a radial pattern within each cell (position). Explored cells have low rewards, while unexplored cells and the discovered target are assigned high rewards. (third row) Predicted rewards averaged over the 8 orientations. (fourth row) Predicted values following the same convention. Values are higher facing the direction of unexplored cells and the target (if discovered). (fifth row) Predicted values averaged over the 8 orientations.	156
A.2	Example rollout of embodied VIN after 20 steps (left column), 40 steps (middle column) and 60 steps (right column). VIN kept oscillating between the same two states after 57 steps. The convention is the same as for Figure A.1, except that a single reward map is shared across all orientations. (first row) Input visualisation. (second row) Predicted rewards. (third row) Predicted rewards averaged over the 8 orientations. (fourth row) Predicted values.	157
A.3	Example rollout of embodied GPPN after 15 steps (left column), 30 steps (middle column) and 45 steps (right column). GPPN revisits the same sequences of states leading to a dead end after 45 steps. The convention is the same as for Figure A.1. (first row) Input visualisation. (second row) Predicted rewards. (third row) Predicted rewards averaged over the 8 orientations.	158
D.1	Transition matrix of IIC clusters. If there exists a transition from one cluster to the other, the corresponding cell in the adjacency matrix is coloured in white. (left) Transition within a single expert demonstration of the Obtain Diamond task, with 30 IIC clusters. (right) All transitions that appear in 122 expert demonstration trajectories of the Obtain Diamond task, with 128 IIC clusters.	184

D.2 Expert demonstration for the Obtain Diamond task. The RGB observations are shown at the top. Next, a coloured strip indicates an IIC cluster assignment for 128 clusters. The next row is a visual encoding of the action taken per frame (in the order of: “right”, “forward”, “left”, “back”, “jump”, “sprint”, “attack”, “sneak”). The following rows show the number of items in the inventory for every item relevant to the Obtain Diamond task. The amount of each item is shown in a progress bar style. A blue strip beneath If the item is being used with a “use” action. Finally, a coloured strip at the bottom indicates an IIC cluster assignment for 30 clusters. [Spans multiple pages] 185

List of Tables

3.1	Comparison of individual components in the implementation of CALVIN for positional states and for embodied pose states. X and Y are the size of the internal spatial discretisation of the environment, Θ is the internal discretisation of the orientation, A is the number of actions, K_x and K_y are the kernel dimensions for spatial locality, and M is the map of embeddings given as input with channel dimension C	48
3.2	Navigation success rate (fraction of trajectories that reach the target) on unseen 2D mazes. Partial observations (exploring an environment gradually) and embodied navigation (translation-rotation state space) are important yet challenging steps towards full 3D environments.	56
3.3	Navigation success rate of CALVIN in partially observable 2D mazes with loss components removed.	59
3.4	Navigation success rate on unseen 3D mazes (MiniWorld). Note that the baselines do not generalise to larger mazes.	61
3.5	Navigation success rate on unseen 3D mazes (MiniWorld), comparing the CNN backbone against the LPN backbone (also in Table 3.4). Most methods do not generalise to larger mazes. The proposed LPN demonstrates robust performance in larger unseen mazes.	63
3.6	Navigation success rate on AVD, with real robot images taken in indoor spaces. The task is to navigate to an object of a learned class. All methods use the proposed LPN backbone, as they fail without it.	65
4.1	Normalised performance comparison of Reinforcement Learning (RL) agents. The agent scores are the returns after the maximum environment steps during training ($100k$ for CartPole, $1M$ for LunarLander and MuJoCo environments, and $10M$ for Atari environments), normalised so that the score of a random agent is 0 and the score of the best performing model is 1. Scores are averaged per environment class (i.e. results for the corridor environments, Atari, and MuJoCo are grouped together) and the bold fonts show the best average normalised score per environment class, while the blue fonts show the best normalised score per environment.....	88
5.1	A breakdown of the number of routes per town (“num”), the average length of the routes per town (“avg. dist.”), and traffic density for the training routes (“density”), testing routes, and the Longest6 benchmark.	122
5.2	Driving performance of expert drivers in CARLA version 0.9.10. The driving score is a product of the route completion percentage \bar{R} and the infraction factor \bar{I} . IL and RL stand for Imitation Learning and Reinforcement Learning. DAGger uses both online and offline data.	123

6.1	Leaderboard: normalised TrueSkill scores according to [136]. The top three teams were GoUp, UniTeam, and Voggite (ours). Scores for BC-Baseline, two expert humans, and a random agent are also included. ...	141
B.1	The returns of RL agents after the maximum environment training steps (100 <i>k</i> for CartPole, 1 <i>M</i> for LunarLander and MuJoCo, and 10 <i>M</i> for Atari).	159

List of Abbreviations

A2C	Advantage Actor-Critic.
AI	Artificial Intelligence.
AVD	Active Vision Dataset.
AWM	Abstract World Model.
BC	Behavioural Cloning.
CALVIN	Collision Avoidance Long-term Value Iteration Network.
CMP	Cognitive Mapper and Planner.
CNN	Convolutional Neural Network.
DAC	Double Actor-Critic.
DAG	Directed Acyclic Graph.
DDPG	Deep Deterministic Policy Gradients.
DQN	Deep Q-Network.
EM	Expectation Maximisation.
GAE	Generalised Advantage Estimate.
GAN	Generative Adversarial Network.
GOA	Generalised Option Advantage.
GPPN	Gated Path Planning Network.
GPU	Graphical Processing Unit.
GRU	Gated Recurrent Unit.
HMM	Hidden Markov Model.
HRL	Hierarchical Reinforcement Learning.
IDM	Inverse Dynamics Model.
IIC	Invariant Information Clustering.

IL	Imitation Learning.
IRL	Inverse Reinforcement Learning.
KL	Kullback–Leibler.
LLM	Large Language Model.
LPN	Lattice PointNet.
LSTM	Long Short-Term Memory.
MCTS	Monte Carlo Tree Search.
MDP	Markov Decision Process.
Meta-RL	Meta Reinforcement Learning.
POMDP	Partially Observable Markov Decision Process.
PPG	Phasic Policy Gradient.
PPO	Proximal Policy Optimisation.
PPO-LSTM	Proximal Policy Optimisation with Long Short-Term Memory.
PPOC	Proximal Policy Option-Critic.
PPOEM	Proximal Policy Optimisation via Expectation Maximisation.
ReLU	Rectified Linear Unit.
RL	Reinforcement Learning.
SAC	Soft Actor-Critic.
Semi-MDP	Semi-Markov Decision Process.
SLAM	Simultaneous Localisation and Mapping.
SOAP	Sequential Option Advantage Propagation.
TD	Temporal Difference.
TD3	Twin-Delayed Deep Deterministic Policy Gradients.
TRPO	Trust Region Policy Optimisation.
UCB	Upper Confidence Bound.
VAE	Variational Auto-Encoder.

VI	Value Iteration.
VIN	Value Iteration Network.
VLM	Vision-Language Model.
VPT	Video PreTraining.

Chapter 1

Introduction

1.1 Motivation

The ability to plan, reason and forecast outcomes of actions, even in novel environments, are remarkable human capabilities that are instrumental in performing complex tasks with long-term objectives. Whenever we encounter a novel scenario, whether that be a new game, sport or location, even though we have never experienced that specific case, we can still strategise by extrapolating from our prior experiences, taking advantage of transferable knowledge and skills.

With modern planning algorithms, it is possible to find a near-optimal solution to a planning problem, if the environment dynamics (specifically the state transition and reward dynamics) are fully known, the states and actions can be enumerated, and unlimited compute is available. Unfortunately, it is often the case that all three of the assumptions do not hold. The agent often only has access to a local or partial observation of the environment, and has to estimate the underlying environment state and dynamics based on this. States and actions are often continuous rather than discrete, so an estimator is required that can map continuous inputs to meaningful representations that generalise to novel inputs. Finally, since compute is finite and enumeration of states and actions is often infeasible, an efficient strategy is necessary to explore the state-action space within limited computational resources and agent lifetime.

Many real-world problems involving strategic decision making require the agent to learn transferable knowledge of the environment that can be applied to novel scenarios with a limited budget of additional trial and error. Conceiving an algorithm that achieves the same level of performance and efficiency as humans in the open domain remains an open question. Autonomous driving [251], for instance, is still an ongoing and unsolved area of research, due to the high complexity of the dynamic environment in a multi-agent problem setting, together with the challenge of imperfect information and noisy sensor inputs. This is in stark contrast to industrial robots which have been in effective operation for many preceding decades, helped by the fact that the environment is controlled, predictable, and in many cases fully known. Combined with the repetitiveness of the task, this allows humans to hard-code the system to handle commonly anticipated scenarios.

Markov Decision Process (MDP) and Reinforcement Learning (RL) are powerful frameworks that formulate decision-making as a learnable problem with a mathematically defined objective [213]. These frameworks capture the sequential and time-evolving nature of interacting with an environment.

Advances in neural networks and their successful integration into RL [138, 139, 201] have transformed the field of computer vision and robotics, giving rise to learning-based approaches for problems traditionally solved by humans manually implementing expert systems. Learning-based approaches have two major advantages. Firstly, learning-based algorithms can keep improving and adapting to the application domain with more availability of data, whereas manually implemented methods are fixed and do not learn to adapt. Secondly, learning-based methods are capable of automatically discovering inherent regularities and characteristics of the application domain and exploiting them to improve their performances without having such strategies hardcoded.

While RL is highly effective in solving complex strategic problems [10, 12, 138, 202, 229], sample efficiency and generalisability are challenges that still need to be addressed.

Current state-of-the-art RL algorithms are highly performant in tasks they have been trained on or can solve with a reactive policy, but do not explicitly learn easily transferable skills [145, 162, 163, 174, 198]. Unlike games or tasks in a simulation where samples can be drawn easily, collecting samples can be expensive and possibly unsafe in real-world problems. Humans can work around these issues by learning transferable knowledge and skills that can be applied to novel situations, thereby increasing the chances of success with less trial-and-error and avoiding catastrophic failure, e.g. falling off a cliff or being run over by a car. This research aims to suggest ways of acquiring skills that allow agents to learn to perform tasks more efficiently and effectively.

1.2 Research objectives

This research addresses the challenge of solving tasks involving spatial reasoning, planning, and decision making in a data-driven manner, while simultaneously making the learning more efficient, interpretable and transferable. This research objective can be further broken down into five research goals, which are described in detail in the following.

1.2.1 Learn a generalisable planner

One of the core objectives of this research is to develop learnable planners that generalise to novel scenarios. A distinction between a *reactive* Markovian policy and a policy with a *plan* is that a reactive policy makes immediate decisions given a current state or local observation, whereas planning involves a more long-term analysis of the given situation to propose a spatially and temporally coherent solution.

The differences between the two approaches are analogous to the System 1 (fast, unconscious, and automatic decisions) and System 2 (slow, conscious, and rigorous decisions) thinking presented in [106]. Both decision processes are important, since reactive policies are useful for making many decisions in real-time, whereas planning is

important to ensure that the decisions made are consistent and coherent. For example, Monte Carlo Tree Search (MCTS)-based algorithms [201, 202] alternate between learning a reactive policy and using them for long-term planning; rollouts of a Monte Carlo tree [40] are simulated and the return estimates are back-propagated using a light-weight reactive policy, which is then updated according to the rollout results.

While the dynamics of games such as Go and simulated environments are known, this is not the case for many real-world problems. Model-based RL approaches [75, 79, 190] address this by learning models of the environment that could be used for simulated rollouts. Chapter 3 explores related alternative avenues to learn a differentiable planner that solves navigation tasks in novel environments that are not effectively solvable with reactive policies. Chapter 5 proposes a novel paradigm of learning algorithmic decision-making from data by treating code as learnable policies with the use of Large Language Models (LLMs). By making algorithms learnable, high-level and long-term plans that were hitherto too complex for RL agents to learn can now be learnt using Imitation Learning (IL) and RL techniques. In addition, Chapter 4 and Chapter 6 demonstrate how temporal abstraction using options [166, 214] can help agents make informed long-term decisions, discussed in Section 1.2.2 and Section 1.2.3.

1.2.2 Discover reusable skills

Skill learning is another important component for efficient exploration, decision making and task-solving. With skills, it is possible to conceive a high-level plan that combines and orchestrates low-level skill policies. These skills are specialised to solve a subset of the task so that the agent may learn to solve complex novel tasks from fewer training samples by composing these skills together. Ways in which these skills can be learnt in an unsupervised way, using rewards from the environment as a learning signal, are explored in Chapter 4. The agent trajectory is segmented into *options* [166, 214] that correspond to

skill-specific sub-policies.

1.2.3 Solve POMDP environments with memory-augmented policies

In relation to Section 1.2.2, options can be used not just to learn skills, but also to learn temporally consistent behaviour. It functions as a memory carried forward as a discrete latent variable, allowing the agent to perform tasks in a Partially Observable Markov Decision Process (POMDP) environment where the underlying state of the environment cannot be determined from current observations alone. The true environment state can be better determined by maintaining a history of the agent trajectory, since past observations are often correlated with future observations by hidden variables. Chapter 4 examines the effectiveness and robustness of options discovered by algorithms with different training objectives, demonstrating the advantage of the proposed solution over classical recurrent policies and Option-Critic policies [9, 111].

In Chapter 6, the concepts of skills and trajectory segmentation are employed to make the agent change its policy for different stages of task completion. Breaking a complex task down to subcomponents and performing them stage-wise allowed the agent to perform temporally consistent behaviour that adheres to a high-level plan.

1.2.4 Explain the behaviour of experts and agents

Another theme explored in this research is the explainability of the learnt policy. Skill learning discussed above is one approach that ensures better explainability, given the options segment the agent trajectory in a semantically interpretable way. Another approach to interpretability is explored in Chapter 3; a differentiable planner learns targets, obstacles, and motion dynamics from expert trajectories of robot navigation. It also computes a reward map and value map during its decision-making process, similarly to Inverse Reinforcement Learning (IRL) [6, 148, 260, 261]. An even more explicit way of representing

the policy as human-readable code is suggested in Chapter 5. Performance issues of the policies can be directly diagnosed by reading the code, making this approach a valuable technique in explainable Artificial Intelligence (AI) research.

1.2.5 Train embodied agents to perform complex tasks

Finally, the aim of this research is to apply developed techniques to problems relevant to embodied agents, e.g. robotics. In Chapter 3, Chapter 5 and Chapter 6, the challenges of robot navigation, autonomous driving and task execution in the virtual world of Minecraft [208] are addressed. These challenges all have navigation and spatial reasoning as key elements to accomplishing the tasks. Navigation is a real-world problem that has traditionally been solved by expert-designed systems, but could be made more efficient by leveraging data-driven learning. For instance, lane changing and cooperation with other vehicles are tasks for autonomous vehicles which require complex planning. The problem is made especially difficult, since human cooperative behaviour is difficult to model due to compounding factors and subtle cues, and there is not always a deterministic strategy to follow. Learning cooperative behaviour from real-world data could be beneficial to optimising these tasks.

1.3 Main contributions

The contributions in this thesis can be summarised as follows.

1. Developed a differentiable planner named Collision Avoidance Long-term Value Iteration Network (CALVIN), which learns to navigate unseen 3D environments by performing differentiable value iteration. State transitions and reward models are learnt from expert demonstrations, similarly to Value Iteration Network (VIN). However, VIN struggles to penalise invalid actions leading to collisions with ob-

stacles and walls, making the value estimate inaccurate. CALVIN resolves this issue by learning action affordance to constrain the agent transitions and rewards. CALVIN can navigate novel 2D and 3D environments and significantly outperforms other learnable planners based on VIN. Published at the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2022 [97]. Details are in Chapter 3.

2. Based on analysis of the Options Framework and the forward-backward algorithm [14], algorithms were developed to learn temporally-consistent options and associated sub-policies in order to solve POMDP tasks that require long-term memory. Two learning objectives for unsupervised option discovery were proposed and studied: Proximal Policy Optimisation via Expectation Maximisation (PPOEM) and Sequential Option Advantage Propagation (SOAP). PPOEM applies the forward-backward algorithm [14] to optimise the expected returns for an option-augmented policy. However, it was shown that this learning approach is unstable for learning causal policies without the knowledge of future trajectories, since option assignments are optimised for the entire episode. As an alternative approach, SOAP evaluates the policy gradient for an optimal option assignment. It extends the concept of the Generalised Advantage Estimate (GAE) to propagate *option advantages* through time, which is an analytical equivalent to performing temporal back-propagation of option policy gradients. With this approach, the option policy is only conditional on the history of the agent. Evaluated against competing baselines, SOAP exhibited the most robust performance, correctly discovering options for POMDP corridor environments, as well as on standard benchmarks including Atari [16] and MuJoCo [222]. The paper is available on *arXiv* [98]. Details are in Chapter 4.
3. Proposed LangProp. a framework for iteratively optimising code generated by LLMs.

LangProp automatically evaluates the code performance on a dataset of input-output pairs, catches any exceptions, and feeds the results back to the LLM in the training loop, so that the LLM can iteratively improve the code it generates. The LangProp training module can be used in both supervised and reinforcement learning settings. LangProp successfully solves Sudoku and CartPole, as well as generates driving code with comparable or superior performance to human-implemented expert systems in the CARLA driving benchmark [48]. LangProp can generate interpretable and transparent policies that can be verified and improved in a metric- and data-driven way. Accepted at the International Conference on Learning Representations (ICLR) 2024 Workshop on Large Language Model (LLM) Agents [100]. This work was conducted during an internship at Wayve Technologies. Details are in Chapter 5.

4. Developed Voggite, an embodied agent that performs tasks in Minecraft, an open-ended virtual world. As a backbone, Voggite uses OpenAI Video PreTraining (VPT) [12], a transformer-based agent pre-trained on online videos labelled by a supervised Inverse Dynamics Model (IDM). The VPT policy takes in 128 frames of past observations, equivalent to 6.4 s of history. While effective for many reactive tasks, the VPT agent struggles to disambiguate different stages of task execution. Voggite resolves this issue by dividing the task into separate stages. Voggite achieved 3rd place out of 63 teams in the MineRL BASALT Competition on Fine-Tuning from Human Feedback at NeurIPS 2022. In the competition, the agents are tasked to find caves and make waterfalls, farms and buildings in Minecraft. A co-authored retrospective of the competition is available on *arXiv* [136]. Details are in Chapter 6.

Work not included in this thesis: “You are what you eat? Feeding foundation models a regionally diverse food dataset of World Wide Dishes” [132].

1.4 Outline

Chapter 2 introduces background material and key concepts relevant to this thesis. Chapter 3 proposes CALVIN, a differentiable planner that learns to plan for long-term navigation in unseen 2D and 3D environments. Chapter 4 introduces PPOEM and SOAP for unsupervised option discovery. Chapter 5 proposes LangProp, a framework for iteratively optimising code generated by LLMs. Chapter 6 discusses data-driven decision-making for embodied agents with composite tasks, and develops Voggite, an embodied agent that performs tasks in Minecraft. Finally, Chapter 7 concludes this thesis with discussions of findings and future research directions.

Chapter 2

Background on planning and data-driven decision making

2.1 Planning algorithms

Planning concerns strategically inducing changes to an environment state with the means of actions to achieve a certain objective [116]. In robotics, motion and trajectory planning are essential to translating high-level specifications of tasks to low-level descriptions of motion. In the field of Artificial Intelligence (AI) and Reinforcement Learning (RL), early successes were achieved in puzzle solving and competing in games [24, 138, 201, 220].

While there are diverse ranges of planning problems, they share some commonalities. Firstly, they have a notion of a *state* s in *state space* \mathcal{S} that captures all details of a situation relevant to a particular problem. Secondly, the environment state can change over time, and the only way for the agent to influence its transition to another state is by taking an *action* $a \in \mathcal{A}(s)$. The transition from the current state s to the next state s' is conditional on a . The plan is expected to meet some criteria and/or optimise a certain objective. In robotics, it is typical to formulate this as a path-finding problem to a target that minimises the total cost, with additional constraints so that illegal configurations and obstacles are not encountered. In RL, the convention is to maximise the cumulative discounted rewards (returns), defined in Section 2.1.1.

2.1.1 Markov Decision Process

Markov Decision Process (MDP) [17, 213] is a standard formulation for sequential decision-making and planning. An MDP consists of states $s \in \mathcal{S}$, actions available at each state $a \in \mathcal{A}(s)$, a transition probability $P(s'|s, a)$ (the probability next state s' given the current state s and action a), and a reward function $R(s, a, s')$, which (either deterministically or stochastically) determines the reward r given to the agent during the transition.¹ An agent starts at state $s_0 \sim \rho(s)$ and at each time step t , takes an action $a_t \in \mathcal{A}(s_t)$, collects a reward r_t and moves to the next state s_{t+1} where $(r_t, s_{t+1}) \sim P(\cdot|s_t, a_t)$ until episode termination at timestep T when a boolean termination indicator d_t is set to 1.

In the RL paradigm, it is useful to think of the decision-making component (the agent) separately from the environment. The environment has an inner state which defines its configuration, but the agent often can only measure the state indirectly through observations given by the environment. The agent can induce changes to the state by taking actions at every time step, and it receives a reward and a new observation. The objective is to learn a policy $\pi(a|s)$ that chooses an action that maximises the expected return at every time step t . A return is a sum of discounted rewards, $\mathcal{R}_t(\tau) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$, where τ denotes the trajectory (a set of states, actions and rewards visited in an episode), and a discount factor $\gamma \in (0, 1]$ is applied to mitigate the infinite horizon problem encountered in continuous tasks, in which the sum of rewards can diverge to infinity.²

2.1.2 Partially Observable Markov Decision Process

Partially Observable Markov Decision Process (POMDP) is a special case of an MDP where the observation available to the agent only contains partial information of the

¹It is common to assume discrete planning time steps denoted with t (unitless integer), and denote the current state and next state as either s_t and s_{t+1} , or with a shorthand s and s' . The same convention applies to actions, rewards, etc. In this thesis, these two notations are used interchangeably.

²Some literature uses G_t to denote returns.

underlying state. In this thesis, s is used to denote the (partial) state given to the agent, which may or may not contain the full information of the environment (which shall be distinguished from state s as the underlying state \mathfrak{s}).³ This implies that the “state” transitions are no longer fully Markovian in a POMDP setting, and may be correlated with past observations and actions. Hence, $p(r_t, s_{t+1} | s_{0:t}, a_{0:t})$ describes the full state and reward dynamics in the case of POMDPs, where $s_{t_1:t_2}$ is a shorthand for $\{s_t | t_1 \leq t \leq t_2\}$, and similarly with $a_{t_1:t_2}$.

2.1.3 Classical approaches to planning

Before the rise of data-driven learning methods, the assumption in planning was that a model of the system or the environment is available. Planning in fully known state spaces with deterministic transitions has partially been solved by graph-search algorithms such as Dijkstra’s algorithm [46] and the A* algorithm [81], mainly in the domain of navigation [99]. However, many assumptions exist for such algorithms to work, such as that the environment is static, the state space is relatively low-dimensional, and the state space is enumerable. For instance, manipulation tasks are high dimensional and can be computationally expensive to solve with graph search algorithms. The D* algorithm [112, 206, 207] mitigated the static environment assumption by incrementally replanning when the agent’s knowledge of the environment is updated. Sampling methods such as Probabilistic Roadmaps (PRM) [109] and Rapidly-exploring Random Trees (RRT) [115, 117] presented practical solutions that can work in high-dimensional state spaces, which are asymptotically optimal. For problems such as Chess and Go where the state space is large and expensive to perform brute-force search, Monte Carlo Tree Search (MCTS) [40] is used to selectively explore states and evaluate the expected returns.

³In other literature, o is used to denote the partial observation to distinguish from the underlying state s . While this makes the distinction explicit, many works on standard RL algorithms assume a fully observable MDP for their formulation, leading to conflicting notations.

For many real-world applications, however, the environment model cannot be accessed directly, and has to be inferred by interacting with the environment and collecting experiences. Moreover, the state transition and reward dynamics can be probabilistic. The need to accommodate these requirements gave rise to RL.

2.2 Reinforcement Learning

RL is a framework that formalises sequential decision-making as cumulative future reward maximisation in an MDP [17, 213]. RL addresses problems that involve agents that interact with the environment to accomplish tasks when either the full MDP is unknown or it is not feasible for classical planning algorithms [116] to solve.

The objective of an agent is to learn a policy $\pi(a_t|s_t)$ that maximises the expectation of return \mathcal{R}_t , where the policy specifies the probability of choosing action a_t in state s_t . Approaches in RL can be broadly classified into policy gradient methods (Section 2.2.3) that directly optimises the policy based on on-policy feedback, value-based methods (Section 2.2.2) that learn a separate value function to estimate the expectation of returns (can be either on-policy or off-policy), and Actor-Critic methods (Section 2.2.4) that combines the advantages of policy learning and value learning. The distinction between on-policy and off-policy methods is explained in Section 2.2.1.

Advances in RL and their effective integration with neural networks as high-dimensional policy and value function approximators have transformed the fields of computer vision, robotics and games. Deep RL has outperformed humans in numerous strategic games [139, 201, 229].

2.2.1 On-policy and off-policy

An RL agent needs to interact with the environment and collect experiences to learn about the MDP (*exploration*), while also trying to optimise the return (*exploitation*). A recurring

theme in RL is to balance exploration and exploitation. This relates to an important distinction in RL algorithms: *on-policy* vs *off-policy*. An algorithm is on-policy if it can only be trained on experiences generated by the current policy π . While this often results in a simpler algorithm, this implies that the learnt policy has to retain a level of stochasticity, since it has to explore and experience sub-optimal behaviour as well. Off-policy learning, enabled by techniques such as importance sampling, allows decoupling of the target policy that is optimised from the behaviour policy which the agent uses to collect experiences. This also allows training upon all experiences that are collected in the past, rather than clearing the rollout buffer after every policy update.

2.2.2 Value-based methods

A value function [213] is a fundamental concept in RL, used to estimate the expected return following a policy π . The (state-)value function $V^\pi(s) \doteq \mathbb{E}_\pi[\mathcal{R}_t | s_t = s]$ evaluates the expected return starting from the current state s , while the action-value function $Q^\pi(s, a) \doteq \mathbb{E}_\pi[\mathcal{R}_t | s_t = s, a_t = a]$ considers both the current state s and the action to be taken a . An optimal policy π^* is a policy that maximises the expected return for all states, i.e. $\forall s \in \mathcal{S}, V^*(s) = \max_\pi V^\pi(s)$. Computing a value function for a given policy is called policy evaluation, and improving the policy to obtain a better return is called policy improvement.

Value Iteration

Value Iteration (VI) [17, 213] is an algorithm to obtain an optimal policy when the environment model (state transition $P(s'|s, a)$ and reward function $R(s, a, s')$) is known and the state-action space is relatively small and discrete so that they can be repeatedly enumerated (i.e. a tabular setting). It performs policy evaluation and policy improvement

for each update step. Updates at step k can be described as follows:

$$\begin{aligned}
 Q^{(k)}(s, a) &= \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V^{(k-1)}(s')], \quad \forall a \in \mathcal{A}(s), \\
 V^{(k)}(s) &= \max_{a \in \mathcal{A}(s)} Q^{(k)}(s, a), \\
 \pi^{(k)}(s) &= \operatorname{argmax}_{a \in \mathcal{A}(s)} Q^{(k)}(s, a).
 \end{aligned} \tag{2.1}$$

This update is repeated until the value function $V^{(k)}$ converges. The final policy π is optimal, which means that the policy gives the maximum expected return.

Value function approximation

While VI provides valuable insight into how value functions can be used to obtain an optimal policy, it is seldom the case that the full MDP is known and fully expandable. Value-based methods aim to approximate this value function from experiences without having to know or expand the full MDP.

Neural networks are often used as value function approximators. There are several approaches to computing the target value to regress towards. Monte-Carlo methods [204, 213] use the returns of fully rolled out episodes to estimate the expectation of returns (see Section 2.2.2), whereas Temporal Difference (TD) learning [184, 213, 235] is at the opposite end of the spectrum, taking a bootstrapped approach with a one-step lookahead to estimate the values (see Section 2.2.2). n -step TD [235] generalises TD learning to n -step lookahead (here, Monte Carlo methods can be considered as ∞ -step TD), and $\text{TD}(\lambda)$ [211] takes an exponentially weighted average of n -step TD with varying n (from 1-step TD to a Monte Carlo estimate) to provide a return estimate with reduced variance.

Monte Carlo methods

Monte Carlo methods [204, 213] use the returns of the episode $\mathcal{R}_t(\tau) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$ as a target value for a value function approximator. While Monte Carlo methods can

update all values predicted by the approximator at each time step along a trajectory with actual returns, the returns cannot be computed until the end of each episode, and have high variance due to different trajectories having varying returns as a result of aggregating all rewards along the trajectory, making it difficult to solve the problem of credit assignment (i.e. which specific actions contributed to improving/reducing the rewards).

Temporal Difference learning

TD learning performs a Bellman update by bootstrapping values of the next state. In the simplest one-step on-policy form, the update of a value function $V^\pi(s)$ can be written as:

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha [r + \gamma V(s') - V(s)], \quad (2.2)$$

where the reward-state transitions are sampled by the current policy π . Here, $V^\pi(s)$ is the value estimator to be learnt, α is the learning rate, and $r + \gamma V(s')$ is the target values to regress $V^\pi(s)$ towards. $\delta_t \doteq r_t + \gamma V(s_{t+1}) - V(s_t)$ is a quantity called TD error. TD learning can also be applied to incomplete episodes or infinite-horizon continual problems in which an episode may have infinite time steps.

TD learning can also be applied to learning action-values $Q(s, a)$. SARSA [184] is an on-policy method, for which the current state s , current action a , reward r , next state s' and next action a' must be known. The TD target is $r + \gamma Q(s', a')$. Q-learning [235] is an off-policy method, since it only requires s , a and r , and the TD target is $r + \gamma \max_{\bar{a}} Q(s', \bar{a})$. Deep Q-Network (DQN) [138] was a major breakthrough for deep reinforcement learning. Several improvements have been made to stabilise training. A target network [139] uses a lagged copy of the Q-network as the target so that the target is approximately fixed during training. Double Q-learning [83, 225] uses two Q-networks, one to select the actions and one to estimate the next step Q-value, to counter the problem of systematic overestimation of Q-values. Prioritised Experience Replay [188] sample informative experiences more frequently to accelerate learning.

2.2.3 Policy gradient methods

Policy gradient methods [210, 238, 239] directly learn and improve a policy $\pi(a|s)$ to maximise the expected returns. The key idea is to reinforce actions that resulted in high returns, while suppressing actions with low returns.

The objective $J(\pi_\theta)$ is the expected return over all completed trajectories generated by the agent following policy $\pi_\theta(a|s)$:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [\mathcal{R}(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \gamma^t r_t \right], \quad (2.3)$$

where θ is a set of learnable parameters. Neural networks are suitable to model $\pi_\theta(a|s)$.

The policy gradient algorithm maximises the objective $J(\pi_\theta)$ by performing gradient ascent on the policy parameters θ according to $\nabla_\theta J(\pi_\theta)$, which can be shown to be Equation (2.4) [213, 238]:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \mathcal{R}_t(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t) \right]. \quad (2.4)$$

The probability of the action $\pi_\theta(a_t|s_t)$ corresponding to a positive $\mathcal{R}_t(\tau)$ is increased, while that corresponding to a negative $\mathcal{R}_t(\tau)$ is decreased.

REINFORCE

The REINFORCE algorithm [238, 239] is a direct application of the policy gradient algorithm, which uses Monte-Carlo approximation for the expectation in Equation (2.4).

While value-based methods tend to struggle with continuous action spaces, since they have to find an argmax action over the values (see Equation (2.1)), policy gradient methods work with both discrete and continuous action spaces. The disadvantage is that they tend to have high variance because returns vary significantly from trajectory to trajectory, and credit assignment for long trajectories is non-trivial.⁴

⁴Credit assignment is the process of determining how individual actions contributed to the overall

2.2.4 Actor-Critic methods

Actor-Critic methods have two components that are learnt jointly: an *actor* – a parameterised policy used to take actions, and a *critic* – a value function used to estimate returns by bootstrapping.

Advantage

Advantage Actor-Critic (A2C) [140] combines policy gradient methods and value-based methods. It is similar to REINFORCE [238, 239], but instead of using the returns from Monte Carlo sampled experiences, it uses the advantage function $A(s_t, a_t) = \mathcal{R}_t - V(s_t)$ that measures the relative return of taking an action a_t in a given state s_t . It subtracts the baseline term $V(s)$ to stabilise the learning of the policy. In practice, the advantage function is trained by regressing towards the TD error $\delta_t = r + \gamma V^\pi(s') - V^\pi(s)$ as a reinforcing signal. The gradient of the A2C objective can be written as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s, a \sim \pi_\theta} [A(s, a) \nabla_\theta \log \pi_\theta(a|s)]. \quad (2.5)$$

Generalised Advantage Estimate

Generalised Advantage Estimate (GAE) [193] provides a robust and low-variance estimate of the advantage function. GAE can be expressed as a sum of exponentially weighted multi-step TD errors:

$$A_t^{\text{GAE}} = \sum_{t'=t}^T (\gamma \lambda)^{t'-t} \delta_{t'}, \quad (2.6)$$

where $\delta_t = r_t + \gamma(1-d_t)V(s_{t+1}) - V(s_t)$ is the TD error at time t , and λ is a hyperparameter that controls the trade-off of bias and variance. When $\lambda = 0$, the GAE is equivalent to the TD error δ_t . When $\lambda = 1$, it is equivalent to a Monte Carlo estimate of the advantage.

success or failure of an agent's strategy, particularly in problems with delayed rewards. Since REINFORCE uses the returns of full trajectories, actions that were irrelevant to the overall success or failure of the agent may still be reinforced or penalised.

Policy Optimisation

A direct implementation of policy gradient algorithms such as A2C is susceptible to policy collapses, in which the policy optimisation overshoots and the resulting new policy significantly deviates from the old policy. To counter this, Trust Region Policy Optimisation (TRPO) [192] updates policies while ensuring that the old and new policies are within a certain proximity, measured in terms of Kullback–Leibler (KL) divergence. Proximal Policy Optimisation (PPO) [194] addresses the same problem, using a first-order method where TRPO uses a more complex second-order method. Furthermore, PPO proposes to replace the KL divergence regularisation with a simpler clipped objective, which has a similar performance. PPO’s clipped objective is:

$$\mathcal{L}_{\text{PPO}}(\theta) = \mathbb{E}_{s, a \sim \pi} \left[\min \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A_t^{\text{GAE}}, \text{clip} \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A_t^{\text{GAE}} \right) \right], \quad (2.7)$$

where A_t^{GAE} is the GAE at time step t , and $\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ is a ratio of probabilities of taking action a at state s with the new policy against that with the old policy.

Q-learning-based Actor Critic

Deep Deterministic Policy Gradients (DDPG) [128, 200] adapts Deep Q-learning to work in continuous action spaces. Instead of evaluating the maximum Q-value over actions, which is a challenge in continuous action spaces, DDPG learns a target policy network $\mu_{\theta}(s)$ to deterministically predict an action that maximises $Q_{\phi}(s, a)$, and use its result to evaluate the TD-error. This is done by solving $\max_{\theta} \mathbb{E}[Q_{\phi}(s, \mu_{\theta}(s))]$.

Twin-Delayed DDPG (TD3) [63] improved upon DDPG by introducing delayed policy updates, policy smoothing, and clipped double-Q learning that learns two Q-value estimates and take their minimum to reduce the effect of over-estimation of the Q-values, while Soft Actor-Critic (SAC) [76] replaces DDPG’s deterministic policy with a stochastic policy and adds entropy regularisation to the RL objective, rewarding the agent for visiting

states where the agent’s policy has high entropy, since having high entropy may be an indication that the policy is uncertain for this state input and has room for improvement.

2.2.5 Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) [143, 160, 228, 242, 253] is a branch of RL that involves decomposing a complex long-horizon task into a hierarchy of subproblems or subtasks so that the lower-level policies can learn more primitive actions with a shorter-term focus, while the higher-level policies select temporally abstracted macro-actions with a longer-term focus. Where multiple tasks share many subtasks, an agent may learn a reusable lower-level policy with a higher-level policy learning to optimise the task-specific objective. This allows the agent to learn more efficiently and generalise better to novel situations.

In HRL, subgoals [65], credit assignment [227], and options [9, 111, 166, 214, 255] are key building blocks in achieving this decomposition. Subgoals are intermediate goals that the agent needs to achieve in order to complete the overall task. The agent uses subgoals to incentivise the agent to visit state regions which brings the agent closer to achieving its ultimate objective. Credit assignment is the process of assessing the level of each action’s contribution to the overall performance of the agent. HRL introduces temporal abstraction to task execution, which naturally allows long-term credit assignment (e.g. advantages computed for higher-level actions).

The Options Framework

Options [166, 214] are temporally extended actions that allow the agent to make high-level decisions in the environment. Each option corresponds to a specialised low-level policy that the agent can use to achieve a specific subtask. In the Options Framework, the inter-option policy $\pi(z_t|s_t)$ and an option termination probability $\varpi(s_{t+1}, z_t)$ govern the

transition of options, where z_t is the current option, and are chosen from an n number of discrete options $\{Z_1, \dots, Z_n\}$.⁵ Options are especially valuable when there are multiple stages in a task that must be taken sequentially (e.g. following a recipe) and the agent must obey different policies given similar observations, depending on the stage of the task.

Earlier works have built upon the Options Framework by either learning optimal option selection over a pre-determined set of options [161] or using heuristics for option segmentation [114, 143], rather than a fully end-to-end approach. While effective, such approaches constrain the agent’s ability to discover useful skills automatically.

The Option-Critic architecture [9] proposed end-to-end trainable systems which learn option assignment. It formulates the problem such that inter-option policies and termination conditions are learned jointly in the process of maximising the expected returns. Double Actor-Critic (DAC) [255] is an HRL approach to discovering options. DAC reformulates the Semi-Markov Decision Process (Semi-MDP) [214] of the option framework as two hierarchical MDPs (high-MDP and low-MDP). The low-MDP concerns the selection of primitive actions within the currently active option. The high-MDP handles high-level decision-making, such as selecting and terminating options.

Applications of the forward-backward algorithm

The forward-backward algorithm, also known as the Baum-Welch algorithm [14], is an algorithm that finds an optimal assignment of latent variables in a Hidden Markov Model (HMM) given sequential observations. Several works [42, 59, 64, 258] have explored the possibility of applying the forward-backward algorithm to the options framework in a TD learning setting.

⁵The notations have been adapted to match the convention in this thesis.

2.2.6 Transfer and generalisation in Reinforcement Learning

Many findings and techniques for standard deep learning in terms of transfer learning [217], multi-task learning [41], and Meta Learning [90] are applicable or translatable in the context of deep RL [244]. Policy and value networks, which often have shared parameters, can be pre-trained on general tasks before being fine-tuned on a more specific task [12] or by learning better representations from auxiliary tasks such as learning the environment dynamics and reward models [75, 79, 80, 144], sometimes referred to as world modelling (see Section 2.2.7). Recent approaches model the RL problem as a more general sequential modelling task, motivating applications of large transformer models [12, 32, 158, 175, 226]. This formulation allows foundation models [21] and large pre-trained models to be repurposed or fine-tuned on specific tasks [12, 49, 191, 248].

Meta Reinforcement Learning (Meta-RL) [15, 50, 165, 171, 231, 250] focuses on developing methods that enable agents to learn a general policy that can efficiently be adapted to solve new tasks with a small amount of data. The aim is to learn a shared representation across many different tasks in order to learn transferable skills. Large transformer policies are used in recent work such as AdA [165] so that the agent can infer and retain knowledge about the current task, which may be used to exhibit task-specific behaviour.

Similarly, Skill-based RL [145, 163, 198] aims to develop agents that can learn a set of reusable skills that can be composed to solve new tasks. These skills are designed to be generalisable and can be often learned from offline data [162, 174, 186]. This approach allows agents to solve new tasks with few-shot learning and adapt to changes in the environment more efficiently. These approaches are compatible with HRL, outlined in Section 2.2.5.

2.2.7 Model-based methods

Algorithms considered so far fall under the category of model-free methods, which do not require explicit modelling of the environment's state transition dynamics. In some problems, especially in the case of simulations, games and robotics, the transition dynamics may be readily accessible as system definitions, in which case it could be beneficial to take advantage of that knowledge. In other problems where the transition dynamics are not known, modelling this has its pros and cons. The advantage is that, once a good model is learnt, it can be used to forecast moves without having to act in the environment. This increases sample efficiency, which is highly beneficial when collecting experiences is costly. On the other hand, it is difficult to learn an accurate and comprehensive model, and long-term planning using inaccurate models can lead to compounding errors, making the problem intractable. Moerland et al. [141] performs a detailed analysis of model-based methods.

Monte Carlo Tree Search

MCTS [40] is a widely used model-based method for problems with deterministic discrete state spaces with known transition functions. It samples candidate actions from the current policy and performs Monte Carlo rollouts to explore states and estimate their values. MCTS was used as a core planning component in AlphaGo [201] to solve Go. It used neural networks to learn a policy to guide the search and a value function to estimate the values of leaf nodes. A small and fast policy network is used for rapid MCTS rollouts, while a large and more expressive network is used as the main strong policy network to be trained. In AlphaZero [202] that solved Go, Chess and Shogi, rollout simulations with a small network were replaced with a greedy strategy of expanding a child node with the highest Upper Confidence Bound (UCB) score.⁶

⁶In AlphaZero, a variant of PUCT [179] was used to evaluate the UCB score for Monte-Carlo Trees.

Efficient sampling from learnt models

It is possible to learn the transition and reward models $P(s'|s, a)$ and $R(s, a, s')$ from experiences, even when they are not readily available. The earliest of this approach has been demonstrated in Dyna [212], where samples are used both to update the policy function directly and to learn a transition model used to generate additional imagined samples. Model-based Value Expansion [56] performs imagined rollouts up to a fixed depth, and from there uses the value function. Model-Based Policy Optimisation (MBPO) [103] chooses rollout length based on estimated model generalisation capacity to attain the best of both worlds of model-free and model-based methods. Model-Based Reinforcement Learning via Meta-Policy Optimisation (MB-MPO) [37] meta-learns a policy that can quickly adapt to any model in the ensemble with one policy gradient step. This approach builds on the gradient-based meta-learning framework called Model-Agnostic Meta-Learning (MAML) [57].

Jointly learnt latent models

Simulated Policy Learning (SimPLe) [263] uses similar techniques to stochastic video prediction to learn a stochastic world model with discrete latent space, used as a simulator, and directly applies PPO [194] to acquire the policy.

TreeQN and ATreeC [54] incorporate recursive tree-structured neural networks. They learn a transition function that, given a state representation and an action, predicts the next state representation and corresponding rewards. This is applied recursively for all possible sequences of actions up to some predefined depth, at which a value function approximator is used to estimate values at each leaf node. Finally, $TD(\lambda)$ [211] is used to refine the estimate of $Q(s, a)$. While TreeQN is based on DQN [138], ATreeC is an actor-critic variant.

Imagination-Augmented Agent (I2A) [168] learns an environment model, a recurrent

architecture to predict next-step observations, which can be trained in an unsupervised fashion from agent trajectories. The environment model is used to roll out imagined trajectories, which are augmented with real samples. The policy network is trained with A3C [140]. World Models [75] similarly learn to generate states for simulation with a variational autoencoder and a generative recurrent network. MuZero [190] extends the work of AlphaZero [202] so that it can learn the transition models for Atari, Go, chess and shogi using one architecture, without any knowledge of the game dynamics.

Deep Planning Network (PlaNet) [78] learns the environment dynamics from images and chooses actions through fast online planning in latent space. It uses a Recurrent State Space Model (RSSM), a form of a sequential Variational Auto-Encoder (VAE) that relates transitions, observations and rewards. Model Predictive Control [176] is used to adapt its plan based on new observations, and Cross-Entropy-Method (CEM) [183] is used to search for the best action sequence. Unlike model-free and hybrid RL algorithms, no policy or value networks are used.

Dreamer [79] leverages the PlaNet world model, and allows the agent to imagine the outcomes of potential action sequences without executing them in the environment. Unlike PlaNet, it uses an actor-critic approach to learn a policy that predicts actions that result in state trajectories with high value estimates. This is similar to DDPG [128, 200] and SAC [76], but is applied to multi-step state transitions rather than maximising the immediate Q-values. DreamerV3 [80] improves the latent representation and reward/value activation function of Dreamer to solve a variety of tasks including finding diamonds in Minecraft, a complex task involving navigating a virtual terrain, gathering materials, and crafting increasingly more powerful tools for mining minerals (see Section 2.4.4).

With the rise of diffusion models [88, 108, 172, 178], recent works have shown that diffusion world models [3, 47] can also be used effectively to train RL agents.

2.3 Other learning methods

2.3.1 Imitation Learning

Imitation Learning (IL) learns a policy from expert trajectories rather than from trial and error. Perhaps the most intuitive and straightforward way of learning a policy, Behavioural Cloning (BC) is a simple form of IL that uses supervised learning to train an agent policy $\pi(a|s)$ given a dataset of state and action pairs as expert demonstrations. A policy trained only with BC is prone to failure, however, since compounding errors during rollout can take the agent to states that are not encountered during training. DAgger [180] is a method to overcome the limitation of BC by iteratively collecting expert labels for online rollouts and adding these samples to the training dataset, ensuring better coverage of the state space likely to be visited by the agent policy.

Inverse Reinforcement Learning (IRL) [148, 260, 261] is a class of methods that aims to recover an underlying reward function that best explains the expert trajectories. Once the reward function is estimated, standard RL techniques can be applied. This approach generalises better than BC in tasks in which inferring the reward mechanism is easier than learning the policy directly. However, IRL is an ill-posed problem since there are many possible reward functions that can explain the given trajectories. Hence, additional assumptions must be introduced to constrain the learning problem.

Generative Adversarial Imitation Learning (GAIL) [87] applies Generative Adversarial Networks (GANs) [66] to the problem of IL, training a policy to generate behaviour indistinguishable from expert demonstrations. This setup enables the model to learn complex behaviours.

Pre-training via imitation learning on demonstrations [12] is an effective strategy for learning representations that are transferrable to solving more general tasks (see Section 2.2.6). World Modelling (see Section 2.2.7) learns to reconstruct the observations

as well as predict the next action, and is shown to be an effective training strategy [91, 93].

2.3.2 Value Iteration Networks

Value Iteration Network (VIN) proposed by Tamar et al. [216] is a model-based IL method that aims to learn the entire planning procedure implicitly and in an end-to-end differentiable way. It generates a plan online by approximating VI (Section 2.2.2), and back-propagates errors through the plan to train estimators of the rewards. It models the underlying MDP with neural network parameters learnt with back-propagation by matching the evaluated policy with expert trajectories. In the case of a gridded state space (e.g. a discretisation of a robot's position in space), the VIN takes the form of a recurrent convolutional network with cross-channel max-pooling. The convolutional kernels approximate transition and reward models that are used to generate Q-values online. The Convolutional Neural Network (CNN) [119] representing the MDP can leverage repeating structural patterns in the state space's connectivity. This enables the VIN to predict an appropriate MDP and plan without further training in novel environments for grid-based state spaces, such as those used in robot navigation, given that the observation characteristics and the local transition dynamics are similar to the training domain.

2.3.3 Fine-tuning and Curriculum Learning

RL relies on trial and error to uncover the underlying MDP which is often sample-inefficient, and involves risk in real-world scenarios where a wrong decision can potentially be catastrophic. On the other hand, IL is limited by the coverage of states explored in expert demonstrations for training. Hence, a policy is often pre-trained using IL and fine-tuned using RL.

Curriculum Learning [146] is another training technique to improve sample-efficiency. It trains the agent in stages, starting with simpler tasks and progressively introducing harder

tasks. This process is designed to help the agent acquire basic skills early on in the training, which can later be applied to solve complex problems.

2.4 Embodied agents

While embodied tasks such as navigation and manipulation can be solved with classical path planning methods (Section 2.1.3) for a fully known environment and known motion dynamics of the agent, in many embodied agent problems the agent must learn to solve tasks without having access to such privileged information. Simultaneous Localisation and Mapping (SLAM) [62, 142] can be used to construct a map of the environment online from the agent observations if the camera parameters are known and the environment is mostly static and is embedded in Euclidean geometry. However, many real-world problems have (a) dynamic components, (b) non-Euclidean environment dynamics, and (c) unknown camera and/or agent dynamics. For instance, a coordinate space in motion, such as a train or an aeroplane, is locally Euclidean, but the entrance/exit may lead to a different global location because there is a temporal component to the planning problem. In these examples, constructing a globally consistent map of the environment from the agent observations alone is infeasible and unscalable. Humans can work with such dynamic environments from partial observations to solve a variety of real-world tasks with various characteristics. In contrast, algorithmically solving embodied task execution over a large spatial and temporal horizon is still an unsolved problem. This motivates research on learning-based techniques for embodied navigation, planning and task execution.

2.4.1 Visual navigation

Visual navigation is a particular application of data-driven planning, in which the observation data is an image stream captured by a mobile robot. Observations retrieved from 3D environments have spatial consistency, and state transitions are often translation-invariant

in the case of spatial states. This opens up an opportunity for data-driven planning algorithms to take advantage of these regularities to make forecasts during planning, thereby increasing their performance and sample efficiency.

Several works have made advances in training deep networks for navigation. Parisotto et al. [157] developed the Neural Map, an A2C [140] agent that reads and writes to a differentiable memory (i.e. a map). Similarly, Mirowski et al. [137] trained reactive policies which are specific to an environment. The Value Prediction Network [153] learns an MDP and its state space from data, and plans with a single roll-out over a short horizon. Hausknecht et al. [84] added recurrence to deep Q-learning to address partially-observable environments, but considered only single-frame occlusions. Several works treated planning as a non-differentiable module, and focused on training neural networks for other aspects of the navigation system. For instance, Savinov et al. [187] composed siamese networks and value estimators trained on proxy tasks, and using an initial map built from footage of a walk-through of the environment. Active Neural SLAM [29] trained a localisation and mapping component (outputting free space and obstacles), a policy network to select a target for the non-differentiable planner, and another to perform low-level control. Subsequent work [28] complemented this map with semantic segmentation. The latter two works navigated successfully in large simulations of 3D-scanned environments, and were transferred to real robots.

AI2-THOR [113], VirtualHome [167], Matterport3D [27], Gibson [247], iGibson [123], Habitat [134, 215], and RoboTHOR [44] are some of the well-known benchmarks for embodied navigation in naturalistic simulated environments.

2.4.2 Decision making in autonomous driving

Approaches to autonomous driving can be broadly classified into modular systems and end-to-end systems [251]. Most systems take a modular approach [122, 131, 224, 237],

which has human-defined rules that orchestrate separately engineered components for localisation and mapping, object detection, tracking, behaviour prediction, planning, and vehicle control. Such systems allow compartmentalisation and better interpretability, but can be complex and require domain knowledge to maintain and update. Another challenge is error propagation [135], i.e. the upstream outputs can be erroneous and must be corrected downstream. Recent work has harnessed end-to-end learning to address these issues. IL [13, 20] optimises the policy to match actions taken by experts, and is the most widely used approach. However, its performance is upper-bounded by the expert. Deep RL has also shown successes in simulation [185], on the road [110], and in combination with IL [130].

CARLA driving benchmark

CARLA [48] is a widely used open-sourced 3D simulator for autonomous driving research, and provides a benchmark [25] to evaluate driving performances of both privileged expert agents and sensor-only agents. Privileged agents have access to traffic light states and ground truth motions of vehicles, bicycles, and pedestrians. These are often implemented manually and are used to generate expert demonstrations for IL. Sensor-only agents, on the other hand, have to make decisions based on RGB-D images, lidar, and accelerometer measurements, which are often achieved by learnable components. Many prior works on CARLA have open-sourced their expert agents. Roach [257] trained a PPO agent [194] on handcrafted reward signals with privileged information. The heavy lifting is done at the reward shaping level, where hazardous agents are identified and the desired speed and pose are computed. Roach expert is also used in MILE [92] and TCP [241], where TCP has an additional emergency braking upon detecting potential collisions. TransFuser [35], InterFuser [196] and TF++ [102] implement their handcrafted expert systems, either using cuboid intersections or line intersections for hazard detection. TransFuser also introduced the Longest6 benchmark [35], which consists of longer routes compared to the official

CARLA benchmark and is less saturated.

2.4.3 LLMs for automating compositional tasks

Large Language Model (LLM) [23, 33, 155, 156]-powered agents have demonstrated sophisticated decision making and planning capabilities. Sequential prompting with the history of observation, action, and the reason for the action was proposed by ReAct [249] as an improvement to Chain-of-Thought prompting [236], which has also been applied to autonomous driving [60]. Auto-GPT [177] automated tasks by iteratively generating a sequence of subtasks in finer detail until they are executable. A similar strategy was applied to robotics [94]. SayCan [96] used LLMs to generate candidate subgoals and assessed their affordances with a value function given visual observations to ground the agent's behaviour. Socratic Models [252] demonstrated zero-shot performance on language-conditioned robot manipulation tasks by orchestrating pre-trained multimodal foundation models. Reed et al. developed Gato [175], a multi-task agent with a multi-modal foundation model backbone that can perform a variety of tasks including image captioning, chat, playing Atari and real-world robot arm manipulation. VIMA [105] and PaLM-E [49] demonstrated profound reasoning and execution capabilities on multi-modal tasks such as Visual Q&A and robotics by fine-tuning LLMs to allow multi-modal prompting. Inner Monologue [95] used environment and user feedback to replan for embodied tasks. Liang et al. [126] and Singh et al. [203] used LLMs to directly generate code for robotics, while ViperGPT [209] and VisProg [73] composed pre-trained vision-and-language models to solve challenging vision tasks which require reasoning and domain knowledge.

2.4.4 Task execution in virtual worlds

Significant innovations have been made in recent decades on research in embodied agents that can interact with video games and complex simulated 3D environments, including

Atari [16, 139], Dota 2 [18], StarCraft II [229], and Minecraft [12, 74]. More recently, SIMA [219] trained an agent to follow free-form instructions across a diverse range of virtual 3D environments, including curated research environments as well as open-ended commercial video games. In this thesis, the Minecraft environment is explored in depth.

Minecraft is an open-ended 3D virtual gaming environment where human players and agents can perform many life-like tasks to survive and live, such as exploring, mining, crafting, building, raising animals, and harvesting crops. MineRL [74] provided a behaviour dataset to train embodied agents, and hosted a competition for RL agents to obtain diamonds in Minecraft. This is a complex task, since it involves many stages of subtasks of gathering wood, cobblestones and iron, and crafting a crafting table, furnace, and pickaxes of different materials (i.e. wood, stone and iron).

OpenAI presented Video PreTraining (VPT) [12], a foundation model trained on a large collection of videos on the Internet, which successfully managed to discover diamonds using RGB pixel inputs as observations and predicting cursor movements, mouse clicks, and keyboard presses as outputs. DreamerV3 [80] achieved the same task but trained just on its own experiences without external datasets, by learning a world model to improve sample efficiency.

Many of the tasks in Minecraft are compositional, and hence require long-term planning, guided by intuition from real-world experiences. The MineRL BASALT Competition [136] evaluated agents on their life-like task execution capabilities in four domains: finding a cave, making a waterfall, creating an animal pen, and building a house.

Recent works on Minecraft explore ways to use LLMs and multi-modal foundation models to interface the agent with language instructions and address such long-term planning problems. MineDojo [53] provides an Internet-scale multimodal knowledge base of images, videos and text, accompanied by a benchmarking suite with thousands of diverse open-ended tasks specified in natural language prompts. The paper also proposes

MineCLIP, a video-text contrastive model based on CLIP [169], which associates natural language subtitles with their time-aligned video segments and uses the correlation score as an open-vocabulary reward function for RL training. Steve-1 [127] builds on MineCLIP, combining with methods from unCLIP [173] to generate language-guided behaviours.

VOYAGER [230] takes a unique approach to generating code as directly executable policies. Assuming privileged information about the environment (i.e. types and coordinates of objects and materials nearby, the items in the inventory, and access to a path planner given a target coordinate), it integrates environment feedback, execution errors, and self-verification into an iterative prompting mechanism for embodied control in Minecraft. VOYAGER maintains a *skill library*, a collection of verified reusable code, which can be considered as *checkpoints* that can be saved, shared, loaded to the agent to determine its strategy, and used as a for further training.

Approaches that use LLM to generate plans and subgoals are also suggested. DECKARD [152] learns modular sub-policies trained on RL objectives to work with a high-level plan generated by the LLM, referred to as an Abstract World Model (AWM). The AWM is a Directed Acyclic Graph (DAG), where the nodes represent subgoals (e.g. crafting specific items in Minecraft), and the edges represent dependencies between these subgoals. Subgoal policies predict actions over a large multi-discrete action space from a pixel-only observation, while the overall policy initiates transitions between subgoals based on the agent’s current inventory. “Describe, Explain, Plan and Select” (DEPS) [233] uses a LLM to interactively adjust the agent plan upon failure to complete a subgoal. A Vision-Language Model (VLM) (e.g. CLIP [169]) is used to provide the agent’s state descriptions in natural language. Zhu et al. [259] similarly implements an LLM Decomposer, LLM Planner, and LLM Interface to perform interactive planning. Their method manages to obtain all items in Minecraft Overworld, although it also assumes a more privileged observation space including voxels, GPS location, and inventory information.

JARVIS-1 [234], a multi-modal agent that takes visual observations and human instructions, combines MineCLIP [53] with an LLM, and further augments the agent with a multimodal memory, which stores successful plans in the past with corresponding scenarios to enhance the correctness and consistency of planning. JARVIS-1 employs self-instruct [232] to generate a dynamic curriculum for the exploration of diverse tasks.

Chapter 3

Towards real-world navigation with deep differentiable planners

This chapter introduces Collision Avoidance Long-term Value Iteration Network (CALVIN), an embodied neural network trained to plan and navigate unseen complex 2D and 3D environments. Rather than requiring prior knowledge of the agent or environment, the planner learns to model the state transitions and rewards. To avoid the potentially hazardous trial-and-error of Reinforcement Learning (RL), this work focuses on differentiable planners from the family of Value Iteration Networks (VINs) [216], which are trained offline from safe expert demonstrations. Although they work well in small simulations, two major limitations hinder their deployment. Firstly, current differentiable planners struggle to plan long-term in environments with a high branching complexity. While they should ideally learn to assign low rewards to obstacles to avoid collisions, these penalties are not strong enough to guarantee collision-free operation. CALVIN thus imposes a structural constraint on the value iteration, which explicitly learns to model impossible actions and noisy motion. Secondly, CALVIN extends the model to plan exploration with a limited perspective camera under translation and fine rotations, which is crucial for real robot deployment. These proposed modifications significantly improve semantic navigation and exploration on several 2D and 3D environments, including the Active Vision Dataset (AVD) that consists of real images captured from a robot, succeeding in settings that are

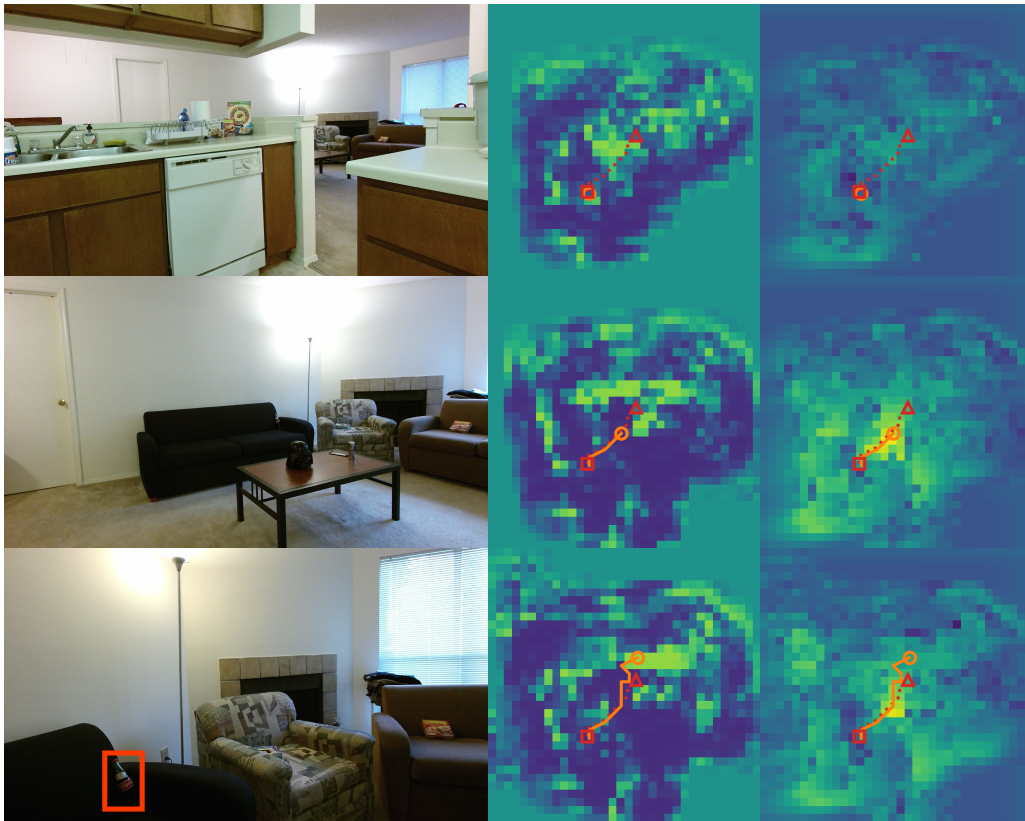


Figure 3.1: **(1st column)** Input images seen during a run of CALVIN on AVD (Section 3.5.5). This embodied neural network has learned to efficiently explore and navigate unseen indoor environments, to seek objects of a given class (highlighted in the last image). **(2nd-3rd columns)** Predicted rewards and values (resp.), for each spatial location (higher for brighter values). The unknown optimal trajectory is dashed, while the robot’s trajectory is solid.

otherwise challenging for differentiable planners.

This work was published at the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2022 [97].

3.1 Introduction

Continuous advances in robotics have enabled robots to be deployed to a wide range of scenarios, from manufacturing in factories and cleaning in households, to the emerging applications of autonomous vehicles and delivery drones [5]. Improving their autonomy is met with many challenges, due to the difficulty of planning from uncertain sensory

data. In classical robotics, the study of planning has a long tradition [116, 221], using detailed knowledge of a robot’s configuration and sensors, with little emphasis on learning from data. An orthogonal approach is to use deep learning, an intensely data-driven approach [67]. Modern deep neural networks excel at pattern recognition [197], although they do not offer a direct path to planning applications. While one approach would be to parse a scene into pre-defined elements (e.g. object classes and their poses) to be passed to a more classical planner, an end-to-end approach where all modules are learnable has the chance to improve with data and adapted to novel settings without manual tuning. Because of the data-driven setup, a deep network has the potential to learn behaviour that leverages the biases of the environment, such as likely locations for certain types of rooms. For example, while the best generic strategy to reach the exit in a maze may be to follow a wall on one side, a better one for office buildings could be to first exit any room and then follow the corridors to the end [5].

VINs [216] emerged as an elegant way to merge classical planning and data-driven deep networks, by defining a *differentiable* planner. The end-to-end differentiability of the network allows the planner to include sub-components learnable from demonstrations. For example, it can learn to identify and avoid obstacles, and to recognise and seek classes of target objects, without explicitly labelled examples. However, there are gaps between VIN’s idealised formulation and realistic robotics scenarios. The Convolutional Neural Network (CNN)-based VIN considers the full environment to be visible and expressible as a 2D grid. As such, it does not account for embodied (first-person) observations in 3D spaces, unexplored and partially visible environments, or the mismatch between egocentric observations and idealised world-space discrete states. It is also observed that VINs are less robust in complex environments where the degree of branching is high, e.g. in a maze as opposed to an obstacle map used in [216].

This work addresses these challenges, and closes the gap between current differen-

differentiable planners and realistic robot navigation applications. The main contributions of this work are:

1. A constrained transition model for Value Iteration (VI), following a rigorous probabilistic formulation, which explicitly models illegal actions and task termination (Section 3.3.1).
2. A 3D state space for embodied planning through the robot’s translation and rotation (Section 3.3.2). Planning through *fine-grained* rotations is often overlooked (Section 3.2.3), requiring better priors for transition modelling (Section 3.3.1).
3. A trajectory reweighting scheme that addresses the unbalanced nature of navigation training distributions (Section 3.3.1).
4. An efficient representation that fuses observations of the same spatial location across time and space (different points-of-view), allowing differentiable planners to handle long time horizons and free-form trajectories (Section 3.3.2).
5. Training the differentiable planner to learn to navigate in both complex 3D mazes, and the challenging AVD [4], with images from a real robot platform (Section 3.5.5), showing that the planner can be trained with limited data collected offline.

3.2 Background

3.2.1 Value Iteration

VI [17, 213] is an algorithm to obtain an optimal policy, by alternating a refinement of both value (V) and action-value (Q) function estimates in each iteration k . The algorithm and the update rule are described in Section 2.2.2.

When s and a are discrete, $Q^{(k)}$ and $V^{(k)}$ can be implemented as simple tables (tensors). The cells of a 2D grid are considered as states, corresponding to discretised

locations in an environment, i.e. $s = (i, j) \in \mathcal{S} = \{1, \dots, X\} \times \{1, \dots, Y\}$. Furthermore, transitions are local (only to coordinates offset by $\delta \in \{-1, 0, 1\}^2$):

$$Q^{(k)}(s, a) = \sum_{\delta \in K} P(s + \delta | s, a) [R(s, a, \delta) + \gamma V^{(k-1)}(s + \delta)], \quad \forall a \in \mathcal{A}(s),$$

$$V^{(k)}(s) = \max_{a \in \mathcal{A}(s)} Q^{(k)}(s, a),$$
(3.1)

Note that to avoid repetitive notation s and δ are 2D indices, and V is a 2D matrix. The policy $\pi^{(k)}(a|s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^{(k)}(s, a)$ chooses the action with the highest action-value. While the sum in Equation (3.1) resembles a convolution, the filters (P) are space-varying (depend on $s = (i, j)$), so it is not directly expressible as such. Equation (3.1) represents the “ideal” VI for local motion on a 2D grid, without further assumptions. For this reason, P and R are unconstrained 5-dimensional tensors. Contrasting Equation (3.1) to any proposed modification will be instructive, by revealing explicitly any additional assumptions.

3.2.2 Value Iteration Network

While VI guarantees the optimal policy, it requires that the functions for transition probability P and reward R are known. Tamar et al. [216] pointed out that all VI operations are (sub-)differentiable, and as such a model of P and R can be *trained from data* by back-propagation. For the case of planning on a 2D grid (navigation), they related Equation (3.1) to a CNN, as:

$$Q_{s,a}^{(k)} = \sum_{\delta \in K} \left(P_{a,\delta}^R \widehat{R}_{s+\delta} + P_{a,\delta}^V V_{s+\delta}^{(k-1)} \right), \quad \forall a \in \mathcal{A},$$

$$V_s^{(k)} = \max_{a \in \mathcal{A}} Q_{s,a}^{(k)},$$
(3.2)

where $P^R, P^V \in \mathbb{R}^{A \times |K|}$ are two learned convolutional filters that represent the transitions (P in Equation (3.1)), and \widehat{R} is a predicted 2D reward map. Note that \mathcal{A} is independent of s , i.e. all actions are allowed in all states. This turns out to be problematic (see

Section 3.2.4). The reward map \hat{R} is predicted by a CNN, from an input of the same size that represents the available observations. In Tamar et al.’s experiments [216], the observations were a fully visible top-down view map of the environment, from which negative rewards such as obstacles and positive rewards such as navigation targets can be located. Each action channel in \mathcal{A} corresponds to a move in the 2D grid, typically 8-directional or 4-directional. Equation (3.2) is attractive because it can be implemented as a CNN consisting of alternating convolutional layers (Q) and max-pooling along the actions (channels) dimension (V).

3.2.3 Applications of Value Iteration Networks

VINs were applied to localisation from partial observations by Karkus et al. [107], but assuming a full map of the environment. Gated Path Planning Network (GPPN) [120] replaced the maximum over actions in the VI formula with a Long Short-Term Memory (LSTM). The evaluation included an extension to rotation, but assumed a fully known four-directional view for every gridded state, which is often not available in practice. A subtle difference is that they handle rotation by applying a linear policy layer to the hidden channels of a 2D state space grid, which is less interpretable than the 3D translation-rotation grid implemented in CALVIN

Most previous work on deploying VIN to robots [150, 189] assumed an occupancy map and goal map rather than learning the map embedding and goal themselves from data. Gupta et al. [72] evaluated a differentiable planner called Cognitive Mapper and Planner (CMP) on real-world data using map embeddings learned end-to-end. CMP uses a hierarchical VIN [216] to plan on larger environments, and updates an egocentric map. It handles rotation as a warping operation external to the VIN (i.e. it plans in a 2D translation state space, not 3D translation-rotation space). Warping an egocentric map per update achieves scalability at the cost of progressive blurring of the embeddings. CMP requires

gathering new trajectories online during training with DAgger [180]. In contrast, CALVIN is trained solely on a fixed set of trajectories, foregoing the need for extra expert labels.

3.2.4 Limitations of Value Iteration Networks

Differences between the VIN and the idealised VI on a grid.

There are several differences between the VIN (Equation (3.2)) and an idealised VI on a grid (Equation (3.1)):

1. The VIN value estimate V takes the maximum action-value Q across *all possible actions* \mathcal{A} , even illegal ones (e.g. moving into an obstacle). Similarly, the Q estimate is also updated even for illegal actions. In contrast, VI only considers legal actions for each state (cell), i.e. $\mathcal{A}(i, j)$.
2. The VIN reward \hat{R} is assumed independent of the action. This means that, for example, a transition between two states cannot be penalised directly; a penalty must be assigned to one of the states (regardless of the action taken to enter it).
3. The VIN transition probability is expanded into 2 terms, P^R which affects the reward and P^V which affects the estimated values. This decoupling means that they do not enjoy the physical interpretability of VI's P (i.e. probability of state transitions), and rewards and values can undergo very different transition dynamics.
4. Unlike the VI, the VIN considers the state transition translation-invariant. This means that it cannot model obstacles (illegal transitions) using P , and must rely on assigning a high penalty to the reward \hat{R} of those states instead.

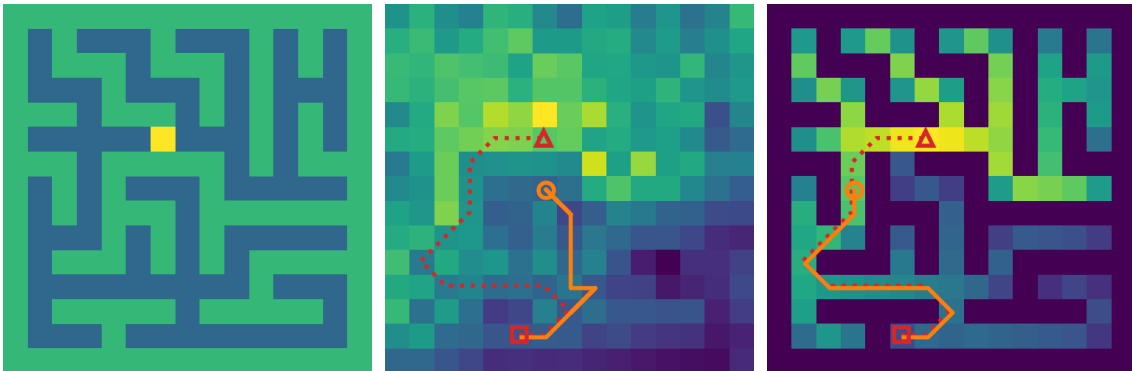


Figure 3.2: **(left)** A 2D maze, with the target in yellow. **(middle)** Values produced by the VIN for each 2D state (actions are taken towards the highest value). Higher values are brighter. The correct trajectory is dashed, and the current one is solid. The agent (orange circle) is stuck due to the local maximum below it. **(right)** Same values for CALVIN. There are no spurious maxima, and the values of walls are correctly considered low (dark).

Motivating experiment

Since the VIN allows all actions at all states (\mathcal{A} does not depend on s), collisions must be modelled as states with low rewards. In practice, the VIN does not learn to forbid such actions completely, resulting in the propagation of values from states which cannot be reached directly due to collision along the way. This is verified experimentally by training a VIN according to Tamar et al. [216] on 4K mazes (see Section 3.5 for details). For each state, predicted scores are measured to identify whether the scores for all valid actions are larger than those for all invalid actions (i.e. collisions). Intuitively, this means the network always prefers free space over collisions. Surprisingly, it turned out that this was not true for 24.6% of the states. For a real-world robot to work reliably, this is an unacceptably high chance of collisions. As a comparison, for the proposed method CALVIN, this rate is only 1.6%. In the same experiment, the VIN was often trapped in local minima of the value function and failed to move (Figure 3.2), which is another failure mode. CALVIN addresses these issues, and push VINs towards realistic scenarios. An alternative would be to employ online retraining (Dagger) [180], but this trades off the benefit of training solely on a fixed set of offline trajectories.

3.3 Collision Avoidance Long-term Value Iteration

Network

CALVIN implements a transition model that accounts for illegal actions and termination. The state space is extended to rotations and incorporates distant observations into a spatial map of embeddings, both necessary for real 3D environments.

3.3.1 Augmented navigation state-action space

A probabilistic transition model is derived in this section from first principles, with only two assumptions and no extra hyper-parameters. The first assumption is locality and translation invariance of the agent motion, which was introduced in the VIN to allow efficient learning with shared parameters. Unlike with the VIN, the transition model $P(s'|s, a)$ is decomposed into two components: the agent motion model $\hat{P}(s' - s|a)$, which is translation invariant and shared across states (depending only on the spatial difference between states $s' - s$); and an observation-dependent predictor $\hat{A}(s, a) \in [0, 1]$ that evaluates whether action a is available from state s to disqualify illegal actions.

In robotics, it is essential that the agent understands that the current task has been completed to move on to the next one. Since there is a high chance that a randomly-acting agent will stumble upon the target in small environments, Anderson et al. [5] suggested that an explicit termination action must be taken at the target to finish successfully. Therefore, in addition to positional states, success and failure states are defined, where a success state W (“win”) is reached only by triggering a termination action D (“done”), and a failure state F (“fail”) is reached upon triggering an incorrect action. The reward for reaching F is denoted as R_F , and the translation-invariant rewards is denoted as $\hat{R}(a, s' - s)$. For simplicity, the reward for success is considered as a special case of $\hat{R}(a, s' - s)$ where

$a = D$ and $s' = W$. With these assumptions, the reward function $R(s, a, s')$ is:

$$R(s, a, s') = \begin{cases} \widehat{R}_F, & s' = F, \\ \widehat{R}(a, s' - s), & s' \neq F. \end{cases} \quad (3.3)$$

Together with the agent motion model $\widehat{P}(s' - s|a)$, action validity predictor $\widehat{A}(s, a)$ and the definition of a failure state F , the transition model $P(s'|s, a)$ can be derived as

$$P(s'|s, a) = \begin{cases} 1 - \widehat{A}(s, a), & s' = F, \\ \widehat{A}(s, a)\widehat{P}(s' - s|a), & s' \neq F. \end{cases} \quad (3.4)$$

From the above, the reward function $R(s, a)$ is evaluated by marginalising over the neighbour states s' :

$$\begin{aligned} R(s, a) &= \sum_{s'} P(s'|s, a)R(s, a, s') \\ &= \widehat{R}_F(1 - \widehat{A}(s, a)) + \widehat{A}(s, a) \sum_{s'} \widehat{P}(s' - s|a)\widehat{R}(a, s' - s). \end{aligned} \quad (3.5)$$

Finally, Equation (3.5) can be substituted into Equation (3.1) to obtain a proposed method for evaluating $Q(s, a)$:

$$\begin{aligned} Q(s, a) &= R(s, a) + \gamma \mathbb{I}_{a \in \mathcal{A}} \sum_{s'} P(s'|s, a)V(s') \\ &= R(s, a) + \gamma \widehat{A}(s, a) \mathbb{I}_{a \neq D} \sum_{s'} \widehat{P}(s' - s|a)V(s'), \end{aligned} \quad (3.6)$$

where \mathbb{I} is an indicator function. Equations (3.5) and (3.6) essentially express a constrained VI, which models the case of a Markov Decision Process (MDP) on a grid with unknown illegal states and termination at a goal state. The inputs to this model are three learnable functions — the motion model $\widehat{P}(s' - s|a)$, the action validity $\widehat{A}(s, a)$ (i.e. obstacle predictions), and the rewards $\widehat{R}(a, s' - s)$ and \widehat{R}_F . These are implemented as CNNs with the observations as inputs (\widehat{R}_F is a single learned scalar). All the constraints follow from a well-defined world model, with interpretable predicted quantities, unlike previous proposals [120, 216]. The resulting method is named Collision Avoidance Long-term Value Iteration Network (CALVIN).

Training

Similarly to Tamar et al. [216], the model is trained with a softmax cross-entropy loss L , comparing an example trajectory $\{(s_t, a_t^*) : t \in T\}$ with predicted action scores $Q(s, a)$:

$$\min_{\hat{P}, \hat{A}, \hat{R}} \frac{1}{|T|} \sum_{t \in T} w_t L(Q(s_t), a_t^*), \quad (3.7)$$

where $Q(s)$ is a vector with one element per action ($Q(s, a)$), and w_t is an optional weight that can be used to bias the loss if $w_t \neq 1$ (Section 3.3.1). Example trajectories are the shortest paths computed from random starting points to the target (also chosen randomly). Note that these shortest paths are computable only during training time, since it is assumed that the ground-truth layout of the map and the obstacle/target positions are not readily available during test time.

The learned functions can be conditioned on input observations. These are 2D grids of features, with the same size as the considered state space (i.e. a map of observations), which is convenient since it enables \hat{P} , \hat{A} and \hat{R} to be implemented as CNNs.

Transition modelling

Similarly to the VIN (Equation (3.2)), the motion model $\hat{P}(s' - s|a)$ is implemented as a convolutional kernel $\mathbb{R}^{|\mathcal{A}| \times K_x \times K_y}$, where K_x and K_y are the x- and y- dimensions of the kernel, so it only depends on the relative spatial displacement $s' - s$ between the two states s and s' . Transitions already observed in the example trajectory can be used to constrain the model by adding a cross-entropy loss term $L(\hat{P}(a_t^*), s_{t+1} - s_t)$ for each step in the example trajectory. After training, the filter $\hat{P}(s' - s|a)$ will consist of a distribution over possible state transitions for each action (visualised in Figure 3.6).

Action availability

Although the action availability $\hat{A}(s, a)$ could be learned completely end-to-end in theory, additional regularisation is necessary in practice. Assuming a reliable log-probability of

each action being taken ($\widehat{A}_{\text{logit}}(s, a)$), available and unavailable actions can be distinguished by thresholding $\widehat{A}_{\text{logit}}(s, a)$ at a learnt threshold $\widehat{A}_{\text{thresh}}(s)$. Using a sigmoid function σ as a soft threshold, $\widehat{A}(s, a)$ could be written as:

$$\widehat{A}(s, a) = \sigma \left(\widehat{A}_{\text{logit}}(s, a) - \widehat{A}_{\text{thresh}}(s) \right). \quad (3.8)$$

Both $\widehat{A}_{\text{logit}}(s, a)$ and $\widehat{A}_{\text{thresh}}(s)$ are predicted by the network, given the observations at each time step. In order to ground the probabilities of each action being taken, the action logit $\widehat{A}_{\text{logit}}(s_t)$ is trained to match the example trajectory action a_t^* for all steps t , with an additional cross-entropy loss $L(\widehat{A}_{\text{logit}}(s), a^*)$. Note that *there is no additional ground truth supervision* – CALVIN strictly uses the same training data as a VIN.

Fully vs. partially observable environments

Some previous works [120, 216] assume that the entire environment is fully observable, which is often unrealistic. Partially observable environments, involving unknown scenes, require exploring to gather information and are thus more challenging. This is accounted for in CALVIN with a simple but significant modification. Note that $Q(s_t)$ in Equation (3.7) depends on the learned functions (CNNs) \widehat{P} , \widehat{A} and \widehat{R} through Equation (3.6), and these in turn are computed from the observations. CALVIN extends the VIN framework to the case of partial observations by ensuring that $Q(s_t|O_{\leq t})$ depends *only* on the observations up to time t , $O_{\leq t}$, which are aggregated into a map of observation embeddings M_t . This means that the VIN recomputes a plan at each step t (since the observations are different), instead of once for a whole trajectory. Unobserved locations have their features set to zero, so in practice knowledge of the environment is slowly built up during an expert demonstration, which enables exploration behaviour to be learned. Finally, because it is not possible to infer an optimal move in a region that was not observed yet, loss terms for unexplored cells are not considered in Equation (3.7).

Trajectory reweighting

Exploration provides observations $O_{\leq t}$ of the same locations at different times (Section 3.3.1). Thus, Equation (3.7) can be augmented with these partial observations as extra samples. The sum in Equation (3.7) becomes $\sum_{t' \in T} \sum_{t \in T_{1:t'}} w_t L(Q(s_t | O_{\leq t'}), a_t^*)$. These constitute extra samples for Equation (3.7), without needing more annotations. For long trajectories, the training data then becomes severely imbalanced between the exploration phase (target is not visible, hence the agent explores the environment) and goal completion phase (target is visible, hence the agent directly navigates towards it), with a large proportion of the former (90% of the data in Section 3.5.2). This is addressed in CALVIN by reweighting the samples. The weights are set such that $w_t = \beta^{d_t} / \max_{j \in T} \beta^{d_j}$ in Equation (3.7), i.e. a geometric decay scaling with the topological distance to the target d_t . Since expert trajectories are shortest paths, this simplifies to $w_t = \beta^{|T|-t}$.

3.3.2 Embodied navigation in 3D environments

Embodied agents such as robots have a pose in 3D space. This work considers a particular category of robots that operate in Special Euclidean group 2, which means that the agent can be both translated and rotated. While holonomic robots can navigate omnidirectionally, this is not the case for non-holonomic robots, and their navigation actions are constrained relative to their orientation (e.g. moving forward or rotating). They may also have a limited sensor suite, such as a perspective camera aimed in one direction, which will limit their observations.

Embodied pose states (position and orientation)

To address the limitation on the available action space (e.g. only able to move forward or rotate, rather than navigating omnidirectional), the 2D state space is augmented with an extra dimension, which corresponds to Θ discretised orientations: $\mathcal{S} = \{1, \dots, \Theta\} \times$

Table 3.1: Comparison of individual components in the implementation of CALVIN for positional states and for embodied pose states. X and Y are the size of the internal spatial discretisation of the environment, Θ is the internal discretisation of the orientation, A is the number of actions, K_x and K_y are the kernel dimensions for spatial locality, and M is the map of embeddings given as input with channel dimension C .

	Positional	Position + Orientation
State s	(x, y)	(θ, x, y)
Map of embeddings M	$C \times X \times Y$	$C \times X \times Y$
VI step	Conv2d	Conv3d
$V(s)$	$X \times Y$	$\Theta \times X \times Y$
$Q(s, a)$	$A \times X \times Y$	$A \times \Theta \times X \times Y$
$\hat{A}(s, a)$	$A \times X \times Y$	$A \times \Theta \times X \times Y$
\hat{P}	$A \times K_x \times K_y$	$A \times \Theta \times \Theta \times K_x \times K_y$
\hat{R}	$A \times K_x \times K_y$	$A \times \Theta \times \Theta \times K_x \times K_y$

$\{1, \dots, X\} \times \{1, \dots, Y\}$. This can be achieved by directly adding one extra dimension to each spatial tensor in Equations (3.5) and (3.6). Likewise, the loss (Equation (3.7)) must compute the cross-entropy over the new action space \mathcal{A} of size $\Theta \times K_x \times K_y$, which includes both transitions in rotation as well as spatial translation. Table 3.1 shows correspondences between tensor dimensions of the positional method and the embodied method for each component of the architecture.

The VI step in CALVIN performs a 2D convolution of \hat{P} over a 2D value map in the case of positional states and a 3D convolution over a 3D value map with orientation in the case of embodied pose states. In the embodied case, the second dimension of \hat{P} corresponds to the orientation of the current state, and the third dimension corresponds to that of the next state.

Note that the much larger state space makes long-term planning more difficult to learn. It can be observed that when naively augmenting the state space in this way, the models fail to learn correct motion kernels $\hat{P}(s' - s|a)$. This further reinforces the need for an auxiliary motion loss (Section 3.3.1), which overcomes this obstacle, and may explain

why prior works did not plan through fine-grained rotations.

3D embeddings for geometric reasoning

While the VIN performs its operations on a grid map – a discretisation of an absolute (world-centric) reference frame – generally an agent only has access to observations in a local (camera-centric) reference frame. Since the learnable functions (\widehat{P} , \widehat{A} and \widehat{R}) in CALVIN (and other VIN-based methods) are CNNs, their natural input is a grid of m -dimensional embeddings, denoted $e_{thij} \in \mathbb{R}^m$, for time t and discrete world-space coordinates $(h, i, j) \in \mathbf{Z}^{Z \times X \times Y}$, where Z is a discretisation of the height in 3D space. A resulting map of embeddings M_t is a spatio-temporal tensor of dimensions $(m \times Z) \times X \times Y$. CNN embeddings $\phi(I_t)$ obtained from images I_t in first-person view must be projected and aggregated into this 3D lattice map where VIN performs planning. A method named Lattice PointNet (LPN) is proposed in this work to accomplish the above.

Each embedding is associated with a 3D point in world-space via projective geometry, assuming that the camera position c_t and rotation matrix R_t are known (as assumed in prior work [26, 72, 120, 121, 216], which can be estimated from monocular vision [142]). Spatial projection also requires knowing (or estimating) the depths $d_t(p)$ of each pixel p in I_t (either with a RGB-D camera as in this work, or monocular depth estimation [61]).

The homogenous 3D coordinates of each pixel $p = (p_1, p_2)$ in the absolute reference frame can be written using projective geometry [82]:

$$[x_t(p), y_t(p), z_t(p), 1] = c_t + R_t K [p_1, p_2, d_t(p), 1]^\top, \quad (3.9)$$

where K is the camera’s intrinsics matrix. Given these absolute coordinates of pixels p , for each cell (h, i, j) in a 3D lattice map M_t , CNN embeddings $\phi(I_t)(p)$ of pixels close to the lattice cell is aggregated with the current map embedding e_{thij} . Inspired by PointNet [30], the embeddings of 3D points *from all past frames* that fall into each cell of the world-space lattice are aggregated with mean-pooling. This framework of spatial

aggregation can be easily extended to work spatio-temporally, aggregating information from past frames $t' \leq t$. More formally:

$$\begin{aligned}
 e_{thij} &= \text{avg}_{t' \leq t} \{ \phi_p(I_{t'}) : \tau_x i \leq x_{t'}(p) < \tau_x(i+1), \\
 &\quad \tau_y j \leq y_{t'}(p) < \tau_y(j+1), \\
 &\quad \tau_z h \leq z_{t'}(p) < \tau_z(h+1), p \in I_{t'} \},
 \end{aligned} \tag{3.10}$$

where τ_x , τ_y and τ_z are the absolute dimensions of each grid voxel, ‘‘avg’’ averages the elements of a set, and $\phi_p(I_{t'})$ retrieves the CNN embedding of image $I_{t'}$ for pixel p .

As opposed to the PointNet’s unstructured multi-layer perceptrons [30], the LPN takes advantage of the spatial structure by applying CNNs for feature extraction. It also maintains a spatially meaningful lattice output with a causal temporal constraint ($t' \leq t$), which is beneficial for downstream CNN predictors: $(\hat{P}(e_t), \hat{A}(e_t) \text{ and } \hat{R}(e_t))$.

A related proposal for Simultaneous Localisation and Mapping (SLAM) used spatial max-pooling but with more complex LSTMs / Gated Recurrent Units (GRUs) for temporal aggregation [26, 86]. Another related work on end-to-end trainable spatial embeddings uses ego-spherical memory [121].

Memory-efficient mapping

The LPN has some appealing properties in the context of navigation: 1) it allows reasoning about far away, observed but yet unvisited locations; 2) it fuses multiple observations of the same location, whether from different points-of-view or different times.

Temporal aggregation during rollout can be computed recursively as $e_{t,h,i,j}/n_{t,h,i,j}$ for $e_{t,h,i,j} = e_{t-1,h,i,j} + e'_{t,h,i,j}$ and $n_{t,h,i,j} = n_{t-1,h,i,j} + n'_{t,h,i,j}$, where $e'_{t,h,i,j}$ is the summed embedding for the points in cell (h, i, j) at time t , and $n'_{t,h,i,j}$ is the number of points per cell. Only the previous map $e_{t-1,h,i,j}$ and previous counts $n_{t-1,h,i,j}$ must be kept, not all past observations. Thus at run-time the memory cost is *constant* over time, allowing unbounded operation (unlike methods that do not have an explicit map [187]).

3.4 Experiment setup

3.4.1 Evaluation benchmarks

The capabilities of CALVIN are tested on increasingly challenging environments, leading up to unseen 3D environments emulating the real world. The measured metric of performance is the navigation success rate (fraction of trajectories that reach the target) in unseen environments.

Randomly generated 2D maze environments

Two types of 2D grid environments are considered: obstacle maps similar to the ones used in [216], and mazes generated using Wilson’s algorithm [240] (examples shown in Figures 3.2 to 3.4). The maze environments are more challenging due to their high branching complexity, especially with local observability, where backtracking is required. Hence, the focus of this work is on maze environments. Implements of both environments are available in an open-sourced code repository accompanying this work (Section 3.5.6).

CALVIN is tested against other VIN-based baselines. Realistic constraints of limited observability and embodied navigation with orientation are progressively introduced.

MiniWorld

MiniWorld [34] is a minimalistic 3D interior environment simulator for RL and robotics research. It can be used to simulate environments with rooms, doors, hallways and various objects. In particular, randomly generated maze environments of 3×3 and 8×8 are used for evaluating the agents in a 3D setting with first-person view RGB-D observations from a monocular camera.

Active Vision Dataset

Active Vision Dataset (AVD) [4] is a dataset of over $30K$ images of indoor environments with objects in the scene labelled with bounding boxes. Images are densely collected from a monocular RGB-D camera onboard a robot at 30 cm intervals and at 30° rotations in 19 unique household and office environments. It could be used for benchmarking object recognition methods and simulating motion in real-world environments by creating arbitrary trajectories connecting the poses of the robot and associated observations that are present in the dataset. This allows interactive navigation with real image streams, without synthetic rendering (as opposed to [134]).

3.4.2 Baselines and hyper-parameter choices

CALVIN is compared against the standard VIN by Tamar et al. [216], as well as against GPPN [120] that uses an LSTM as a recurrent operation to propagate values.

Similarly to VIN [216] which uses a 2-layer CNN to predict the reward map, and GPPN [120], which uses a 2-layer CNN to produce inputs to the LSTM, CALVIN uses a 2-layer CNN as an available actions predictor $\hat{A}(s, a)$. For each experiment, the size of the hidden layer was chosen from $\{40, 80, 150\}$. 150 was used for all the grid environments, 80 for MiniWorld and 40 for AVD, partially due to memory constraints.

VIN has an additional hyperparameter for the number of hidden action channels, which is set to 40, sufficiently bigger than the number of actual actions in all the experiments. While the kernel sizes K for VIN and CALVIN were set to 3 for experiments in the grid environment, it was noted in [120] that GPPN works better with larger kernel size. Therefore, the best kernel size is chosen out of $\{3, 5, 7, 9, 11\}$ for GPPN. For experiments on MiniWorld and AVD, there are state transitions with step size of 2, hence a kernel size of $K = 5$ is chosen for VIN and CALVIN.

The number of value iteration steps k was chosen from $\{20, 40, 60, 80, 100\}$. For

trajectory reweighting, β was chosen from $\{0.1, 0.25, 0.5, 0.75, 1.0\}$.

3.4.3 Expert trajectory generation

Expert trajectories are generated by running an A* [81] planner from the start state to the target state. Euclidean costs are assigned to every transition in the 2D grid environments, and a cost of 1 per move for the MiniWorld and AVD environments. In the case of MiniWorld, an additional cost is assigned to locations near obstacles to ensure that the trajectories are not in close proximity to the walls.

3.4.4 Inference time rollouts

The performance of the model is tested by running navigation trials (rollouts) on a randomly generated environment. At every time step, the model is queried the set of Q -values $\{Q(s, a) : a \in \mathcal{A}\}$ for the current state s , and an action which gives the maximum predicted Q -value is taken.

While the VIN is trained with $V^{(0)}$ initialised with zeros, in a true VI algorithm, the value function must converge for an optimal policy to be obtained. To help the value function converge faster under a time and compute budget, the value function is initialised with predicted values from the previous time step at test time with online navigation.

A limit is set to the maximum number of steps taken by the agent, which were 200 for the fully known 15×15 grid, 500 for the partially known grid, 300 for MiniWorld (3×3), 1000 for MiniWorld (8×8), and 100 for AVD.

Architectural design of Lattice PointNet

The LPN consists of three stages: a CNN that extracts embeddings from observations in image-space (image encoder), a spatial aggregation step (Equation (3.10)) that performs mean pooling of embeddings for each map cell, and another CNN that refines the map

embedding (map encoder). The image encoder consists of two CNN blocks, each consisting of the following layers in order: optional group normalisation, 2D convolution, dropout, Rectified Linear Unit (ReLU) and 2D max pooling. The map encoder consists of 2D convolution, dropout, ReLU, optional group normalisation, and finally, another 2D convolution. The number of channels of each convolutional layer are (80, 80, 80, 40) for MiniWorld and (40, 40, 40, 20) for AVD respectively. The point clouds can consume a significant amount of memory for long trajectories. Hence, the most recent 40 frames are used for the 8×8 MiniWorld maze.

The input to the LPN is a 3-channel RGB image for the MiniWorld experiment, and a 128-channel embedding extracted using the first 2 blocks of ResNet18 pre-trained on ImageNet for the AVD experiment.

Architectural design of the CNN backbone

A CNN backbone is used in a control experiment in Section 3.5.4 to show the effectiveness of the LPN backbone. In contrast to LPN which performs spatial aggregation of embeddings, the CNN backbone is a direct application of an encoder-decoder architecture that transforms image-space observations into map-space embeddings. Gupta et al. [72] employed a similar architecture to obtain their map embeddings. While they use ResNet50 as the encoder network, a simple CNN is used in the control experiment in Section 3.5.4 to match the result obtained with LPN in the MiniWorld experiment (Section 3.5.4).

The CNN backbone consists of three stages: a CNN encoder, two fully-connected layers with ReLU to transform embeddings from image-space to map-space, and a CNN decoder. The encoder consists of 3 blocks of batch normalisation, 2D convolution, dropout, ReLU and 2D max pooling, and a final block with just batch normalisation and 2D convolution. The number of channels of each convolutional layer is (64, 128, 128, 128), respectively.

The fully-connected layers take in an encoder output of size $128 \times 5 \times 7$, reduce it to

a hidden size of 128, and output either $128 \times 5 \times 5$ for the smaller maze or $128 \times 4 \times 4$ for the larger maze, which is then passed to the decoder.

The decoder consists of 3 blocks of batch normalisation, 2D deconvolution, dropout and ReLU, and a final block with just 2D deconvolution. The number of channels of each deconvolution layer is (128, 128, 64, 20), respectively. The output size of the decoder depends on the map resolution, hence appropriate strides, kernel sizes and paddings are chosen for the decoder network to match the output sizes of 30×30 and 80×80 . This approach is not scalable to maps with high resolution or with arbitrary size, which is one of the drawbacks of this approach.

3.5 Experiments

3.5.1 Fully observable 2D grid environment

CALVIN was first evaluated on 2D environments with positional states, where the observations are top-down views of the whole scene, and which thus do not require dealing with perspective images (Section 3.3.2). Since Tamar et al. [216] obtain near-perfect performance in their 2D environments, after reproducing their results, evaluation was primarily performed on 2D mazes, which are much more challenging since they require frequent backtracking to navigate if exploration is required. Mazes of size 15×15 are used. The allowed moves \mathcal{A} are to any of the 8 neighbours of a cell. As discussed in Section 3.3, a termination action D must be triggered at the target to successfully complete the task. The target is placed in a free cell chosen uniformly at random, with a minimum topological distance from the (random) start location equal to the environment size to avoid trivial tasks. This was necessary to avoid trivial tasks (i.e. starting close to the target).

Table 3.2: Navigation success rate (fraction of trajectories that reach the target) on unseen 2D mazes. Partial observations (exploring an environment gradually) and embodied navigation (translation-rotation state space) are important yet challenging steps towards full 3D environments.

Environment	Standard loss			Reweighted loss (ours)		
	VIN	GPPN	CALVIN (ours)	VIN	GPPN	CALVIN (ours)
Fully observable	75.6±20.6	91.3±8.1	99.0 ±1.0	77.5±26.6	96.6±4.0	99.7 ±0.5
Partially observable	3.6±0.6	8.5±3.5	48.0 ±5.2	1.7±1.7	11.25±3.7	92.2 ±1.3
Embodied	11.0±1.0	14.5±2.1	90.0 ±7.9	15.2±3.6	28.5±3.5	93.7 ±6.2

Baselines and training.

For the first experiment, CALVIN was compared with other differentiable planners: the VIN [216] and the more recent GPPN [120], on fully-observed environments. Other than using mazes instead of convex obstacles, this setting is close to Tamar et al.’s [216]. The VIN, GPPN and CALVIN all use 2-layer CNNs to predict their inputs, as described in Section 3.4.2. All networks are trained with $4K$ example trajectories in an equal number of different mazes, using the Adam optimiser with the optimal learning rate chosen from $\{0.01, 0.005, 0.001\}$, until convergence (up to 30 epochs). Navigation success rates (fraction of trajectories that reach the target) for epochs with minimum validation loss are reported. Reweighted loss (Section 3.3.1) is equally applicable to all differentiable planners, so results are reported both with and without it.

Results

Table 3.2 (first row) shows the navigation success rate, averaged over 3 random seeds (and the standard deviation). The VIN has a low success rate, showing that it does not scale to large mazes. GPPN achieves a high success rate, and CALVIN performs near-perfectly.

The low performance of VIN is likely due to the value function not being learnt correctly, as outlined in Section 3.2.4. A comparison of values learnt by the VIN and CALVIN are shown in Figure 3.2.

The high performance of GPPN may be explained by its higher capacity, as it contains a LSTM with more parameters. Nevertheless, CALVIN has a more constrained architecture, so its higher performance hints at a better inductive bias for navigation. It is interesting to note that the proposed reweighted loss has a beneficial effect on all methods, not just CALVIN. With the correct data distribution, any method with sufficient capacity can fit the objective. This shows that addressing the imbalanced nature of the data is an important, complementary factor.

3.5.2 Partially observable 2D grid environment

Next, the same methods are compared in unknown environments, where the observation maps only contain observed features up to the current time step (Section 3.3.1). To simulate local observations, ray-casting is performed to identify cells that are visible from the current position, up to 2 cells away. No information about the location of the target is given until it is within view of the agent – this corresponds to object class-based navigation [5], i.e. exploring to find an object of a given class.

Results

In this case, the agent has to take significantly more steps to explore compared to a direct route to the target. From Table 3.2 (2nd row) it could be observed that partial observability causes most methods to fail catastrophically. The sole exception is CALVIN with the reweighted loss (proposed), which performs well. Note that to succeed, an agent must acquire several complex behaviours: directing exploration to large unseen areas; backtracking from dead ends; and seeking the target when seen. CALVIN displays all of these behaviours.

Figure 3.3 shows a trajectory taken by CALVIN at runtime, with corresponding observation maps and predicted values. At each rollout step, CALVIN performs inference

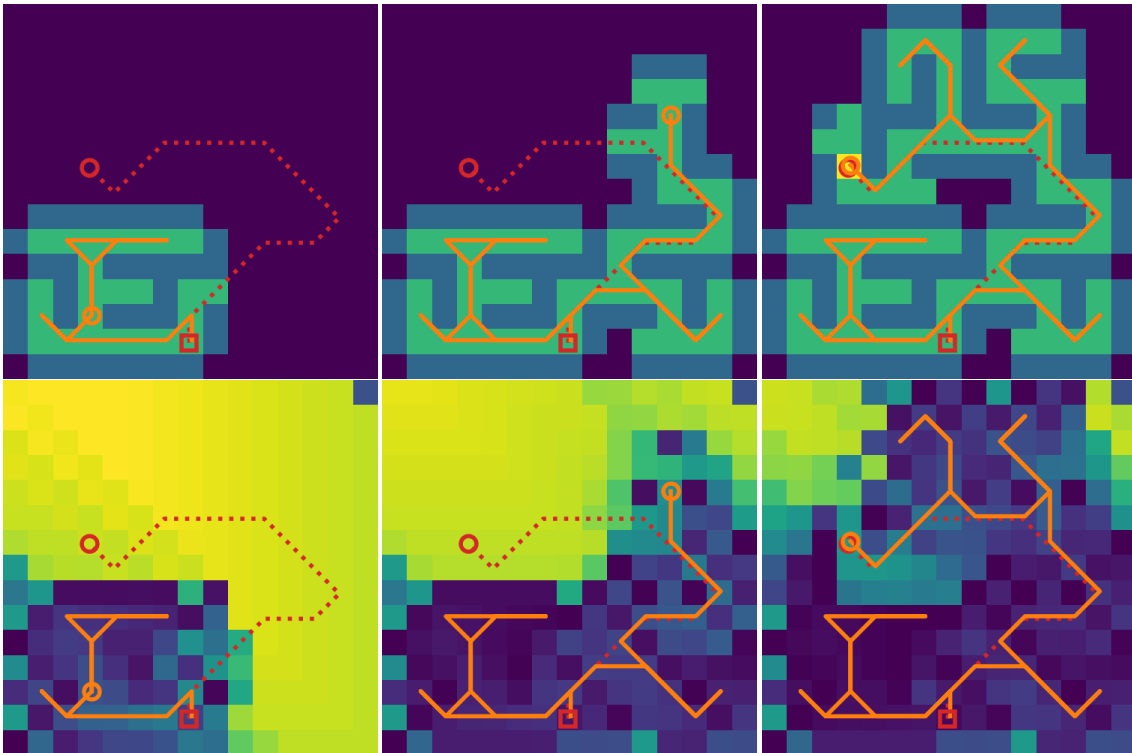


Figure 3.3: Example rollout of CALVIN after 21 steps (left column), 43 steps (middle column) and 65 steps (right column). CALVIN successfully terminated at 65 steps. **(top row)** Input visualisation: unexplored cells are dark, and the discovered target is yellow. The correct trajectory is dashed, and the current one is solid. The orange circle shows the position of the agent. **(bottom row)** Predicted values (higher values are brighter). Explored cells have low values, while unexplored cells and the discovered target are assigned high values.

on the best action to take based on its current observation map. No information about the location of the target is given until it is within view of the agent. This makes the problem challenging, since the agent may have to take significantly more steps compared to an optimal route to reach the target. In this example, the agent managed to backtrack every time it encountered a dead end, successfully reaching the target after 65 steps. The model initially assigns high values to all unexplored states. When the target comes into view, the model assigns a high probability to the availability of the “done” action at the corresponding state. The agent learns a sufficiently high reward for a successful termination so that the “done” action is triggered at the target.

Since only a combination of CALVIN and a reweighted loss works at all, it could be

inferred that a correct inductive bias and a balanced data distribution are both necessary for success. One explanation is that the agent needs to explore, which typically is to navigate away from locations that it has previously observed, and the reweighting encourages such behaviour by placing larger weights closer to the target. CALVIN trained with the reweighted loss managed to backtrack when it encountered a dead end, successfully reaching the target.

Ablation study of removing loss components

CALVIN is trained on three additive loss components: a loss term for the predicted Q-values L_Q (Section 3.3.1), a loss term for the transition models L_P (Section 3.3.1), and a loss term for the action availability L_A (Section 3.3.1). The aim is to assess the contributions of each loss component to the overall performance.

Experiments are conducted on the partially observable grid environment. The results in Table 3.3 indicate that all loss components, in particular the transition model loss, contribute to the robust performance of the network.

Table 3.3: Navigation success rate of CALVIN in partially observable 2D mazes with loss components removed.

Loss	$L_Q + L_P + L_A$	$L_Q + L_P$	$L_Q + L_A$
Success rate	92.2	84.1	8.3

3.5.3 Embodied navigation with orientation

Next, the agents are evaluated on embodied navigation, in which transitions depend on the agent’s orientation (Section 3.3.2). The state space of all methods is augmented with 8 orientations at 45° intervals following Section 3.3.2, and the action space is defined as 4 move actions \mathcal{A} : forward, backward, turn left, and turn right.

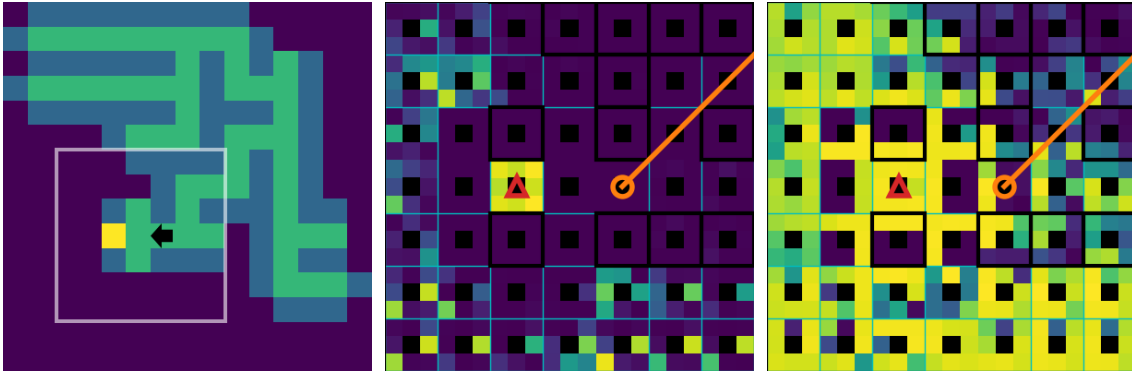


Figure 3.4: CALVIN’s learnt rewards and values on partially observable 2D mazes with embodied navigation (Section 3.5.3). **(left)** Input visualisation: unexplored cells are dark, the target is yellow (just found by the agent), and a black arrow shows the agent’s position and orientation. **(middle)** Close-up of predicted rewards (higher values are brighter) inside the white rectangle of the left panel. The 3D state space (position/orientation) is shown, with rewards for the 8 orientations in a radial pattern within each cell (position). Explored cells have low rewards, with the highest reward at the target. **(right)** Close-up of predicted values. They are higher facing the direction of the target. Obstacles (black border) have low values.

Results

Table 3.2 (3rd row) shows that VIN and GPPN perform slightly better, but still have a low chance of success. CALVIN outperforms them by a large margin. A typical run is visualised in Figure 3.4 (refer to the caption for more detailed analysis). A high reward is predicted for the (visible) target, with lower but still high rewards for unexplored regions; in this snapshot the agent just reached the target. One advantage of CALVIN displayed in Figures 3.2 to 3.4 is that values and rewards are fully interpretable and play the expected roles in VI (Equation (3.1)). Less constrained architectures [120, 216] insert operators that deviate from the value iteration formulation, and thus lose their interpretability as rewards and values.

3.5.4 Synthetically-rendered 3D environments

Having validated embodied navigation and exploration, CALVIN is now integrated with the LPN (Section 3.3.2) to handle first-person views of 3D environments.

Table 3.4: Navigation success rate on unseen 3D mazes (MiniWorld). Note that the baselines do not generalise to larger mazes.

Size	CNN backbone		LPN backbone (ours)		
	A2C	PPO	VIN	GPPN	CALVIN (ours)
3×3	98.7 \pm 1.9	81.0 \pm 26.9	90.3 \pm 3.1	91.3 \pm 4.7	97.7 \pm 1.7
8×8	23.6 \pm 4.9	14.7 \pm 6.2	41.2 \pm 9.5	33.3 \pm 8.6	69.2 \pm 5.3

The MiniWorld simulator [34] is used to generate 3D maze environments with arbitrary layouts (Section 3.4.1). Only a monocular camera is considered (not 360° views [120]). The training trajectories now consist of first-person videos of the shortest path to the target, visualised in Figure 3.5. $1K$ random trajectories are generated in mazes of both small (3×3) and large (8×8) grids by adding or removing walls at the boundaries of this grid’s cells. Note that the maze’s layout and the agent’s location do not necessarily align with the 2D grids used by the planners (as in [120]). Thus, planning happens on a fine discretisation of the state space (30×30 for small mazes and 80×80 for large ones, with 8 orientations). This allows smooth motions and no privileged information about the environment. Both translation and rotation are perturbed by Gaussian noise, forcing all agents to model uncertain dynamics. To accommodate such probabilistic transitions, motion dynamics are modelled with a 5×5 kernel (see Section 3.4.2).

Baselines and training

Several baselines are compared: two popular RL methods, Advantage Actor-Critic (A2C) [140] and Proximal Policy Optimisation (PPO) [194], as well as the VIN, GPPN and the proposed CALVIN. Since A2C and PPO are difficult to train if triggering the “done” action is strictly required, this assumption is relaxed, allowing the agent to terminate once it is near the target. All methods use as a first stage a simple 2-layer CNN (details in Section 3.4.4). Since this CNN was not enough to get the VIN, GPPN and CALVIN to work well, they all use the proposed LPN backbone (Section 3.4.4). It was not feasible to

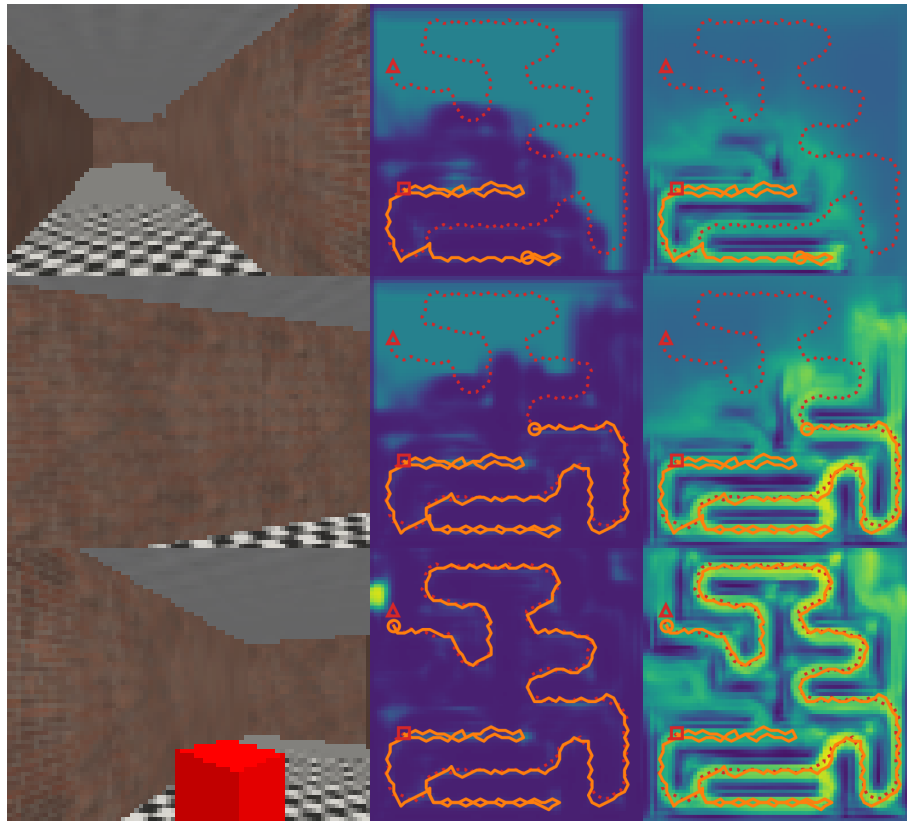


Figure 3.5: Example results on MiniWorld (Section 3.5.4). Left to right: input images, predicted rewards and values. The format is as in Figure 3.1. Notice the high reward on unexplored regions, replaced with a single peak around the target when it is seen (last row).

adopt GPPN’s strategy of taking views at all possible states as input [120], due to the high memory requirements and the environment not being fully visible (only a forward camera input of RGB-D is available). Other training details are identical to Section 3.5.1.

Results

Results in Table 3.4 indicate that although the RL methods are successful in small mazes, their reactive policies do not scale to large mazes. CALVIN succeeds reliably even for longer trajectories, outperforming the others. It is interesting to note that the proposed LPN backbone is important for all differentiable planners, and it allows them to achieve very high success rates for small environments (though CALVIN performs slightly better). An example run of CALVIN is shown in Figure 3.5. The CALVIN agent found the target after

an efficient exploration period, despite not knowing its location and never encountering this maze before.



Figure 3.6: Transition model learnt from MiniWorld trajectories for the *move forward* action at each discretised orientation, at 45° intervals. Higher values are brighter (yellow for a probability of 1), and lower values are darker (purple for a probability of 0).

Figure 3.6 visualises the learnt state transitions $\hat{P}(s' - s|a)$ for the *move forward* action in CALVIN. It could be observed that the learning mechanism outlined in Section 3.3.1 works even for transitions with added noise, which is the case for MiniWorld experiments. The network learns to propagate values probabilistically from the possible next states.

Comparison of LPN against CNN backbone

In this ablation study, the proposed LPN backbone is compared against a typical encoder-decoder CNN backbone as a component that maps observations to map embeddings. The performance of the two methods is evaluated for VIN, GPPN and CALVIN. It could be seen in Table 3.5 that the LPN backbone is highly effective, especially for larger environments where long-term planning based on spatially aggregated embeddings is necessary.

Table 3.5: Navigation success rate on unseen 3D mazes (MiniWorld), comparing the CNN backbone against the LPN backbone (also in Table 3.4). Most methods do not generalise to larger mazes. The proposed LPN demonstrates robust performance in larger unseen mazes.

Size	CNN backbone			LPN backbone (ours)		
	VIN	GPPN	CALVIN (ours)	VIN	GPPN	CALVIN (ours)
3×3	89.4	73.1	75.2	90.3	91.3	97.7
8×8	0.6	18.3	8.6	41.2	33.3	69.2

3.5.5 Indoor images from a real-world robot

Finally, CALVIN is tested on real images obtained with a robotic platform. The AVD [4] is used (Section 3.4.1). Images can be composed to simulate any trajectory, up to some spatial granularity. There are also bounding box annotations of object instances, which are used to evaluate semantic navigation. The last four indoor environments in AVD are kept as a validation set, and the other 18 scenes are used for training (by sampling $1K$ shortest paths to the target). A visualisation is shown in Figure 3.1.

Tasks and training

A semantic navigation task of seeking an object of a learnt class is considered. The most common class (“soda bottle”) is chosen as a target object. The training follows Section 3.5.4.

Results

Performances are reported for VIN, GPPN and CALVIN after 8 epochs of training in Table 3.6. As in Section 3.5.4, they also fail without the LPN, so all results are with the LPN backbone. A similar conclusion to that for synthetic environments can be drawn: proper spatio-temporal aggregation of local observations is essential for differentiable planners to scale realistically. CALVIN achieves a significantly higher success rate on the training set than the other methods. On the other hand, while it has higher mean validation success, the high variance of this estimate does not allow the result to be as conclusive as for training. This could be attributed to the small size of AVD in general, and of the validation set in particular, which contains only 3 different indoor scenes. Nevertheless, this shows that CALVIN learns effective generic strategies to seek a specific object, and that VIN and GPPN equipped with the LPN backbone can achieve partial success in several training environments. Figure 3.1 shows an example of a successful navigation sequence.

Table 3.6: Navigation success rate on AVD, with real robot images taken in indoor spaces. The task is to navigate to an object of a learned class. All methods use the proposed LPN backbone, as they fail without it.

Subset	VIN	GPPN	CALVIN (ours)
Training	61.6±4.5	50.6±9.2	70.3±4.9
Validation	45.0±1.0	44.0±3.5	47.6±6.0

3.5.6 Implementation

The code used in this paper is open-sourced, and can be found at <https://github.com/shuishida/calvin>. This includes code for CALVIN as well as baselines such as VIN and GPPN, and for different training environments (i.e. grid world, MiniWorld and AVD).

3.6 Conclusion

This work proposed CALVIN, which addresses limitations of current VIN implementations with an extended state-action space, an action availability mechanism, and additional constraints to the motion model. Contributions include robust embodied navigation (position and orientation) in unexplored environments and 3D environments from a single camera, as well as a dense sampling scheme that achieves higher sample efficiency in some scenarios. The LPN backbone was also proposed, which efficiently fuses spatio-temporal information in a differentiable way.

CALVIN achieved high performance in challenging 3D navigation environments by learning transition models and rewards in a data-driven manner. All components of the model are end-to-end trainable, but they also have clear definitions of what they mathematically represent. This internal representation grants robustness to randomised and unknown environments, unlike reactive policies learnt by RL agents.

Chapter 4

Option discovery via Expectation

Maximisation and Policy Gradients

This chapter aims to propose ways to overcome the limitations of Reinforcement Learning (RL) algorithms designed for Markov Decision Processes (MDPs) with the means of *options*. Options are temporally abstracted macro-actions that enable hierarchical planning and decision-making over multiple time steps. Each option is associated with a sub-policy which dictates how a sequence of primitive actions should be taken during the execution of the option. Options also function as a memory that allows the agent to retain historical information beyond the policy's context window. While option assignment could be handled using heuristics and hand-crafted objectives, learning an optimal option assignment for any given task is an unsolved challenge. In this chapter, two algorithms, Proximal Policy Optimisation via Expectation Maximisation (PPOEM) and Sequential Option Advantage Propagation (SOAP), are proposed and investigated in depth to address this problem.

The first approach, PPOEM, applies Expectation Maximisation (EM) to a Hidden Markov Model (HMM) describing a Partially Observable Markov Decision Process (POMDP) for the options framework. The method is an extension of the forward-backward algorithm, also known as the Baum-Welch algorithm [14], applied to options. While this approach has previously been explored [42, 59, 64, 258], these applications were limited to 1-step Temporal Difference (TD) learning. In addition, the learnt options have limited

expressivity due to how the option transitions are defined (see Section 4.2.4). In contrast, PPOEM augments the forward-backward algorithm with Generalised Advantage Estimate (GAE), which is a temporal generalisation of TD learning, and extends the Proximal Policy Optimisation (PPO) [194] to work with options. While this approach was shown to be effective in a limited setting of a corridor environment requiring memory, the performance degraded with longer corridors. It could be hypothesised that this is due to the learning objective being misaligned with the true RL objective, as the approach assumes access to the full trajectory of the agent for the optimal assignment of options, even though the agent only has access to its past trajectory (and not its future) at inference time.

As an alternative approach, SOAP evaluates and maximises the policy gradient for an optimal option assignment directly. With this approach, the option policy is only conditional on the history of the agent. The derived objective has a surprising resemblance to the forward-backward algorithm, but showed more robustness when tested in longer corridor environments. The algorithms were also evaluated on the Atari [16] and MuJoCo [222] benchmarks. Results demonstrated that using SOAP for option learning is more effective and robust than using the standard approach for learning options, proposed by the Option-Critic architecture [9, 111].

The proposed approach can improve the efficiency of skill discovery in skill-based RL algorithms, allowing them to adapt efficiently to complex novel environments. The paper is available on *arXiv* [98].

4.1 Introduction

While deep RL has seen rapid advancements in recent years, with numerous real-world applications such as robotics [2, 70, 77], gaming [7, 12, 225], and autonomous vehicles [110, 130], many algorithms are limited by the amount of observation history they condition their policy on, due to the increase in computational complexity. Developing

Chapter 4. Option discovery via Expectation Maximisation and Policy Gradients 68

learnable embodied agents that plan over a wide spatial and temporal horizon has been a longstanding challenge in RL.

With a simple Markovian policy $\pi(a_t|s_t)$, the agent’s ability to make decisions is limited by only having access to the current state as input. Early advances in RL were made on tasks that either adhere to the Markov assumption that the policy and state transitions only depend on the current state, or those can be solved by frame stacking [139] that grants the policy access to a short history. However, many real-world tasks are better modelled as POMDPs [262] with a long-term temporal dependency, motivating solutions that use a bounded working memory for computational scalability.

For POMDP tasks, the entire history of the agent’s trajectory may contain signals to inform the agent to make a more optimal decision. This is due to the reward and next state distribution $p(r_t, s_{t+1}|s_{0:t}, a_{0:t})$ being conditional on the past states and actions, not just on the current state and action.

A common approach of accommodating POMDPs is to learn a latent representation using sequential policies, typically using a Long Short-Term Memory (LSTM) [89], Gated Recurrent Unit (GRU) [36] or Transformer [226]. This will allow the policy to gain access to signals from the past. However, this approach has an inherent trade-off between the duration of history it can retain (defined by the policy’s context window size) and the compute and training data required to learn the policy. This is because the entire history of observations within the context window have to be included in the forward pass at training time to propagate useful gradients back to the sequential policy. Another caveat is that, with larger context windows, the input space is less constrained and it becomes increasingly unlikely that the agent will revisit the same combination of states, which makes learning the policy and value function sample-expensive, and potentially unstable at inference time if the policy distribution has changed during training.

Training RL agents to work with longer working memory is a non-trivial task, es-

pecially when the content of the memory is not pre-determined and the agent also has to learn to store information relevant to each task. With the tasks that the RL algorithms are expected to handle becoming increasingly complex [31, 51, 136], there is a vital need to develop algorithms that learn policies and skills that generalise to dynamic and novel environments. Many real-world tasks are performed over long time horizons, which makes it crucial that the algorithm can be efficiently trained and quickly adapted to changes in the environment. This gives motivation to develop an algorithm that (a) can solve problems modelled as POMDP using options, (b) has a bounded context length input for the policy and value function so that they can be trained more sample-efficiently, (c) only requires the current observation to be forward-passed through a neural network at training time to reduce the Graphical Processing Unit (GPU) memory and computational requirements.

There has been considerable effort in making RL more generalisable and efficient. Relevant research fields include Hierarchical Reinforcement Learning (HRL) [143, 160, 228, 253], skill learning [145, 162, 163, 198], Meta Reinforcement Learning (Meta-RL) [15, 50, 171, 231] and the options framework [166, 214], with a shared focus on learning reusable policies. In particular, this research focuses on the options framework, which extends the RL paradigm with a HMM that uses options to execute long-term behaviour.

Options are instrumental in abstract reasoning and high-level decision-making, since they enable temporal abstraction, credit assignment, and identification of skills and sub-goals. Acquiring transferable skills and composing them to execute plans, even in novel environments, are remarkable human capabilities that are instrumental in performing complex tasks with long-term objectives. Whenever one encounters a novel situation, one can still strategise by applying prior knowledge with a limited budget of additional trial and error. One way of achieving this is by abstracting away the complexity of long-term planning by delegating short-term decisions to a set of specialised low-level policies, while

the high-level policy focuses on achieving the ultimate objective by orchestrating these low-level policies.

The Option-Critic architecture [9] presents a well-formulated solution for end-to-end option discovery. The authors showed that once the option policies are learned, the Option-Critic agent can quickly adapt when the environment dynamics are changed, whereas other algorithms suffer from the changes in reward distributions.

However, there are challenges with regard to automatically learning options. A common issue is that the agent may converge to a single option that approximates the optimal policy under a Markov assumption. Additionally, learning options from scratch can be less sample-efficient due to the need to learn multiple option policies. (The data-efficiency may be improved if the options are learned off-policy [186, 243]. This work mostly focused on on-policy algorithms due to their robustness and ease of analysis.)

In the following sections, two candidate training objectives are proposed and derived to learn an optimal option assignment. The first objective, which will be referred to as PPOEM, is derived by using the forward-backward algorithm [14], applied to a POMDP according to the options framework. However, this objective is developed to assign latent variables for offline sequences, and is less suitable for RL with on-policy rollouts. As an alternative training objective, SOAP is proposed by adapting the policy gradient algorithm to work with options, which exhibits a resemblance to the forward-backward algorithm.

4.2 Background

4.2.1 Option-Critic architecture

As mentioned in Section 2.2.5, the options framework [166, 214] formalises the idea of temporally extended actions that allow agents to make high-level decisions. Let there be n discrete options $\{\mathcal{Z}_1, \dots, \mathcal{Z}_n\}$ from which z_t is chosen and assigned at every time

step t . Each option corresponds to a specialised sub-policy $\pi_\theta(a_t|s_t, z_t)$ that the agent can use to achieve a specific subtask. At $t = 0$, the agent chooses an option according to its inter-option policy $\pi_\phi(z_t|s_t)$ (policy over options), then follows the option sub-policy until termination, which is dictated by the termination probability function $\varpi_\psi(s_t, z_{t-1})$. Once the option is terminated, a new option z_t is sampled from the inter-option policy and the procedure is repeated.

The Option-Critic architecture [9] learns option assignments end-to-end. It formulates the problem such that the option sub-policies $\pi_\theta(a_t|s_t, z_t)$ and termination function $\varpi_\psi(s_{t+1}, z_t)$ are learned jointly in the process of maximising the expected returns. The inter-option policy $\pi_\phi(z_t|s_t)$ is an ϵ -greedy policy that takes an argmax z of the option value function $Q_\phi(s, z)$ with $1 - \epsilon$ probability, and uniformly randomly samples options with ϵ probability. In every step of the Option-Critic algorithm, the following updates are performed for a current state s , option z , reward r , episode termination indicator $d \in \{0, 1\}$, next state s' , and discount factor $\gamma \in [0, 1)$:

$$\begin{aligned}
 \delta &\leftarrow r + \gamma(1 - d) \left[(1 - \varpi_\psi(s', z)) Q_\phi(s', z) + \varpi_\psi(s', z) \max_z Q_\phi(s', z) \right] - Q_\phi(s, z), \\
 Q_\phi(s, z) &\leftarrow Q_\phi(s, z) + \alpha_\phi \delta, \\
 \theta &\leftarrow \theta + \alpha_\theta \frac{\partial \log \pi_\theta(a|s, z)}{\partial \theta} [r + \gamma Q_\phi(s', z)], \\
 \psi &\leftarrow \psi - \alpha_\psi \frac{\partial \varpi_\psi(s', z)}{\partial \psi} [Q_\phi(s', z) - \max_z Q_\phi(s', z)].
 \end{aligned}
 \tag{4.1}$$

Here, α_ϕ , α_θ and α_ψ are learning rates for $Q_\phi(s, z)$, $\pi_\theta(a|s, z)$, and $\varpi_\psi(s, z)$, respectively.

Proximal Policy Option-Critic (PPOC) [111] builds on top of the Option-Critic architecture [9], replacing the ϵ -greedy policy over the option-values with a policy network $\pi_\varphi(z|s)$ parametrised by φ with corresponding learning rate α_φ , substituting the policy gradient algorithm with PPO [194] to optimise the sub-policies $\pi_\theta(a|s, z)$, and introducing GAE [193] for the advantage estimate and value function updates. The policy loss function for PPO is given as $\mathcal{L}_{\text{PPO}}(\theta)$ in Equation (2.7). Extending the definition of GAE given in

Section 2.2.4 to work with options,

$$\begin{aligned}
 A^{\text{GAE}}(s, z) &\leftarrow r + \gamma V(s', z') - V(s, z) + \lambda \gamma (1 - d) A^{\text{GAE}}(s', z'), \\
 Q_\phi(s, z) &\leftarrow Q_\phi(s, z) + \alpha_\phi A^{\text{GAE}}(s, z), \\
 \theta &\leftarrow \theta + \alpha_\theta \frac{\partial \mathcal{L}_{\text{PPO}}(\theta)}{\partial \theta}, \\
 \psi &\leftarrow \psi - \alpha_\psi \frac{\partial \varpi_\psi(s, z)}{\partial \psi} A^{\text{GAE}}(s, z), \\
 \varphi &\leftarrow \varphi + \alpha_\varphi \frac{\partial \log \pi_\varphi(z|s)}{\partial \varphi} A^{\text{GAE}}(s, z).
 \end{aligned} \tag{4.2}$$

PPOC is used as one of the baselines in this work.

4.2.2 Double Actor-Critic

Double Actor-Critic (DAC) [255] is an HRL approach to discovering options. DAC reformulates the Semi-Markov Decision Process (Semi-MDP) [214] of the option framework as two hierarchical MDPs (high-MDP and low-MDP).

The low-MDP concerns the selection of low-level actions within the currently active option. Given the current state s_t , selected option z_t , and action a_t , the state and action spaces of the low-MDP can be defined as $S_t^L = (s_t, z_t)$ and $A_t^L = a_t$. With this definition, the transition function, reward function and policy for the low-MDP can be written as:

$$\begin{aligned}
 P_L(S_{t+1}^L | S_t^L, A_t^L) &= p((s_{t+1}, z_{t+1}) | (s_t, z_t), a_t) = P(s_{t+1} | s_t, a_t) \cdot p(z_{t+1} | s_{t+1}, z_t), \\
 R_L(S_t^L, A_t^L) &= r(s_t, a_t), \\
 \pi_L(A_t^L | S_t^L) &= \pi_L(a_t | s_t, z_t).
 \end{aligned} \tag{4.3}$$

The high-MDP handles high-level decision-making, such as selecting and terminating options. The state and action spaces of the high-MDP can be defined as $S_t^H = (s_t, z_{t-1})$ and $A_t^H = z_t$. With this definition, the transition function, reward function and policy for

the high-MDP can be written as:

$$\begin{aligned}
 P_H(S_{t+1}^H | S_t^H, A_t^H) &= p((s_{t+1}, z_t) | (s_t, z_{t-1}), A_t^H) = \mathbb{I}_{A_t^H = z_t} p(s_{t+1} | s_t, z_t), \\
 R_H(S_t^H, A_t^H) &= r(s_t, z_t), \\
 \pi_H(A_t^H | S_t^H) &= \pi_H(z_t | s_t, z_{t-1}).
 \end{aligned} \tag{4.4}$$

The key idea is to treat the inter-option and intra-option policies as independent actors in their respective MDPs. Under this formulation, standard policy optimisation algorithms such as PPO can be used to optimise the policies (high-level and low-level) for both MDPs.

4.2.3 Expectation Maximisation algorithm

The EM algorithm [45] is a well-known method for learning the assignment of latent variables, often used for unsupervised clustering and segmentation. The k -means clustering algorithm [58] can be considered a special case of EM. The following explanation in this section is a partial summary of Chapter 9 of Bishop’s book [19].

The objective of EM is to find a maximum likelihood solution for models with latent variables. Denoting the set of all observed data as \mathbf{X} , the set of all latent variables as \mathbf{Z} , and the set of all model parameters as Θ , the log-likelihood function is given by:

$$\log p(\mathbf{X} | \Theta) = \log \left\{ \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z} | \Theta) \right\}. \tag{4.5}$$

However, evaluating the above summation (or integral for a continuous \mathbf{Z}) over all possible latents is intractable. The EM algorithm is a way to strictly increase the likelihood function by alternating between the E-step that evaluates the expectation of a joint log-likelihood $\log p(\mathbf{X}, \mathbf{Z} | \Theta)$, and the M-step that maximises this expectation.

In the E-step, the current parameter estimate Θ_{old} (using random initialisation in the first iteration, or the most recent updated parameters in subsequent iterations) is used to determine the posterior of the latents $p(\mathbf{Z} | \mathbf{X}, \Theta_{\text{old}})$. The joint log-likelihood is obtained

under this prior. The expectation, denoted as $Q(\Theta; \Theta_{\text{old}})$, is given by:

$$Q(\Theta; \Theta_{\text{old}}) = \mathbb{E}_{\mathbf{Z} \sim p(\cdot | \mathbf{X}, \Theta_{\text{old}})} [\log p(\mathbf{X}, \mathbf{Z} | \Theta)] = \sum_{\mathbf{Z}} p(\mathbf{Z} | \mathbf{X}, \Theta_{\text{old}}) \log p(\mathbf{X}, \mathbf{Z} | \Theta). \quad (4.6)$$

In the M-step, an updated parameter estimate Θ_{new} is obtained by maximising the expectation:

$$\Theta_{\text{new}} = \underset{\Theta}{\operatorname{argmax}} Q(\Theta, \Theta_{\text{old}}). \quad (4.7)$$

The E-step and the M-step are performed alternately until a convergence criterion is satisfied. The EM algorithm makes obtaining a maximum likelihood solution tractable.

4.2.4 Forward-backward algorithm

The EM algorithm can also be applied in an HMM setting for sequential data, resulting in the forward-backward algorithm, also known as the Baum-Welch algorithm [14]. Figure 4.1 shows the graph of the HMM of interest. At every time step $t \in \{0, \dots, T\}$, a latent z_t is chosen out of n number of discrete options $\{Z_1, \dots, Z_n\}$, which is an underlying conditioning variable for an observation x_t . In the following derivation, $\{x_t | t_1 \leq t \leq t_2\}$ is denoted with a shorthand $x_{t_1:t_2}$, and similarly for other variables. Chapter 13 of Bishop's book [19] offers a comprehensive explanation for this algorithm.

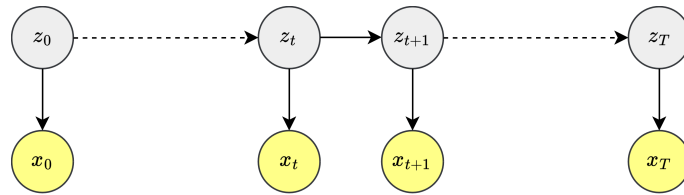


Figure 4.1: An HMM for sequential data \mathbf{X} of length T , given latent variables \mathbf{Z} .

For this HMM, the joint likelihood function for the observed sequence $\mathbf{X} = \{x_0, \dots, x_T\}$ and latent variables $\mathbf{Z} = \{z_0, \dots, z_T\}$ is given by:

$$p(\mathbf{X}, \mathbf{Z} | \Theta) = p(z_0 | \Theta) \prod_{t=0}^{T-1} p(x_t | z_t, \Theta) \prod_{t=1}^T p(z_t | z_{t-1}, \Theta). \quad (4.8)$$

Using the above, EM objective can be simplified as:

$$\begin{aligned}
 \mathcal{Q}(\Theta; \Theta_{\text{old}}) &= \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \Theta_{\text{old}}) \log p(\mathbf{X}, \mathbf{Z}|\Theta) \\
 &= \sum_{z_0} p(z_0|\Theta_{\text{old}}) \log p(z_0|\Theta) + \sum_{t=0}^T \sum_{z_t} p(z_t|\mathbf{X}, \Theta_{\text{old}}) \log p(x_t|z_t, \Theta) \\
 &\quad + \sum_{t=1}^T \sum_{z_{t-1}, z_t} p(z_{t-1}, z_t|\mathbf{X}, \Theta_{\text{old}}) \log p(x_t, z_t|z_{t-1}, \Theta).
 \end{aligned} \tag{4.9}$$

E-step

In the E-step, $p(z_t|\mathbf{X})$ and $p(z_{t-1}, z_t|\mathbf{X})$ are evaluated. Note that in the following derivation, it is assumed that the probability distributions are conditioned on Θ . Defining $\alpha(z_t) := p(z_t|x_{0:t})$, $\beta(z_t) := \frac{p(x_{t+1:T}|z_t)}{p(x_{t+1:T}|x_{0:t})}$ and normalising constant $c_t := p(x_t|x_{0:t-1})$,

$$p(z_t|\mathbf{X}) = \frac{p(x_{0:T}, z_t)}{p(x_{0:T})} = \frac{p(x_{0:t}, z_t)p(x_{t+1:T}|z_t)}{p(x_{0:t})p(x_{t+1:T}|x_{0:t})} = \alpha(z_t)\beta(z_t), \tag{4.10}$$

$$\begin{aligned}
 p(z_{t-1}, z_t|\mathbf{X}) &= \frac{p(x_{0:T}, z_{t-1}, z_t)}{p(x_{0:T})} = \frac{p(x_{0:t-1}, z_{t-1})p(x_t|z_t)p(z_t|z_{t-1})p(x_{t+1:T}|z_t)}{p(x_{0:t-1})p(x_t|x_{0:t-1})p(x_{t+1:T}|x_{0:t})} \\
 &= p(x_t|z_t)p(z_t|z_{t-1}) \frac{\alpha(z_t)\beta(z_t)}{c_t}.
 \end{aligned} \tag{4.11}$$

Recursively evaluating $\alpha(z_t)$, $\beta(z_t)$ and c_t ,

$$\begin{aligned}
 \alpha(z_t) &= \frac{p(x_{0:t}, z_t)}{p(x_{0:t})} = \frac{p(x_t, z_t|x_{0:t-1})}{p(x_t|x_{0:t-1})} = \frac{\sum_{z_{t-1}} [p(z_{t-1}|x_{0:t-1})p(x_t|z_t)p(z_t|z_{t-1})]}{p(x_t|x_{0:t-1})} \\
 &= \frac{p(x_t|z_t) \sum_{z_{t-1}} [\alpha(z_{t-1})p(z_t|z_{t-1})]}{c_t},
 \end{aligned} \tag{4.12}$$

$$\begin{aligned}
 \beta(z_t) &= \frac{p(x_{t+1:T}|z_t)}{p(x_{t+1:T}|x_{0:t})} = \frac{\sum_{z_{t+1}} [p(x_{t+2:T}|z_{t+1})p(x_{t+1}|z_{t+1})p(z_{t+1}|z_t)]}{p(x_{t+2:T}|x_{0:t+1})p(x_{t+1}|x_{0:t})} \\
 &= \frac{\sum_{z_{t+1}} [\beta(z_{t+1})p(x_{t+1}|z_{t+1})p(z_{t+1}|z_t)]}{c_{t+1}},
 \end{aligned} \tag{4.13}$$

$$\begin{aligned}
 c_t &= p(x_t|x_{0:t-1}) = \sum_{z_{t-1}, z_t} [p(z_{t-1}|x_{0:t-1})p(x_t|z_t)p(z_t|z_{t-1})] \\
 &= \sum_{z_{t-1}, z_t} [\alpha(z_{t-1})p(x_t|z_t)p(z_t|z_{t-1})].
 \end{aligned} \tag{4.14}$$

Initial conditions are $\alpha(z_0) = \frac{p(x_0|z_0)p(z_0)}{\sum_{z_0} [p(x_0|z_0)p(z_0)]}$, $\beta(z_T) = 1$.

M-step

In the M-step, the parameter set Θ is updated by maximising $\mathcal{Q}(\Theta; \Theta_{\text{old}})$, which can be rewritten by substituting $p(z_t|\mathbf{X})$ and $p(z_{t-1}, z_t|\mathbf{X})$ in Equation (4.9) with $\alpha(z)$ and $\beta(z)$ (ignoring the constants) as derived in Section 4.2.4.

Option discovery via the forward-backward algorithm

The idea of applying the forward-backward algorithm to learn option assignments is first introduced in [42], and has later been applied in both Imitation Learning (IL) settings [64, 258] and RL settings [59]. However, in previous literature, the option policy is decoupled into an option termination probability $\varpi(s_t, z_{t-1})$, and an inter-option policy $\pi(z_t|s_t)$. Due to the inter-option policy being unconditional on the previous option z_{t-1} , the choice of a new option z_t will be uninformed of the previous option z_{t-1} . This may be problematic for learning POMDP tasks as demonstrated in Section 4.6.1. Previous literature also does not address the issues of exponentially diminishing magnitudes which arise from recursively applying the formula. This is known as the scaling factor problem [19].

This chapter presents a concise derivation of the forward-backward algorithm applied to an improved version of the options framework. The scaling factor is also built into the derivation.

4.3 Option assignment formulation

The aim is to learn a diverse set of options with corresponding policy and value estimates, such that each option is responsible for accomplishing a well-defined subtask, such as reaching a certain state region. At every time step t , the agent chooses an option z_t out of n number of discrete options $\{Z_1, \dots, Z_n\}$.

4.3.1 Option policy and sub-policy

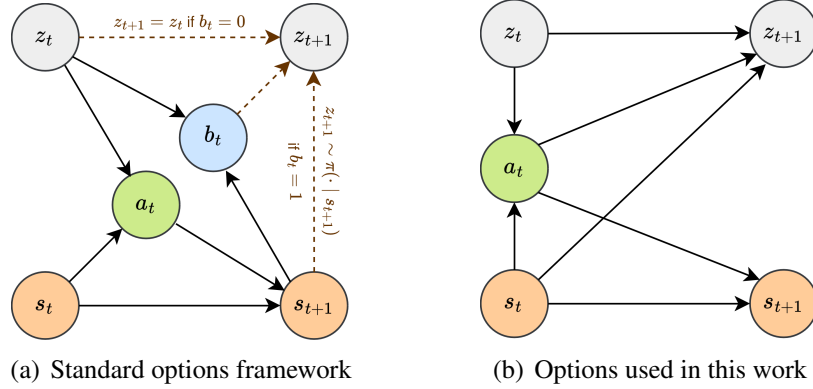


Figure 4.2: Probabilistic graphical models showing the relationships between options z , actions a and states s at time step t . b_t in the standard options framework denotes a boolean variable that initiates the switching of options when activated. This work adopts a more general formulation compared to the options framework, as defined in Equation (4.15).

The goal is to learn a sub-policy $\pi_{\theta}(a|s, z)$ conditional to a latent option variable z , and an option policy $\pi_{\psi}(z'|s, a, z)$ used to iteratively assign options at each time step, to model the joint option policy

$$p_{\Theta}(a_t, z_{t+1}|s_t, z_t) = \pi_{\theta}(a_t|s_t, z_t)\pi_{\psi}(z_{t+1}|s_t, a_t, z_t). \quad (4.15)$$

Here, the learnable parameter set of the policy is denoted as $\Theta = \{\theta, \psi\}$.

A comparison of the option policy used in this work and the standard options framework is shown in Figure 4.2. Unlike the options framework, which further decouples the option policy π_{ψ} into an option termination probability $\varpi(s_t, z_{t-1})$, and an unconditional inter-option policy $\pi(z_t|s_t)$, in this work the option policy is modelled π_{ψ} with one network so that the inter-option policy is informed by the previous option z_t upon choosing the next z_{t+1} . A graphical model for the full HMM is shown in Figure 4.3.

4.3.2 Evaluating the probability of latents

Let us define an auto-regressive action probability $\alpha_t := p(a_t|s_{0:t}, a_{0:t-1})$, an auto-regressive option forward distribution $\zeta(z_t) := p(z_t|s_{0:t}, a_{0:t-1})$, and an option backward feedback

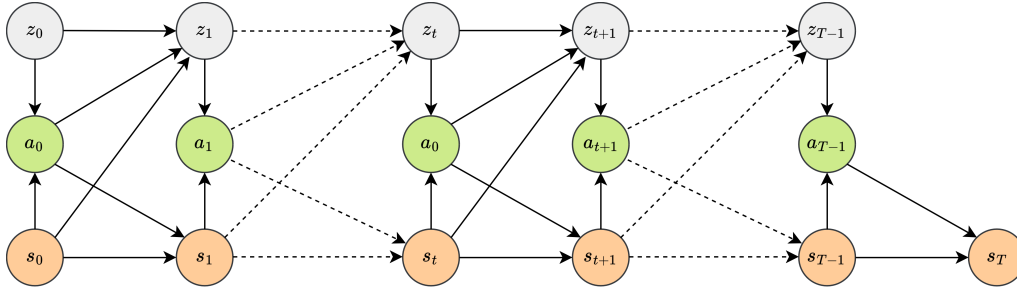


Figure 4.3: An HMM showing the relationships between options z , actions a and states s . The dotted arrows indicate that the same pattern repeats where the intermediate time steps are abbreviated.

$\beta(z_t) := \frac{p(s_{t:T}, a_{t:T-1} | s_{t-1}, a_{t-1}, z_t)}{p(s_{t:T}, a_{t:T-1} | s_{0:t-1}, a_{0:t-1})}$. Notice that the definitions of action probability α , option forward $\zeta(z_t)$, and option backward $\beta(z_t)$ resemble c_t , $\alpha(z_t)$ and $\beta(z_t)$ defined in Section 4.2.4, respectively. While it is common practice to denote the forward and backward quantities as α and β in the forward-backward algorithm (also known as the α - β algorithm), here α_t is redefined to denote the action probability (corresponding to the normalising constant c_t), and $\zeta(z_t)$ for the option forward distribution, to draw attention to the fact that these are probabilities of option z_t and action a_t , respectively.

α_t , $\zeta(z_t)$ and $\beta(z_t)$ can be recursively evaluated as follows:

$$\begin{aligned} \alpha_t &= p(a_t | s_{0:t}, a_{0:t-1}) = \sum_{z_t, z_{t+1}} p(z_t | s_{0:t}, a_{0:t-1}) p_{\Theta}(a_t, z_{t+1} | s_t, z_t) \\ &= \sum_{z_t, z_{t+1}} \zeta(z_t) p_{\Theta}(a_t, z_{t+1} | s_t, z_t), \end{aligned} \quad (4.16)$$

$$\begin{aligned} \zeta(z_{t+1}) &= \frac{p(z_{t+1}, s_{t+1}, a_t | s_{0:t}, a_{0:t-1})}{p(s_{t+1}, a_t | s_{0:t}, a_{0:t-1})} \\ &= \frac{\sum_{z_t} p(z_t | s_{0:t}, a_{0:t-1}) p_{\Theta}(a_t, z_{t+1} | s_t, z_t) P(s_{t+1} | s_{0:t}, a_{0:t})}{p(a_t | s_{0:t}, a_{0:t-1}) P(s_{t+1} | s_{0:t}, a_{0:t})} \\ &= \frac{\sum_{z_t} \zeta(z_t) p_{\Theta}(a_t, z_{t+1} | s_t, z_t)}{\alpha_t}, \end{aligned} \quad (4.17)$$

$$\begin{aligned} \beta(z_t) &= \frac{p(s_{t:T}, a_{t:T-1} | s_{t-1}, a_{t-1}, z_t)}{p(s_{t:T}, a_{t:T-1} | s_{0:t-1}, a_{0:t-1})} \\ &= \frac{\sum_{z_{t+1}} [p(s_{t+1:T}, a_{t+1:T-1} | s_t, a_t, z_{t+1}) p_{\Theta}(a_t, z_{t+1} | s_t, z_t) P(s_t | s_{0:t-1}, a_{0:t-1})]}{p(s_{t+1:T}, a_{t+1:T-1} | s_{0:t}, a_{0:t}) p(a_t | s_{0:t}, a_{0:t-1}) P(s_t | s_{0:t-1}, a_{0:t-1})} \\ &= \frac{\sum_{z_{t+1}} [\beta(z_{t+1}) p_{\Theta}(a_t, z_{t+1} | s_t, z_t)]}{\alpha_t}. \end{aligned} \quad (4.18)$$

Initial conditions are $\zeta(z_0) = p(z_0) = \frac{1}{n}$ for all possible z_0 , indicating a uniform distribution over the options initially when no observations or actions are available, and $\beta(z_T) = \frac{p(s_T|s_{T-1}, a_{T-1}, z_T)}{p(s_T|s_{0:T-1}, a_{0:T-1})} = 1$.

4.4 Proximal Policy Optimisation via Expectation Maximisation

In this section, PPOEM is introduced, an algorithm that extends PPO for option discovery with an EM objective. The expectation of the returns is taken over the joint probability distribution of states, actions and options, sampled by the policy. This objective gives a tractable objective to maximise, which has a close resemblance to the forward-backward algorithm.

4.4.1 Expected return maximisation objective with options

The objective is to maximise the expectation of returns $R(\tau)$ for an agent policy π over a trajectory τ with latent option z_t at each time step t . The definition of a trajectory τ is a set of states, actions and rewards visited by the agent policy in an episode. The objective $J[\pi]$ can be written as:

$$J[\pi_\Theta] = \mathbb{E}_{\tau, \mathbf{Z} \sim \pi} [R(\tau)] = \int_{\tau, \mathbf{Z}} R(\tau) p(\tau, \mathbf{Z} | \Theta). \quad (4.19)$$

Taking the gradient of the maximisation objective,

$$\begin{aligned} \nabla_{\Theta} J[\pi_\Theta] &= \int_{\tau, \mathbf{Z}} R(\tau) \nabla_{\Theta} p(\tau, \mathbf{Z} | \Theta) = \int_{\tau, \mathbf{Z}} R(\tau) \frac{\nabla_{\Theta} p(\tau, \mathbf{Z} | \Theta)}{p(\tau, \mathbf{Z} | \Theta)} p(\tau, \mathbf{Z} | \Theta) \\ &= \mathbb{E}_{\tau, \mathbf{Z}} [R(\tau) \nabla_{\Theta} \log p(\tau, \mathbf{Z} | \Theta)]. \end{aligned} \quad (4.20)$$

To simplify the derivation, let us focus on the states and actions that appear in the trajectory. The joint likelihood function for the trajectory τ and latent options $\mathbf{Z} =$

$\{z_0, \dots, z_{T-1}\}$ is given by:

$$p(\tau, \mathbf{Z}|\Theta) = p(s_{0:T}, a_{0:T-1}, z_{0:T}|\Theta) = p(s_0, z_0) \prod_{t=0}^{T-1} [p_{\Theta}(a_t, z_{t+1}|s_t, z_t) P(s_{t+1}|s_{0:t}, a_{0:t})], \quad (4.21)$$

Evaluating $\nabla_{\Theta} \log p(\tau, \mathbf{Z}|\Theta)$, the log converts the products into sums, and the terms which are constant with respect to Θ are eliminated upon taking the gradient, leaving

$$\nabla_{\Theta} \log p(\tau, \mathbf{Z}|\Theta) = \sum_{t=0}^{T-1} \nabla_{\Theta} \log [\pi_{\theta}(a_t|s_t, z_t) \pi_{\psi}(z_{t+1}|s_t, a_t, z_t, s_{t+1})]. \quad (4.22)$$

Substituting Equation (4.22) into Equation (4.20) and explicitly evaluating the expectation over the joint option probabilities,

$$\begin{aligned} \nabla_{\Theta} J[\pi_{\Theta}] &= \mathbb{E}_{\tau, \mathbf{Z} \sim \pi} \left[\sum_{t=0}^{T-1} R(\tau) \nabla_{\Theta} \log p_{\Theta}(a_t, z_{t+1}|s_t, z_t) \right] \\ &= \mathbb{E}_{\tau \sim \pi} \int_{\mathbf{Z}} \left[\sum_{t=0}^{T-1} [R(\tau) \nabla_{\Theta} \log p_{\Theta}(a_t, z_{t+1}|s_t, z_t)] \right] p(\mathbf{Z}|\tau) \\ &= \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T-1} \sum_{z_t, z_{t+1}} [R(\tau) p(z_t, z_{t+1}|\tau) \nabla_{\Theta} \log p_{\Theta}(a_t, z_{t+1}|s_t, z_t)] \right]. \end{aligned} \quad (4.23)$$

Using the action probability $\alpha_t := p(a_t|s_{0:t}, a_{0:t-1})$, option forward distribution $\zeta(z_t) := p(z_t|s_{0:t}, a_{0:t-1})$, and option backward feedback $\beta(z_t) := \frac{p(s_{t:T}, a_{t:T-1}|s_{t-1}, a_{t-1}, z_t)}{p(s_{t:T}, a_{t:T-1}|s_{0:t-1}, a_{0:t-1})}$ evaluated in Section 4.3.2, $p(z_t, z_{t+1}|\tau)$ can be evaluated as

$$\begin{aligned} p(z_t, z_{t+1}|\tau) &= \frac{p(s_{0:T}, a_{0:T-1}, z_t, z_{t+1})}{p(s_{0:T}, a_{0:T-1})} \\ &= \frac{p(s_{0:t}, a_{0:t-1}, z_t) p_{\Theta}(a_t, z_{t+1}|s_t, z_t) p(s_{t+1:T}, a_{t+1:T-1}|s_t, a_t, z_{t+1})}{p(s_{0:t}, a_{0:t-1}) p(a_t|s_{0:t}, a_{0:t-1}) p(s_{t+1:T}, a_{t+1:T-1}|s_{0:t}, a_{0:t})} \\ &= p_{\Theta}(a_t, z_{t+1}|s_t, z_t) \frac{\zeta(z_t) \beta(z_{t+1})}{\alpha_t}. \end{aligned} \quad (4.24)$$

Using this, Equation (4.23) can be evaluated and maximised with gradient descent.

Relationship with Expectation Maximisation

The objective derived in Equation (4.23) closely resembles the objective of the EM algorithm applied to the HMM with options as latent variables. The expectation of

the marginal log-likelihood $\mathcal{Q}(\Theta; \Theta_{\text{old}})$, which gives the lower-bound of the marginal log-likelihood $\log p(\tau|\Theta)$, is given by

$$\begin{aligned} \mathcal{Q}(\Theta; \Theta_{\text{old}}) &= \mathbb{E}_{\mathbf{Z} \sim p(\cdot|\tau, \Theta_{\text{old}})} [\ln p(\tau, \mathbf{Z}|\Theta)] = \mathbb{E}_{\tau \sim \pi} \int_{\mathbf{Z}} p(\mathbf{Z}|\tau, \Theta_{\text{old}}) \ln p(\tau, \mathbf{Z}|\Theta) d\mathbf{Z} \\ &= \mathbb{E}_{\tau \sim \pi} \sum_{t=0}^{T-1} \sum_{z_t, z_{t+1}} [p(z_t, z_{t+1}|\tau, \Theta_{\text{old}}) \log p_{\Theta}(a_t, z_{t+1}|s_t, z_t)] + \text{const.} \end{aligned} \quad (4.25)$$

The difference is that the expected return maximisation objective in Equation (4.23) weights the log probabilities of the policy according to the returns, whereas the objective of Equation (4.25) is to find a parameter set Θ that maximises the probability that the states and actions that appeared in the trajectory are visited by the joint option policy p_{Θ} .

4.4.2 PPO objective with Generalised Advantage Estimation

A standard optimisation technique for neural networks using gradient descent can be applied to optimise the policy network. Noticing that the optimisation objective in Equation (4.23) resembles the policy gradient algorithm, the joint option policy can be optimised using the PPO algorithm instead to prevent the updated policy $p_{\Theta}(a_t, z_{t+1}|s_t, z_t)$ from deviating from the original policy too much.

Several changes have to be made to adapt the training objective to PPO. Firstly, $\nabla \log p_{\Theta}$ is replaced by $\frac{\nabla p_{\Theta}}{p_{\Theta_{\text{old}}}}$, its first order approximation, to easily introduce clipping constraints to the policy ratios. Secondly, the return $R(\tau)$ is replaced with the GAE, A_t^{GAE} , as introduced in Section 2.2.4.

Extending the definition of GAE to work with options,

$$A_t^{\text{GAE}}(z_t, z_{t+1}|\tau) = r_t + \gamma V(s_{t+1}, z_{t+1}) - V(s_t, z_t) + \lambda \gamma (1 - d_t) A_{t+1}^{\text{GAE}}(z_{t+1}|\tau) \quad (4.26)$$

$$A_t^{\text{GAE}}(z_t|\tau) = \sum_{z_{t+1}} p(z_{t+1}|z_t, \tau) A_t^{\text{GAE}}(z_t, z_{t+1}|\tau). \quad (4.27)$$

The GAE could be evaluated backwards iteratively, starting from $t = T$ with the initial condition $A_T^{\text{GAE}}(z_{t+1}|\tau) = 0$. The option transition function $p(z_{t+1}|z_t, \tau)$ can be

evaluated using $p(z_t, z_{t+1}|\tau)$ (Equation (4.24)) as:

$$p(z_{t+1}|z_t, \tau) = \frac{p(z_t, z_{t+1}|\tau)}{\sum_{z_{t+1}} p(z_t, z_{t+1}|\tau)}. \quad (4.28)$$

The target value $V_{\text{target}}(s_t, z_t)$ to regress the estimated value function towards can be defined in terms of the GAE and the current value estimate as:

$$V_{\text{target}}(s_t, z_t) = V^\pi(s_t, z_t) + A_t^{\text{GAE}}(z_t|\tau). \quad (4.29)$$

4.5 Sequential Option Advantage Propagation

In the previous section, assignments of the latent option variables \mathbf{Z} were determined by maximising the expected return for complete trajectories. The derived algorithm resembles the forward-backward algorithm closely, and requires the backward pass of $\beta(z_t)$ in order to fully evaluate the option probability $p(\mathbf{Z}|\tau)$. During rollouts of the agent policy, however, knowing the optimal assignment of latents $p(z_t|\tau)$ in advance is not possible, since the trajectory is incomplete and the backward pass has not been initiated. Therefore, the policy must rely on the current best estimate of the options given its available past trajectory $\{s_{0:t}, a_{0:t}\}$ during its rollout. This option distribution conditional only on its past is equivalent to the auto-regressive option forward distribution $\zeta(z_t) := p(z_t|s_{0:t}, a_{0:t-1})$.

Since the optimal option assignment can only be achieved in hindsight once the trajectory is complete, this information is not helpful for the agent policy upon making its decisions. A more useful source of information for the agent, therefore, is the current best estimate of the option assignment $\zeta(z_t)$. It is sensible, therefore, to directly optimise for the expected returns evaluated over the option assignments $\zeta(z_t)$ to find an optimal option policy, rather than optimising the expected returns for an option assignment $p(\mathbf{Z}|\tau)$, which can only be known in hindsight.

The following section proposes a new option optimisation objective that does not involve the backward pass of the EM algorithm. Instead, the option policy gradient for an

optimal forward option assignment is evaluated analytically. This results in a temporal gradient propagation, which corresponds to a backward pass, but with a slightly different outcome. Notably, this improved algorithm, SOAP, applies a normalisation of the option advantages in every back-propagation step through time.

As far as the author is aware, this work is the first to derive the back-propagation of policy gradients in the context of option discovery.

4.5.1 Policy Gradient objective with options

Let us start by deriving the policy gradient objective assuming options. The maximisation objective $J[\pi]$ for the agent can be defined as:

$$J[\pi_{\Theta}] = \mathbb{E}_{\tau \sim \pi} [R(\tau)] = \int_{\tau} R(\tau) p(\tau|\Theta) d\tau. \quad (4.30)$$

Taking the gradient of the maximisation objective,

$$\nabla_{\Theta} J[\pi_{\Theta}] = \int_{\tau} R(\tau) \nabla_{\Theta} p(\tau|\Theta) d\tau = \int_{\tau} R(\tau) \frac{\nabla_{\Theta} p(\tau|\Theta)}{p(\tau|\Theta)} p(\tau|\Theta) d\tau = \mathbb{E}_{\tau} [R(\tau) \nabla_{\Theta} \log p(\tau|\Theta)]. \quad (4.31)$$

So far, the above derivation is the same as the normal policy gradient objective without options. Next, the likelihood for the trajectory τ is given by:

$$p(\tau|\Theta) = p(s_{0:T}, a_{0:T-1}|\Theta) = \rho(s_0) \prod_{t=0}^{T-1} [p(a_t|s_{0:t}, a_{0:t-1}, \Theta) P(s_{t+1}|s_{0:t}, a_{0:t})]. \quad (4.32)$$

This is where options become relevant, as the standard formulation assumes that the policy $\pi(a|s)$ is only dependent on the current state without history, and similarly that the state transition environment dynamics $P(s'|s, a)$ is Markovian given the current state and action. In many applications, however, the *states* that are observed do not contain the entire information about the underlying dynamics of the environment¹, and therefore,

¹Some literature on POMDP choose to make this explicit by denoting the partial observation available to the agent as observation o , distinguishing from the underlying ground truth state s . However, since o can also stand for *options*, and is used in other literature on options, here the input to the agent's policy and value functions is denoted using the conventional s to prevent confusion.

Chapter 4. Option discovery via Expectation Maximisation and Policy Gradients 84

conditioning on the history yields a different distribution of future states compared to conditioning on just the current state. To capture this, the policy and state transitions are now denoted to be $p(a_t|s_{0:t}, a_{0:t-1})$ and $P(s_{t+1}|s_{0:t}, a_{0:t})$, respectively. Here, the probabilities are conditional on the historical observations ($s_{0:t}$) and historical actions (e.g. $a_{0:t}$), rather than just the immediate state s_t and action a_t . Note that $p(a_t|s_{0:t}, a_{0:t-1})$ is a quantity α_t that has already been evaluated in Section 4.3.2.

Evaluating $\nabla_{\Theta} \log p(\tau|\Theta)$, the log converts the products into sums, and the terms which are constant with respect to Θ are eliminated upon taking the gradient, leaving

$$\nabla_{\Theta} \log p(\tau|\Theta) = \sum_{t=0}^{T-1} \nabla_{\Theta} \log p(a_t|s_{0:t}, a_{0:t-1}, \Theta) = \sum_{t=0}^{T-1} \nabla_{\Theta} \log \alpha_t = \sum_{t=0}^{T-1} \frac{\nabla_{\Theta} \alpha_t}{\alpha_t}, \quad (4.33)$$

where α_t is substituted following its definition.

Substituting Equation (4.33) into Equation (4.31),

$$\nabla_{\Theta} J[\pi_{\Theta}] = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T-1} R(\tau) \frac{\nabla_{\Theta} \alpha_t}{\alpha_t} \right]. \quad (4.34)$$

Similarly to Section 4.4.2, it is possible to substitute the return $R(\tau)$ with GAE, thereby reducing the variance in the return estimate. Extending the definition of GAE to work with options,

$$A_t^{\text{GAE}}(z_t, z_{t+1}) = r_t + \gamma V(s_{t+1}, z_{t+1}) - V(s_t, z_t) + \lambda \gamma (1 - d_t) A_{t+1}^{\text{GAE}}(z_{t+1}), \quad (4.35)$$

$$A_t^{\text{GAE}}(z_t) = \sum_{z_{t+1}} p(z_{t+1}|s_t, a_t, z_t) A_t^{\text{GAE}}(z_t, z_{t+1}), \quad (4.36)$$

$$V_{\text{target}}(s_t, z_t) = V^{\pi}(s_t, z_t) + A_t^{\text{GAE}}(z_t). \quad (4.37)$$

Notice that, while the definition of these estimates is almost identical to Section 4.4.2, the advantages are now propagated backwards via the option transition $p(z_{t+1}|s_t, a_t, z_t)$ rather than $p(z_{t+1}|z_t, \tau)$.

Substituting the GAE into Equation (4.34),

$$\begin{aligned} \nabla_{\Theta} J[\pi_{\Theta}] &= \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T-1} \frac{\sum_{z_t} A_t^{\text{GAE}}(z_t) \zeta(z_t)}{\alpha_t} \nabla_{\Theta} \alpha_t \right] \\ &= \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T-1} \frac{\sum_{z_t} A_t^{\text{GAE}}(z_t) \zeta(z_t)}{\alpha_t} \sum_{z_t, z_{t+1}} [p_{\Theta}(a_t, z_{t+1} | s_t, z_t) \nabla \zeta(z_t) + \zeta(z_t) \nabla p_{\Theta}(a_t, z_{t+1} | s_t, z_t)] \right]. \end{aligned} \quad (4.38)$$

4.5.2 Analytic back-propagation of the policy gradient

If a forward pass of the policy can be made in one step over the entire trajectory, a gradient optimisation on the objective can be performed directly. However, this would require storing the entire trajectory in GPU memory, which is highly computationally intensive. Instead, this section analytically evaluates the back-propagation of gradients of the objective so that the model can be trained on single time-step rollout samples during training.

Gradient terms appearing in Equation (4.38) are either $\nabla \zeta(z_t)$ or $\nabla p_{\Theta}(a_t, z_{t+1} | s_t, z_t)$ for $0 \leq t \leq T - 1$. While $p_{\Theta}(a_t, z_{t+1} | s_t, z_t)$ is approximated by neural networks and can be differentiated directly, $\nabla \zeta(z_{t+1})$ has to be further expanded to evaluate the gradient in recursive form as:

$$\begin{aligned} \nabla \zeta(z_{t+1}) &= \frac{\nabla \sum_{z_t} \zeta(z_t) p_{\Theta}(a_t, z_{t+1} | s_t, z_t)}{\alpha_t} - \zeta(z_{t+1}) \frac{\nabla \alpha_t}{\alpha_t} \\ &= \frac{1}{\alpha_t} \left[\sum_{z_t} \nabla [\zeta(z_t) p_{\Theta}(a_t, z_{t+1} | s_t, z_t)] - \zeta(z_{t+1}) \sum_{z'_t, z'_{t+1}} \nabla [\zeta(z'_t) p_{\Theta}(a_t, z'_{t+1} | s_t, z'_t)] \right]. \end{aligned} \quad (4.39)$$

Using Equation (4.39), it is possible to rewrite the $\nabla \zeta(z_{t+1})$ terms appearing in Equation (4.38) in terms of $\nabla \zeta(z_t)$ and $\nabla p_{\Theta}(a_t, z_{t+1} | s_t, z_t)$. Defining the coefficients of $\nabla \zeta(z_t)$ in Equation (4.38) as option utility $U(z_t)$,

$$\begin{aligned}
 \sum_{z_{t+1}} U(z_{t+1}) \nabla \zeta(z_{t+1}) &= \frac{1}{\alpha_t} \sum_{z_t, z_{t+1}} \left[U(z_{t+1}) - \sum_{z'_{t+1}} U(z'_{t+1}) \zeta(z'_{t+1}) \right] \nabla [\zeta(z_t) p_{\Theta}(a_t, z_{t+1} | s_t, z_t)] \\
 &= \frac{1}{\alpha_t} \sum_{z_t, z_{t+1}} \left[U(z_{t+1}) - \sum_{z'_{t+1}} U(z'_{t+1}) \zeta(z'_{t+1}) \right] \\
 &\quad \cdot [p_{\Theta}(a_t, z_{t+1} | s_t, z_t) \nabla \zeta(z_t) + \zeta(z_t) \nabla p_{\Theta}(a_t, z_{t+1} | s_t, z_t)].
 \end{aligned} \tag{4.40}$$

Applying this iteratively to Equation (4.38), starting with $t = T - 1$ in reverse order, Equation (4.38) could be expressed solely in terms of gradients $\nabla p_{\Theta}(a_t, z_{t+1} | s_t, z_t)$. Defining the coefficients of $\nabla p_{\Theta}(a_t, z_{t+1} | s_t, z_t)$ as policy gradient weighting $W_t(z_t, z_{t+1})$,

$$\begin{aligned}
 A_t^{\text{GOA}}(z_{t+1}) &= \sum_{z_t} A_t^{\text{GAE}}(z_t) \zeta(z_t) + (1 - d_t) \left[U(z_{t+1}) - \sum_{z'_{t+1}} U(z'_{t+1}) \zeta(z'_{t+1}) \right], \\
 U(z_t) &= \frac{\sum_{z_{t+1}} A_t^{\text{GOA}}(z_{t+1}) p_{\Theta}(a_t, z_{t+1} | s_t, z_t)}{\alpha_t}, \\
 W(z_t, z_{t+1}) &= \frac{A_t^{\text{GOA}}(z_{t+1}) \zeta(z_t)}{\alpha_t}.
 \end{aligned} \tag{4.41}$$

where $A_t^{\text{GOA}}(z_{t+1})$ is a new quantity derived and introduced in this work as Generalised Option Advantage (GOA), which is a term that appears in evaluating $U(z_t)$ and $W(z_t, z_{t+1})$.

Rewriting the policy gradient objective in Equation (4.38) with the policy gradient weighting,

$$\nabla_{\Theta} J[\pi_{\Theta}] = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T-1} \sum_{z_t, z_{t+1}} \frac{A_t^{\text{GOA}}(z_{t+1}) \zeta(z_t)}{\alpha_t} \nabla_{\Theta} p_{\Theta}(a_t, z_{t+1} | s_t, z_t) \right]. \tag{4.42}$$

4.5.3 Learning objective for option-specific policies and values

The training objective given in Equation (4.42) is modified so that it could be optimised with PPO. Unlike in Section 4.4.2, the training objective is written in terms of ∇p_{Θ} and not $\nabla \log p_{\Theta}$. Therefore, the clipping constraints are applied to p_{Θ} directly, limiting it to

the range of $(1 - \epsilon)p_{\Theta_{\text{old}}}$ and $(1 + \epsilon)p_{\Theta_{\text{old}}}$. The resulting PPO objective is:

$$J_{\Theta} = \mathbb{E}_{s_t, a_t \sim \pi} \sum_{z_t, z_{t+1}} \left[\frac{\zeta(z_t)}{\alpha_t} \min \left(\pi_{\Theta}(a_t, z_{t+1} | s_t, z_t) A_t^{\text{GOA}}(z_{t+1}), \right. \right. \\ \left. \left. \text{clip} \left(\pi_{\Theta}(a_t, z_{t+1} | s_t, z_t), (1 - \epsilon)\pi_{\Theta_{\text{old}}}(a_t, z_{t+1} | s_t, z_t), (1 + \epsilon)\pi_{\Theta_{\text{old}}}(a_t, z_{t+1} | s_t, z_t) \right) A_t^{\text{GOA}}(z_{t+1}) \right) \right]. \quad (4.43)$$

The option-specific value function $V_{\phi}^{\pi}(s_t, z_t)$ parameterised by ϕ can be learnt by regressing towards the target values $V_{\text{target}}(s_t, z_t)$ evaluated in Equation (4.37) for each state s_t and option z_t sampled from the policy and option-forward probability, respectively. Defining the objective function for the value regression as J_{ϕ} ,

$$J_{\phi} = - \mathbb{E}_{s_t \sim \pi, z_t \sim \zeta} [V_{\text{target}}(s_t, z_t) - V_{\phi}^{\pi}(s_t, z_t)]^2. \quad (4.44)$$

The final training objective is to maximise the following:

$$J_{\text{SOAP}} = J_{\Theta} + J_{\phi}. \quad (4.45)$$

4.6 Experiments

Experiments were conducted on a variety of RL agents: PPO, PPOC, Proximal Policy Optimisation with Long Short-Term Memory (PPO-LSTM), DAC, PPOEM (ours), and SOAP (ours). PPO [194] is a baseline without memory, PPOC [111] implements the Option-Critic algorithm using PPO for policy optimisation, PPO-LSTM implements a recurrent policy with latent states using an LSTM, DAC [255] optimises both the inter- and intra-option policies on hierarchical MDPs, PPOEM is the algorithm developed in the first half of this chapter that optimises the expected returns using the forward-backward algorithm, and SOAP is the final algorithm proposed in this chapter that uses an option advantage derived by analytically evaluating the temporal propagation of the option policy gradients. SOAP mitigates the deficiency of PPOEM that the training objective optimises the option assignments over a full trajectory which is typically only available in hindsight;

Table 4.1: Normalised performance comparison of RL agents. The agent scores are the returns after the maximum environment steps during training (100k for CartPole, 1M for LunarLander and MuJoCo environments, and 10M for Atari environments), normalised so that the score of a random agent is 0 and the score of the best performing model is 1. Scores are averaged per environment class (i.e. results for the corridor environments, Atari, and MuJoCo are grouped together) and the **bold fonts** show the best average normalised score per environment class, while the **blue fonts** show the best normalised score per environment.

Environment	PPO	PPOC	PPO-LSTM	DAC	PPOEM (ours)	SOAP (ours)
Corridor	-0.05	0.31	0.43	0.65	0.60	1.00
$L = 3$	-0.08	0.76	1.00	0.90	0.99	1.00
$L = 10$	-0.01	0.06	0.36	0.63	0.70	1.00
$L = 20$	-0.05	0.11	-0.06	0.41	0.12	1.00
CartPole	1.00	0.80	0.98	1.00	0.98	1.00
LunarLander	0.86	0.74	1.00	0.78	0.99	0.99
Atari	0.93	0.22	0.78	0.85	0.74	0.89
Asteroids	0.83	0.81	0.64	1.00	0.93	0.89
Beam Rider	0.93	0.13	0.37	0.70	0.66	1.00
Breakout	1.00	0.01	0.68	0.95	0.14	0.92
Enduro	0.97	0.00	0.90	0.93	1.00	0.82
Ms Pacman	0.88	0.15	0.74	0.87	0.69	1.00
Pong	1.00	0.48	1.00	1.00	0.94	1.00
Qbert	1.00	0.00	0.97	0.65	0.70	0.90
Road Runner	1.00	0.15	0.92	0.88	0.52	0.81
Seaquest	0.89	0.18	0.87	0.69	1.00	0.55
Space Invaders	0.82	0.32	0.73	0.82	0.82	1.00
MuJoCo	0.97	0.60	0.75	0.82	0.43	0.93
Ant	1.00	0.07	0.46	0.64	0.07	0.87
Half Cheetah	1.00	0.80	0.83	0.79	0.01	0.92
Humanoid	0.98	0.96	0.96	1.00	0.13	0.90
Reacher	0.99	0.99	0.99	1.00	0.98	1.00
Swimmer	0.96	0.34	0.97	1.00	0.95	0.90
Walker	0.85	0.47	0.31	0.52	0.44	1.00

SOAP optimises the option assignments given only the history of the trajectory instead, making the optimisation objective better aligned with the task objective.

The aim is to (a) show and compare the option learning capability of the newly developed algorithms, and (b) assess the stability of the algorithms on standard RL en-

vironments. All algorithms use PPO as the base policy optimiser, and share the same backbone and hyperparameters, making it a fair comparison. All algorithms use Stable Baselines 3 [170] as a base implementation with the recommended tuned hyperparameters for each environment. In the following experiments, the number of options was set to 4.

4.6.1 Option learning in corridor environments

A simple environment of a corridor with a fork at the end is designed as a minimalistic and concrete example where making effective use of latent variables to retain information over a sequence is necessary to achieve the agent’s goal.

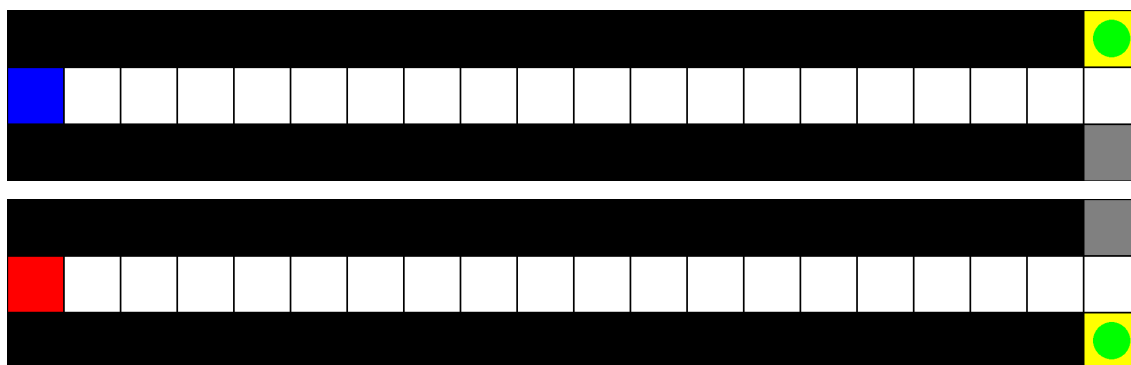


Figure 4.4: A corridor environment. The above example has a length $L = 20$. The agent represented as a green circle starts at the left end of the corridor, and moves towards the right. When it reaches the right end, the agent can either take an up action or a down action. This will either take the agent to a yellow cell or a grey cell. The yellow cell gives a reward of 1, while the grey cell gives a reward of -1 . All other cells give a reward of 0. The location of a rewarding yellow cell and the penalising grey cell are determined by the colour of the starting cell (either "blue" or "red"), as shown, and this is randomised, each with 50% probability. The agent only has access to the colour of the current cell as observation. For simplicity of implementation, the agent’s action space is {"up", "down"}, and apart from the fork at the right end, taking either of the actions at each time step will move the agent one cell to the right. The images shown are taken from rollouts of the SOAP agent after training for $100k$ steps. The agent successfully navigated to the rewarding cell in both cases.

Figure 4.4 describes the corridor environment, in which the agent has to determine whether the rewarding cell (coloured yellow) is at the top or bottom, based on the colour of the cell it has seen at the start (either "blue" or "red"). However, the agent only has access to the colour of the current cell, and does not have a bird’s-eye-view of the environment.

Chapter 4. Option discovery via Expectation Maximisation and Policy Gradients 90

Hence, the agent must retain the information of the colour of the starting cell in memory, whilst discarding all other information irrelevant to the completion of the task. The agent must learn that the information of the colour of the starting cell is important to task completion in an unsupervised way, just from the reward signals. This makes the task challenging, as only in hindsight (after reaching the far end of the corridor) is it clear that this information is useful to retain in memory, but if this information was not written in memory in the first place then credit assignment becomes infeasible.

The length of the corridor L can be varied to adjust the difficulty of the task. It is increasingly challenging to retain the information of the starting cell colour with longer corridors. In theory, this environment can be solved by techniques such as frame stacking, where the entire history of the agent observations is provided to the policy. However, the computational complexity of this approach scales proportionally to corridor length L , which makes this approach unscalable.

Algorithms with options present an alternative solution, where in theory, the options can be used as latent variables to carry the information relevant to the task. In this experiment, PPOC, PPO-LSTM, DAC, PPOEM and SOAP are compared against a standard PPO algorithm. The results are shown for corridors with lengths $L = 3$, $L = 10$ and $L = 20$. Due to the increasing level of difficulty of the task, the agents are trained with $8k$, $40k$ and $100k$ time steps of environment interaction, respectively.

The results are shown in Figure 4.5 and Table B.1, and a performance score normalised to the range of the returns of a random agent score and the returns of the best agent is shown in Section 4.6. As expected, the vanilla PPO agent does not have any memory component so it learnt a policy that takes one action deterministically regardless of the colour of the first cell. Since the location of the rewarding cell is randomised, this results in an expected return of 0.

With PPOC that implements the Option-Critic architecture, and DAC that implements

HRL using options, while the options should in theory be able to retain information from the past, it could be observed that the training objective was not sufficient to learn a useful option assignment to complete the task. PPOEM and SOAP, on the other hand, were able to learn to select a different option for a different starting cell colour. From Figure 4.5, it could be seen that the two algorithms had identical performance for a short corridor, but as the corridor length L increased, the performance of PPOEM deteriorated, while SOAP was able to reliably find a correct option assignment, albeit with more training steps.

There are several major differences between the baseline option-based algorithms (PPOC and DAC) and the proposed algorithms (PPOEM and SOAP) which could be contributing to their significant differences in performance. Firstly, while the option transition function in PPOEM and SOAP are in the form of $\pi_\phi(z_{t+1}|s_t, a_t, z_t)$, which allows the assignment of the new option to be conditional on the current option, the option transition in the Option-Critic architecture is decoupled into an option termination probability $\varpi(s_t, z_{t-1})$, and an unconditional inter-option policy $\pi(z_t|s_t)$. This means that whenever the previous option z_{t-1} is terminated with probability $\varpi(s_t, z_{t-1})$, the choice of the new option z_t will be uninformed of the previous option z_{t-1} , whereas in PPOEM and SOAP the probability of the next option z_{t+1} is conditional on the previous option z_t . The formulation of DAC [255] does not have this specific constraint; however, the original implementation by the authors similarly decouples the option transition function such that a new option cannot be fully conditioned on the previous option.

Secondly, both PPOC and DAC rely on learning an option-value function (a value function $V(s, o)$ that is both conditional on the current state s and the current option o) to learn the high-level inter-option policy. However, learning an option-value function for an optimal policy can only happen once the inter-option policy is properly learned, but since the inter-option policy is randomly initialised and the option assignment carries little information, it is not possible for the sub-policies to learn optimal policies. In the

case where the agents’ history carries information about the future (e.g. the corridor environment), it is important that the option assignment correctly captures this information without it being lost for the agent to be able to learn an optimal policy. Due to this chicken-and-an-egg problem of learning the inter- and the intra-option policies, neither of these approaches succeeds. In contrast, SOAP directly propagates gradients backwards over multiple timesteps so that the option assignments are directly updated.

Thirdly, in PPOC and DAC, a new option is sampled at every time step, but the complete option forward distribution given the history is not available as a probability distribution. In contrast, in PPOEM and SOAP this is available as $\zeta(z_t) := p(z_t | s_{0:t}, a_{0:t-1})$. Evaluating expectations over distributions gives a more robust estimate of the objective function compared to taking a Monte Carlo estimate of the expectations with the sampled options, which is another explanation of why PPOEM and SOAP were able to learn better option assignments than PPOC or DAC.

SOAP’s training objective maximises the expectation of returns taken over an option probability conditioned only on the agent’s past history, whereas PPOEM’s objective assumes a fully known trajectory to be able to evaluate the option assignment probability. Since option assignments have to be determined online during rollouts, the training objective of SOAP better reflects the task objective. This explains its more reliable performance for longer sequences.

PPO-LSTM achieved competitive performance in a corridor with $L = 3$, demonstrating the capability of latent states to retain past information, but its performance quickly deteriorated for longer corridors. It could be hypothesised that this is because the latent state space of the recurrent policies is not well constrained, unlike options which take discrete values. Learning a correct value function $V(s, z)$ requires revisiting the same state-latent pair. It is conceivable that with longer sequence lengths during inference time, the latent state will fall within a region that has not been trained well due to compounding

noise, leading to an inaccurate estimate of the values and sub-policy.

4.6.2 Stability of the algorithms on CartPole, LunarLander, Atari, and MuJoCo environments

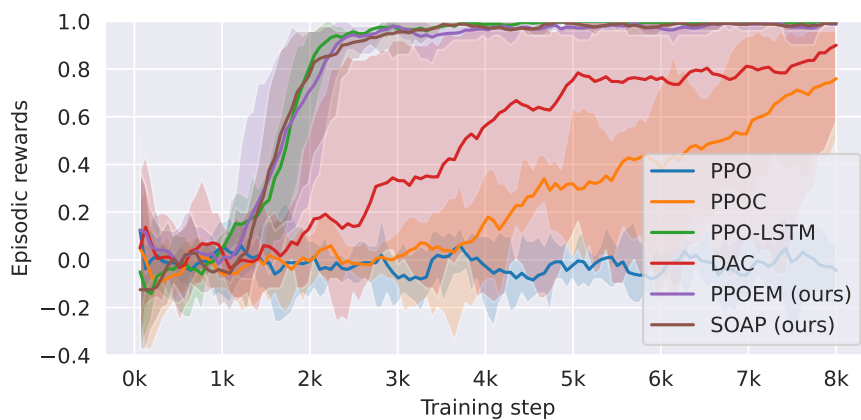
Experiments were also conducted on standard RL environments to evaluate the stability of the algorithms with options. Results for CartPole-v1 and LunarLander-v2 are shown in Figure 4.6, and results on 10 Atari environments [16] and 6 MuJoCo environments [222] are shown in Figure 4.7 and Figure 4.8, respectively. Table B.1 summarises the agent scores after the maximum environment steps during training ($100k$ for CartPole, $1M$ for LunarLander and MuJoCo environments, and $10M$ for Atari environments), and Section 4.6 shows the scores normalised so that the score of a random agent is 0 and the score of the best performing model is 1. (If an agent’s final score is lower than a random agent it can have negative normalised scores.) There was no significant difference in performances amongst the algorithms for simpler environments such as CartPole and LunarLander, with PPOC and DAC having slightly worse performance than others. For the Atari and MuJoCo environments, however, there was a consistent trend that SOAP achieves similar performances (slightly better in some cases, slightly worse in others) to the vanilla PPO, while PPOEM, PPO-LSTM and PPOC were significantly less stable to train. It could be hypothesised that, similarly to Section 4.6.1, the policy of PPOC disregarded the information of past options when choosing the next option, which is why the performance was unstable with larger environments. Another point of consideration is that, with N number of options, there are N number of sub-policies to train, which becomes increasingly computationally expensive and requires many visits to the state-option pair in the training data, especially when using a Monte Carlo estimate by sampling the next option as is done in PPOC and DAC instead of maintaining a distribution of the option $\zeta(z_t)$ as in PPOEM and SOAP. As for PPO-LSTM, similar reasoning as in Section 4.6.1

suggests that with complex environments with a variety of trajectories that can be taken through the state space, the latent states that could be visited increases combinatorially, making it challenging to learn a robust sub-policy and value functions.

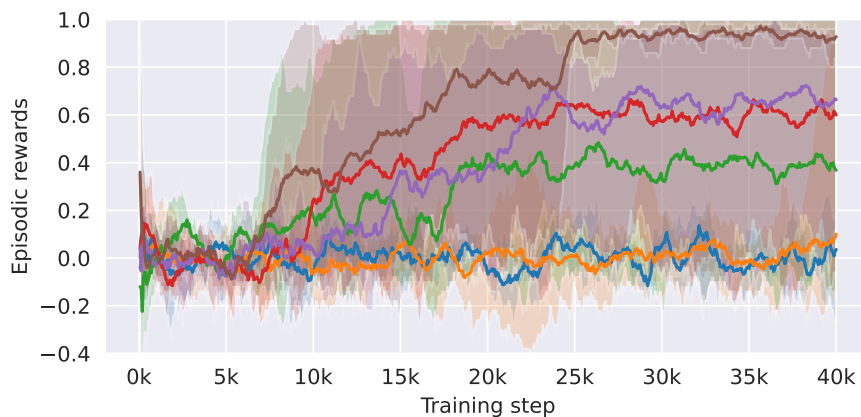
4.7 Conclusion

Two competing algorithms, PPOEM and SOAP, are proposed to solve the problem of option discovery and assignments in an unsupervised way. PPOEM implements a training objective of maximising the expected returns using the EM algorithm, while SOAP analytically evaluates the policy gradient of the option policy to derive an option advantage function that facilitates temporal propagation of the policy gradients. These approaches have an advantage over Option-Critic architecture in that (a) the option distribution is analytically evaluated rather than sampled, and (b) the option transitions are fully conditional on the previous option, allowing historical information to propagate forward in time beyond the temporal window provided as observations.

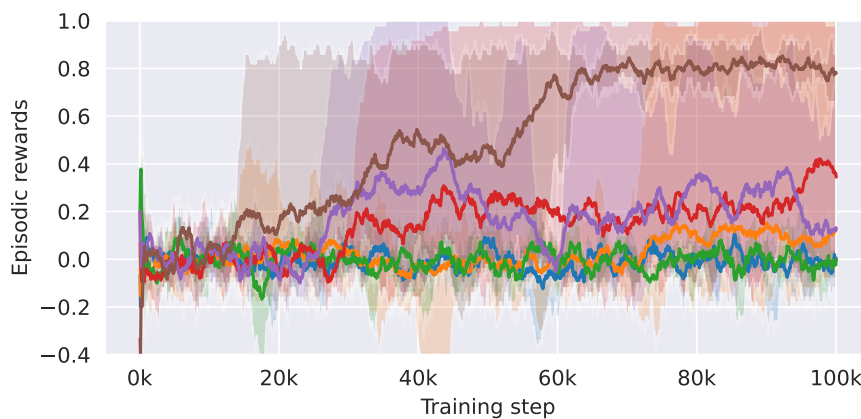
Experiments in POMDP corridor environments designed to require options showed that SOAP is the most robust way of learning option assignments that adhere to the task objective. SOAP also maintained its performance when solving MDP tasks without the need for options (e.g. Atari with frame-stacking), whereas PPOC, DAC, PPO-LSTM and PPOEM were less stable when solving these problems.



(a) Corridor of length $L = 3$



(b) Corridor of length $L = 10$



(c) Corridor of length $L = 20$

Figure 4.5: Training curves of RL agents showing the episodic rewards obtained in the corridor environment with varying corridor lengths. The mean (solid line) and the min-max range (coloured shadow) for 5 seeds per algorithm are shown.

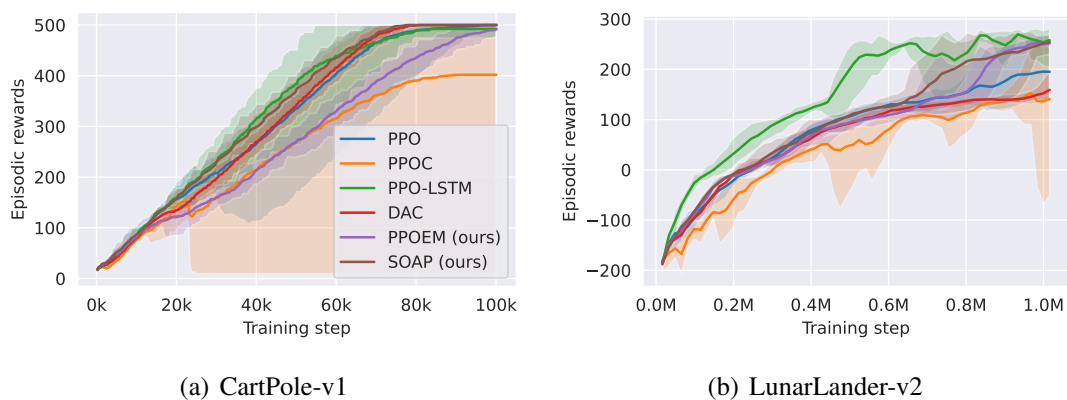


Figure 4.6: Training curves of RL agents showing the episodic rewards obtained in the CartPole-v1 and LunarLander-v2 environments. The mean (solid line) and the min-max range (coloured shadow) for 5 seeds per algorithm are shown.

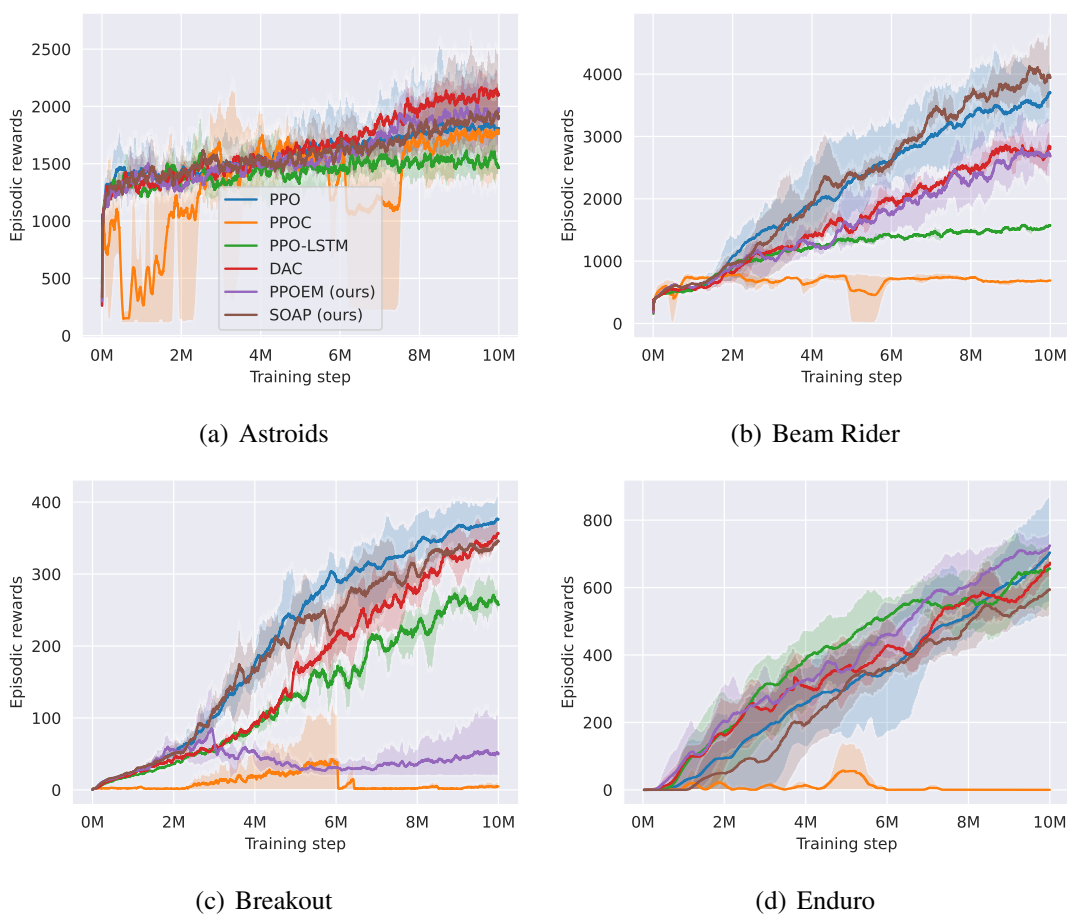
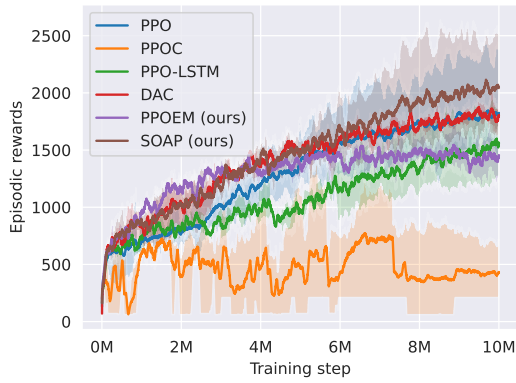
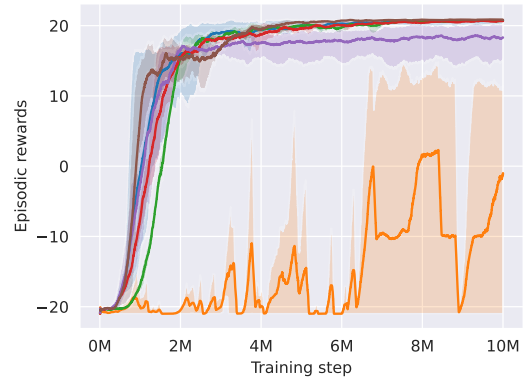


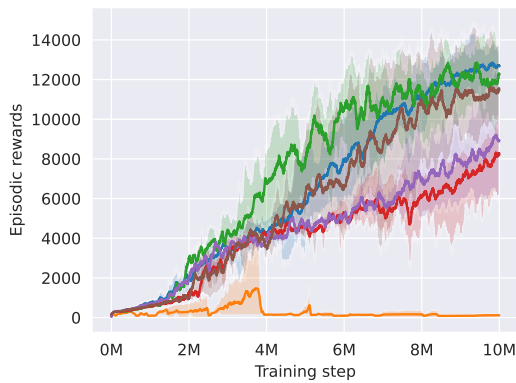
Figure 4.7: Training curves of RL agents showing the episodic rewards obtained in the Atari environments. The mean (solid line) and the min-max range (coloured shadow) for 3 seeds per algorithm are shown. [Spans multiple pages]



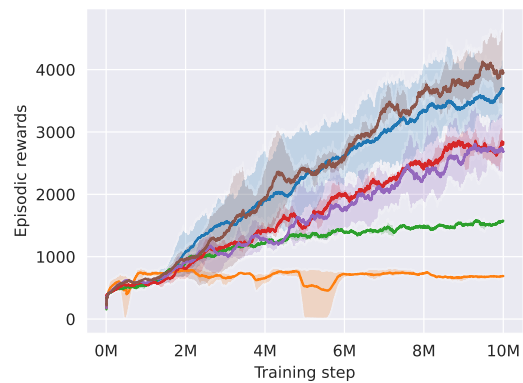
(e) Ms Pacman



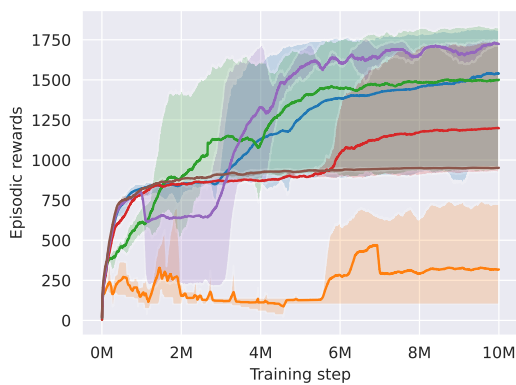
(f) Pong



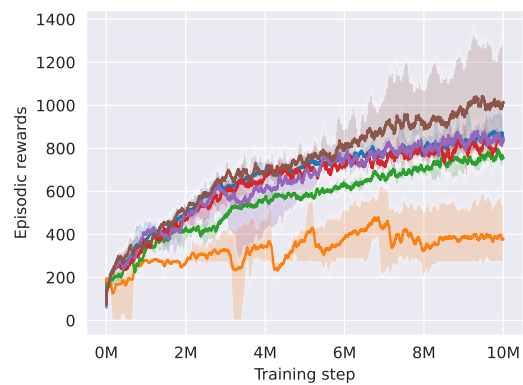
(g) Qbert



(h) Road Runner



(i) Seaquest



(j) Space Invader

[Continued] Training curves of RL agents showing the episodic rewards obtained in the Atari environments. The mean (solid line) and the min-max range (coloured shadow) for 3 seeds per algorithm are shown.

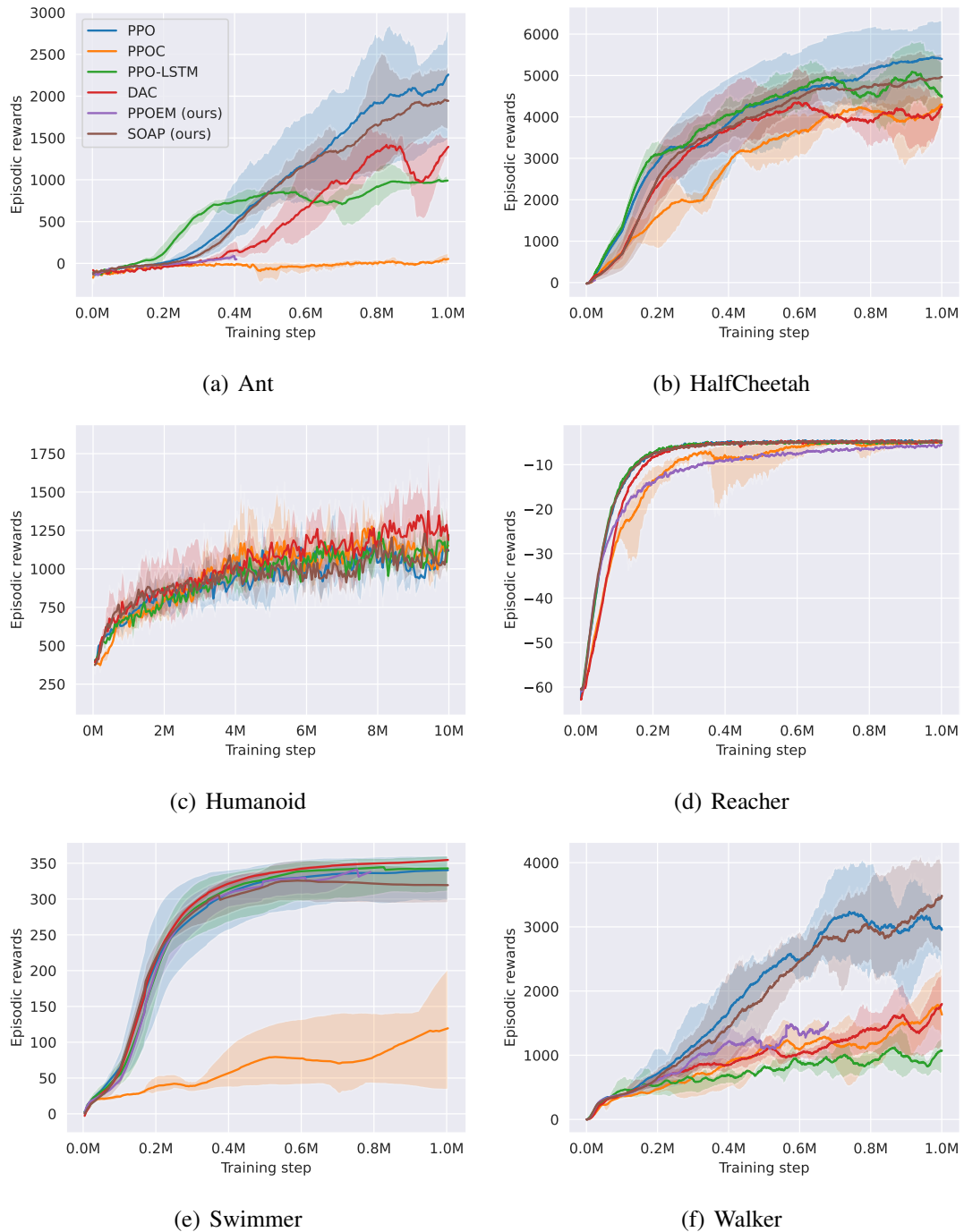


Figure 4.8: Training curves of RL agents showing the episodic rewards obtained in the MuJoCo environments. The mean (solid line) and the min-max range (coloured shadow) for 3 seeds per algorithm are shown. Note that the PPOEM algorithm failed mid-way in some cases due to training instabilities.

Chapter 5

A code optimisation framework using Large Language Models

This chapter proposes LangProp, a framework for iteratively optimising code generated by Large Language Models (LLMs), in both supervised and Reinforcement Learning (RL) settings. While LLMs can generate sensible coding solutions zero-shot (i.e. without having to fine-tune the model to a specific problem domain), they are often sub-optimal. Especially for code generation tasks, it is likely that the initial code will fail on certain edge cases. LangProp automatically evaluates the code performance on a dataset of input-output pairs, catches any exceptions, and feeds the results back to the LLM in the training loop, so that the LLM can iteratively improve the code it generates. By adopting a metric- and data-driven training paradigm for this code optimisation procedure, one could easily adapt findings from traditional machine learning techniques such as Behavioural Cloning (BC), DAgger, and RL.

LangProp demonstrates applicability to general domains such as Sudoku and CartPole, as well as a first proof of concept of automated code optimisation for autonomous driving in CARLA. LangProp can generate interpretable and transparent policies that can be verified and improved in a metric- and data-driven way.

This research was conducted during an internship at Wayve Technologies as a work placement for the Autonomous Intelligent Machines and Systems Centre for Doctoral

Training programme (AIMS CDT). The work was performed under the supervision of Anthony Hu and Gianluca Corrado, with mentorship by members of the world modelling team at Wayve. The development of the algorithm and the code is entirely my own work. Our paper was accepted at the International Conference on Learning Representations (ICLR) 2024 Workshop on Large Language Model (LLM) Agents [100].

5.1 Introduction

Building systems that can self-improve with data is at the core of the machine learning paradigm. By leveraging vast amounts of data and having an automated feedback loop to update models according to an objective function, machine learning methods can directly optimise the metrics of interest, thus outperforming systems that are handcrafted by experts. In the early history of Artificial Intelligence (AI), Symbolic AI, e.g. rule-based expert systems [85, 101], was a dominant and perhaps a more intuitive and explainable approach to solving tasks in an automated way, and is still widely used in fields such as medicine [1] and autonomous driving [11]. However, there have been numerous successes in recent decades in machine learning, e.g. deep neural networks, that demonstrate the advantage of data-driven learning.

Advances in LLMs [23, 155, 223] were enabled by neural networks. Trained on both natural language and code, they can translate human intent and logic into executable code and back, expanding the boundaries of applying logic and reasoning. Unlike other machine learning techniques, LLMs have an affinity with Symbolic AI since they operate in discrete symbolic input-output spaces. The generated outputs are interpretable, even though the internal representation of these tokens is in a continuous embedding space. This observation led to the question of whether it is possible to have the best of both worlds – having an interpretable and transparent system, characteristic of Symbolic AI, which can self-improve in a data-driven manner, following the machine learning paradigm. This

work hypothesises that LLMs provides the missing piece of the puzzle; the optimisation mechanism.

A direct analogy can be drawn from training neural networks, and *train* symbolic systems by leveraging the power of LLMs to interpret and generate scripts. Using this analogy, an LLM can be considered as an *optimiser* equivalent to stochastic gradient descent or Adam. The actual *model* in this new paradigm is an object that handles the initialisation and updates of *parameters* as well as the forward pass logic, where the *parameters* are a collection of symbolic scripts that the LLM generates. At every iteration, a forward pass through the model is performed, comparing it against the ground truth in the dataset, and passing the scores and feedback into the LLM which interprets the results and updates the scripts in a way that fixes the issues raised.

While many methods use LLMs for code generation, and systems such as Auto-GPT [177] iteratively query LLMs to execute tasks in an agent-like manner, LangProp is the first to completely translate and apply the training paradigm used in machine learning for iterative code generation. This work draws inspiration from VOYAGER [230], which introduced the idea that a collection of LLM-generated code (skill library) can be considered as sharable and fine-tunable *checkpoints*. However, VOYAGER's method is specific to Minecraft, and additional work is needed to apply its approach to other domains. LangProp is proposed, a code optimisation framework that is easily adaptable to many application domains.

LangProp is formulated as a general code optimisation framework, decoupled from a specific application domain. It is applied first to the simple settings of Sudoku puzzles and inverted pendulum control (CartPole) to show its task-agnostic nature. Then, the framework is applied to find a driving policy for a more complex autonomous driving challenge.

Autonomous driving is a key area in which model interpretability and transparency

are critical. LangProp is a valuable proof of concept for building interpretable and language-instructable systems in a more automated and learnable way. This work combines both the benefit of interpretability of expert systems while also taking a data-driven approach, exposing the system to potential failure modes and adverse scenarios during training time and iteratively optimising the system towards a well-defined driving metric so that the resulting system is robust to adverse events and potential errors in intermediate components.

The main hypotheses of this work are: (a) LangProp can generate interpretable code that learns to control a vehicle, (b) LangProp can improve driving performance with more training data in comparison to zero-shot code generation, and (c) machine learning training paradigms such as BC, RL [213] and DAgger [180] can be easily transferred and applied to LangProp training.

5.2 Background

5.2.1 LLMs for code generation

Transformers [226] trained on code generation tasks have shown outstanding performances [33, 71, 124, 125, 151, 182]. Furthermore, general-purpose LLMs [155, 156] trained on a large corpus of books and online text have shown remarkable capabilities of translating between natural language and code. However, there is no guarantee that the generated code is error-free. Benchmarks have been suggested to evaluate LLMs on the code generation quality [33, 129].

Code generation with execution is highly relevant to this work. Cobbe et al. [38] and Li et al. [125] used majority voting on the execution results to select code from a pool of candidates. but this is prone to favouring common wrong solutions over correct solutions. Ni et al. [149] suggested a ranking mechanism using a learned verifier to assess code

correctness. CLAIRIFY [205] implemented automatic iterative prompting that catches errors and provides feedback to the LLM until all issues are resolved.

Tangentially related fields are Automated Program Repair [245, 246], unit test generation [181], and planning for code generation [118, 256]. APR is typically solved as a text infill task by identifying an erroneous block of code, masking it out, and querying an LLM, providing the surrounding code as context. Planning for LLMs formulates code generation as a sequence generation task and applies RL techniques. , considering the current code as the state and the action is either generation of the next token in the code [118] or transition to a refined code [256]. While orthogonal to the approach in this work of iteratively generating code using a pre-trained general-purpose LLM as an optimiser, findings from these fields may be compatible with LangProp for future work.

5.2.2 LLMs for automated task completion

Literature on the use of LLMs for automated task completion is discussed in Section 2.4.3. Unlike this work, the above methods require an LLM in the loop during inference, whereas the method proposed in this work only requires access to an LLM during the code optimisation stage.

This work is inspired by VOYAGER [230], which integrates environment feedback, execution errors, and self-verification into an iterative prompting mechanism for embodied control in Minecraft. VOYAGER maintains a *skill library*, a collection of verified reusable code, which can be considered as *checkpoints*. However, there is no mechanism to optimise or remove a sub-optimal skill in the skill library. This limitation is addressed in this work, which presents a more general code optimisation framework that can be applied to a variety of domains, including autonomous driving.

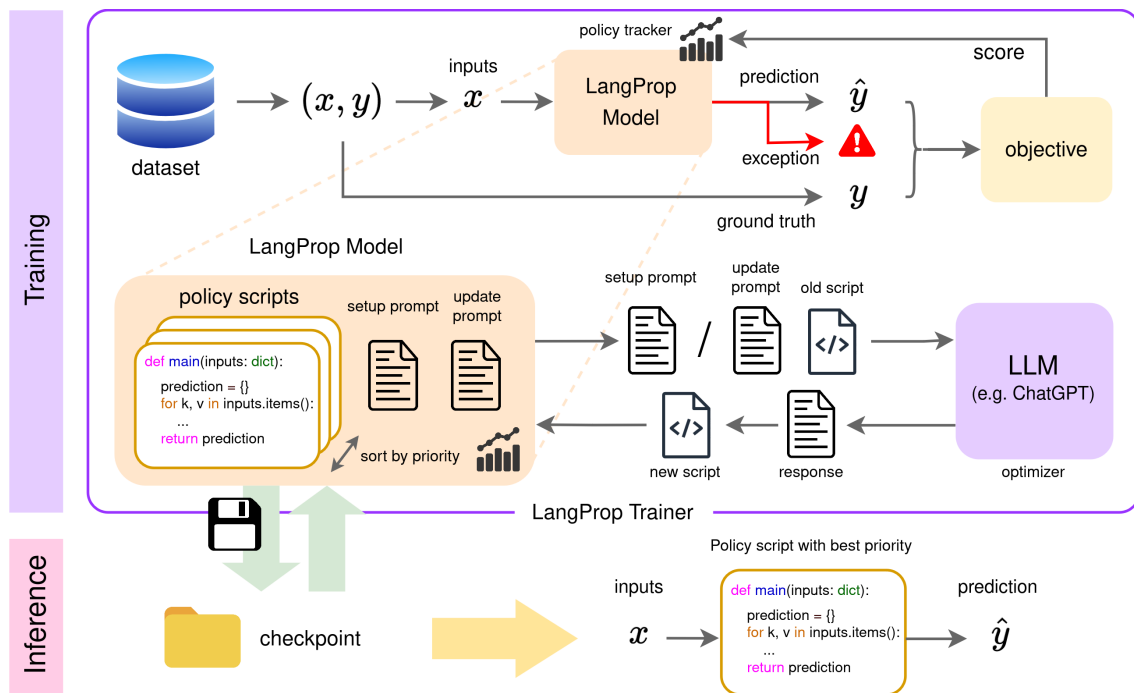


Figure 5.1: An overview of the LangProp framework, which consists of a LangProp model, an LLM optimiser, and a LangProp trainer. During training, the LLM generates and updates the policy scripts which are evaluated against a training objective. Policies with higher performances are selected for updates, and the best policy is used for inference.

5.3 The LangProp Framework

The LangProp framework, shown in Figure 5.1, addresses a general task of optimising code on a given metric of success in a data-driven way, similar to how a neural network is optimised on an objective function. LangProp performs iterative prompting to improve code performance, using the inputs, outputs, exceptions, metric scores, and any environmental feedback to inform the LLM upon updates. The updates in LangProp are performed using a form of an evolutionary algorithm [8]. The following sections describe the key concepts in LangProp in more detail.

5.3.1 Model definition

The LangProp model consists of a setup prompt, an update prompt, and a collection of executable code generated by the LLM, which this work will refer to as *policies*. While

neural models are parameterised by floating-point weights, the *parameters* of a LangProp model is the set of policies. Each policy is associated with an executable *script* as well as a statistics tracker, which updates the *priority*, an aggregate measure of the policy's performance with respect to the training objective. The priority is used to rerank policies so that the best-performing policies are used for updates and inference.

Policy setup

The initialisation of the policies is done similarly to zero-shot code generation. The definition and specification of the requested function are given as a docstring of the function, including the names and types of the inputs and outputs, what the function is supposed to achieve, and a template for the function. Chain-of-Thought prompting [236] is also adopted. Examples of setup prompts can be found in Appendix C.2.1. Responses from the LLM are parsed to extract the solution code snippets. Multiple responses are collected to ensure the diversity of the initial policies.

Training objective

The difference between LangProp over typical usage of LLMs for code generation is that it performs code optimisation in a metric- and data-driven manner. In many tasks, it is easier to provide a dataset of inputs and ground truth corresponding outputs rather than to accurately specify the requirements for a valid solution or write comprehensive unit tests. Similar to how neural networks are trained, the user defines an objective function that measures how accurate the policy prediction is against the ground truth, e.g. L1 or L2 loss. A penalty is given if the policy raises any exception (e.g. syntax error) while executing the code.

Forward-pass and feedback

Similar to training neural networks, LangProp assumes a dataset of inputs and associated ground truth labels for supervised learning (or rewards for RL, discussed in Section 5.5.2). For every batch update, the inputs are fed into all the policies currently in the LangProp model to make predictions, equivalent to a *forward-pass*. For each policy, the prediction is evaluated by the objective function which returns a *score*. If an exception is raised during execution of a policy script, it is caught by the model and an exception penalty is returned as a score instead.

The execution results, which include the score, exception trace, and any printed messages from the execution, are fed back into the model and are recorded by the policy tracker. This is analogous to how parameters in a neural network are assigned gradients during back-propagation.¹ This information stored by the tracker is used in the policy update step in Section 5.3.2.

5.3.2 Model forward pass definition

The LangProp module captures printed outputs and exceptions and stores them in the policy tracker along with the corresponding inputs during a forward pass. The Python code snippet extracted from the LLM's response and saved as a text string is executed using the `exec` function in Python. The local scope variables can be accessed via `locals`.

```
1 class LPModule:
2     ...
3     def __call__(self, *args, **kwargs) -> Any:
```

¹The current LangProp implementation is limited to an update of a single module, i.e. it does not yet accommodate for chaining of modules. This was attempted by making the LLM generate docstrings of helper functions to initiate submodules, and tracking submodule priorities. However, version tracking of submodules and the mechanism of providing feedback for submodule updates were substantial challenges. While LangProp v1 does not implement the full back-propagation algorithm, a single-layer feedback operation is referred to as *back-prop* to highlight the similarities and encourage future research.

```
4     if not self.training:
5         return self.forward(self.script_records[0].script, *args,
6                               ↪ **kwargs)
7
8     inputs = (args, kwargs)
9     script = self.run_config.active_tracker.record.script
10    with CapturePrint() as p:
11        try:
12            output = self.forward(script, *args, **kwargs)
13            self.run_config.active_tracker.forward(inputs, output,
14                                                    ↪ "\n".join(p))
15        except KeyboardInterrupt as e:
16            raise e
17        except Exception as e:
18            trace = "\n".join(traceback.format_exc().split('\n')[-3:])
19            detail = f"{type(e).__name__}: {trace}"
20            self.run_config.active_tracker.store_exception(inputs, e,
21                                                         ↪ detail, "\n".join(p))
22            raise e
23    return output
24
25 def forward(self, script, *args, **kwargs):
26     exec(script, locals(), locals())
27     output = locals()[self.name](*deepcopy(args), **deepcopy(kwargs))
28     return output
```

Listing 1: Forward passing mechanism of the LangProp module (extract)

Priority

The priority is, simply put, an average of scores with respect to the training objective. In case a small batch size is required for faster computation, a running average of the scores is used as the priority rather than ranking the policies' performance based on scores from the current batch alone, which may result in highly stochastic results. This is sufficient for supervised learning with a fixed-size dataset. As discussed later in Section 5.5.2, however,

a more complex training method such as RL or DAgger [180] has a non-stationary training distribution. Therefore, an exponential averaging with a discount factor of $\gamma \in (0, 1]$ is used:

$$P_{i,k} = \frac{\left(\sum_{j=1}^{N_k^B} s_{i,j,k}\right) + W_{i,k-1}P_{i,k-1}}{N_k^B + W_{i,k-1}}, \quad (5.1)$$

$$W_{i,k} = \gamma (N_k^B + W_{i,k-1}).$$

Here, N_k^B , $P_{i,k}$ and $W_{i,k}$ are the batch size, priority, and priority weighting of the k -th batch for the i -th policy, respectively, and $s_{i,j,k}$ is the objective score of the i -th policy for the j -th element in the k -th batch. Initial conditions are $P_{i,0} = 0$ and $W_{i,0} = 0$. Weighting recent scores higher ensures that policies with higher priorities have high performance on the most up-to-date dataset.

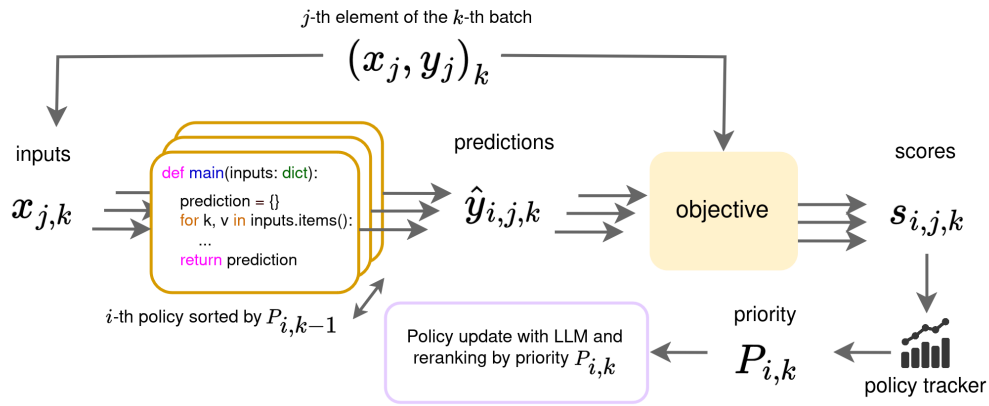


Figure 5.2: The policy evaluation and update mechanism. The performances of the policies are monitored and aggregated over time by a policy tracker as *priorities*, used to rerank the policies.

Policy reranking and update

This step updates the model based on the most recent forward-backward pass and updated priorities. This corresponds to the optimisation step in neural network training, where parameters are updated based on gradients computed on the most recent batch. The update step is illustrated in Figure 5.2. First, the policies are reranked by the priorities and the top

N^K number of policies are kept, out of which the top N^U policies are selected for updates. For each of these policies, the policy tracker is queried for the worst-case input-output pairs in the training batch, namely that with the lowest objective score. The tracker returns the corresponding input, output and score, along with any exception or print messages during the execution. This information, together with the old policy script, is embedded into the update prompt by a prompt template engine (Section 5.3.3). The update prompt is passed to the LLM, which returns N^R responses containing new policy scripts for each of the N^U policies chosen for updates.

After the update, there are $N^U \times N^R$ new policies and up to N^K old policies. To initialise the new policies with sensible priorities, objective scores for the new policies are evaluated by performing the forward-backward pass, using the same training samples as the current update. Finally, all the policies are sorted by their priorities, ready for inference or training on a new batch.

5.3.3 Prompt template engine

During the policy update stage, a dynamic prompting mechanism is required to embed information about the input, predicted output, ground truth, exception, print messages, and the policy script to be revised. The logic to generate these prompts is sometimes complex, for example, predictions are only made when there are no exceptions. To enable flexible prompt generation while avoiding any hardcoding of the prompts in the codebase, A simple yet powerful prompt template is developed that can parse variables, execute Python code embedded within the prompt, and import sub-prompts from other files, and are included in the open-sourced solution (Section 5.6). The update prompt examples shown in Appendix C.2.2 make extensive use of the policy template engine’s capabilities.

In the template engine, every line that begins with “#” is treated as comments. Every line that begins with “\$ ” or line blocks in between “\$begin” and “\$end” are treated

as executable Python code, as well as everything surrounded by `{{ }}` in a single line. If a “print” function is used within the prompt template, it will execute the Python code inside the print function and render the resulting string as a part of the prompt. Variables can be passed to the prompt template engine, and are made accessible in the local scope of the prompt template.

As an example, consider the following prompt template.

```
1 {" and ".join(p for p in people)} {"are" if len(people) > 1 else "is"}
   ↪ work here.
2 $begin
3 for i, p in enumerate(people):
4     print(f"{p} is employee No. {i + 1}.")
5 $end
```

Listing 2: Example prompt template

In this case, `read_template("example", people=["Tom", "Jerry"])` resolves to: “Tom and Jerry work here.\nTom is employee No. 1.\nJerry is employee No. 2.”.

5.3.4 Trainer forward-backward definition

The trainer has a similar abstraction to deep learning training. At every step, it triggers a forward method that calls the policy and stores the inputs, the policy’s prediction, and the expected output, and a backward method that updates the policy tracker with the scores, exceptions, or any feedback.

```
1 class LPTrainer:
2     ...
3     def step(self, tracker: RecordTracker, func_args, func_kwargs, label,
   ↪ feedback=""):
4         with self.run_config.activate(tracker):
5             score, exception_detail = self.forward(func_args, func_kwargs,
   ↪ label)
```

```
6         tracker.backward(score, label, feedback + exception_detail)
7
8     def forward(self, func_args, func_kwargs, label):
9         try:
10            with set_timeout(self.run_config.forward_timeout):
11                output = self.module(*func_args, **func_kwargs)
12                self.test_output(output, func_args, func_kwargs, label)
13                score = self.score(output, label)
14                exception_detail = ""
15            except KeyboardInterrupt as e:
16                raise e
17            except Exception as e:
18                score = self.run_config.exception_score
19                trace = "\n".join(traceback.format_exc().split('\n')[-3:])
20                exception_detail = f"""\nThere was an exception of the
21                ↪ following:\n{type(e).__name__}: {trace}"""
22            return score, exception_detail
```

Listing 3: Forward-backward pass in the LangProp Trainer (extract)

5.3.5 Training paradigm

LangProp mirrors the code abstraction of PyTorch [159] and PyTorch Lightning [52] for the module and trainer interfaces. This allows LangProp as a framework to be task-agnostic, making it easily applicable to a range of domains and use cases. Moreover, it helps highlight the similarities between neural network optimisation and code optimisation using LangProp and facilitates a smooth integration of other neural network training paradigms.

Importantly, LangProp’s internal implementation does not depend on PyTorch or PyTorch Lightning. LangProp supports PyTorch datasets and data loaders, as well as any iterable dataset object for training and validation. Listing 4 shows an example of a standard LangProp training script. The design of the module and trainer interfaces are inspired by PyTorch [159] and PyTorch Lightning [52], respectively.

```
1 train_loader = DataLoader(train_data, batch_size, shuffle=True,
    ↪ collate_fn=lambda x: x)
2 val_loader = DataLoader(val_data, batch_size, shuffle=True,
    ↪ collate_fn=lambda x: x)
3 model = LPModule.from_template(name, root)
4 trainer = LPTrainer(model, RunConfig(run_name))
5 trainer.fit(train_loader, val_loader, epochs=epochs)
```

Listing 4: Training a LangProp model with a trainer. The model can be instantiated from a path to the setup and update prompts that specify the task to be learned.

After every training step on a mini-batch, the trainer saves a *checkpoint*, which consists of the setup prompt, update prompt template, the policy scripts (maximum of $N^K + N^U \times N^R$), and the statistics monitored by the policy tracker (priorities P and priority weights W). Since these can be stored as text or JSON files, the size of a checkpoint is in the order of a few hundred kilobytes. Checkpoints can be used to resume training, fine-tune the model, or for inference.

```
1 model = LPModule.from_checkpoint(checkpoint)
2 model.setup(config=RunConfig())
3 prediction = model(*input_args, **input_kwargs)
```

Listing 5: Inference with a LangProp model checkpoint.

Listing 5 shows how a LangProp checkpoint can be loaded and used for inference. The policy with the highest priority is used. Since policies are *parameterised* as executable code, the use of an LLM is only required during training, not during inference. Since querying LLMs is both expensive and slow, this is a key advantage of the LangProp approach, which makes integration of LLMs more feasible for real-time applications, such as robotics and autonomous driving.

5.4 General domain experiments

LangProp’s code optimisation capability is demonstrated in three domains with increasing complexity. The GPT 3.5 Turbo 16k model [154] is used as a backbone LLM.

5.4.1 Generalised Sudoku

A generalised Sudoku puzzle consists of $W \times H$ subblocks, each with $H \times W$ elements, where H and W represent height and width, respectively. A valid solution places numbers from 1 to WH in each cell, such that each row, column and subblock contains no repeated numbers. LangProp is trained on this problem given 100 samples of unsolved Sudoku puzzles as input and corresponding solutions as output. The training objective is defined as the correctness of the arrived solution, i.e. whether the puzzle completed by a LangProp-learned policy is a valid Sudoku puzzle solution. The setup and update prompts are in Appendix C.2. Due to the complexity of the task specification, the LLM queried zero-shot occasionally failed on the first attempt, confusing the task with a standard 3×3 Sudoku. LangProp filtered out incorrect results during training and identified a fully working solution. Samples of an incorrect zero-shot solution and a correct solution after LangProp training can be found in Appendix C.3.1.

5.4.2 CartPole

CartPole [22] is a widely used environment to train and test RL algorithms. To make it feasible for LangProp to generate a policy to solve this task, the observation and action specifications are provided, following the Gymnasium documentation for CartPole-v1. The setup and update prompts are in Appendix C.2. Queried zero-shot, the LLM generated a solution that is simplistic and does not balance the CartPole, achieving a score of 9.9 out of 500. With a simple Monte-Carlo method of optimising the policy for the total rewards, improved policies were obtained using LangProp, achieving the maximum score of 500.0.

Interestingly, LangProp learned a policy that implemented a PID controller to solve the task.

Figure 5.3 shows learning curves of the LangProp policy for 10 different seeds. Training hyperparameters were $N^U = N^R = N^K = 3$. Out of 10 seeds, 9 converged to an optimal solution within 10 LangProp updates, and within $10k$ total steps in the CartPole environment. Learning curves of Proximal Policy Optimisation (PPO) [194] are provided for comparison, which converges at around $80k$ environment steps. This shows that certain tasks may be more sample-efficient to solve with

LangProp. While it is infeasible to arrive at a correct solution zero-shot, the LangProp optimisation loop allows the LLM to discover a correct solution.

Sample results can be found in Appendix C.3.2. Implementations, prompts, checkpoints, and examples of zero-shot and trained policies are available in the open-sourced repository.

5.5 Driving in CARLA

This section describes how the LangProp framework can be used in the context of autonomous driving in CARLA. CARLA [48] is a widely used open-sourced 3D simulator for autonomous driving research, and many prior works on CARLA have open-sourced their expert agents. CARLA is chosen as a benchmark since (a) autonomous driving requires interpretable driving policies, (b) CARLA has a rich collection of human-implemented expert

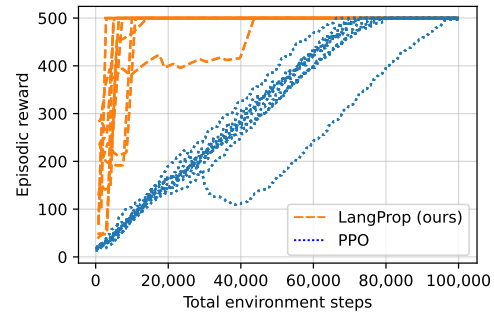


Figure 5.3: The total number of *environment* steps required to learn CartPole-v1 (10 seeds per method) in comparison to an RL method, PPO. Most seeds converged to an optimal solution within 10 LangProp updates.

agents to compare against, and (c) a metric-driven learnable approach would be beneficial since driving decisions are challenging planning problems, and even human-implemented experts have sub-optimal performance. Section 2.4.2 discusses related work on autonomous driving.

5.5.1 Data collection

Data agent

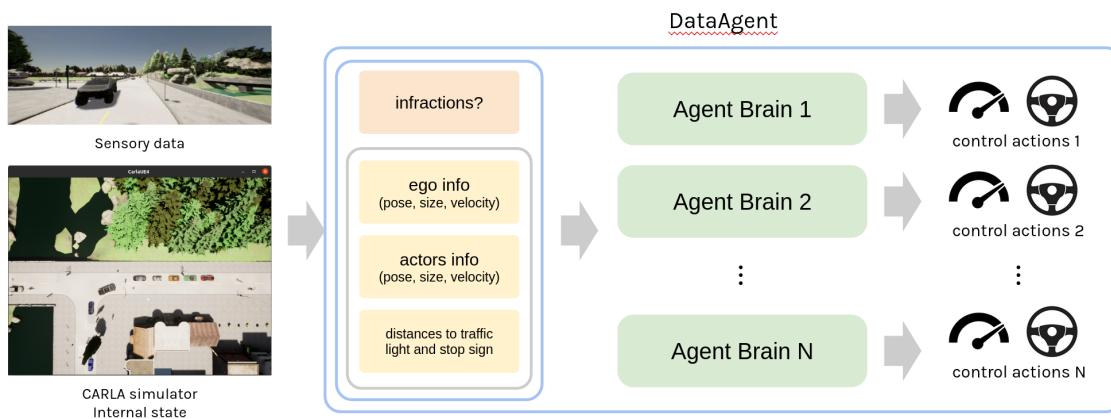


Figure 5.4: An overview of the CARLA agents for data collection and evaluation. All agents are implemented as a standardised `DataAgent` class, with one or multiple `AgentBrain` that takes in the preprocessed observations and outputs a control action. These inputs and outputs are processed, recorded and saved by the `DataAgent` so that they can be used for data collection or for online training.

To standardise the data collection and evaluation pipeline for both the expert agent and the LangProp agent, a generic `DataAgent` is implemented. An overview is shown in Figure 5.4. It collects basic privileged information from the CARLA environment: the 3D bounding box coordinates of the actors in the scene (pedestrians, vehicles, traffic lights, and stop signs), the velocity of the pedestrians and vehicles, distances to the next traffic light and stop sign in the current lane, and the next waypoint to navigate towards. In addition, it also collects the RGB, depth, lidar, segmentation, top-down view, and the expert’s control actions which can be used to train image-based driving policies. This

standardised data collection agent is decoupled from the expert agent and the LangProp agent, and has the option of turning off sensors that are not used for data collection to save computation time and data storage.

The data collection agent itself does not have a driving policy. It expects a separate `AgentBrain` that takes a dictionary of scene information curated by the data agent as input and outputs a vehicle control action (throttle, brake, and steering). All driving agents inherit from the `DataAgent` class, each with an `AgentBrain` that implements its driving policy. It is also possible to chain multiple agent brains as an array, where the previous agent brain's control decision is provided as an additional input to the next agent brain. This is useful for the `Dagger` agent and online agent, which require expert supervision during online rollouts.

Expert design

An expert agent is implemented for data collection and to provide action labels to train the LangProp agent with Imitation Learning (IL). While `TransFuser` [35] and `TF++` [102] use a computationally expensive 3D bounding box collision detection algorithm, and `InterFuser` [196] uses line collision which is faster but less accurate, the LangProp expert uses an efficient polygon collision detection algorithm between ground-projected bounding boxes.

The LangProp expert only uses the data collected by the data agent to ensure that the LangProp agent has access to the same privileged information as the expert agent. For every interval of 0.25 s up to 2 s into the future, whether the ego vehicle polygon will intersect any of the actor polygons is evaluated, assuming that the ego vehicle will maintain velocity, and the other actors will move in the current orientation with a speed less than or equal to the current speed. The ego vehicle polygon is padded forward by 2 m , and by 2 m either left or right upon lane changes. Apart from lane changing, only actors that are ahead of the ego vehicle are considered, i.e. with a field of view of 180° . A traffic light and/or

stop sign that affects the vehicle is identified by querying the associated waypoints in the CARLA simulator. For pedestrians, vehicles, traffic lights, and stop signs, the distances to the obstacles are calculated. The normal driving speed is 6 m/s (“MOVE”). If any of the distances are reachable within 2 s with a 2 m margin (“SLOW”), the target speed is set to the speed which allows a 2 s margin, and if the distance is below 2 m (“STOP”), the target speed is set to 0 m/s . Steering is evaluated by calculating the angle to the next waypoint, which is 4 m ahead of the current position of the ego vehicle. A PID controller is used for low-level control to convert the target speed and angle to throttle, brake, and steering.

5.5.2 Training the LangProp agent

LangProp agent

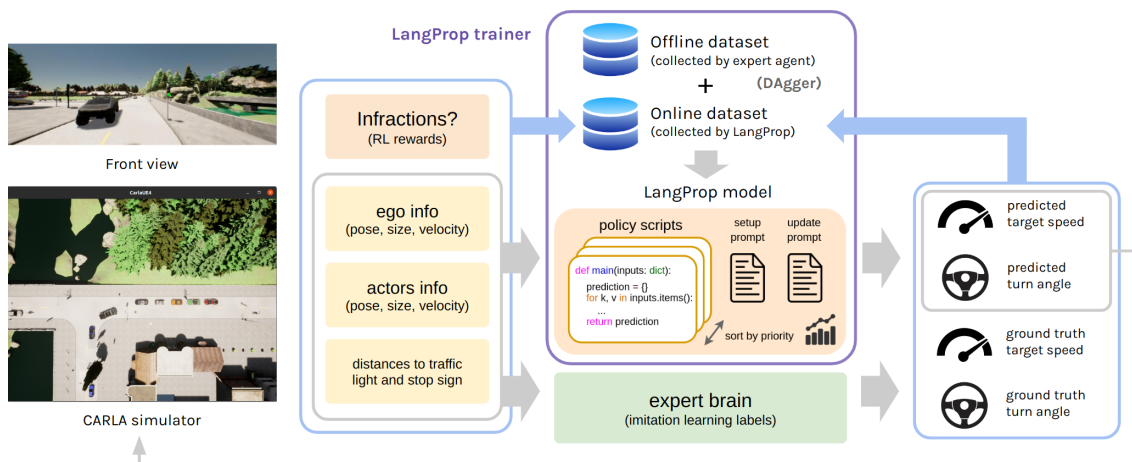


Figure 5.5: An overview of the LangProp agent training pipeline. The LangProp model is updated on a dataset that includes both offline expert data as well as online LangProp data annotated with expert actions, similar to DAgger. The agent is given negative rewards upon infraction.

Similarly to all the baseline experts, privileged information from the CARLA simulator to the agent is provided. Unlike the baseline experts where post-processing is manually implemented, LangProp can decide how the information should be handled (e.g. converting world coordinates into the ego-centric frame). For the ego vehicle, as well as for all vehicles and pedestrians within a 50 m radius, the following information

is provided: the location (in world coordinates), orientation, speed, length, width of the actor, the target waypoint (4 m ahead, used by other baseline experts), and the distances to a red traffic light and stop sign along the current lane if they exist. Importantly, all actors are provided without filtering, even if they are irrelevant to the driving agent. Given this information, the LangProp policy is expected to return a desired speed level (“MOVE”: 6 m/s, “SLOW”: 1 m/s, “STOP”: 0 m/s)² and a turning angle for the ego vehicle. These are passed to an external PID controller to convert them into throttle, brake, and steering. The function specification in the setup prompt is given in Listing Appendix C.2.3 as a docstring. Given this function definition, an LLM generates policy script candidates that satisfy the specification and updates them following the procedures in Section 5.3.

Behavioural Cloning, DAgger, and RL

Three major training paradigms often used to train embodied agents are explored – BC, DAgger [180], and RL. In BC, the accuracy of the policy outputs is measured against ground truth expert actions for a pre-collected dataset. BC is known to have issues with out-of-distribution inputs at inference time, since the expert’s policy is used to collect the training data, while the learned policy is used for rollouts at inference time. DAgger addresses this issue by labelling newly collected *online* data with expert actions, and adding them to the expert-collected *offline* data to form an aggregate replay buffer. CARLA runs at a frame rate of 20 Hz. LangProp adds training samples to the replay buffer every 10 frames, and a batch update is performed after every batch of 100 new samples.

While DAgger solves the issue of distribution mismatch, the performance of the learned policy is still upper-bounded by the accuracy of the expert. It also does not take into account that certain inaccuracies are more critical than others. In the context of

²While it is straightforward for the policy to directly predict the speed or acceleration as numeric values, this makes the task of designing a suitable loss function for IL more challenging and open-ended. Therefore, a categorical output is chosen, simplifying the scoring function.

autonomous driving, actions that result in infractions such as collisions should be heavily penalised. RL offers a way of training a policy from reward signals from the environment, which is convenient since penalties can be assigned directly upon any infractions according to the CARLA leaderboard [25]. While RL typically optimises for maximum returns (discounted sum of future rewards), this setting is simplified by assigning an infraction penalty if there is an infraction in the next 2 s window. The agent monitors infractions every 10 frames, and triggers an update upon infractions.

Since infraction penalties are sparse signals, and will become rarer as the policies improve, two strategies are adopted; (a) RL training is combined with IL training that provides denser signals, and (b) training data with infractions are sampled with 100 times higher sampling probability.

Training objective

The training objective for the LangProp driving agent is given as

$$\begin{aligned}
 S(a^\pi, a^{\pi_e}, a^{\pi_b}, I, E) = & \mathbb{1} \left[(a_{\text{speed}}^\pi = a_{\text{speed}}^{\pi_e}) \wedge [\neg I \vee \{(a_{\text{speed}}^\pi \neq a_{\text{speed}}^{\pi_b}) \wedge (a_{\text{speed}}^{\pi_e} \neq a_{\text{speed}}^{\pi_b})\}] \right] \\
 & + r_{\text{infrac}} \mathbb{1} (I \wedge (a_{\text{speed}}^\pi = a_{\text{speed}}^{\pi_b}) \wedge (a_{\text{speed}}^{\pi_e} \neq a_{\text{speed}}^{\pi_b})) \\
 & + r_{\text{angle}} \mathbb{1} (|a_{\text{angle}}^\pi - a_{\text{angle}}^{\pi_e}| > \theta_{\text{max}}) + r_{\text{error}} \mathbb{1} (E),
 \end{aligned} \tag{5.2}$$

where a^π , a^{π_e} and a^{π_b} are actions taken by the current policy, expert policy, and behaviour policy used to collect the training sample, respectively, I and E are boolean variables for infraction and exception occurrences, $r_{\text{infrac}} = r_{\text{error}} = r_{\text{angle}} = -10$ are penalties for infraction, exception, and exceeding angle error of $\theta_{\text{max}} = 10^\circ$, and $\mathbb{1}$ equates to 1 if the boolean argument is true, and 0 otherwise. The expert is only imitated when there are no infractions, or if the expert was not the behaviour policy that incurred the infraction, and an infraction cost is only given when the current policy takes the same action as the behavioural policy that caused the infraction when the expert chose a different action.

Training strategy

The training pipeline for the LangProp driving agent is shown in Figure 5.5. For all the LangProp agents, the training data was collected only on the training routes in CARLA leaderboard [25], and data collected on the test routes by the expert agent with expert action labels is used as the validation dataset. See Section 5.5.3 for more details on the routes. For the LangProp agent trained offline, only samples collected by the expert agent were used as training data. For the online training, only samples collected by the current LangProp model’s inference policy were used, i.e. the policy code with the highest priority at the time of rollout. For DAgger training, a mix of 1000 training samples collected offline and 1000 samples collected online in every replay batch were used to evaluate the objective score. Strictly speaking, DAgger [180] should incrementally add new online samples to a buffer initialised with offline samples. However, this prevented the LangProp model from learning from infractions during the early stages of the training, since online samples with infractions are the minority of all the samples. For this reason, an even split between offline and online samples was maintained throughout the training, with a sampling weight of 100 for samples with infractions. Sampling is without replacements, so that a particular training sample is only sampled once per replay batch.

Hyperparameters

Notable training hyperparameters are the number of policies chosen for updates $N^U = 2$, the number of responses per query $N^R = 2$, the number of policies to keep $N^K = 20$, the frequency of batch updates (every 100 new samples in the replay buffer), batch sizes for online replay data (1000) and offline expert data (1000), the sampling weight for infractions (100), and the infraction, exception, and angle penalties ($r_{\text{infrac}} = r_{\text{error}} = r_{\text{angle}} = -10$). For better performance, it is possible to increase N^U , N^R , and N^K , but with a trade-off of computational time and the cost of using OpenAI API. With this experiment setting, around

700 training steps are taken, 1400 queries are made, and 2800 responses are received from GPT 3.5 per training job, which costs roughly \$150.

5.5.3 Benchmark

Baselines

The LangProp agent was compared against RL agents with privileged information (Roach [257], TCP [241]) as well as human-written experts (TransFuser [35], InterFuser [196], TF++ [102], ours). The official training and testing routes provided by the CARLA leaderboard [25] were used, as well as the Longest6 benchmark [35] that has longer routes with denser traffic. For the LangProp agent, only the training routes are used for imitation/RL at training time, and the saved checkpoints are used for inference during evaluation runs on different routes.

CARLA leaderboard

The driving scores are computed by the CARLA leaderboard evaluator [25], using the official training and test routes, and the Longest6 benchmark provided by Chitta et al. [35]. There are towns 1 – 6 across the benchmarks. Towns 7 – 10 are also used in the official online leaderboard. A breakdown of routes for each benchmark is shown in Table 5.1. Towns 2 and 5 are withheld in the training routes and only appear in the testing routes and the Longest6 benchmark. The Longest6 benchmark has longer routes with denser traffic.

The main metric of the leaderboard is the driving score, which is a product of the route completion percentage \bar{R} and the infraction factor \bar{I} . It is evaluated as $\frac{1}{N} \sum_i^N (R_i I_i)$, where i denotes the index of the N routes used for evaluation, R_i is the percentage of route completion of the i -th route, and I_i is the infraction factor of the i -th route. The infraction factor is a product of infraction coefficients for pedestrian collision (0.5), vehicle collision (0.60), collision with static objects (0.65), running a red light (0.70), and running a stop

Table 5.1: A breakdown of the number of routes per town (“num”), the average length of the routes per town (“avg. dist.”), and traffic density for the training routes (“density”), testing routes, and the Longest6 benchmark.

Routes	Training routes			Testing routes			Longest6		
	num	avg. dist.	density	num	avg. dist.	density	num	avg. dist.	density
Town 1	10	776.3	120	-	-	120	6	898.8	500
Town 2	-	-	100	6	911.7	100	6	911.7	500
Town 3	20	1392.5	120	-	-	120	6	1797.5	500
Town 4	10	2262.6	200	10	2177.8	200	6	2102.4	500
Town 5	-	-	120	10	1230.1	120	6	1444.7	500
Town 6	10	1915.4	150	-	-	150	6	2116.7	500

sign (0.80). The driving score per route is equal to the route completion R_i when there are no infractions, and is discounted for every infraction by a corresponding infraction factor. Note that in the Longest6 benchmark, the authors decided to remove the stop sign penalty by setting its infraction coefficient to 1.0, which is adhered to in the following experiments.

CARLA version 0.9.10 is used for the experiments to maintain consistency with other baseline experts that assume this version. The LangProp expert has been tested both on CARLA version 0.9.10 and version 0.9.11.

5.5.4 Experiments

Results

The results are shown in Table 5.2. The LangProp expert and the TF++ expert significantly outperformed all other expert agents in all routes, and the LangProp expert outperformed TF++ by a margin on the test routes. The core collision avoidance logic is just 100 lines of code, with additional preprocessing and tooling for data collection. From the breakdown

Table 5.2: Driving performance of expert drivers in CARLA version 0.9.10. The driving score is a product of the route completion percentage \bar{R} and the infraction factor \bar{I} . IL and RL stand for Imitation Learning and Reinforcement Learning. DAgger uses both online and offline data.

Method	Training routes			Testing routes			Longest6		
	Score \uparrow	\bar{R} \uparrow	\bar{I} \uparrow	Score \uparrow	\bar{R} \uparrow	\bar{I} \uparrow	Score \uparrow	\bar{R} \uparrow	\bar{I} \uparrow
Roach expert	57.8	95.9	0.61	63.4	98.8	0.64	54.9	81.7	0.67
TCP expert	64.3	92.3	0.71	72.9	93.2	0.77	46.9	63.1	0.76
TransFuser expert	69.8	94.5	0.74	73.1	91.3	0.80	70.8	81.2	0.88
InterFuser expert	69.6	83.1	0.86	78.6	81.7	0.97	48.0	56.0	0.89
TF++ expert	90.8	95.9	0.94	86.1	91.5	0.94	76.4	84.4	0.90
Our expert	88.9	92.8	0.95	95.2	98.3	0.97	72.7	78.6	0.92
LangProp Agents									
Offline IL	0.07	0.37	0.97	0.00	0.00	1.00	0.00	0.00	1.00
DAgger IL	36.2	94.5	0.40	41.3	95.3	0.44	22.6	87.4	0.30
DAgger IL/RL	64.2	90.0	0.72	61.2	95.2	0.64	43.7	71.1	0.65
Online IL/RL	70.3	90.5	0.78	80.9	92.0	0.89	55.0	75.7	0.73

of the scores, the LangProp expert seems to prioritise safer driving with fewer infractions (higher infraction factor \bar{I}) by trading off route completion compared to TF++ in the Longest6 benchmark.

For the LangProp agent, it could be observed that training using offline samples, DAgger, and online samples improves performance in this order. Adding the infraction penalties as an additional RL objective further improved the performance. The best-performing agent, LangProp trained on online data with IL and RL, achieved better performance than the Roach expert (trained with PPO) as well as the TransFuser and InterFuser experts (both written by researchers) on all benchmarks apart from TransFuser on the Longest6 benchmark. Note that TransFuser has an advantage over the Longest6 benchmark since LangProp has never seen this benchmark during training. The driving policy generated

using LangProp is shown in Appendix C.3.3.

The result has two important implications. Firstly, the code selection metric (the training objective) plays a large role in the ultimate performance of the code. This is an important finding since prior work on code generation mostly focused on error correction given exceptions. The results demonstrate that for complex tasks, it is important to treat code generation as an iterative optimisation process rather than a zero-shot task. Secondly, training using LangProp exhibits similar characteristics as training in deep learning; in deep learning, it is a well-studied problem that policies trained with BC on offline datasets do not generalise to out-of-distribution online data. DAgger and RL are two of the common ways of addressing this problem. The results show that these training paradigms can also be effective when used in LangProp.

Analysis of training methods

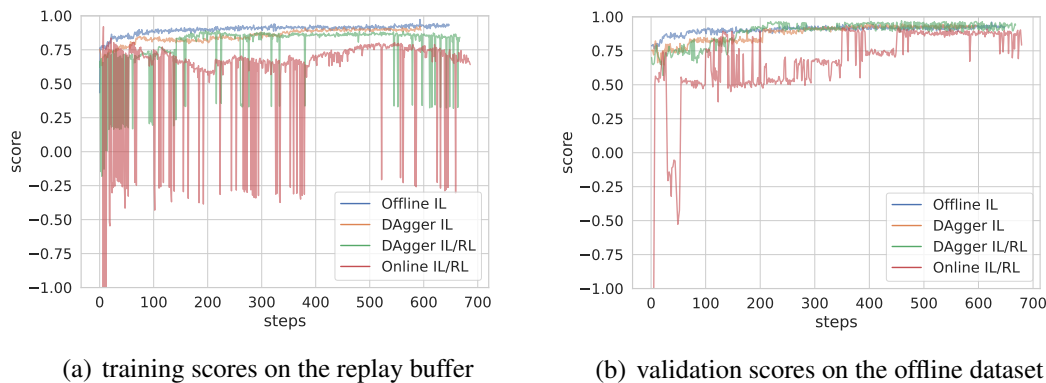


Figure 5.6: Training curves for the different training methods of the LangProp agent. The training scores are evaluated on 1000 samples from the offline training dataset and/or online replay buffer, and the validation scores are evaluated on 1000 samples from the offline validation dataset. Updates are performed every 1000 frames of agent driving, and upon infractions in the RL setting. The score is in the range of $[-10, 1]$ due to exception penalties. The axis is limited to $[-1, 1]$ in the plots.

A common failure mode of offline trained models was that the agent remained stationary indefinitely until the timeout was reached. Upon inspection of the policy code that was generated, the cause of failure was identified to be a phenomenon known as causal

confusion in IL [43]. A snippet of code responsible for such failure in one of the runs is shown in Listing 6.

This exemplifies the interpretability of LangProp models, allowing us to directly assess the source of failure. The code predicts 0 speed when the agent's current speed is already close to 0. Note that this is not a failure of the LangProp algorithm, but due to such a policy maximising the IL objective on an offline dataset, bypassing the need to learn a more complex policy. This phenomenon is also common in the context of *deep* IL, and can be avoided by employing training on online data, e.g. using DAgger or RL. This work is thought to be the first to report a similar phenomenon using LLMs for policy optimisation.

```
1 # General rule: if the ego vehicle is stopped or moving very slowly, set the
  ↪ speed level to "STOP"
2 if np.abs(scene_info["ego_forward_speed"]) < DELTA_V_THRESHOLD:
3     speed_level = "STOP"
```

Listing 6: Causal confusion in offline-trained policy

The use of online training samples alleviated the issue of causal confusion, leading to selecting policies where the agent has a sensible driving performance. This is because if the agent remains stationary, those samples will accumulate in the replay buffer, resulting in a lower priority for the causally confused policy. Comparing the results in Table 5.2 and the validation scores in Figure 5.6(b), it seems that the scores on the offline dataset are not indicative of the agent's driving performance. From the training scores on the replay buffer and/or offline dataset in Figure 5.6(a), it could be seen that the agents trained with RL on infractions have spikes corresponding to infractions. This is due to oversampling infractions when they occur, allowing the policy update to immediately address the issue. DAgger has a milder response compared to training just on online data because the offline dataset does not include on-policy infractions. The higher rate of infractions in the training

distribution may be why the online trained agent has a lower training score but has a higher driving performance.

5.6 Implementation

The code used in this paper is open-sourced, and can be found at <https://github.com/shuishida/LangProp/>. This includes code for the general LangProp framework, applying LangProp to tasks such as Sudoku and CartPole, and training and evaluating the LangProp agent in CARLA. Pre-trained checkpoints using LangProp and videos of sample runs by the LangProp agent are also included.

5.7 Conclusion

This work presented LangProp, a framework that uses LLMs for data-driven code optimisation, and demonstrated its capability of generating and improving policies in the domains of Sudoku, CartPole and CARLA. In particular, LangProp generated driving policies in CARLA that outperform those that existed when the backbone GPT 3.5 was trained. It was shown that classical training paradigms such as BC, DAgger, and RL directly translate to training with LangProp, and the choices of the objective function and the training data distribution can be used to guide which policies are selected. Automatically optimising the code to maximise a given performance metric has been a key missing feature in few-shot code generation. The LangProp framework provides this feature by reformulating the machine learning training paradigm in the context of using LLMs as code optimisers and treating policy code as parameters of the model. The LangProp paradigm opens up many possibilities for data-driven machine learning with more interpretability and transparency.

Chapter 6

Embodied task execution in a virtual world

This chapter addresses the long-standing challenge of performing complex compositional tasks in Minecraft that involve spatial reasoning and planning. Minecraft is a popular game in a 3D virtual world, in which players can explore the terrain, collect resources, craft tools, and build structures, as well as gather food and combat malicious game characters for survival. It serves as a perfect evaluation benchmark for embodied agents due to the diversity of its randomly generated landscape and the open-ended and life-like nature of the tasks presented. This chapter introduces Voggite, an embodied agent that performs tasks in Minecraft using OpenAI Video PreTraining (VPT) [12] as a backbone. Unlike VPT, which is only retained to retain short-term memory with a transformer policy and struggles to disambiguate different stages of task execution, Voggite decomposes complex tasks into a sequence of subtasks, thereby solving the problem of ambiguity in the VPT policy. The VPT policy is fine-tuned to solve a range of life-like tasks that the original VPT was not specifically trained for. Voggite was submitted to the MineRL BASALT Competition 2022 [136] and achieved 3rd place out of 63 teams. This chapter also discusses other approaches that were considered to solve the problem of learning reusable skill-learning for embodied agents from a limited dataset of expert demonstrations.

6.1 Introduction

Preceding chapters addressed the problems of learning to plan in partially observable environments for robot navigation, learning to segment skills in a self-supervised way, and learning interpretable policies for autonomous driving using Large Language Models (LLMs) to iteratively improve code policies. These are all steps towards enabling embodied agents to operate in a real-world environment and acquire skills reusable for task execution and long-term consistent planning. In this penultimate chapter, the game of Minecraft is considered a platform to practically evaluate embodied agents on their life-like task execution capabilities.

Minecraft is an open-ended 3D gaming environment where human players and agents can perform many life-like tasks. In Minecraft, agents explore a terrain of pre-historic natural scenery in a block-like virtual world. Players can gather raw materials from nature and process them to obtain tools, which could be used to craft and build more complex tools, structures and machines. Minecraft does not define a fixed set of challenges or requirements to complete the game; rather, it is an open-ended sandbox environment where players can unleash their curiosity and creativity. Many of the tasks that could be performed are relevant to surviving, living, exploring and creating, such as mining, crafting, building, raising animals, harvesting crops, and occasional combats with unfriendly game characters.

Many of the tasks in Minecraft are compositional. For instance, finding a diamond requires the agent to chop wood, make a crafting table from wooden planks, make a wooden pickaxe to collect cobblestones, make a stone pickaxe to collect iron, create a furnace to melt the iron into an iron pickaxe, and finally use the iron pickaxe to obtain a diamond. This requires long-term planning, guided by intuition from real-world experiences.

It is easy for human players to draw the analogy between the real world and the Minecraft world, and learn to complete tasks in this virtual world, since many of the objects

and characters in Minecraft reflect those in the real world and obey similar rules. Learning the environment dynamics from scratch using trial and error such as with Reinforcement Learning (RL) would be computationally expensive. On the other hand, some environment dynamics, such as how object blocks can float in mid-air, do not reflect the real world and must be learnt either from external resources (e.g. documentation or manual) or through gameplay. Humans can combine the three modes of learning to adapt to new environments: transfer of prior knowledge, gathering experiences through trial and error, and integration of external or collective knowledge. A natural approach to designing artificial embodied agents for real-world tasks would be to equip them with such learning capabilities.

Research on learning-based agents in Minecraft has been accelerated by the developments of Malmö (a Minecraft simulator for RL agents) and MineRL [74] (a state-action paired dataset of over 60 million frames across a diverse set of tasks), along with competitions accompanying the MineRL dataset. Starting with the task of obtaining a diamond in the first competition [74], the MineRL Competition in 2021 [195] introduced more life-like tasks in the BASALT track, some with difficult-to-define reward functions: finding a cave, making a waterfall, creating an animal pen, and building a house.

While earlier works on Minecraft (see Section 2.4.4) relied on semantically rich observations and knowing the agent's inventory (i.e. what items and resources the agent has in possession), the recent VPT [12], a transformer foundation model trained on a large collection of videos on the Internet, successfully managed to discover diamonds taking RGB pixel inputs as observations and predicting cursor movements, mouse clicks and keyboard presses as outputs. Following the success of VPT, the MineRL BASALT Competition was significantly updated in 2022, defining the observations to be RGB pixels and actions to be cursor movements and mouse clicks in the same way as VPT. This work builds on top of VPT, addresses its weaknesses and fine-tunes it to solve challenges specified in the MineRL BASALT Competition 2022.

6.2 Background

6.2.1 OpenAI VPT

VPT [12] is a transformer-based foundation model that is trained to take in a video sequence as input and output a sequence of actions. Training such large models with Behavioural Cloning (BC) (i.e. directly supervising them with expert action labels) requires a vast amount of expert-labelled collection of videos, which are not readily available. OpenAI solved this problem by collecting a relatively small amount of expert labels to train an Inverse Dynamics Model (IDM), a separate transformer model that also predicts a sequence of actions from a video sequence, except that while a transformer policy must predict actions auto-regressively without having access to future sequences of frames, an IDM can be trained non-causally and is provided with video frames from future time steps. Aided with such privileged information, an IDM can predict actions that were taken by the agent to produce the corresponding observations much more accurately with less training data compared to a causally trained policy. An IDM trained on around $2K$ hours of expert play was used to label $70K$ hours of gameplay video footage without action labels. Since any public video footage could be labelled by the IDM, this method of data labelling makes it possible to train a large transformer policy on vast amounts of publicly available video footage, e.g. on YouTube.

While online videos are useful for learning semantic feature representations and primary tasks, since Minecraft is an open-ended and undirected environment, the resulting policy would also be generic and undirected. To encourage the policy to learn useful task-specific behaviours, a combination of Imitation Learning (IL) and RL was used. In the IL setting, a VPT foundation model was fine-tuned with BC from expert demonstrations specifically for early game resource gathering and tool crafting to build a basic house. In the RL setting, the agent was trained on rewards designed for the task of crafting a

diamond pickaxe. Phasic Policy Gradient (PPG) [39], a policy gradient algorithm similar to Proximal Policy Optimisation (PPO) [194] but with improved sample efficiency is used.

One of the limitations of the VPT agent is that, despite its transformer architecture which allows the agent to retain temporal information, the VPT policy takes in 128 frames of past observations, equivalent to only 6.4 *s* of history. While this is far outstanding compared to prior work, it is still insufficient for long-term planning, and it fails to work well with tasks with multiple stages and dependencies.

6.2.2 MineRL BASALT Competition 2022

The MineRL BASALT Competition [136], standing for “Benchmark for Agents that Solve Almost Lifelike Tasks”, challenged embodied agents to solve tasks with hard-to-specify reward functions in Minecraft. The competition evaluated agents on their life-like task execution capabilities in four domains: finding a cave, making a waterfall, creating an animal pen, and building a house. Unlike many competitions with automated evaluation, agent performances are evaluated by human judges in the MineRL BASALT Competition. 600 hours of labelled expert demonstrations are provided for the four tasks. An OpenAI VPT agent individually fine-tuned on the four tasks is treated as a baseline for the competition.

6.3 Segmenting stages of task execution in Minecraft

6.3.1 Motivation

Inspecting the performances and rollouts of the VPT agent fine-tuned for tasks such as making a waterfall, creating an animal pen and building a house, it was noted that the agent’s short temporal window of observations is hindering the agent from executing long-term plans. Let us inspect each scenario in turn.

Finding a cave

Finding a cave is a task simple enough that it could be partially solved by chance even by the baseline VPT agent with a reactive policy; the agent must explore the environment, discover a cave-like structure, enter it, and press ESCAPE for task completion. Nevertheless, navigating a terrain with pitfalls, cliffs and pools of water, also with the threat of mobs (antagonistic game characters) is a non-trivial task.

While the baseline agent was relatively good at exploration, it had a tendency to start to dig whenever it was trapped in a dead end or encountered obstacles. However, finding a cave by digging was not considered an allowed strategy in the competition. The agent was also not competent at backtracking or planning a long-term trajectory.

Making a waterfall

In the task of making a waterfall, the agent holds a bucket of water that it must carry up a mountain to flip over at the summit. Once the agent creates a waterfall, it must climb down and turn around to capture the scenery in its first-person view camera.

The baseline had a tendency to flip over the bucket at random locations and continue on its exploration without a clear intention of climbing up or down a mountain. This is not surprising, as whether the agent should climb up a mountain or down is ambiguous from the sequence of observations alone (which is a close-up view of the mountainous terrain), and the information of whether the bucket is full of water or empty is a minor detail in the observations. Learning two highly different sub-policies (one climbing up, one climbing down) based on these small differences in input is challenging.

Creating an animal pen

Creating an animal pen consists of multiple stages of task execution: finding a suitable village house to build a pen next to, finding some wandering animals of the same species

and luring them to the chosen location, and finally building a fence around them. Other orderings of stages to complete the task may also be possible.

The baseline agent proceeded to explore whilst occasionally throwing fences onto the ground and nearby walls inconsistently. This may be because (a) the agent was fine-tuned from a general-purpose VPT agent that was trained to explore, and (b) the agent only has a short-term memory of 6.4 s, and is not able to disambiguate between whether it is meant to explore, build or lure animals at a particular time. Hence, the policy may be confused with conflicting training signals and fail to learn a temporally coherent strategy.

Building a house

Similarly to Section 6.3.1, the task of building a house also has multiple stages: exploration to find a cleared space, building the walls, making a door using a crafting table, attaching the door, and giving a tour of the house. Resources required to build the house are available in the inventory, although some specific items may need to be crafted. In this example as well, the baseline agent manifested inconsistent behaviour of running around the terrain while occasionally dropping resources in random locations on the way, which is not beneficial for the task of building a house.

From the above examples, it is evident that the baseline VPT lacks long-term memory and planning capabilities, which is significantly impacting the competence of an otherwise well-trained large transformer policy with rich prior knowledge of the Minecraft environment. In the following sections, two strategies are considered to segment the task into multiple stages to aid the policy in making temporally consistent decisions.

6.3.2 Segmenting by states via Invariant Information Clustering

An initially considered approach was to cluster the observations in the expert demonstrations. Since the observations in the demonstrations are RGB images and are only annotated

with corresponding actions taken by the expert, a separate means to obtain such clustering must be considered.

One approach would be to manually annotate the observations with human-identified subtasks. However, this approach would be expensive and the definition of “subtask” is unclear. It would be more desirable to discover temporal clusters automatically through data. As an alternative, a contrastive method with temporal constraints was conceived.

Background on Invariant Information Clustering

Invariant Information Clustering (IIC) [104] was used as a contrastive training objective to learn cluster assignments. IIC is a method with a simple objective to maximise the mutual information between the class assignments of positively paired data samples. It assumes a mapping function $\Phi : \mathcal{X} \rightarrow \mathcal{Z}$, where \mathcal{X} is the input space, \mathcal{Z} is a categorical space $\mathcal{Z} = \{1, \dots, C\}$, and C is the number of clusters.

Mutual information between random variables z and z' is defined as:

$$I(z; z') = \int_z \int_{z'} p(z, z') \log \frac{p(z, z')}{p(z)p(z')} dz dz'. \quad (6.1)$$

Given N number of positive data samples $\{(x_n, y_n) : 1 \leq n < N\}$, where x_n and y_n are known to belong to the same cluster, the maximisation objective is:

$$\max_{\Phi} I(\Phi(x); \Phi(y)). \quad (6.2)$$

Since the goal is to learn representations with a deep neural network, $\Phi(x)$ performs soft rather than hard clustering with a softmax layer as an output. The output $\Phi(x) \in [0, 1]^C$ can be interpreted as the distribution of a discrete random variable z over C classes, formally given by $P(z = c|x) = \Phi_c(x)$. Similarly, $P(z' = c'|y) = \Phi_c(y)$. By marginalising over the dataset (or a batch of size N in practice), a joint probability $\mathbf{P}_{cc'} = P(z = c, z' = c')$ could be considered; $\mathbf{P}_{cc'}$ denotes the probability that input x is assigned to cluster c and input y is assigned to cluster c' for a random positive pair (x, y)

in the dataset. \mathbf{P} is a $C \times C$ matrix given by:

$$\mathbf{P} = \frac{1}{N} \sum_n^N \Phi(x_n) \cdot \Phi(y_n)^T. \quad (6.3)$$

For most problems, for every positive pair (x, y) , a pair (y, x) should also be a positive pair. Therefore, \mathbf{P} could be symmetrised by $(\mathbf{P} + \mathbf{P}^T)/2$. The marginals $\mathbf{P}_c = P(z = c)$ and $\mathbf{P}_{c'} = P(z' = c')$ can be obtained by summing over the rows and columns of this matrix.

Finally, the IIC objective function is obtained by substituting the joint probability matrix \mathbf{P} into the mutual information objective in Equation (6.1) as:

$$I(z; z') = I(\mathbf{P}) = \sum_{c=1}^C \sum_{c'=1}^C \mathbf{P}_{cc'} \cdot \log \frac{\mathbf{P}_{cc'}}{\mathbf{P}_c \cdot \mathbf{P}_{c'}}. \quad (6.4)$$

Application of IIC to Minecraft

Similarly to how Ji et al. [104] applied IIC to the task of image classification by pairing images to an image-augmented version of itself, an embodied agent applying actions to the environment and receiving subsequent observations can be considered as a form of applying augmentation to the observations. Temporally proximate observations are more likely to come from the same stage of a task execution. This prior could be used to harvest positive pairs from demonstrations for contrastive learning.

Given a sequence of observations o_0, o_1, \dots, o_T , where T is the length of an episode, it could be assumed that it is likely that neighbouring observations o_t and o_{t+l} (where $0 \leq l < L$ for lookahead $L \ll T$) are classified into the same stage of a task, compared to a randomly selected pair of observations in the dataset. Using this temporal constraint, contrastive learning can be applied to classify observations into clusters.

Preliminary results

Figure 6.1 shows an example of applying IIC to expert demonstrations to cluster the observations. As can be seen, there is some temporal consistency in the cluster assignments



Figure 6.1: Applying IIC to a video sequence of expert demonstration for the Obtain Diamond task. The example shown is for lookahead $L = 1$. The video sequence is subsampled for every 10 frames and shown from left to right, top to bottom. The coloured strips for every tile of observation correspond to clusters found by applying IIC to the observations. There are 30 clusters in this case.

due to similarities in the observations. Some clusters appear early on in the demonstration, whereas others appear later. Identifying these correlations between clusters and task progression may be useful in identifying stages and subtasks within expert demonstrations.

While these qualitative experiments applied IIC clustering on the RGB observations, preliminary experiments were also conducted on the inventory vector space as observations, following the convention in the MineRL BASALT 2021 Competition [195] (as opposed to 2022 [136]). Visualisations are included in Appendix D.

Limitations

While this research direction of segmenting demonstration trajectories based on temporal constraints on observations was promising, several shortcomings were identified which resulted in a pivot in this work. The reasons for this pivot were as follows:

1. Cluster assignments determined by the function $\Phi(x)$ only consider the current observation, or a short window of observations at most if x is a latent embedding

from a transformer encoder. This hinders the aim of using learnt cluster assignments as task stage indicators to inform long-term planning.

2. The latent representations learnt from training $\Phi(x)$ on the IIC objective may be useful; however, this method was conceived before the release of VPT. Since VPT already serves as a semantically sound pre-training method, pre-training with IIC was made somewhat redundant.
3. A unimodal cluster assignment per observation may not be a reasonable constraint, since a scene may contain multiple objects or triggers, each with a different cluster association.

For the above reasons, this research direction was not pursued. However, IIC pre-training may serve as a complementary technique to VPT for learning a semantically meaningful task representation.

6.3.3 Segmenting by actions via task-specific heuristics

To fulfil the original purpose of assigning stages to agent trajectories for long-term planning and temporal consistent strategies, an alternative method is considered. This method has similarities to the Options Framework Section 2.2.5 in that it switches between stages only upon certain triggers. Unlike the IIC-based method proposed in Section 6.3.2, which relies on a clustering function $\Phi(x)$ to perform a classification at every step in the agent rollout, this newly suggested method propagates the stage information forward temporally.

While learning these trigger points is both attractive and desirable, this would require some other way of constraining the learning problem; if supervised learning is employed, some form of data annotation is required; if stage segmentation is to be learnt via self-supervised learning, what constitutes as distinct “stages” must be defined as a

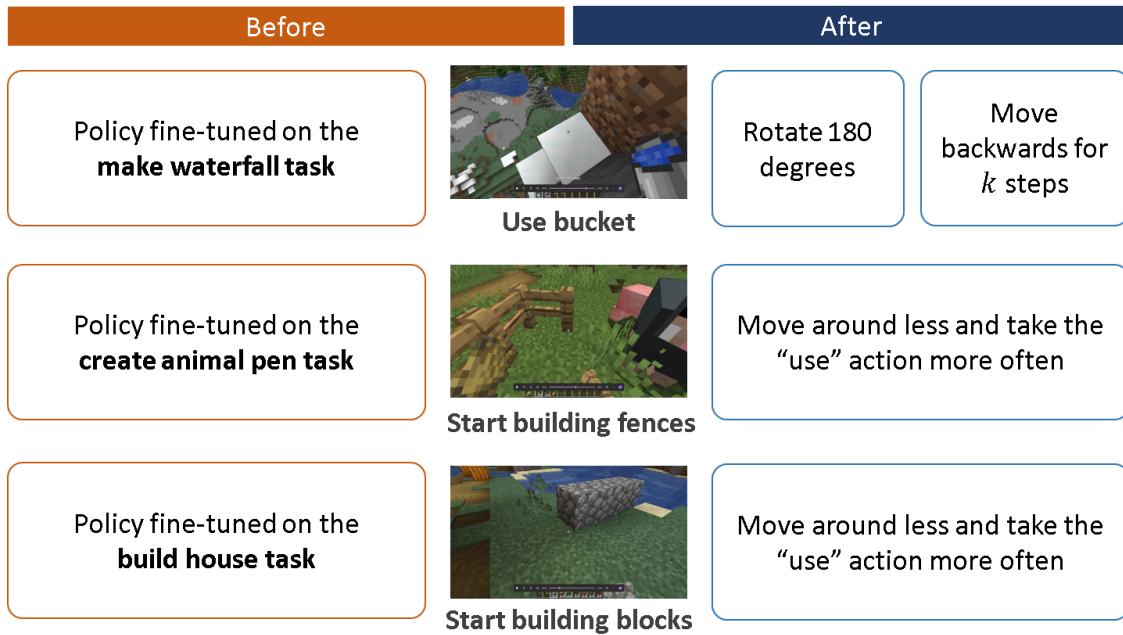


Figure 6.2: Trigger actions defined for the Voggite agent to switch to a different policy during task execution. For the “find a cave task”, no triggers are defined and the VPT policy is fine-tuned as normal (with improvements to training techniques as outlined in Section 6.4.3). For “make a waterfall” task, the agent changes its behaviour to climbing down a mountain after the bucket has been used. For “create an animal pen” and “build a house” tasks, the policy distribution is shifted to move around less and commit to building structures after taking the first “use” action.

training constraint. This problem, also known as change-point detection, is a complex research question on its own.

On the other hand, defining a trigger for each task in the MineRL BASALT Competition is straightforward. It was noted that certain actions that the agent takes (e.g. a “use bucket” in the waterfall task, as well as a “use item” action in the building tasks) is more indicative of a stage change in a task than inspecting changes in the observations. While the observation space is a high-dimensional RGB image, the action space of the VPT model is categorical, which is then converted to cursor movement, mouse click and keyboard actions. The “use” action in particular is easy to track, and is an ideal trigger for a stage change. Figure 6.2 summarises these triggers and the shift in the agent’s policy for each task. This was implemented in the submission to the MineRL BASALT Competition, which is discussed next.

6.4 Submission to the MineRL BASALT Competition

6.4.1 Voggite

The embodied agent submitted to the competition was named *Voggite* after a mineral, following the tradition set in works related to Minecraft (e.g. the *BASALT* competition). This specific mineral was chosen due to its spelt resemblance to VGG (Visual Geometry Group).

Voggite modified and improved upon VPT’s training setup in four aspects. The changes were:

1. pre-computing VPT embeddings such that only a small policy head network has to be trained and fine-tuned via BC, making training faster,
2. enabling reshuffling of training samples since VPT embeddings are pre-computed and no longer have to be evaluated auto-regressively,
3. training setup in PyTorch Lightning for faster training,
4. reweighting actions according to action frequency so that rare actions are effectively sampled more often.

These changes allowed a much faster and more effective iteration of ideas and methods.

Furthermore, the “use” action was identified to be a trigger action for the waterfall, animal pen and house-building tasks. Until the trigger action is taken, the agent follows a standard VPT policy fine-tuned on individual tasks. Once the “use” action is triggered, however, the agent’s behaviours are modified to adapt to needs for different tasks. In the waterfall task, the agent must retreat down the mountain once a bucket is used. The agent is rotated 180° once the trigger action is taken, and then moved backwards for the next 5 seconds before terminating. For the animal pen and housing tasks, once the “use” action is triggered, the agent’s move action probability is reduced, while the “use” action

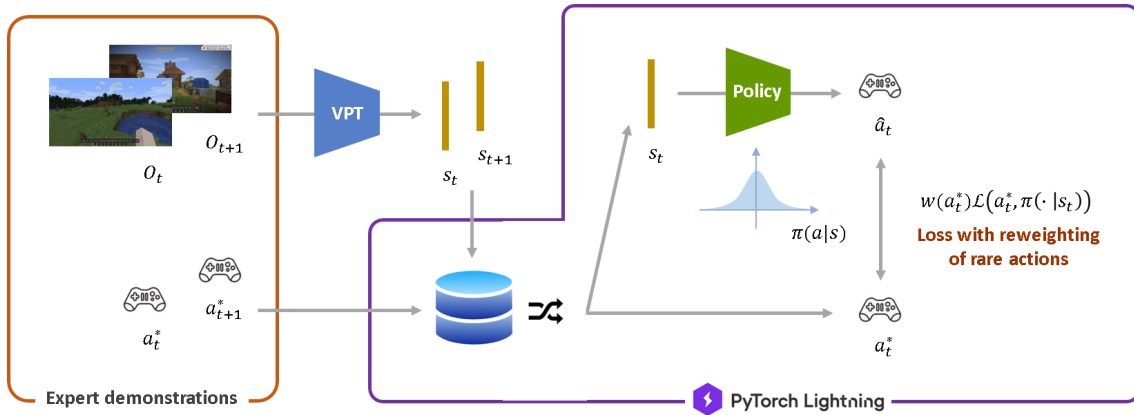


Figure 6.3: Diagram of the Voggitte training pipeline. VPT embeddings are pre-computed for the expert demonstrations and stored as a permutable dataset, removing the sequential constraint of forward-passing through the VPT transformer. A policy head is trained and fine-tuned on each task in the MineRL BASALT Competition, given the VPT embeddings and expert actions labels. The categorical losses are reweighted inversely to the frequency of the expert actions’ occurrences.

probability is increased. This is to ensure that the agent commits to building a house once it has identified a suitable location, rather than aimlessly alternating between exploring and building, as with the baseline VPT.

6.4.2 Implementation

The code for the submitted Voggitte agent is open-sourced, and can be found at https://github.com/shuishida/minerl_basalt_2022.

6.4.3 Submission and results

The competition team consisted of myself (Shu Ishida) as the main contributor, as well as my supervisor (Dr. João F. Henriques) who provided guidance and contributed research ideas. The code for this research and submission was developed entirely by Shu Ishida.

Voggitte was submitted to the MineRL BASALT Competition at NeurIPS 2022 [136]. Voggitte successfully created a waterfall in many of the runs, and improved consistency in building activities over the baseline VPT. Our solution achieved 3rd place out of 63 teams, 446 individual participants and 504 submissions overall. Results are shown in Table 6.1.

Table 6.1: Leaderboard: normalised TrueSkill scores according to [136]. The top three teams were GoUp, UniTeam, and Voggite (ours). Scores for BC-Baseline, two expert humans, and a random agent are also included.

Team	FindCave	MakeWaterfall	AnimalPen	House	Average
GoUp	0.31	1.21	0.28	1.11	0.73
UniTeam	0.56	-0.10	0.02	0.04	0.13
Voggite (ours)	0.21	0.43	-0.20	-0.18	0.06
JustATry	-0.31	-0.02	-0.15	-0.14	-0.15
TheRealMiners	0.07	-0.03	-0.28	-0.38	-0.16
yamato.kataoka	-0.33	-0.20	-0.27	-0.18	-0.25
corianas	-0.05	-0.26	-0.45	-0.24	-0.25
Li and Ivan	-0.15	-0.72	-0.14	-0.22	-0.31
KAIROS	-0.35	-0.32	-0.41	-0.36	-0.36
Miner007	-0.07	-0.76	-0.12	-0.52	-0.37
KABasalt	-0.57	-0.23	-0.41	-0.31	-0.38
Human2	2.52	2.42	2.46	2.34	2.43
Human1	1.94	1.94	2.52	2.28	2.17
BC-Baseline	-0.43	-0.23	-0.19	-0.42	-0.32
Random	-1.80	-1.29	-1.14	-1.16	-1.35

6.5 Conclusion

It was found that executing plans in *stages* is essential for the coherent behaviour of an agent in compositional tasks. *Stages* in this context share similarities with options discussed in Chapter 4. Due to time constraints leading up to the competition, some aspects of the submitted Voggite solution are hard-coded rather than end-to-end learnable, namely the stage switch trigger detection and policy distribution change that follows. However, it served as a proof of concept that a multi-staged policy is necessary to make the agent behaviour more consistent and task-specific. It also retains many of the advantageous properties of VPT, such as knowing how to navigate and craft tools from pre-training without any hard-coding. In future work (see Section 7.2.3), we hope to apply a more learning-oriented approach such as Sequential Option Advantage Propagation (SOAP) (Chapter 4) and TACO [199] to the challenges in Minecraft and other complex tasks for embodied agents.

Chapter 7

Conclusion

The big picture for this thesis was to solve spatial reasoning and planning tasks in a data-driven manner, while simultaneously making the learning more efficient, interpretable and generalisable. For this, Imitation Learning (IL) and Reinforcement Learning (RL) methods were explored and developed to learn from regularities in spatial tasks, acquire transferable skills, and solve novel problems with zero-shot or few-shot learning. The main contributions of the works covered in this thesis are summarised in Section 1.3. In this final chapter, the initial research objectives outlined in Section 1.2 are revisited, and future work is discussed.

7.1 Discussion

Learning a generalisable planner

This thesis explored two avenues of learning a generalisable planner: learning the underlying Markov Decision Process (MDP) directly for an end-to-end differentiable planner (Collision Avoidance Long-term Value Iteration Network (CALVIN), Chapter 3), and iteratively optimising algorithmic plans in the form of code (LangProp, Chapter 5).

The approach presented by CALVIN proved robust to complex structures such as large mazes, correctly learning a translationally invariant transition and reward kernels to model the MDP and solve navigation tasks in novel environments. A valid value map is computed

in an Inverse Reinforcement Learning (IRL) setting of recovering a reward function from expert demonstrations. The original Value Iteration Network (VIN) [216] learnt inaccurate models of the world, inept at handling long-term planning in environments with high branching complexities. In particular, illegal actions leading to collisions were not given large enough penalties. This work mitigated this issue by imposing a structural constraint on the value iteration to explicitly model any impossible actions.

LangProp proposed a novel paradigm of learning algorithmic decision-making from data by treating code as learnable policies with the use of Large Language Models (LLMs). While there has been exciting work in end-to-end differentiable algorithms [69, 147, 164], algorithms are discrete in nature and are hard to learn in an end-to-end differentiable way. Works using RL for algorithm discovery had successes in finding faster algorithms (e.g. AlphaTensor [55], AlphaDev [133]), but at the expense of massive compute and highly tuned task-specific training. LangProp presents an alternative method of tuning algorithms; rather than discovering “novel” algorithms, LangProp is best suited to discover known algorithms and tune them to task-specific needs, leveraging the prior knowledge of foundation models. By making algorithms learnable, high-level and long-term plans that were hitherto too complex for RL agents to learn can now be learnt using IL and RL techniques, as demonstrated in experiments with the CARLA benchmark [25] for autonomous driving.

Furthermore, Chapter 4 and Chapter 6 demonstrate how temporal abstraction using options can help agents make informed long-term decisions. These are further discussed in the following sections on reusable skill learning and memory-augmented policies.

While the challenge of generalisable planning is still not completely solved, these works indicate a path towards learnable algorithms for data-driven long-term planning.

Discovering reusable skills

Chapter 4 addressed the problem of automatic discovery of reusable skills. Ways of discovering options as temporally consistent macroscopic actions were considered. While multiple formulations for this problem have been proposed in prior work [9, 59, 64, 111, 258], experiments in Chapter 4 show that option learning using the forward-backward algorithm [14] or a standard formulation of options following the Options Framework Section 2.2.5 do not effectively learn option assignments that are both causally sound and temporally consistent.

Sequential Option Advantage Propagation (SOAP) presented an alternative formulation by analytically evaluating the policy gradient for an optimal option assignment. While policy gradients cannot be explicitly propagated temporally without keeping all observations in memory, as transformer policies [226] do, SOAP achieves an equivalent effect by extending the concept of the Generalised Advantage Estimate (GAE) to propagate *option advantages* through time. Unlike the forward-backward algorithm which optimises the option assignment over a Hidden Markov Model (HMM) of options, observations and actions, assuming a fully known trajectory, SOAP’s formulation is causal, assuming only the knowledge of historical observations and actions. Since the option policy itself must be causal (only conditional on the history of the agent), the causal SOAP formulation is more robust and appropriate to be used as a causal agent’s learning objective.

Solving POMDP environments with memory-augmented policies

The learnt options could be considered as discretised memory that carries forward historical information of the agent trajectory through time, thereby allowing the agent to disambiguate different states that maps onto the same observations, as is with the case of Partially Observable Markov Decision Process (POMDP) environments.

By learning temporally consistent and distinct options, the SOAP agent was able to

solve POMDP corridor environments that require knowledge beyond the currently available observations. While this is a simple experiment setting, SOAP already outperforms the baselines of Proximal Policy Option-Critic (PPOC) [111] and Double Actor-Critic (DAC) as well as a Long Short-Term Memory (LSTM) recurrent policy. LSTMs and transformers are expressive tools for sequential modelling, often used to solve POMDP environments [12, 218]. However, training such recurrent or sequential policies is computationally expensive, since many frames of historical observations have to be forward-passed simultaneously at training time. The window size for historical observations is limited by the size of the Graphical Processing Unit (GPU) memory available. In comparison, the approach taken by SOAP is memory-efficient, since only the current observation has to be included in the forward-pass. This makes it a promising candidate as an algorithm with the potential to be extended to more long-term planning problems.

In Chapter 6, it was shown that a strategy of options can be extended to solve tasks in more complex POMDP environments such as Minecraft. Having options to disambiguate different stages within a task was essential to achieve consistent task execution without forgetting. While the sub-policies are Video PreTraining (VPT)-based transformers with a local window of observations and do not have long-term memory, the options function as a task-level discrete memory that keeps track of the high-level plan.

Improvements in the training such as pre-computing VPT embeddings and fine-tuning only the policy head with using PyTorch Lightning contributed to faster and more efficient training. Voggite achieved competitive performance in the MineRL BASALT Competition [136], significantly outperforming the VPT baseline, which demonstrated the benefits of using options for execution of embodied tasks with long-term planning.

Explaining the behaviour of experts and agents

CALVIN (Chapter 3) demonstrated an interpretable approach to learnable planning by modelling transitions and learning reward functions to compute explicit reward and value maps. SOAP (Chapter 4) and Voggite (Chapter 6) implemented skill segmentation to separate low-level control from task-level plans. Options serve as a useful indicator that distinguishes between different tasks and subtasks. LangProp (Chapter 5) presented the most interpretable approach of all, representing policies as human-readable code that can also be improved with data-driven learning. Experiments in autonomous driving demonstrated that a known phenomenon called causal confusion when trained with IL can be reproduced with LangProp. Causal confusion was directly diagnosable from the policy that was learnt, even before running the experiments. This could enable faster iteration on ideas and experiments with future work, and empower the field of data-driven explainable Artificial Intelligence (AI).

Training embodied agents to perform complex tasks

This thesis touched upon the problems of robotic navigation (Chapter 3), autonomous driving (Chapter 5), simulated control in Atari [16] and MuJoCo [222] (Chapter 4), and game play in Minecraft (Chapter 6). Methods of learning such complex tasks from data were also suggested. Although these are incomplete solutions, significant improvements to baseline methods were demonstrated by applying techniques of differentiable planning, optimising algorithmic policies with LLMs, hierarchical policies, and option discovery. Learning long-term plans that go beyond reactive policies was a consistent challenge in all the domains. The thesis suggested pathways that may lead to solving this challenge. Details of such pathways are discussed as future work.

7.2 Limitations and future work

7.2.1 CALVIN

While end-to-end differentiable planning is an attractive concept demonstrated in the VINs that this work improves upon, several limitations hinder seamless scaling and deployment to real-world problems. For instance, VINs as well as CALVINs assume the agent’s pose, depth image and the camera parameters to be known. Integration with end-to-end trainable localisation and mapping modules [157, 254] may be considered for future work. A static environment was also assumed where observation embeddings can be aggregated temporally. Extending the method to dynamic environments is non-trivial, since that requires a learnable dynamic mapping system, out of scope for the Lattice PointNet (LPN) approach. The temporally and policy-dependent dynamics of the environment must be modelled as well for an accurate value propagation, which indicates that the VIN approach is fundamentally limited to static environments. Other approaches, such as model-based RL, in particular those on world models [75, 79, 190] may be better suited for modelling such tasks.

A related limitation is that these methods operate in discrete predefined state and action spaces. Extending Value Iteration (VI) to continuous state and action spaces is challenging. Policy gradient and Actor-Critic methods typically solve such problems by learning a policy function that maps continuous states to continuous actions. However, value propagation can no longer be achieved by enumeration, so it is not possible to find and execute an optimal policy zero-shot on novel environments using these methods. Learning a near-optimal policy and an accurate value function with a high zero-shot capability will require a large amount of training data. This makes it challenging for VIN-like architectures to be deployed beyond a navigation setting, such as robot manipulation and planning in higher dimensional spaces.

In order to preserve the benefits of planning by enumeration while extending this to continuous state and action spaces, Hierarchical Reinforcement Learning (HRL) may be required, where low-level sub-policies that handle continuous control are orchestrated by high-level policies that implement long-term plans. The Options Framework [214], analysed in Chapter 4. is one method of achieving this. It may be possible to extend techniques suggested in CALVIN to these settings.

Both VIN and CALVIN are trained only in an IL setting, where the optimal action is explicitly labelled by an expert. While this is helpful in increasing sample efficiency, it limits the scope of application since the optimal action is not always known in many real-world problems. Furthermore, the agent loses the opportunity to learn from its failures. To bridge this gap, an improved form of CALVIN was considered, which allowed training both in IL and RL setups. While the results were not included in this work, preliminary results showed that CALVIN could be extended easily to training in RL settings due to its enhanced interpretability compared to VIN.

Finally, although neural networks hold promise for making navigation more robust under uncertainty, their uninterpretable failure modes mean that they are not yet mature enough for safety-critical applications, and more research to close this gap is still needed. CALVIN is a more interpretable MDP structure for VINs, trained solely on safe offline demonstrations. However, their reliability is far from guaranteed, and complementary safety systems in hardware must be considered in any deployment. In addition to further improving robustness to failures, future work may investigate more complex tasks involving real-world deployment, and study the effect of sensor drift on navigation performance.

7.2.2 SOAP

SOAP demonstrated capabilities of learning options in a POMDP environment of corridors, and showed equivalent performances to the baseline Proximal Policy Optimisation (PPO)

agent in other environments. However, even in simple settings, it took the agent many samples before a correct option assignment was learnt. Option discovery is a difficult chicken and an egg problem, since options need to be assigned correctly in order for the rewards to be obtained and passed onto the options, but without the rewards a correct option assignment may not be learnt. Furthermore, learning to segment episodes into options in an unsupervised way without any pre-training is an ill-defined problem, since there could be many equally valid solutions. Combining the learning objective of SOAP with methods such as curriculum learning to pre-train diverse sub-policies specialised to different tasks may stabilise training.

In the current formulation of SOAP, the options are discrete variables, and are less expressive compared to latent variables in recurrent policies and transformers. This greatly reduces the memory capacity and could hinder learning in POMDP environments. Further research in extending the derivations of SOAP to work with continuous or multi-discrete variables as latents would be desirable, and may lead to making the method scalable to more complex problems.

7.2.3 LangProp

While LangProp successfully harnessed the capability of LLMs to apply data-driven optimisation techniques to code optimisation, LangProp may not be the most appropriate solution for all problems - in fact, neural networks excel in working with continuous state-action spaces and low-level control, whereas LLMs are better at handling high-level planning and reasoning tasks. LangProp intends to propose an alternative learning paradigm that allows LLMs to be used to learn high-level planning which has hitherto been a difficult problem for other machine learning approaches (e.g. neural networks).

There are numerous future research directions that could improve the capability of LangProp as a training framework, as well as give a better theoretical foundation, such as

(a) chaining of modules with a full back-propagation algorithm, (b) improvements to the evolutionary algorithm (e.g. priority mechanism), (c) a robust sampling mechanism for failed examples upon updates, (d) incorporating human feedback in natural language during policy updates, and (e) using LangProp with LLMs fine-tuned for code correction and optimisation tasks. In particular, scaling this approach to larger repositories and complex systems would require a multi-modular approach that can propagate useful learning signals to subcomponents if there are multiple failure points in the system.

Applying LangProp to RL tasks has open questions in credit assignment and value estimation. LangProp demonstrated improvements in RL policies written as code when (a) the policy can be optimised on episodic returns with a Monte-Carlo method (e.g. CartPole), or (b) there is immediate feedback from the environment (e.g. infractions in CARLA). However, for complex tasks that have delayed rewards, it is necessary to have an accurate value/advantage estimator for credit assignment. Since replacing a neural value estimator with a code-based function is not feasible, it is most likely that a hybrid method (having an interpretable code-based actor policy trained with LangProp that uses a value function estimated by a neural network as a critic) would be a way to apply LangProp to complex RL scenarios. However, this is also an open-ended question, which calls for further exploration.

Having an LLM in the RL optimisation means that more useful signals could be potentially harvested from the environment, rather than relying just on sparse scalar rewards for updates. For instance, having descriptive feedback from the Gymnasium environment on the failure modes of the agent, given either as a warning or natural language feedback, can significantly accelerate the learning of the RL agent. This also allows a more seamless integration of human-in-the-loop feedback.

Finally, more investigation is required in terms of the robustness and safety of LLM-written applications. This is applicable to all systems that involve code genera-

tion. While this framework iteratively improves the quality of the code and filters out potential errors that make the final code policy less likely to contain errors, additional safety mechanisms and firewalls are necessary during the training process, since the code is evaluated based on execution, which could potentially be a source of attacks or risk. It is worth emphasising the importance of additional safety precautions before deployment.

LangProp opens up new possibilities for data-driven code development. While zero-shot applications of LLMs have enabled tools such as GitHub Copilot, some suggestions are inaccurate or misaligned with the user’s intentions, whereas if there are data or unit tests that the code needs to satisfy, the code suggestions can be made much more accurate by first running evaluations on these test suites and choosing the best possible suggestion that satisfies the requirements. Planning is one aspect of autonomous driving that has not yet successfully adopted a data-driven approach, for good reasons, since neural networks often struggle to produce generalisable high-level planning rules and are less interpretable. Therefore, most methods currently in deployment have human-engineered planning algorithms. The LangProp framework is insufficient to replace such systems since it lacks the robustness that human-designed systems have to offer, and more research needs to be done in this direction. This work is hoped to provide inspiration for future research to make the framework more robust and safely deployable in the real world.

7.2.4 Voggite

The agent Voggite developed in this work is a prototype example of applying options to task-solving in Minecraft. The work was limited in that there were only two options per task, and the option switching trigger and distribution shifts in sub-policies were manually hard-coded. Further work is required to (a) allow multiple options, (b) make sub-policies fully learnable (in Voggite the sub-policies are manually manipulated from a fine-tuned VPT policy), (c) make triggers observation-dependent as well as action-dependent, and

(d) learn transitions between these multiple options. (a) and (b) are relatively straightforward implementations, since the option segmentation themselves could still be defined task-by-task by a human expert by defining the triggers. Implementing (c) and (d) that involve learning the triggers themselves are trickier, and may require integration of techniques that were suggested in Chapter 4, as well as other pre-training and regularisation techniques to constrain the learning problem. For instance, curriculum learning may be employed to first learn and fine-tune low-level sub-policies, and which are then used as initialisations to learn a task-specific policy that orchestrates these sub-policies and their fine-tuning.

Prior work on learned task segmentation, such as TACO [199] is highly relevant. TACO addressed and solved similar crafting and manipulation tasks which comprise multiple stages, by formulating the segmentation problem as a sequence alignment problem between demonstration trajectories and an acyclic sequence of task descriptors called task sketch. A sequence alignment strategy that extends Connectionist Temporal Classification [68] is employed. While the work in Chapter 6 had a hard deadline for the NeurIPS BASALT Competition [136], which constrained the scope of the work, comparing methods such as TACO against other temporal abstraction strategies (e.g. Proximal Policy Optimisation via Expectation Maximisation (PPOEM) or SOAP) in the Minecraft setting would be informative.

Related to Chapter 4, making options represent more information than just the stage of a task execution is another challenge of extending Voggite beyond sequential tasks. For instance, if options can be multi-categorical, it could be useful for planning out long-term strategies on this discretised option-space. Information such as which objects are in the inventory are innately multi-categorical, and planning to gather or craft a specific item in the technology tree of the Minecraft universe requires the agent to be able to plan on this technology tree, which are either learnt directly from experiences and trial-and-error, or

could be obtained from natural language inputs or reading the documentation.

Planning in natural language space using multi-modal foundation models [233, 234, 259], as well as interfacing with natural language human inputs [127] or directly generating executable code [230] are some promising research directions for learning a high-level plan. Since Voggite was developed prior to these methods being released, the avenue of using LLMs was not explored in this work. However, being able to take advantage of the priors of the world is crucial for solving complex tasks in Minecraft without an exponential amount of trial-and-error. LLMs are also a natural way for the agent to interact with humans and achieve zero-shot task execution.

Appendix A

CALVIN embodied navigation with comparisons

For a visual comparison of Collision Avoidance Long-term Value Iteration Network (CALVIN), Value Iteration Network (VIN) and Gated Path Planning Network (GPPN), rollouts are performed on a randomly generated maze using each of the algorithms, assuming partial observability and embodied navigation.

A.1 Rollout of CALVIN

Figure A.1 shows an example of a trajectory taken by CALVIN at runtime, with corresponding observation maps, predicted values and predicted rewards for taking the “done” action. Similarly to Section 3.5.2, the agent manages to explore unvisited cells and back-track upon a dead end until the target is discovered. One key difference is that now the agent learns to predict rewards and values for every discretised orientation as well as the discretised location. Upon closer inspection, it could be observed that the predicted values are higher facing the direction of unexplored cells and towards the discovered target. Since rotation is a relatively low-cost operation, in this training example, the network seems to have learnt to assign high rewards to a particular orientation at unexplored cells, from which high values propagate. Rewards and values averaged over orientations yield a more

intuitive visualisation.

A.2 Rollout of VIN

A corresponding visualisation for VIN is shown in Figure A.2. Unlike CALVIN’s implementation of rewards (eq. 5 in sec. 4.1) as a function of discretised states and actions, the “reward map” produced by the VIN does not offer a direct interpretation, as it is shared across all actions as implemented by Tamar et al. [216], and is also shared across all orientations in the case of embodied navigation. The values are also not well learnt, with some of the higher values appearing in obstacle cells. The unexplored cells are not assigned sufficiently high values to incentivise exploration by the agent. In this example, the agent gets stuck and starts oscillating between two orientations after 57 steps.

A.3 Rollout of GPPN

Finally, a visualisation for GPPN is shown in Figure A.3. Unlike VIN and CALVIN, GPPN does not have an explicit reward map predictor, but performs value propagation using an Long Short-Term Memory (LSTM) before outputting a final Q-value prediction. Similarly to Appendix A.2, the values predicted are not very interpretable, and do not incentivise exploration or avoidance of dead ends. In this example, the agent keeps revisiting a dead end that has already been explored in the first 20 steps.

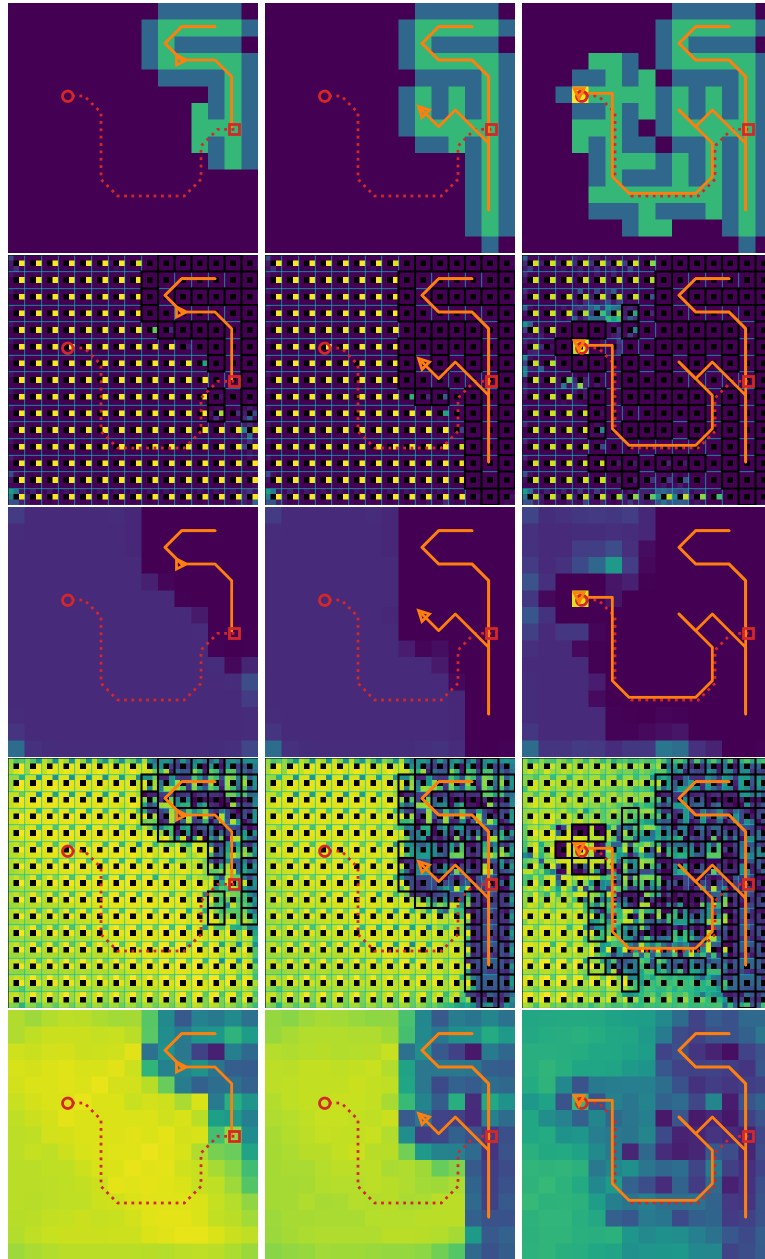


Figure A.1: Example rollout of embodied CALVIN after 30 steps (left column), 60 steps (middle column) and 90 steps (right column). CALVIN successfully terminated at 91 steps. **(first row)** Input visualisation: unexplored cells are dark, and the discovered target is yellow. The correct trajectory is dashed, and the current one is solid. The orange triangle shows the position and the orientation of the agent. **(second row)** Predicted rewards (higher values are brighter). The 3D state-space (position/orientation) is shown, with rewards for the 8 orientations in a radial pattern within each cell (position). Explored cells have low rewards, while unexplored cells and the discovered target are assigned high rewards. **(third row)** Predicted rewards averaged over the 8 orientations. **(fourth row)** Predicted values following the same convention. Values are higher facing the direction of unexplored cells and the target (if discovered). **(fifth row)** Predicted values averaged over the 8 orientations.

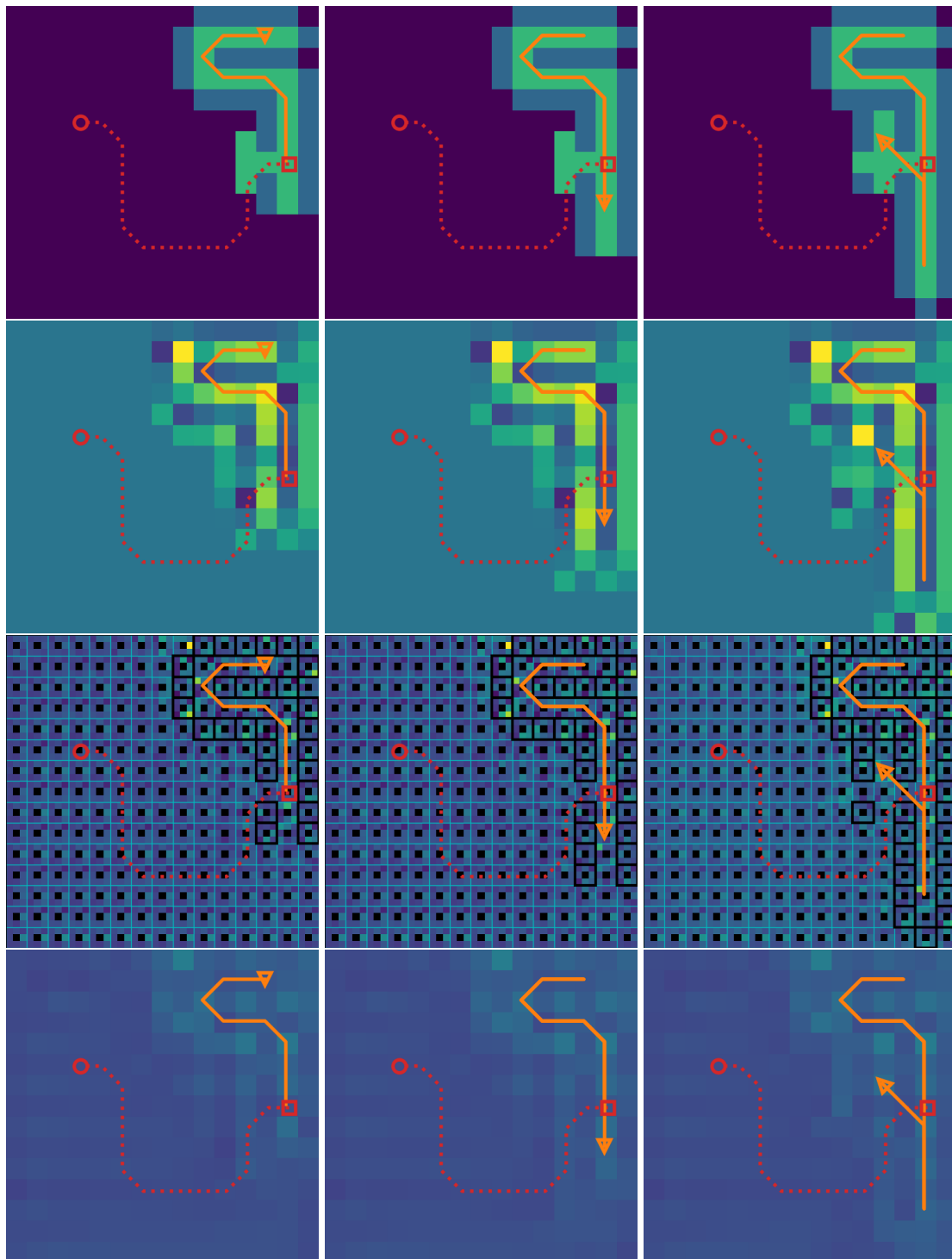


Figure A.2: Example rollout of embodied VIN after 20 steps (left column), 40 steps (middle column) and 60 steps (right column). VIN kept oscillating between the same two states after 57 steps. The convention is the same as for Figure A.1, except that a single reward map is shared across all orientations. **(first row)** Input visualisation. **(second row)** Predicted rewards. **(third row)** Predicted rewards averaged over the 8 orientations. **(fourth row)** Predicted values.

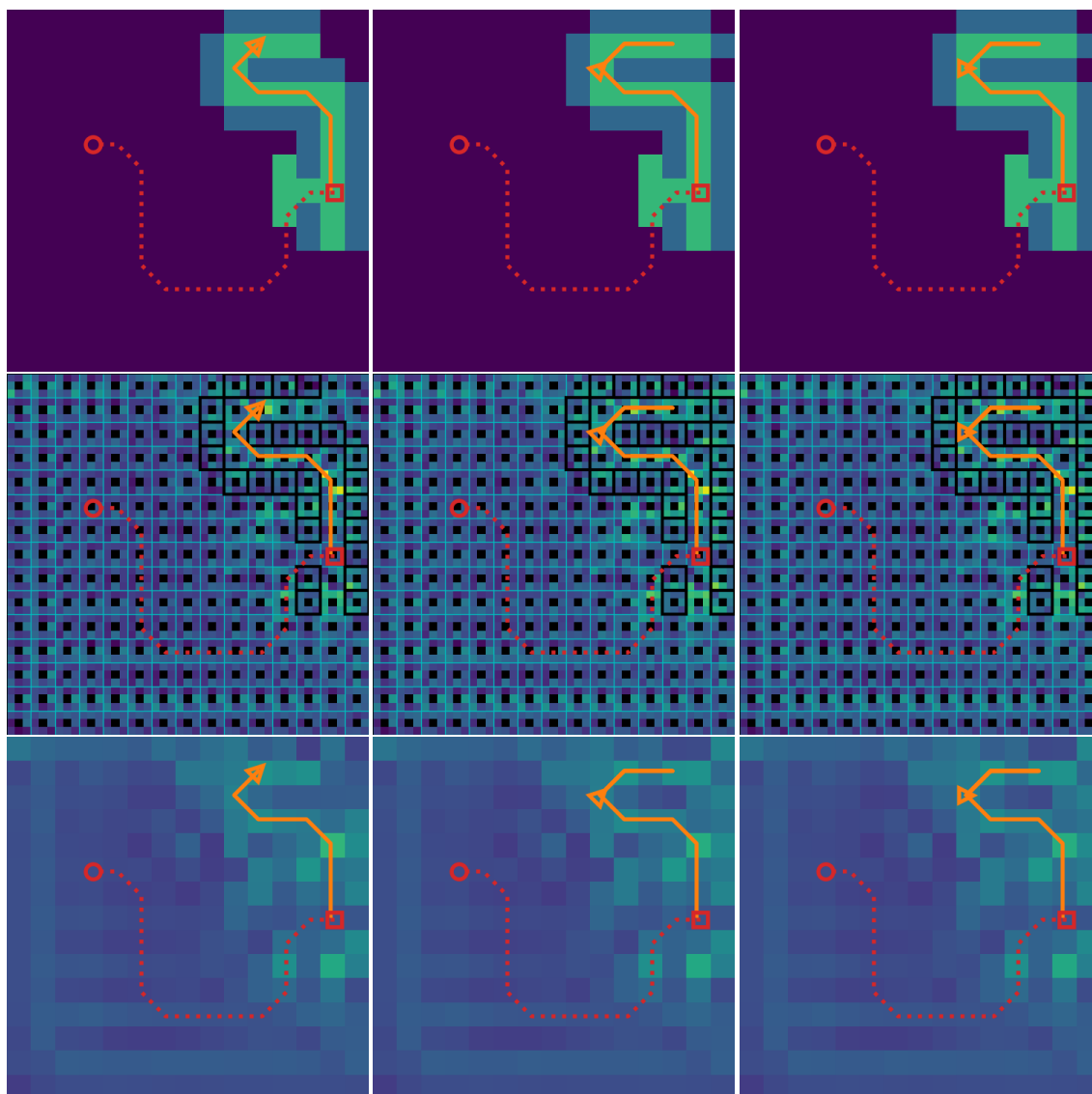


Figure A.3: Example rollout of embodied GPPN after 15 steps (left column), 30 steps (middle column) and 45 steps (right column). GPPN revisits the same sequences of states leading to a dead end after 45 steps. The convention is the same as for Figure A.1. **(first row)** Input visualisation. **(second row)** Predicted rewards. **(third row)** Predicted rewards averaged over the 8 orientations.

Appendix B

Benchmarking option-based Reinforcement Learning agent

Table B.1: The returns of Reinforcement Learning (RL) agents after the maximum environment training steps (100k for CartPole, 1M for LunarLander and MuJoCo, and 10M for Atari).

Environment	PPO	PPOC	PPO-LSTM	DAC	PPOEM	SOAP
Corridor $L = 3$	-0.04	0.76	0.99	0.90	0.99	0.99
Corridor $L = 10$	0.04	0.10	0.37	0.60	0.66	0.93
Corridor $L = 20$	0.00	0.13	0.00	0.34	0.13	0.78
CartPole	499.65	401.87	492.75	500.00	491.30	500.00
LunarLander	195.03	141.07	257.96	159.00	251.71	254.14
Asteroids	1806.14	1760.53	1463.03	2108.87	1985.70	1910.53
Beam Rider	3704.50	689.07	1574.99	2839.36	2691.09	3955.31
Breakout	376.19	4.64	257.57	355.97	51.02	345.68
Enduro	704.17	0.13	655.96	674.03	724.97	593.44
Ms Pacman	1825.40	434.17	1555.80	1806.40	1453.87	2046.87
Pong	20.74	-0.99	20.70	20.65	18.32	20.81
Qbert	12669.59	115.17	12282.50	8212.08	8940.83	11469.25
Road Runner	36247.13	5273.33	33507.00	32042.00	18886.00	29490.33
Seaquest	1539.73	317.80	1501.13	1198.33	1723.50	951.67
Space Invaders	845.73	378.90	756.27	840.63	844.00	1013.40
Ant	2258.28	53.10	987.01	1394.61	49.30	1943.45
Half Cheetah	5398.50	4300.07	4478.37	4251.44	56.07	4962.48
Humanoid	1196.32	1169.26	1169.12	1212.80	275.03	1100.38
Reacher	-4.87	-4.94	-5.04	-4.60	-5.54	-4.87
Swimmer	340.31	119.46	342.84	354.67	337.85	319.45
Walker	2960.18	1631.97	1071.81	1799.50	1517.90	3478.50

Appendix C

LangProp code generation and prompt examples

C.1 Notes on training with LangProp

C.1.1 Choosing the priority discount factor

How the priorities of the policies are calculated has a large effect on the final performance of the trained LangProp model. For a stationary training distribution (e.g. supervised learning on a fixed offline dataset), whether one uses the immediate average score, a running average, or an exponential average does not make a difference except that just using the immediate average score results in a more stochastic result due to fewer numbers of samples. If the computational resources and time are not constrained, one could increase the batch size and just use the immediate average score. If these are constrained, one may adopt a running average with smaller batch sizes. This works when the training distribution is stationary and there are no other changing components other than the policy currently training.

If the training distribution changes or the policy consists of multiple chained modules, each with a learnable sub-policy, it is no longer possible to use a simple running average, and either an objective score of a single large batch or an exponential averaging scheme

must be used. The current implementation of LangProp does not support multiple chained modules, but is a foreseeable and natural extension to the framework. Changes in the training distribution are expected in DAgger or Reinforcement Learning (RL). For training the LangProp agent in Section 5.5.2, a discount factor of $\gamma = 0$ is used, effectively only using the immediate average scores evaluated on a freshly sampled batch. This is because forward passes through the LangProp driving policies are fast due to not having any complex components so it is possible to use a large batch size. However, in applications where forward passes are expensive and the batch size must be small, using exponential averaging with a non-zero discount factor γ is recommended.

C.1.2 Specifying the policy

One of the challenges in the early stages of the project was in specifying the inputs and outputs of the function. Most of the failures in learning a policy were due to misspecification of the inputs, rather than a fundamental problem with the Large Language Model (LLM) or with LangProp. For instance, it was crucial to specify the units of the input values, e.g. m/s , which allowed the LLM to choose sensible values for some internal parameters. It was also important to name input variables explicitly such that it is clear whether the coordinates are given as absolute world coordinates or coordinates relative to the ego vehicle. A useful property of LangProp is that because the LLM has some understanding of the world from natural language, it can easily incorporate this knowledge when generating the code, constraining the search space of feasible code. We can further guide the LLM to generate policies with certain characteristics, e.g. having a larger safety margin, by expressing preferences in the prompts. This adds to the benefits of the LangProp approach, where it is easier to encourage policies to exhibit certain behaviours.

C.2 LangProp prompt definitions

LangProp as a framework can be used to optimize a diverse range of code optimization problems. The functionality of the model is determined by the choices in the setup prompt, the update prompt, and the dataset that the LangProp model is trained on.

C.2.1 Policy setup prompt examples

We provide simple examples of learning a Sudoku algorithm, and learning a policy that plays CartPole-v1, a widely used reinforcement learning environment in [22], to show the generality of the framework. The setup prompt should include the specification of the function’s inputs and outputs and their types in the form of a docstring.

```
1 I am developing code to solve a sudoku puzzle. Please write a function which
  ↳ takes a numpy array of an unsolved sudoku puzzle and return a complete
  ↳ solution.
2
3 Here is the definition of the function.
4
5 ...
6 Given a numpy array of non-negative integers as a starting condition of a
  ↳ sudoku puzzle, the function returns a complete solution, also as a numpy
  ↳ array.
7 The inputs to the function are the incomplete sudoku numpy array, the width
  ↳ of the sudoku, and the height of the sudoku. Note that the overall numpy
  ↳ array has a shape of (height x width, width x height).
8 For example, if we are solving a conventional 3x3 sudoku, the width is 3,
  ↳ the height is 3, and the numpy array is 9x9. The constraint of a sudoku
  ↳ puzzle is that, every row, every column, and every block of dimensions
  ↳ (height, width) should contain unique values of 1 to height x width, one
  ↳ each. The unfinished sudoku puzzle is given as a numpy array of integers
  ↳ where some of the values are filled, and other values which are unsolved
  ↳ are 0. The function returns a complete sudoku puzzle as an numpy array
  ↳ of integers.
```

```
9
10 Args:
11     - sudoku: np.ndarray      # shape of (height x width, width x height)
12     - width: int
13     - height: int
14
15 Returns:
16     - solution: np.ndarray    # shape of (height x width, width x height)
17 ```
18
19
20 This is a template of the code.
21
22 ```python
23 def {{ function_name }}(sudoku: np.ndarray, width: int, height: int) ->
24     ↪ np.ndarray:
25     # Write code here
26     return solution
27 ```
28 Please do the following:
29 Step 1. Describe step by step what the code should do in order to achieve
30     ↪ its task.
31 Step 2. Provide a python code solution that implements your strategy,
32     ↪ including all necessary import statements.
33
34
```

Listing C.1: Setup prompt template to learn an algorithm to solve generalized Sudoku

```
1 I am developing code to solve CartPole. Please write a function which takes
2     ↪ the position and velocity of the cart, and the angle and angular
3     ↪ velocity of the pole, and return the action that the policy should take.
4
5 Here is the definition of the function.
6
7 ```
```

```
6 Given the position and velocity of the cart, and the angle and angular
  ↪ velocity of the pole, return the action that the policy should take to
  ↪ balance the pole on the cart.
7
8 Args:
9   - cart_position: float          # range of -4.8 to 4.8 [m]
10  - cart_velocity: float         # range of -inf to +inf [m/s]
11  - pole_angle: float           # range of -0.418 to 0.418 [radian]
12  - pole_angular_velocity: float # range of -inf to +inf [radian/s]
13
14 Returns:
15  - action: int    # 0 if the cart should be pushed to the left (negative
  ↪ direction), 1 if it should be pushed to the right (positive
  ↪ direction)
16 ```
17
18
19 This is a template of the code.
20
21 ```python
22 def {{ function_name }}(cart_position, cart_velocity, pole_angle,
  ↪ pole_angular_velocity) -> int:
23     # Write code here
24     return action
25 ```
26
27 Please do the following:
28 Step 1. Describe step by step what the code should do in order to achieve
  ↪ its task.
29 Step 2. Provide a python code solution that implements your strategy,
  ↪ including all necessary import statements.
30
```

Listing C.2: Setup prompt template to learn an agent policy to play CartPole

C.2.2 Policy update prompt example

The prompt used to update the policy contains the same information as the setup prompt, but in addition, has example inputs and outputs where the code had failed to produce a valid prediction. If there was an exception or printed messages during the execution of the code, this will also be provided as feedback. The LLM is asked to identify the source of the sub-optimal performance and rewrite the code to achieve a higher score.

```
1 I am developing code to solve a sudoku puzzle. Please write a function which
  ↳ takes a numpy array of an unsolved sudoku puzzle and return a complete
  ↳ solution.
2
3 Here is the definition of the function.
4
5 ```
6 Given a numpy array of non-negative integers as a starting condition of a
  ↳ sudoku puzzle, the function returns a complete solution, also as a numpy
  ↳ array.
7 The inputs to the function are the incomplete sudoku numpy array, the width
  ↳ of the sudoku, and the height of the sudoku.
8 Note that the overall numpy array has a shape of (height x width, width x
  ↳ height).
9 For example, if we are solving a conventional 3x3 sudoku, the width is 3,
  ↳ the height is 3, and the numpy array is 9x9.
10 The constraint of a sudoku puzzle is that, every row, every column, and
  ↳ every block of dimensions (height, width) should contain unique values
  ↳ of 1 to height x width, one each.
11 The unfinished sudoku puzzle is given as a numpy array of integers where
  ↳ some of the values are filled, and other values which are unsolved are
  ↳ 0.
12 The function returns a complete sudoku puzzle as an numpy array of integers.
13
14 Args:
15     - sudoku: np.ndarray          # shape of (height x width, width x height)
16     - width: int
17     - height: int
```

```
18
19 Returns:
20     - solution: np.ndarray      # shape of (height x width, width x height)
21     ```
22
23
24 Here is an example code that I have written. However, it is not working as
    ↪ expected.
25
26 ```python
27 {{ code }}
28 ```
29
30 I executed the code, and got an accuracy of {{ int(avg_score * 100) }}%.
31
32 $begin
33 if printed:
34     print("There was a print message saying: {{ printed }}")
35 if exception:
36     print("""The code failed to run because there was an exception. The
    ↪ exception message was as follows: {{ exception }}""")
37     print("Resolving this exception is the top priority.")
38 else:
39     if {{ same_outputs }}:
40         print("""It also seems that the code produces the same output of
    ↪ ```{{ outputs }}``` for all inputs, which is not the behaviour
    ↪ we want.""")
41
42     print("""
43
44 The code produced incorrect results for the following inputs. The
    ↪ prediction, ground truth label and score were as follows.
45
46 Inputs: sudoku = {{ args[0] }}, width = {{ args[1] }}, height = {{ args[2]
    ↪ }}
47 Incorrect prediction: solution = {{ outputs }}""")
48     if {{ label is not None }}:
49         print("""Ground truth label: solution = {{ label[0] }}""")
```

```
50     print("Score: {{ int(score * 100) }}%")
51 $end
52
53 $begin
54 if feedback:
55     print("""{{ feedback }}""")
56 $end
57
58 Please do the following:
59
60 $begin
61 if exception:
62     print("Step 1. Look at the error message carefully and identify the
        ↪ reason why the code failed, and how it can be corrected.")
63 else:
64     print("Step 1. Given the example input and output, identify the reason
        ↪ why the code made a wrong prediction, and how it can be corrected to
        ↪ achieve a good score.")
65 $end
66
67 Step 2. Describe step by step what the code should do in order to achieve
        ↪ its task.
68 Step 3. Please rewrite the python function `{{ function_name }}` to achieve
        ↪ a higher score, including all necessary import statements.
69
```

Listing C.3: Update prompt template to learn an algorithm to solve generalized Sudoku

```
1 I am developing code to solve CartPole. Please write a function which takes
  ↪ the position and velocity of the cart, and the angle and angular
  ↪ velocity of the pole, and return the action that the policy should take.
2
3 Here is the definition of the function.
4
5 ```
```

```
6 Given the position and velocity of the cart, and the angle and angular
  ↪ velocity of the pole, return the action that the policy should take to
  ↪ balance the pole on the cart.
7
8 Args:
9   - cart_position: float          # range of -4.8 to 4.8 [m]
10  - cart_velocity: float         # range of -inf to +inf [m/s]
11  - pole_angle: float           # range of -0.418 to 0.418 [radian]
12  - pole_angular_velocity: float # range of -inf to +inf [radian/s]
13
14 Returns:
15  - action: int    # 0 if the cart should be pushed to the left (negative
  ↪ direction), 1 if it should be pushed to the right (positive
  ↪ direction)
16 ```
17
18
19 Here is an example code that I have written. However, it is not working as
  ↪ expected.
20
21 ```python
22 {{ code }}
23 ```
24
25 I executed the code, but the performance was not very high. I got a score of
  ↪ {{ int(avg_score) }} where a good score is 500.
26
27 $begin
28 if printed:
29     print("There was a print message saying: {{ printed }}")
30 if exception:
31     print("""The code failed to run because there was an exception. The
  ↪ exception message was as follows: {{ exception }}""")
32     print("Resolving this exception is the top priority.")
33 else:
34     if {{ same_outputs }}:
```

```
35     print("It also seems that the code produces the same output of ```{{
      ↪ outputs ``` for all inputs, which is not the behaviour we
      ↪ want.")
36 $end
37
38 $begin
39 if feedback:
40     print("""{{ feedback }}""")
41 $end
42
43 Please do the following:
44
45 $begin
46 if exception:
47     print("Step 1. Look at the error message carefully and identify the
      ↪ reason why the code failed, and how it can be corrected.")
48 else:
49     print("Step 1. Given the example input and output, identify the reason
      ↪ why the code made a wrong prediction, and how it can be corrected to
      ↪ achieve a good score.")
50 $end
51
52 Step 2. Describe step by step what the code should do in order to achieve
      ↪ its task.
53 Step 3. Please rewrite the python function `{{ function_name }}` to achieve
      ↪ a higher score, including all necessary import statements.
54
```

Listing C.4: Update prompt template to learn an agent policy to play CartPole

C.2.3 Policy definition for the LangProp driving agent in CARLA

The driving policy is given the location, orientation, speed, length, and width of the ego vehicle, other vehicles and pedestrians in the scene, the distances to the next red traffic light and stop sign, and the target waypoint (4 *m* ahead, used by other baseline experts),


all in absolute world coordinates.

```

1 ...
2 Args:
3     - scene_info: dict
4         Contains the following information:
5         {
6             "ego_location_world_coord": np.ndarray,          # numpy array of
              ↳ shape (2,) which contains (x, y) of the center location of
              ↳ the ego vehicle in world coordinates given in [m]
7             "ego_target_location_world_coord": np.ndarray,  # numpy array of
              ↳ shape (2,) which contains (x, y) of the target location of
              ↳ the ego vehicle in world coordinates given in [m]
8             "ego_orientation_unit_vector": np.ndarray,      # numpy array of
              ↳ shape (2,) which contains (x, y) of unit vector orientation
              ↳ of the ego vehicle in world coordinates. The vehicle moves
              ↳ in the direction of the orientation.
9             "ego_forward_speed": float,                    # the speed of
              ↳ the ego vehicle given in [m/s].
10            "ego_length": float,                            # length of the
              ↳ ego vehicle in the orientation direction, given in [m/s].
11            "ego_width": float,                             # width of the
              ↳ ego vehicle perpendicular to the orientation direction,
              ↳ given in [m].
12            "distance_to_red_light": Union[float, None],    # distance to
              ↳ red light given in [m]. None if no traffic lights are
              ↳ affecting the ego vehicle
13            "distance_to_stop_sign": Union[float, None],    # distance to
              ↳ stop sign given in [m]. None if no stop signs are affecting
              ↳ the ego vehicle
14            "vehicles": {                                    # dictionary of nearby
              ↳ vehicles
15                <vehicle_id: int>: {
16                    "location_world_coord": np.ndarray,    # numpy array of
                      ↳ shape (2,) which contains (x, y) of the center
                      ↳ location of vehicle <vehicle_id> in world
                      ↳ coordinates given in [m]

```

```
17         "orientation_unit_vector": np.ndarray, # numpy array of
↪ shape (2,) which contains (x, y) of unit vector
↪ orientation of vehicle <vehicle_id> in world
↪ coordinates. The vehicle moves in the direction of
↪ the orientation.
18         "forward_speed": float, # speed of
↪ vehicle <vehicle_id> given in [m/s].
19         "forward_length": float, # length of the
↪ vehicle <vehicle_id> along the orientation direction,
↪ given in [m].
20         "sideways_width": float, # width of the
↪ vehicle <vehicle_id> perpendicular to the
↪ orientation direction, given in [m].
21     },
22 },
23 "pedestrians": { # dictionary of nearby
↪ pedestrians
24     <pedestrian_id: int>: {
25         "location_world_coord": np.ndarray, # numpy array of
↪ shape (2,) which contains (x, y) of the center
↪ location of pedestrian <pedestrian_id> in world
↪ coordinates given in [m]
26         "orientation_unit_vector": np.ndarray, # numpy array of
↪ shape (2,) which contains (x, y) of unit vector
↪ orientation of pedestrian <pedestrian_id> in world
↪ coordinates. The vehicle moves in the direction of
↪ the orientation.
27         "forward_speed": float, # speed of
↪ pedestrian <pedestrian_id> relative to the
↪ orientation given in [m/s].
28         "forward_length": float, # length of the
↪ pedestrian <pedestrian_id> along the orientation
↪ direction, given in [m].
29         "sideways_width": float, # width of the
↪ pedestrian <pedestrian_id> perpendicular to the
↪ orientation direction, given in [m].
30     },
31 }
```

```
32     }
33
34 Returns:
35 - speed_level: str           # Choose from ("MOVE", "SLOW", "STOP").
36 - turn_angle: float         # Predicted turn angle of the ego vehicle to
    ↪ reach the target waypoint in [degrees]. The range should be between
    ↪ -180 to 180 degrees
37 
```

Listing C.5: Docstring given as part of the setup prompt for the LangProp agent

C.3 Code generation examples

C.3.1 Solutions for Sudoku

Incorrect solution generated zero-shot

```
1 import numpy as np
2
3 def find_empty(sudoku):
4     for i in range(len(sudoku)):
5         for j in range(len(sudoku[0])):
6             if sudoku[i][j] == 0:
7                 return (i, j) # row, col
8     return None
9
10 def is_valid(sudoku, num, pos):
11     # Check row
12     for i in range(len(sudoku[0])):
13         if sudoku[pos[0]][i] == num and pos[1] != i:
14             return False
15
16     # Check column
17     for i in range(len(sudoku)):
```

```
18     if sudoku[i][pos[1]] == num and pos[0] != i:
19         return False
20
21     # Check 3x3 box
22     box_x = pos[1] // 3
23     box_y = pos[0] // 3
24
25     for i in range(box_y*3, box_y*3 + 3):
26         for j in range(box_x * 3, box_x*3 + 3):
27             if sudoku[i][j] == num and (i,j) != pos:
28                 return False
29
30     return True
31
32
33 def solve_sudoku(sudoku: np.ndarray, width: int, height: int) -> np.ndarray:
34     find = find_empty(sudoku)
35     if not find:
36         return sudoku
37     else:
38         row, col = find
39
40     for i in range(1, width*height + 1):
41         if is_valid(sudoku, i, (row, col)):
42             sudoku[row][col] = i
43
44             if solve_sudoku(sudoku, width, height) is not None:
45                 return sudoku
46
47             sudoku[row][col] = 0
48
49     return None
```

Listing C.6: Example code to solve Sudoku generated zero-shot before LangProp optimization. The code is instructed to solve a general Sudoku with subgrids of size $H \times W$, but confuses it with the standard 3×3 Sudoku.

Correct solution after applying LangProp

```
1 import numpy as np
2
3 def find_empty(sudoku):
4     for i in range(len(sudoku)):
5         for j in range(len(sudoku[0])):
6             if sudoku[i][j] == 0:
7                 return (i, j) # row, col
8     return None
9
10 def is_valid(sudoku, num, pos):
11     # Check row
12     for i in range(len(sudoku[0])):
13         if sudoku[pos[0]][i] == num and pos[1] != i:
14             return False
15
16     # Check column
17     for i in range(len(sudoku)):
18         if sudoku[i][pos[1]] == num and pos[0] != i:
19             return False
20
21     # Check 3x3 box
22     box_x = pos[1] // 3
23     box_y = pos[0] // 3
24
25     for i in range(box_y*3, box_y*3 + 3):
26         for j in range(box_x * 3, box_x*3 + 3):
27             if sudoku[i][j] == num and (i,j) != pos:
28                 return False
29
30     return True
31
32
33 def solve_sudoku(sudoku: np.ndarray, width: int, height: int) -> np.ndarray:
34     find = find_empty(sudoku)
35     if not find:
```

```
36     return sudoku
37 else:
38     row, col = find
39
40 for i in range(1, width*height + 1):
41     if is_valid(sudoku, i, (row, col)):
42         sudoku[row][col] = i
43
44         if solve_sudoku(sudoku, width, height) is not None:
45             return sudoku
46
47         sudoku[row][col] = 0
48
49 return None
```

Listing C.7: Example code to solve Sudoku after LangProp optimization. The code outputs a correct solution.

C.3.2 Solutions for CartPole

Incorrect solution generated zero-shot

```
1 import numpy as np
2
3 def solve_cartpole(cart_position, cart_velocity, pole_angle,
4     ↪ pole_angular_velocity) -> int:
5     # Set the threshold for the pole angle
6     threshold_angle = 0.1
7
8     # Check the angle of the pole
9     if np.abs(pole_angle) < threshold_angle:
10         # If the angle is small enough, take no action
11         action = 0
12     else:
```

```
13     # If the angle is too large, push the cart in the direction of the  
    ↔ pole  
14     if pole_angle > 0:  
15         # Push the cart to the right  
16         action = 1  
17     else:  
18         # Push the cart to the left  
19         action = 0  
20  
21     return action
```

Listing C.8: Example policy to solve CartPole generated zero-shot before LangProp optimization.

The overly simplistic policy achieves a mean score of 9.9 out of 500.

Correct solution after applying LangProp

```
1 import numpy as np  
2  
3 def solve_cartpole(cart_position: float, cart_velocity: float, pole_angle:  
    ↔ float, pole_angular_velocity: float) -> int:  
4  
5     # Define the constants for the PD controller  
6     kp_pole = 0.5  
7     kd_pole = 0.1  
8  
9     kp_cart = 0.2  
10    kd_cart = 0.05  
11  
12    # Compute error signals and their derivatives  
13    pole_error = pole_angle  
14    pole_error_derivative = pole_angular_velocity  
15  
16    cart_error = cart_position  
17    cart_error_derivative = cart_velocity  
18  
19    # Compute the control signals for the pole and cart using PD controllers
```

```

20 pole_control_signal = kp_pole * pole_error + kd_pole *
    ↪ pole_error_derivative
21 cart_control_signal = kp_cart * cart_error + kd_cart *
    ↪ cart_error_derivative
22
23 # Combine the control signals to get the final action
24 if pole_control_signal < -cart_control_signal:
25     action = 0
26 elif pole_control_signal > cart_control_signal:
27     action = 1
28 else:
29     action = 1
30
31 return action

```

Listing C.9: Example policy to solve CartPole after LangProp optimization. The policy learns a PID controller and achieves a mean score of 500 out of 500.

C.3.3 Driving code generated by LangProp

We show an example driving policy generated using LangProp, trained with both imitation learning and reinforcement learning, as described in Section 5.5.2. Please refer to the open-sourced code repository (Section 5.6) for the full prompts and code used to train the policy, and pre-trained checkpoints for each training setting used in the evaluation.

```

1 import numpy as np
2 from typing import List, Tuple, Union, Dict
3
4 def predict_speed_and_steering(scene_info: Dict, stopping_time_margin=2.0,
5                               reaction_time=1.0, stopping_speed=0.1,
6                               ↪ slow_speed=2.5, safety_margin=2.0) ->
7                               ↪ Tuple[str, float]:
8
9     """
10    Given the state of the scene, find the required speed level and turn
11    ↪ angle of the ego vehicle.

```

```
8
9  Args:
10     scene_info (dict): Contains information about the scene.
11     stopping_time_margin (float): Time margin to determine when to stop
    ↪ in seconds [default: 2.0]
12     reaction_time (float): Time taken to react to external events in
    ↪ seconds [default: 1.0]
13     stopping_speed (float): Minimum speed below which the vehicle should
    ↪ stop in m/s [default: 0.1]
14     slow_speed (float): Speed at which the vehicle should slow down in
    ↪ m/s [default: 2.5]
15     safety_margin (float): Safety margin in meters between the vehicle
    ↪ and other objects [default: 2.0]
16
17 Returns:
18     (str, float): A tuple containing the required speed level and turn
    ↪ angle of the ego vehicle.
19
    The speed level can be "MOVE", "SLOW", or "STOP". The
    ↪ turn angle should be a float between -180 to 180
    ↪ degrees.
20
21 """
22 # Step 1: Given the input dictionary, extract all the relevant
    ↪ parameters.
23 ego_location = np.array(scene_info['ego_location_world_coord'])
24 ego_target_location =
    ↪ np.array(scene_info['ego_target_location_world_coord'])
25 ego_orientation = np.array(scene_info['ego_orientation_unit_vector'])
26 ego_forward_speed = scene_info['ego_forward_speed']
27 ego_length = scene_info['ego_length'] + safety_margin
28 ego_width = scene_info['ego_width'] + safety_margin
29 vehicles = scene_info['vehicles']
30 pedestrians = scene_info['pedestrians']
31 distance_to_stop_sign = scene_info['distance_to_stop_sign']
32 distance_to_red_light = scene_info['distance_to_red_light']
33
34 # Step 2: Convert all coordinates for vehicles and pedestrians to
    ↪ relative coordinates in terms of the ego vehicle orientation.
35 vehicles_rel = {}
```

```

35     for k, v in vehicles.items():
36         location = np.array(v['location_world_coord']) - ego_location
37         orientation = np.array(v['orientation_unit_vector'])
38         speed = v['forward_speed']
39         length = v['forward_length'] + v['sideways_width'] + safety_margin
40         width = v['sideways_width'] + safety_margin
41         location_rel = np.dot(location, ego_orientation),
         ↪ np.abs(np.dot([-ego_orientation[1], ego_orientation[0]],
         ↪ location))
42     if location_rel[0] > 0:
43         vehicles_rel[k] = {'location_rel': location_rel, 'speed': speed,
         ↪ 'length': length, 'width': width}
44
45     pedestrians_rel = {}
46     for k, v in pedestrians.items():
47         location = np.array(v['location_world_coord']) - ego_location
48         orientation = np.array(v['orientation_unit_vector'])
49         speed = v['forward_speed']
50         length = v['forward_length'] + safety_margin
51         width = v['sideways_width'] + safety_margin
52         location_rel = np.dot(location, ego_orientation),
         ↪ np.abs(np.dot([-ego_orientation[1], ego_orientation[0]],
         ↪ location))
53     if location_rel[0] > 0:
54         pedestrians_rel[k] = {'location_rel': location_rel, 'speed':
         ↪ speed, 'length': length, 'width': width}
55
56     # Step 3: Compute the Euclidean distance from the ego location to the
         ↪ target location.
57     distance_to_target = np.linalg.norm(ego_target_location - ego_location)
58
59     # Step 4: Calculate the threshold stopping distance and threshold slow
         ↪ distance based on the current speed.
60     if ego_forward_speed < stopping_speed:
61         stopping_distance = safety_margin
62         slow_distance = ego_length / 2
63     else:

```

```
64     stopping_distance = ((ego_forward_speed - stopping_speed) ** 2) / (2
    ↪ * 0.7) + safety_margin + ego_length / 2
65     slow_distance = ((ego_forward_speed - slow_speed) ** 2) / (2 * 0.7)
    ↪ + safety_margin + ego_length / 2
66
67     # Step 5: Check if there is a stop sign and the distance is smaller than
    ↪ the stopping distance. If yes, initiate a stop action if the speed
    ↪ is greater than the stopping speed.
68     if distance_to_stop_sign is not None and distance_to_stop_sign <
    ↪ stopping_distance:
69         if ego_forward_speed <= stopping_speed:
70             speed_level = "MOVE"
71         else:
72             stopping_speed_current = max(distance_to_stop_sign / 2,
    ↪ stopping_speed)
73             if ego_forward_speed > stopping_speed_current:
74                 ego_forward_speed = stopping_speed_current
75                 speed_level = "STOP"
76             else:
77                 speed_level = "MOVE"
78
79     # Step 6: Check if there is a red light and the distance is smaller than
    ↪ the stopping distance. If yes, initiate a stop action.
80     elif distance_to_red_light is not None and distance_to_red_light <
    ↪ stopping_distance:
81         speed_level = "STOP"
82
83     # Step 7: Check for vehicles and pedestrians that may cause collision
    ↪ course, and decide whether to STOP, SLOW or MOVE the ego vehicle.
84     else:
85         collision_vehicle = False
86         collision_pedestrian = False
87         min_longitudinal_distance = stopping_distance
88         min_lateral_distance = float('inf')
89         for k, v in vehicles_rel.items():
90             location_rel = v['location_rel']
91             speed = v['speed']
92             length = v['length']
```

```

93     width = v['width']
94     longitudinal_distance = location_rel[0] - v['length'] / 2 -
    ↪ ego_length / 2
95
96     # check if there is a collision course with the ego vehicle
97     if np.abs(location_rel[1]) <= width / 2 + ego_width / 2 and
    ↪ longitudinal_distance <= stopping_distance:
98         collision_vehicle = True
99         if longitudinal_distance <= 0:
100             speed_level = "STOP"
101             break
102     # check if the vehicle is within safety margin
103     if longitudinal_distance < stopping_distance and
    ↪ np.abs(location_rel[1]) <= width / 2 + ego_width / 2:
104         if np.abs(speed - ego_forward_speed) < 0.5 and speed <=
    ↪ ego_forward_speed:
105             continue
106         min_longitudinal_distance = min(longitudinal_distance -
    ↪ v['length'] / 2 - ego_length / 2,
    ↪ min_longitudinal_distance)
107         min_lateral_distance = np.minimum(width / 2 + ego_width / 2 -
    ↪ np.abs(location_rel[1]), min_lateral_distance)
108
109     for k, v in pedestrians_rel.items():
110         location_rel = v['location_rel']
111         speed = v['speed']
112         length = v['length']
113         width = v['width']
114         longitudinal_distance = location_rel[0] - length / 2 -
    ↪ ego_length / 2
115
116         # check if there is a collision course with the ego vehicle
117         if np.abs(location_rel[1]) <= width / 2 + ego_width / 2 and
    ↪ longitudinal_distance <= stopping_distance:
118             collision_pedestrian = True
119             if longitudinal_distance <= 0:
120                 speed_level = "STOP"
121                 break

```

```

122         # check if the pedestrian is within safety margin
123         if longitudinal_distance < stopping_distance and
        ↪ np.abs(location_rel[1]) <= width / 2 + ego_width / 2:
124             if np.abs(speed - ego_forward_speed) < 0.5 and speed <=
        ↪ ego_forward_speed:
125                 continue
126             min_longitudinal_distance = min(longitudinal_distance -
        ↪ length / 2 - ego_length / 2, min_longitudinal_distance)
127             min_lateral_distance = np.minimum(width / 2 + ego_width / 2 -
        ↪ np.abs(location_rel[1]), min_lateral_distance)
128
129         # Step 8: Initiate a stop action if the ego vehicle is about to
        ↪ collide with a nearby vehicle or pedestrian.
130         if collision_vehicle or collision_pedestrian or
        ↪ min_longitudinal_distance <= safety_margin/2 or
        ↪ min_lateral_distance <= safety_margin/2:
131             speed_level = "STOP"
132             ego_forward_speed = 0
133         # Step 9: Initiate a slow action if the vehicles or pedestrian
        ↪ within the safe stopping distance margin.
134         elif min_longitudinal_distance <= slow_distance and
        ↪ min_longitudinal_distance >= stopping_distance and
        ↪ min_lateral_distance <= ego_width:
135             speed_level = "SLOW"
136             if np.abs(min_lateral_distance) > 0 and
        ↪ np.abs(min_lateral_distance - ego_width) > 0:
137                 speed_factor = (min_longitudinal_distance -
        ↪ stopping_distance) / (slow_distance -
        ↪ stopping_distance/2)
138                 speed_factor = min(max(0.0, speed_factor), 1.0)
139                 ego_forward_speed = slow_speed * speed_factor +
        ↪ ego_forward_speed * (1 - speed_factor)
140         # Step 10: Initiate a move action if no obstacles are present
141         else:
142             speed_level = "MOVE"
143             ego_forward_speed = min(ego_forward_speed + 0.2, 6.0)
144

```

```
145     # Step 11: Compute the angle between the ego vehicle orientation and the  
        ↪ vector pointing to the target in world coordinates.  
146     target_direction = ego_target_location - ego_location  
147     target_direction_ego = np.dot(target_direction, ego_orientation),  
        ↪ np.dot([-ego_orientation[1], ego_orientation[0]], target_direction)  
148  
149     # Step 12: Rotate the vector to the coordinate system of the ego vehicle  
        ↪ and return the angle.  
150     target_angle = np.arctan2(target_direction_ego[1],  
        ↪ target_direction_ego[0]) * 180.0 / np.pi if  
        ↪ np.linalg.norm(target_direction_ego) > 0 else 0.0  
151     target_angle = ((target_angle + 180) % 360) - 180  
152  
153     return speed_level, target_angle
```

Listing C.10: Example driving policy generated by LangProp, trained with both imitation learning and reinforcement learning.

Appendix D

Clustering the Minecraft inventories

In the MineRL BASALT Competition 2022 [136], only the RGB observation was given as input to the agent. Hence, Invariant Information Clustering (IIC) [104] was conducted in image space in Section 6.3.2. However, at the start of this project, the specifications of the 2022 competition were not yet available. Since the MineRL BASALT Competition 2021 [195] provided the inventory information in the observation as a vector, this was used instead for IIC clustering early on in this project. An example expert demonstration for the Obtain Diamond task is shown in Figure D.2, along with the breakdown of the inventory

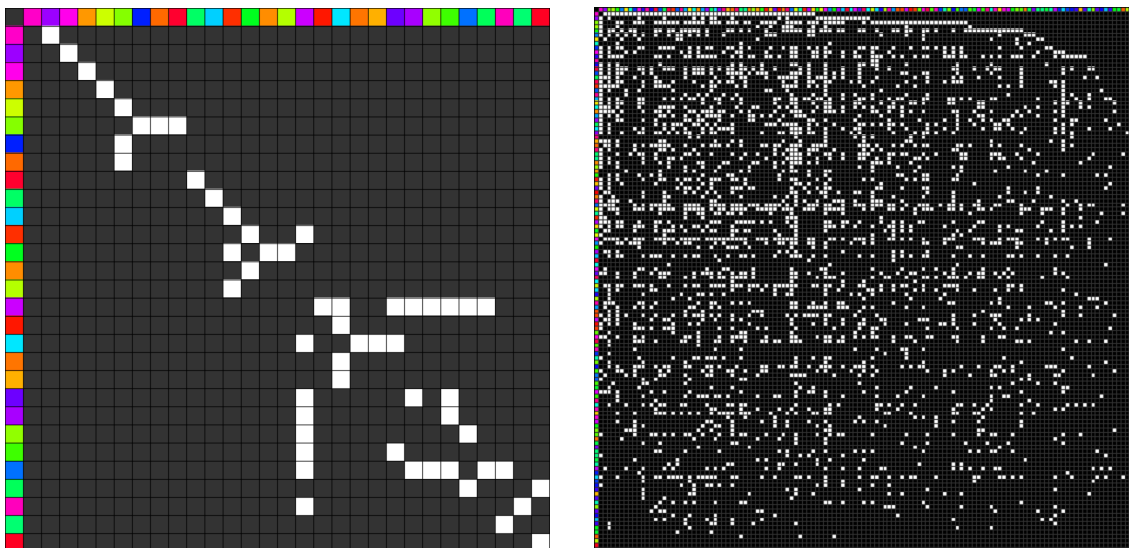


Figure D.1: Transition matrix of IIC clusters. If there exists a transition from one cluster to the other, the corresponding cell in the adjacency matrix is coloured in white. **(left)** Transition within a single expert demonstration of the Obtain Diamond task, with 30 IIC clusters. **(right)** All transitions that appear in 122 expert demonstration trajectories of the Obtain Diamond task, with 128 IIC clusters.

and the learnt IIC cluster assignments. Transitions between the learnt clusters are shown in Figure D.1.

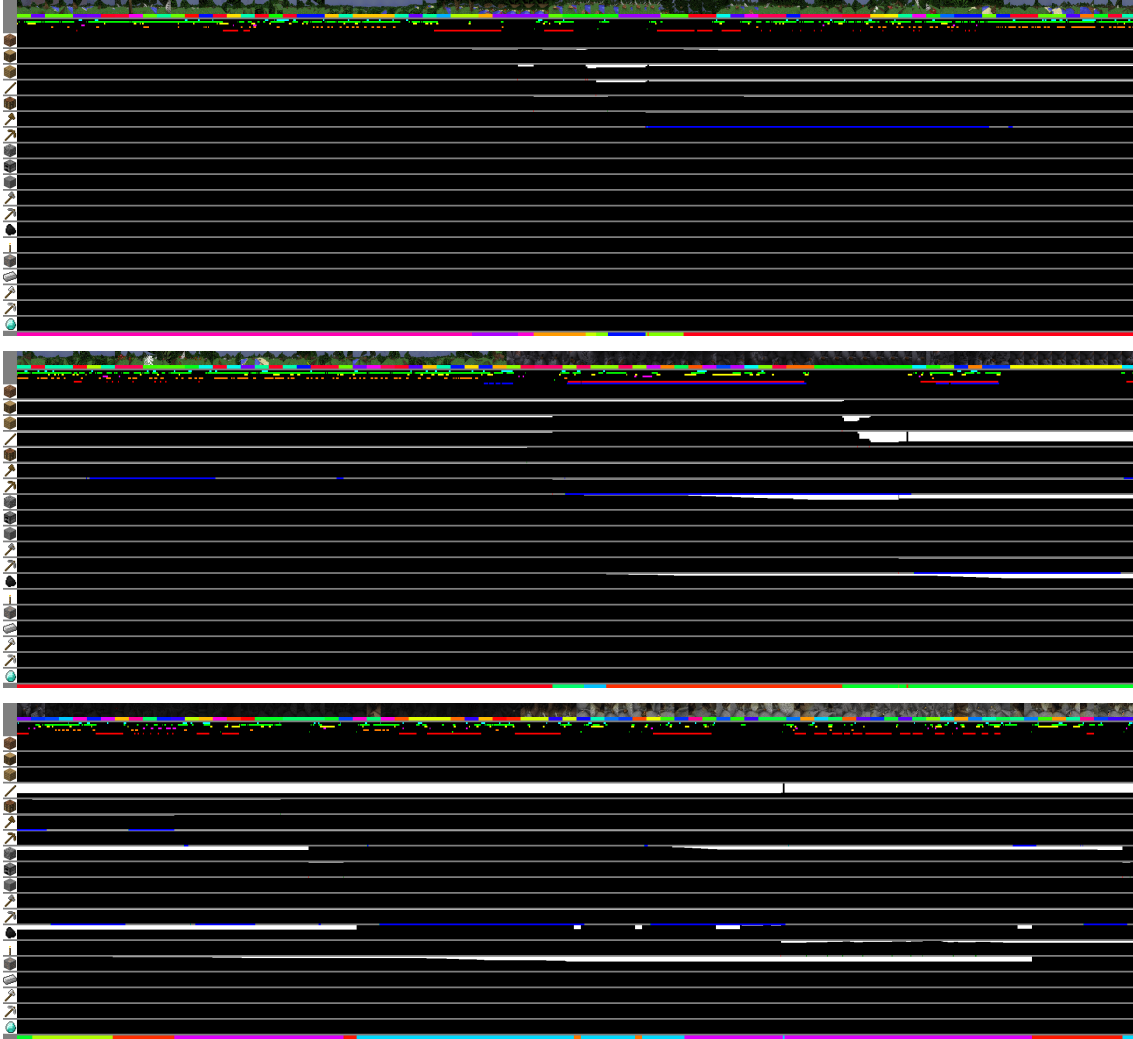
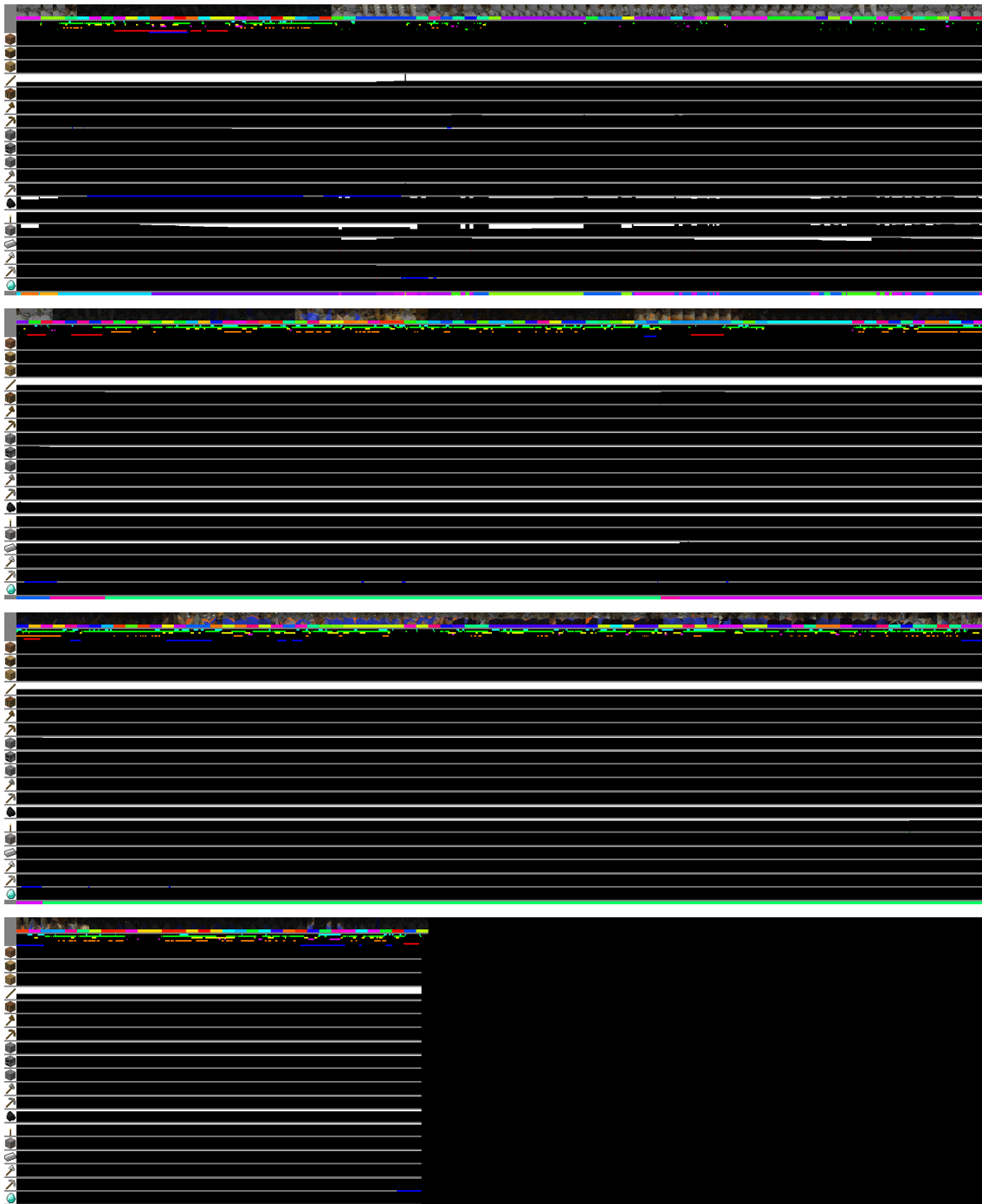


Figure D.2: Expert demonstration for the Obtain Diamond task. The RGB observations are shown at the top. Next, a coloured strip indicates an IIC cluster assignment for 128 clusters. The next row is a visual encoding of the action taken per frame (in the order of: “right”, “forward”, “left”, “back”, “jump”, “sprint”, “attack”, “sneak”). The following rows show the number of items in the inventory for every item relevant to the Obtain Diamond task. The amount of each item is shown in a progress bar style. A blue strip beneath If the item is being used with a “use” action. Finally, a coloured strip at the bottom indicates an IIC cluster assignment for 30 clusters. [Spans multiple pages]



[Continued] Expert demonstration for the Obtain Diamond task. The RGB observations are shown at the top. Next, a coloured strip indicates an IIC cluster assignment for 128 clusters. The next row is a visual encoding of the action taken per frame (in the order of: “right”, “forward”, “left”, “back”, “jump”, “sprint”, “attack”, “sneak”). The following rows show the number of items in the inventory for every item relevant to the Obtain Diamond task. The amount of each item is shown in a progress bar style. A blue strip beneath If the item is being used with a “use” action. Finally, a coloured strip at the bottom indicates an IIC cluster assignment for 30 clusters.

Bibliography

- [1] Bassem Abu-Nasser. Medical expert systems survey. *International Journal of Engineering and Information Systems (IJEAIS)*, 1(7):218–224, 2017.
- [2] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik’s cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.
- [3] Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in atari. *arXiv preprint arXiv:2405.12399*, 2024.
- [4] Phil Ammirato, Patrick Poirson, Eunbyung Park, Jana Košecká, and Alexander C. Berg. A dataset for developing and benchmarking active vision. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1378–1385, 2017. doi: 10.1109/ICRA.2017.7989164.
- [5] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Motlaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.
- [6] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.
- [7] Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective. In *Proceedings of the genetic and evolutionary computation conference companion*, pages 314–315, 2019.
- [8] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.
- [9] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The Option-Critic architecture. In *Proceedings of the AAAI Conference of Artificial Intelligence, AAAI’17*, page 1726–1734. AAAI Press, 2017.
- [10] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskiy, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari human benchmark. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 507–517. PMLR, 2020.
- [11] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M Paixao, Filipe Mutz, et al. Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816, 2021.
- [12] 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 (NeurIPS)*, 35:24639–24654, 2022.

- [13] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.
- [14] Leonard E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In Oved Shisha, editor, *Proceedings of the Symposium on Inequalities*, pages 1–8, University of California, Los Angeles, 1972. Academic Press.
- [15] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.
- [16] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.
- [17] Richard Bellman. A Markovian Decision Process. *Indiana University Mathematics Journal*, 6(4):679–684, 1957.
- [18] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębniak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.
- [19] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.
- [20] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Praseon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiaakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.
- [21] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
- [22] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [23] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

- [24] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep Blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
- [25] CARLA. CARLA autonomous driving leaderboard. <https://leaderboard.carla.org/>, 2020.
- [26] Vincent Cartillier, Zhile Ren, Neha Jain, Stefan Lee, Irfan Essa, and Dhruv Batra. Semantic MapNet: Building allocentric semantic maps and representations from egocentric views. In *Proceedings of the AAAI Conference of Artificial Intelligence*, volume 35, pages 964–972, 2021.
- [27] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niebner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. In *Proceedings of the International Conference on 3D Vision (3DV)*, pages 667–676. IEEE, 2017.
- [28] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [29] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to Explore using Active Neural SLAM. In *Proceedings of the International Conference on Learning Representations (ICLR)*, page 18, 2020.
- [30] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, Honolulu, HI, 2017. IEEE. ISBN 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.16.
- [31] Li Chen, Penghao Wu, Kashyap Chitta, Bernhard Jaeger, Andreas Geiger, and Hongyang Li. End-to-end autonomous driving: Challenges and frontiers. *arXiv preprint arXiv:2306.16927*, 2023.
- [32] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:15084–15097, 2021.
- [33] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- [34] Maxime Chevalier-Boisvert. gym-miniworld environment for OpenAI Gym. <https://github.com/maximecb/gym-miniworld>, 2018.
- [35] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. TransFuser: Imitation with transformer-based sensor fusion for autonomous driving. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
- [36] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro

- Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179.
- [37] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 617–629. PMLR, 2018.
- [38] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- [39] Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2020–2027. PMLR, 2021.
- [40] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo Tree Search. In *Proceedings of the International Conference on Computers and Games*, pages 72–83. Springer, 2006.
- [41] Michael Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*, 2020.
- [42] Christian Daniel, Herke Van Hoof, Jan Peters, and Gerhard Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104:337–357, 2016.
- [43] Pim De Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.
- [44] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, et al. RoboTHOR: An open simulation-to-real embodied AI platform. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3164–3174, 2020.
- [45] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.
- [46] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959.
- [47] Zihan Ding, Amy Zhang, Yuandong Tian, and Qinqing Zheng. Diffusion world model: Future modeling beyond step-by-step rollout for offline reinforcement learning. *arXiv preprint arXiv:2402.03570*, 2024.
- [48] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the Conference on Robot Learning (CoRL)*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 2017.

- [49] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. PaLM-E: An Embodied Multimodal Language Model. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 8469–8488. PMLR, 2023.
- [50] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL²: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.
- [51] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.
- [52] William A Falcon. PyTorch Lightning. <https://github.com/Lightning-AI/lightning>, 2019.
- [53] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. MineDojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:18343–18362, 2022.
- [54] Gregory Farquhar, Tim Rocktaeschel, Maximilian Igl, and Shimon Whiteson. TreeQN and ATreeC: Differentiable tree planning for deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- [55] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatin, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.
- [56] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value expansion for efficient model-free reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.
- [57] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1126–1135. PMLR, 2017.
- [58] E. W. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.
- [59] Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.
- [60] Daocheng Fu, Xin Li, Licheng Wen, Min Dou, Pinlong Cai, Botian Shi, and Yu Qiao. Drive like a human: Rethinking autonomous driving with large language models. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 910–919, 2024.

- [61] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2002–2011, 2018.
- [62] Jorge Fuentes-Pacheco, Jose Ascencio, and J. Rendon-Mancha. Visual Simultaneous Localization and Mapping: A Survey. *Artificial Intelligence Review*, 43, 2015. doi: 10.1007/s10462-012-9365-8.
- [63] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in Actor-Critic methods. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1587–1596. PMLR, 2018.
- [64] Vittorio Giammarino and Ioannis Ch. Paschalidis. Online Baum-Welch algorithm for hierarchical imitation learning. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 3717–3722, 2021. doi: 10.1109/CDC45484.2021.9683044.
- [65] Sandeep Goel and Manfred Huber. Subgoal discovery for hierarchical reinforcement learning using learned policies. In *Proceedings of the Florida Artificial Intelligence Research Society (FLAIRS) Conference*, pages 346–350, 2003.
- [66] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems (NeurIPS)*, 27, 2014.
- [67] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press Cambridge, 2016.
- [68] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 369–376, 2006.
- [69] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. *arXiv:1410.5401 [cs]*, 2014. arXiv: 1410.5401.
- [70] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396. IEEE, 2017.
- [71] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. DeepSeek-Coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.
- [72] Saurabh Gupta, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive Mapping and Planning for Visual Navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. arXiv: 1702.03920.
- [73] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14953–14962, 2023.

- [74] William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. MineRL: a large-scale dataset of Minecraft demonstrations. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2442–2448, 2019.
- [75] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.
- [76] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1861–1870. PMLR, 2018.
- [77] Tuomas Haarnoja, Ben Moran, Guy Lever, Sandy H Huang, Dhruva Tirumala, Jan Humplik, Markus Wulfmeier, Saran Tunyasuvunakool, Noah Y Siegel, Roland Hafner, et al. Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *Science Robotics*, 9(89):eadi8022, 2024.
- [78] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2555–2565. PMLR, 2019.
- [79] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to Control: Learning behaviors by latent imagination. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.
- [80] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.
- [81] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 1968.
- [82] R Hartley and A Zisserman. Multiple view geometry in computer. *Vision, 2nd ed., New York: Cambridge*, 2003.
- [83] Hado Hasselt. Double Q-learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 23:2613–2621, 2010.
- [84] Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable mdps. In *Proceedings of the AAAI fall symposium series*, 2015.
- [85] Frederick Hayes-Roth. Rule-based systems. *Communications of the ACM*, 28(9): 921–932, 1985.
- [86] Joao F. Henriques and Andrea Vedaldi. MapNet: An Allocentric Spatial Memory for Mapping Environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8476–8484, 2018. doi: 10.1109/CVPR.2018.00884. issn: 1063-6919.
- [87] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4565–4573, 2016.

- [88] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:6840–6851, 2020.
- [89] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–80, 1997. doi: 10.1162/neco.1997.9.8.1735.
- [90] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.
- [91] Anthony Hu, Gianluca Corrado, Nicolas Griffiths, Zachary Murez, Corina Gurau, Hudson Yeo, Alex Kendall, Roberto Cipolla, and Jamie Shotton. Model-based imitation learning for urban driving. *Advances in Neural Information Processing Systems*, 35:20703–20716, 2022.
- [92] Anthony Hu, Gianluca Corrado, Nicolas Griffiths, Zachary Murez, Corina Gurau, Hudson Yeo, Alex Kendall, Roberto Cipolla, and Jamie Shotton. Model-based imitation learning for urban driving. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:20703–20716, 2022.
- [93] Anthony Hu, Lloyd Russell, Hudson Yeo, Zak Murez, George Fedoseev, Alex Kendall, Jamie Shotton, and Gianluca Corrado. Gaia-1: A generative world model for autonomous driving. *arXiv preprint arXiv:2309.17080*, 2023.
- [94] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 9118–9147. PMLR, 2022.
- [95] 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 *Proceedings of the Conference on Robot Learning (CoRL)*, pages 1769–1782. PMLR, 2023.
- [96] Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as I can, not as I say: Grounding language in robotic affordances. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of the Conference on Robot Learning (CoRL)*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR, 2023.
- [97] Shu Ishida and João F. Henriques. Towards real-world navigation with deep differentiable planners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17327–17336, 2022.

- [98] Shu Ishida and João F Henriques. SOAP-RL: Sequential Option Advantage Propagation for Reinforcement Learning in POMDP Environments. *arXiv preprint arXiv:2407.18913*, 2024.
- [99] Shu Ishida, Marc Rigter, and Nick Hawes. Robot path planning for multiple target regions. In *Proceedings of the European Conference on Mobile Robots (ECMR)*, pages 1–6, 2019. doi: 10.1109/ECMR.2019.8870971.
- [100] Shu Ishida, Gianluca Corrado, George Fedoseev, Hudson Yeo, Lloyd Russell, Jamie Shotton, João F. Henriques, and Anthony Hu. LangProp: A code optimization framework using Large Language Models applied to driving. In *Proceedings of the Workshop on LLM Agents at the International Conference on Learning Representations (ICLR)*, 2024.
- [101] Peter Jackson. *Introduction to expert systems*. Addison-Wesley Pub. Co., Reading, MA, 1986.
- [102] Bernhard Jaeger, Kashyap Chitta, and Andreas Geiger. Hidden biases of end-to-end driving models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8240–8249, 2023.
- [103] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.
- [104] Xu Ji, Joao F Henriques, and Andrea Vedaldi. Invariant Information Clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9865–9874, 2019.
- [105] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. VIMA: General robot manipulation with multimodal prompts. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.
- [106] Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011. ISBN 978-0-374-27563-1.
- [107] Peter Karkus, David Hsu, and Wee Sun Lee. QMDP-Net: Deep Learning for Planning under Partial Observability. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4694–4704, 2017.
- [108] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:26565–26577, 2022.
- [109] Lydia Kavvaki, Petr Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 1996.
- [110] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

- [111] Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. Learnings options End-to-End for continuous action tasks. *arXiv preprint arXiv:1712.00004*, 2017.
- [112] Sven Koenig and Maxim Likhachev. D* Lite. In *Proceedings of the AAAI Conference of Artificial Intelligence*, 2002.
- [113] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. AI2-THOR: An interactive 3D environment for visual AI. *arXiv preprint arXiv:1712.05474*, 2017.
- [114] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29. Curran Associates, Inc., 2016.
- [115] Steven M. Lavalle. Rapidly-exploring Random Trees: A new tool for path planning. *Computer Science Department, Iowa State University*, 1998.
- [116] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, 2006. ISBN 978-0-511-54687-7 978-0-521-86205-9. doi: 10.1017/CBO9780511546877.
- [117] Steven M. Lavalle and Jr. James J. Kuffner. Rapidly-exploring Random Trees - progress and prospects. *Algorithmic and Computational Robotics: New Directions*, 2000.
- [118] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:21314–21328, 2022.
- [119] 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 (NeurIPS)*, 2, 1989.
- [120] Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated Path Planning Networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2947–2955. PMLR, 2018.
- [121] Daniel James Lenton, Stephen James, Ronald Clark, and Andrew Davison. End-to-End Egospheric Spatial Memory. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
- [122] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Proceedings of the IEEE intelligent Vehicles Symposium*, pages 163–168. IEEE, 2011.
- [123] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Sri-vastava, Bokui Shen, Kent Elliott Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, et al. iGibson 2.0: Object-centric simulation for robot learning of everyday

- household tasks. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 455–465. PMLR, 2022.
- [124] Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Uma-pathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro Von Werra, and Harm de Vries. StarCoder: may the source be with you! *Transactions on Machine Learning Research*, 2023. Reproducibility Certification.
- [125] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with AlphaCode. *Science*, 378(6624): 1092–1097, 2022.
- [126] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.
- [127] Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. Steve-1: A generative model for Text-to-Behavior in Minecraft. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2024.
- [128] 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, *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.
- [129] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2024.
- [130] Yiren Lu, Justin Fu, George Tucker, Xinlei Pan, Eli Bronstein, Rebecca Roelofs, Benjamin Sapp, Brandyn White, Aleksandra Faust, Shimon Whiteson, et al. Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7553–7560. IEEE, 2023.
- [131] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

- [132] Jabez Magomere, Shu Ishida, Tejumade Afonja, Aya Salama, Daniel Kochin, Foutse Yuehgoh, Imane Hamzaoui, Raesetje Sefala, Aisha Alaagib, Elizaveta Semenova, et al. You are what you eat? Feeding foundation models a regionally diverse food dataset of World Wide Dishes. *arXiv preprint arXiv:2406.09496*, 2024.
- [133] Daniel J Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, et al. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.
- [134] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.
- [135] Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4745–4753, 2017.
- [136] Stephanie Milani, Anssi Kanervisto, Karolis Ramanauskas, Sander Schulhoff, Brandon Houghton, Sharada Mohanty, Byron Galbraith, Ke Chen, Yan Song, Tianze Zhou, et al. Towards solving fuzzy tasks with human feedback: A retrospective of the MineRL BASALT 2022 competition. *arXiv preprint arXiv:2303.13512*, 2023.
- [137] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.
- [138] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, 2013. arXiv: 1312.5602.
- [139] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015. doi: 10.1038/nature14236. Number: 7540 Publisher: Nature Publishing Group.
- [140] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016. issn: 1938-7228 Section: Machine Learning.

- [141] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.
- [142] Raul Mur-Artal and Juan D. Tardos. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. doi: 10.1109/TRO.2017.2705103. arXiv: 1610.06475.
- [143] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.
- [144] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 7559–7566. IEEE, 2018.
- [145] Taewook Nam, Shao-Hua Sun, Karl Pertsch, Sung Ju Hwang, and Joseph J Lim. Skill-based meta-reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.
- [146] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21:1–50, 2020.
- [147] Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. Neural Programmer: Inducing latent programs with gradient descent. In Yoshua Bengio and Yann LeCun, editors, *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.
- [148] Andrew Y. Ng and Stuart J. Russell. Algorithms for Inverse Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, ICML ’00, pages 663–670, San Francisco, CA, USA, 2000. ISBN 978-1-55860-707-1.
- [149] Ansong Ni, Srinu Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida Wang, and Xi Victoria Lin. LEVER: Learning to verify language-to-code generation with execution. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 26106–26128. PMLR, 2023.
- [150] Buqing Nie, Yue Gao, Yidong Mei, and Feng Gao. Capability Iteration Network for robot path planning. *International Journal of Robotics and Automation*, 36(0), 2021. doi: 10.2316/j.2021.206-0598.
- [151] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. CodeGen: An open large language model for code with multi-turn program synthesis. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.
- [152] Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling. In

- Proceedings of the International Conference on Machine Learning (ICML)*, pages 26311–26325. PMLR, 2023.
- [153] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value Prediction Network. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.
- [154] OpenAI. ChatGPT. <https://openai.com/blog/chatgpt>, 2022.
- [155] OpenAI. GPT-4 technical report, 2023.
- [156] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022.
- [157] Emilio Parisotto and Ruslan Salakhutdinov. Neural Map: Structured Memory for Deep Reinforcement Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018. arXiv: 1702.08360.
- [158] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pages 7487–7498. PMLR, 2020.
- [159] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.
- [160] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.
- [161] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. MCP: Learning composable hierarchical control with multiplicative compositional policies. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.
- [162] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. ASE: Large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Transactions on Graphs*, 41(4), 2022. doi: 10.1145/3528223.3530110.
- [163] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. Accelerating reinforcement learning with learned skill priors. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2020.
- [164] Felix Petersen. *Learning with Differentiable Algorithms*. PhD thesis, University of Konstanz, 2022.
- [165] Vitchyr H Pong, Ashvin V Nair, Laura M Smith, Catherine Huang, and Sergey Levine. Offline meta-reinforcement learning with online self-supervision. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 17811–17829. PMLR, 2022.

- [166] Doina Precup and Richard S. Sutton. Temporal abstraction in reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2000.
- [167] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8494–8502, 2018.
- [168] Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.
- [169] 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 *Proceedings of the International Conference on Machine Learning (ICML)*, pages 8748–8763. PMLR, 2021.
- [170] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
- [171] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 5331–5340. PMLR, 2019.
- [172] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.
- [173] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*, 2022.
- [174] Dushyant Rao, Fereshteh Sadeghi, Leonard Hasenclever, Markus Wulfmeier, Martina Zambelli, Giulia Vezzani, Dhruva Tirumala, Yusuf Aytar, Josh Merel, Nicolas Heess, et al. Learning transferable motor skills with hierarchical latent mixture policies. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.
- [175] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.
- [176] Arthur George Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005.
- [177] Toran Bruce Richards. Auto-GPT. <https://github.com/Significant-Gravitas/Auto-GPT>, 2023.

- [178] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, 2022.
- [179] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.
- [180] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 2011. PMLR.
- [181] Baptiste Roziere, Jie Zhang, Francois Charton, Mark Harman, Gabriel Synnaeve, and Guillaume Lample. Leveraging automated unit tests for unsupervised code translation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.
- [182] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.
- [183] Reuven Y Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.
- [184] G A Rummery and M Niranjan. Online Q-learning using Connectionist Systems. *Department of Engineering, University of Cambridge*, page 21, 1994.
- [185] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 29: 70–76, 2017.
- [186] Sasha Salter, Markus Wulfmeier, Dhruva Tirumala, Nicolas Heess, Martin Riedmiller, Raia Hadsell, and Dushyant Rao. Mo2: Model-based offline options. In *Conference on Lifelong Learning Agents*, pages 902–919. PMLR, 2022.
- [187] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric Topological Memory for Navigation. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- [188] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.
- [189] Daniel Schleich, Tobias Klamt, and Sven Behnke. Value Iteration Networks on multiple levels of abstraction. *Robotics: Science and Systems*, 2019. doi: 10.15607/rss.2019.xv.014.
- [190] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

- [191] Ingmar Schubert, Jingwei Zhang, Jake Bruce, Sarah Bechtle, Emilio Parisotto, Martin Riedmiller, Jost Tobias Springenberg, Arunkumar Byravan, Leonard Hasenclever, and Nicolas Heess. A generalist dynamics model for control. *arXiv preprint arXiv:2305.10912*, 2023.
- [192] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1889–1897. PMLR, 2015.
- [193] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun, editors, *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.
- [194] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, 2017. arXiv: 1707.06347.
- [195] Rohin Shah, Steven H Wang, Cody Wild, Stephanie Milani, Anssi Kanervisto, Vinicius G Goecks, Nicholas Waytowich, David Watkins-Valls, Bharat Prakash, Edmund Mills, et al. Retrospective on the 2021 BASALT competition on learning from human feedback. *arXiv preprint arXiv:2204.07123*, 2022.
- [196] Hao Shao, Letian Wang, Ruobing Chen, Hongsheng Li, and Yu Liu. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 726–737. PMLR, 2023.
- [197] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. In *Workshop at the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 806–813, 2014.
- [198] Lucy Xiaoyang Shi, Joseph J Lim, and Youngwoon Lee. Skill-based model-based reinforcement learning. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 2262–2272. PMLR, 2023.
- [199] Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner. Taco: Learning task decomposition via temporal alignment for control. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 4654–4663. PMLR, 2018.
- [200] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 387–395. PMLR, 2014.
- [201] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks

- and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961. Number: 7587 Publisher: Nature Publishing Group.
- [202] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.
- [203] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.
- [204] Satinder Singh and Richard Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22, 1996. doi: 10.1023/A:1018012322525.
- [205] Marta Skreta, Naruki Yoshikawa, Sebastian Arellano-Rubach, Zhi Ji, Lasse Bjørn Kristensen, Kourosh Darvish, Alán Aspuru-Guzik, Florian Shkurti, and Animesh Garg. Errors are useful prompts: Instruction guided task programming with verifier-assisted iterative prompting. *arXiv preprint arXiv:2303.14100*, 2023.
- [206] Anthony Stentz. Optimal and Efficient Path Planning for Partially-Known Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, page 8, 1994.
- [207] Anthony Stentz. The Focussed D* Algorithm for Real-Time Replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, page 8, 1995.
- [208] Mojang Studios. Minecraft. <https://www.minecraft.net/>, 2011.
- [209] Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11888–11898, 2023.
- [210] Richard Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems (NeurIPS)*, 12, 2000.
- [211] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.
- [212] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.
- [213] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [214] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.
- [215] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:251–266, 2021.

- [216] Aviv Tamar, YI WU, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value Iteration Networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2154–2162, 2016.
- [217] 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.
- [218] Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023.
- [219] SIMA Team. Scaling instructable agents across many simulated worlds. *Google DeepMind Technical Report*, 2024.
- [220] Gerald Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.
- [221] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.
- [222] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE, 2012.
- [223] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [224] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of field Robotics*, 25(8):425–466, 2008.
- [225] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference of Artificial Intelligence*, volume 30, 2016.
- [226] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.
- [227] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal Networks for hierarchical reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, ICML’17, page 3540–3549. JMLR.org, 2017.
- [228] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical

- reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3540–3549. PMLR, 2017.
- [229] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z. Number: 7782 Publisher: Nature Publishing Group.
- [230] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. In *Workshop at the Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- [231] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.
- [232] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *The Annual Meeting Of The Association For Computational Linguistics*, 2023.
- [233] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, Explain, Plan and Select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023.
- [234] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. JARVIS-1: Open-world multi-task agents with memory-augmented multimodal language models. In *Agent Learning in Open-Endedness Workshop*, 2023.
- [235] Christopher Watkins. *Learning From Delayed Rewards*. PhD thesis, University of Cambridge, 1989.
- [236] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-Thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:24824–24837, 2022.
- [237] Junqing Wei, Jarrod M Snider, Junsung Kim, John M Dolan, Raj Rajkumar, and Bakhtiar Litkouhi. Towards a viable autonomous driving research platform. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 763–770. IEEE, 2013.
- [238] Ronald J. Williams. Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, Northeastern University, College of Computer Science, 1988.

- [239] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004.
- [240] David Bruce Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the ACM symposium on Theory of Computing, STOC '96*, pages 296–303, Philadelphia, Pennsylvania, USA, 1996. ISBN 978-0-89791-785-8. doi: 10.1145/237814.237880.
- [241] Penghao Wu, Xiaosong Jia, Li Chen, Junchi Yan, Hongyang Li, and Yu Qiao. Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:6119–6132, 2022.
- [242] Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Compositional transfer in hierarchical reinforcement learning. *arXiv preprint arXiv:1906.11228*, 2019.
- [243] Markus Wulfmeier, Dushyant Rao, Roland Hafner, Thomas Lampe, Abbas Abdolmaleki, Tim Hertweck, Michael Neunert, Dhruva Tirumala, Noah Siegel, Nicolas Heess, et al. Data-efficient hindsight off-policy option learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 11340–11350. PMLR, 2021.
- [244] Markus Wulfmeier, Arunkumar Byravan, Sarah Bechtle, Karol Hausman, and Nicolas Heess. Foundations for transfer in reinforcement learning: A taxonomy of knowledge modalities. *arXiv preprint arXiv:2312.01939*, 2023.
- [245] Chunqiu Steven Xia and Lingming Zhang. Less training, more repairing please: revisiting automated program repair via zero-shot learning. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 959–971, 2022.
- [246] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Practical program repair in the era of large pre-trained language models. *arXiv preprint arXiv:2210.14179*, 2022.
- [247] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson Env: Real-world perception for embodied agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9068–9079, 2018.
- [248] Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*, 2023.
- [249] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

- [250] Haoqi Yuan and Zongqing Lu. Robust task representations for offline meta-reinforcement learning via contrastive learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 25747–25759. PMLR, 2022.
- [251] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.
- [252] Andy Zeng, Maria Attarian, Krzysztof Marcin Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael S Ryoo, Vikas Sindhwani, Johnny Lee, et al. Socratic Models: Composing zero-shot multimodal reasoning with language. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.
- [253] Jesse Zhang, Haonan Yu, and Wei Xu. Hierarchical reinforcement learning by discovering intrinsic options. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. URL <https://openreview.net/forum?id=r-gPPHEjpmw>.
- [254] Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural SLAM: Learning to Explore with External Memory. *arXiv preprint arXiv:1706.09520*, 2017. arXiv: 1706.09520.
- [255] Shangdong Zhang and Shimon Whiteson. Dac: The double actor-critic architecture for learning options. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.
- [256] Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B Tenenbaum, and Chuang Gan. Planning with large language models for code generation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.
- [257] Zhejun Zhang, Alexander Liniger, Dengxin Dai, Fisher Yu, and Luc Van Gool. End-to-End urban driving by imitating a reinforcement learning coach. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15222–15232, 2021.
- [258] Zhiyu Zhang and Ioannis Ch. Paschalidis. Provable hierarchical imitation learning via em. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.
- [259] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. Ghost in the Minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023.
- [260] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Proceedings of the AAAI Conference of Artificial Intelligence*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.
- [261] Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. Modeling Interaction via the Principle of Maximum Causal Entropy. In *Proceedings of the International Conference on Machine Learning (ICML)*, page 8, 2010.

-
- [262] K.J. Åström. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965. doi: [https://doi.org/10.1016/0022-247X\(65\)90154-X](https://doi.org/10.1016/0022-247X(65)90154-X).
- [263] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłoś, Błażej Osipiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for Atari. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.