UNIVERSIDADE
BEIRA INTERIOR

Improving the Robustness of Demonstration Learning

André Rosa de Sousa Porfírio Correia

Tese para obtenção do Grau de Doutor em
Engenharia Informática
(3º ciclo de estudos)

Orientador: Prof. Doutor Luís Filipe Barbosa de Almeida Alexandre

Covilhã, maio de 2025

Composição do Júri

Hugo P. Proença
Professor Catedrático — Presidente
Universidade da Beira Interior

Armando J. Pinho
Professor Catedrático — Arguente
Universidade de Aveiro

Gabriel F. Fernandes
Professor Associado — Arguente
Faculdade de Ciências e Tecnologia da Universidade de Coimbra

José A. Victor
Professor Catedrático — Arguente
Instituto Superior Técnico da Universidade de Lisboa

João C. Neves
Professor Auxiliar — Arguente
Universidade da Beira Interior

Luís A. Alexandre
Professor Catedrático — Orientador
Universidade da Beira Interior

Provas realizadas a 11 abril 2025 com início às 15:30 horas.

Declaração de Integridade

Eu, André Rosa de Sousa Porfírio Correia, que abaixo assino, estudante com o número de inscrição D2884 do 3º Ciclo de Engenharia Informática da Faculdade de Engenharias, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o **Código de Integridades da Universidade da Beira Interior**.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

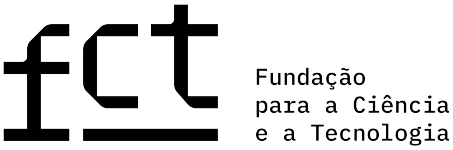
Universidade da Beira Interior, Covilhã 07/05/2025

Assinado por: **ANDRÉ ROSA DE SOUSA PORFÍRIO CORREIA**
Num. de Identificação: 15080894
Data: 2025.05.07 22:05:09+01'00'

CARTÃO DE CIDADÃO

Thesis prepared at NOVA LINCS and Soft Computing and Image Analysis Laboratory (SOCIA Lab), and submitted to Universidade da Beira Interior for defense in a public examination session.

This work is supported by UID/04516/NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) with the financial support of FCT.IP, and also through the research grant 2022.14197.BD.



Agradecimentos

The work reflected in these results stands as a testament to the immeasurable support generously provided by numerous individuals. Throughout this journey, those who extended their support and motivation have emerged as my paramount sources of strength and inspiration. I dedicate this text to express my heartfelt appreciation for their indispensable contributions.

I would like to express my deepest gratitude to my advisor, Professor Luís Alexandre, whose knowledge, guidance, and unwavering support have been instrumental throughout this journey. His unwavering availability and motivational encouragement empowered me to persevere and ultimately achieve the goals set before me. Working alongside Professor Luís over these years has been an honor, providing me with the invaluable opportunity to learn from his extensive knowledge and experience.

I am also grateful to my colleagues at SOCIALab and NOVALincs, especially Vasco, Nuno, Bruno and Saeid, for their support, understanding, and the shared moments of celebration and encouragement. Your friendship has been a vital source of strength throughout these years. It has been a pleasure to work alongside you and learn so many different things. A special token of appreciation to Nuno and Vasco, for their continuous availability and discussions.

To my family, whose constant encouragement and understanding sustained me through the challenges of pursuing a PhD, I am profoundly grateful. In particular, I want to extend a heartfelt thank you to my mother, father and grandparents, for their boundless love, sacrifice, and belief in my capabilities. Their unwavering support has been my anchor, and this accomplishment is as much theirs as it is mine.

A special thanks goes to my circle of friends who provided laughter, encouragement, and a sense of camaraderie during this demanding journey. In particular, I would like to acknowledge Rui, Rafael, and Gaspar for their enduring friendship and the countless times they offered a listening ear or a motivating word when needed the most.

In conclusion, I extend my heartfelt appreciation to everyone who has played a role, big or small, in shaping this academic endeavor. Your contributions have not gone unnoticed, and I am truly thankful.

List of Publications

The development of this work resulted in the creation of five research articles that were published in different international conferences and journals.

Publications included in the thesis resulting from this doctoral research program:

1. Multi-View Contrastive Learning from Demonstrations. **André Correia**, Luís A. Alexandre. In 6th IEEE International Conference on Robotic Computing (IRC), 2022.

2. Hierarchical Decision Transformer. **André Correia**, Luís A. Alexandre. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2023.

3. DEFENDER: DTW-Based Episode Filtering Using Demonstrations for Enhancing RL Safety. **André Correia**, Luís A. Alexandre. In 31st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), 2023.

4. A Survey of Demonstration Learning. **André Correia**, Luís A. Alexandre. In Robotics and Autonomous Systems, Elsevier, 2024.

5. Music to Dance as Language Translation using Sequence Models. **André Correia**, Luís A. Alexandre. In review at a journal. https://arxiv.org/abs/2403.15569

6. Decision Mamba Architectures. **André Correia**, Luís A. Alexandre. In review at a journal. https://arxiv.org/abs/2405.07943

Other publications produced during the development of this thesis:

1. Optimal Algorithm Allocation for Robotic Network Cloud Systems. Saeid Alirezazadeh, **André Correia**, Luís A. Alexandre. In Robotics and Autonomous Systems, Elsevier, 2022.

2. Dynamic Task Scheduling with Optimal Energy Consumption for Robotic Network Cloud System. Saeid Alirezazadeh, **André Correia**, Luís A. Alexandre. In review at a journal.

3. Towards Safe Exploration using Demonstrations. **André Correia**, Luís A. Alexandre. In 28th Portuguese Conference on Pattern Recognition, RECPAD 2022, Leiria, Portugal, October 28th, 2022.

4. Exploring Actor-Critic Algorithms for Controlling a Biped Robot in Simulation. Pedro Batista, **André Correia**, Luís A. Alexandre. In 30th Portuguese Conference on Pattern Recognition, RECPAD 2024, Covilhã, Portugal, October 28th, 2024.

Resumo

Os avanços na Aprendizagem por Reforço (AR) permitiram a automatização de tarefas complexas, impulsionados em grande parte pelos algoritmos de aprendizagem profunda que utilizam redes neuronais para estimar a função de política. O sucesso crescente da AR deve-se principalmente ao engenho e aos esforços de engenharia de especialistas que conceberam e otimizaram algoritmos de aprendizagem por reforço que maximizam os dados de exploração para estimar funções de política poderosas, obtendo resultados sem precedentes numa vasta panóplia de tarefas. Apesar dos notáveis sucessos alcançados, os métodos de AR enfrentam frequentemente ineficiências no uso de dados de treino, e requerem extensivas interações de tentativa e erro com o ambiente. Cada tentativa falhada coloca em causa a segurança do agente bem como a de todos os elementos ao seu redor. Este desafio levou ao surgimento da Aprendizagem por Demonstração (AD), que propõe colecionar os dados de interação antes do uso do robô sobre a forma de um conjunto de dados de demonstração. A política pode ser estimada usando este conjunto de dados, não requererendo que o agente interaja com o ambiente, mitigando preocupações de segurança.

No entanto, os métodos de AD enfrentam desafios próprios, como a dependência da qualidade do conjunto de dados de demonstração que restringem a qualidade da política devido à incompletude ou inadequação do conjunto nos dados. Contudo, a criação de um conjunto de dados de demonstração com qualidade para aprender cada tarefa é um processo difícil. Caso o conjunto de dados não esteja completo existirá um desvio da distribuição representada pelo conjunto de dados para a distribuição real da tarefa. Se a política estimada não for capaz de generalizar através do treino do conjunto de dados, quando o agente encontrar um dado fora da distribuição, irá provávelmente falhar a interação.

Nesta tese, visamos abordar essas lacunas e avançar no campo da AD. Em primeiro lugar, descrevemos os conceitos fundacionais, metodologias e algoritmos utilizados nos métodos propostos nesta tese. Depois conduzimos um estudo abrangente do estado da arte em métodos de AD. Neste estudo identificámos as várias escolhas que envolvem a criação do conjunto de dados de demonstração, seguido da categorização dos diferentes métodos existentes. Neste estudo foram também listadas as vantagens e desvantagens de métodos de AD, bem como as suas áreas de aplicação. Finalmente o estudo concluiu com a identificação dos problemas em aberto no ramo de AD. De seguida, proposemos o método CLfD, que utiliza Aprendizagem Contrastiva para criar representações invariantes em relação ao ponto de vista da câmara usando vídeos de demonstração sincronizados e capturados por camaras em diferentes posições. Neste método mostrámos que estas representações podem ser utilizadas como função de recompensa usando Aprendizagem por Reforço Inverso, para estimar uma política robusta às variações da posição da câmara e concluímos que pode ser usado para generalizar outras características. Adi-

cionalmente, propomosemos o método DEFENDER, que junta as qualidades de AD e AR. Especificamente, pode ser aplicado a qualquer algoritmo de AR e melhora a segurança dos agentes durante o treino. O método usa um conjunto de dados com demonstrações bem sucedidas e falhadas. Antes de cada interação, o método compara a trajetória do agente com as demonstrações, caso a trajetória se enquadre melhor com demonstrações falhadas, a trajetória é terminada evitando uma potencial interação catastrófica. Depois propomosemos o método HDT, que melhora o estado-da-arte de modelos de sequência em AD. Mostrámos que o estado-da-arte requer especificação precisa de um hyper-parâmetro cujo valor não é trivial e depende da tarefa, para que a política tenha um bom desempenho. O método proposto propõe uma arquitetura hierárquica que remove a necessidade desta especificação e melhora o desempenho sobre o estado-da-arte em diversas tarefas. Posteriormente utilizamos a arquitetura Mamba para melhorar o método HDT, bem como o estado-da-arte de modelos de sequência em AD. Esta substituição não só melhora o desempenho dos modelos, como reduz o tempo de inferência. Por fim, propomosemos o método MDLT que aplica modelos de sequência para aprender uma política de dança usando AD. O MDLT propõe modelar a aprendizagem da política como a tradução de linguagem musical para linguagem de dança usando um modelo sequêncial. Neste método, a política estimada demonstra capacidade de generalização para músicas do mesmo género musical em que foi treinada, bem como a capacidade para aprender diversos géneros musicais em simultâneo. Propomosemos duas variantes deste método: utilizando Transformers ou Mamba.

Os resultados obtidos pelos métodos propostos mostram que é possivel melhorar os métodos de AD nos respetivos problemas em que cada um foi aplicado. Todos os métodos propostos foram avaliados em diversos conjuntos de dados e quando comparados com os métodos estado-da-arte dos respetivos problemas, todos mostraram melhorar o desempenho das políticas estimadas, a segurança durante o treino no caso do DEFENDER e a robustez da política no caso do método CLfD.

Palavras-chave

Aprendizagem por Demonstração, Aprendizagem por Imitação, Aprendizagem por Reforço, Robótica, Função de Política, Clonagem de Comportamento, Aprendizagem por Reforço Inverso

Resumo Alargado

Os avanços na Aprendizagem por Reforço (AR) permitiram a automatização de tarefas complexas, impulsionados em grande parte pelos algoritmos de aprendizagem profunda que utilizam redes neuronais para estimar a função de política. O sucesso crescente de AR deve-se principalmente ao engenho e aos esforços de engenharia de especialistas que conceberam e otimizaram algoritmos de aprendizagem por reforço que maximizam os dados de exploração para estimar funções de política poderosas, obtendo resultados sem precedentes numa vasta panóplia de tarefas. Apesar dos notáveis sucessos alcançados, os métodos de AR frequentemente enfrentam ineficiências no uso de dados de treino, e requerem extensivas interações de tentativa e erro com o ambiente. Cada tentativa falhada coloca em causa a segurança do agente bem como a de todos os elementos ao seu redor. Este desafio levou ao surgimento da Aprendizagem por Demonstração (AD), que propõe colecionar os dados de interação antes do uso do robô sobre a forma de um conjunto de dados de demonstração. A política pode ser estimada usando este conjunto de dados, não requererendo que o agente interaja com o ambiente, mitigando preocupações de segurança.

No entanto, os métodos de AD enfrentam desafios próprios, como a dependência da qualidade do conjunto de dados de demonstração que restringem a qualidade da política devido à incompletude ou inadequação do conjunto nos dados. Contudo, a criação de um conjunto de dados de demonstração com qualidade para aprender cada tarefa é um processo difícil. Caso o conjunto de dados não esteja completo existirá um desvio da distribuição representada pelo conjunto de dados para a distribuição real da tarefa. Se a política estimada não for capaz de generalizar através do treino do conjunto de dados, quando o agente encontrar um dado fora da distribuição, irá provávelmente falhar a interação.

O primeiro capítulo descreve conceitos fundacionais, metodologias e algoritmos utilizados nos métodos propostos nesta tese. Começa pela explicação das arquiteturas de modelos sequenciais: os Transformers e os SSSMs. Depois descreve o algoritmo DTW, um algoritmo para análise e comparação de sequências. Posteriormente, são explicados os algoritmos estado-da-arte de AR: DDPG, TD3 e SAC. Finalmente, o algoritmo estado-da-arte de processamento sequencial com AD é descrito, o DT.

No segundo capítulo efetuamos uma revisão extensiva do estado-da-arte de métodos de AD. A aprendizagem por demonstração reduz a sobrecarga de programação ao ensinar ao agente uma tarefa através de demonstrações. O paradigma pode ser dividido em duas fases. A primeira fase consiste na recolha e construção do conjunto de dados de demonstração. A segunda fase consiste em extrair o comportamento representado no conjunto de dados e treinar um agente. O principal desafio na AD é a generalização para cenários não representados pelo conjunto de dados de demonstração. A imitação direta através do

clonagem de comportamento aprende esta distribuição. Se o agente visitar casos fora da distribuição, não saberá o que fazer. Foram estudados diferentes formas de mitigar este desafio. Em seguida, foram explorados métodos para refinar o comportamento. Estes vão desde interações de tentativa e erro com o ambiente usando aprendizagem por reforço, transferindo a habilidade de uma tarefa para outra usando aprendizagem de transferência, questionando ativamente o professor, ou usando algoritmos evolutivos para otimizar o comportamento. Foram listadas as principais aplicações de AD, bem como as vantagens e desvantagens do paradigma. AD é uma área recente com um futuro promissor, e as principais áreas em aberto para pesquisa foram identificadas.

O terceiro capítulo apresenta dois métodos para combinar AR com AD de forma a combater dois problemas de políticas em AR. O primeiro método proposto é o CLfD. Este algoritmo utiliza aprendizagem contrastiva para obter representações de estado invariantes a pontos de vista a partir de demonstrações em vídeo não rotuladas capturadas a partir de múltiplos pontos de vista. Utiliza-se a aprendizagem contrastiva para realçar a informação relevante para a tarefa, suprimindo a informação irrelevante nas características. As representações são obtidas contrastando frames semanticamente alinhadas de diferentes pontos de vista. O alinhamento semântico é garantido pela sincronização das demonstrações em vídeo. Demonstra-se que estas representações são aplicáveis para imitar tarefas robóticas. O método proposto é validado nos conjuntos de dados publicamente disponíveis Multi-View Pouring e Pick and Place, sendo comparado com os baselines TCN e CMC. Estudámos cuidadosamente diferentes backbones para os codificadores e mostrámos os efeitos de diferentes escolhas. Demonstrámos que o nosso algoritmo pode alinhar corretamente frames entre dois pontos de vista e com menos iterações de treino do que as baselines, sendo também mais leve computacionalmente. Também criámos um conjunto de dados de aprendizagem por demonstração que pode ser usado para explorar outras abordagens e compará-las com a nossa proposta. Por último, mostrámos que as representações contêm informação que pode ser utilizada para fins de classificação e fornecemos uma função de recompensa dentro de um algoritmo de aprendizagem por reforço para aprender uma tarefa de manipulação. O método proposto neste capítulo propõe uma solução para um dos problemas em aberto em AD: o problema do contexto, especificamente, o problema das políticas apenas funcionarem se forem executadas no mesmo contexto em que foram treinadas. No mesmo capítulo, propomos o método DEFENDER, que junta as qualidades de AD e AR. Especificamente, pode ser aplicado a qualquer algoritmo de AR e melhora a segurança dos agentes durante o treino. O método usa um conjunto de dados com demonstrações bem sucedidas e falhadas. Antes de cada interação, o método compara a trajetória do agente com as demonstrações, caso a trajetória se enquadre melhor com demonstrações falhadas, a trajetória é terminada evitando uma potencial interação catastrófica. A comparação de trajetórias é realizada usando Dynamic Time Warping (DTW). Foram realizados estudos para determinar as estratégias de comparação de trajetórias apropriadas de um grupo de 576 estratégias. O DEFENDER foi aplicado aos algoritmos SAC e TD3. De seguida comparámos o desempenho das políticas e o

rácio de colisão das políticas com e sem o DEFENDER. Os resultados obtidos demonstram uma melhoria significativa de segurança dos agentes durante o treino acompanhadas de ligeiras melhorias no desempenho. Embora exista espaço para melhorar, devido ao facto das estratégias de comparação poderem ser sobre-protetivas, este trabalho demonstra o potencial de usar demonstrações para aumentar a segurança de algoritmos de AR.

O quarto capítulo apresenta dois métodos sequenciais para melhorar o algoritmo estado-da-arte: o primeiro usando a arquitetura Transformer e o segundo usando a arquitetura Mamba baseada nos SSSMs. O primerio método é o HDT, um algoritmo de clonagem de comportamento hierárquico que melhora o desempenho dos modelos sequênciais em AD. Mostrámos empíricamente que o método sequencial estado-da-arte depende da especificação de um hiper-parâmetro para definir a sequência de retornos que condiciona o modelo. A definição do valor deste hyper-parametro determina o desempenho do modelo, depende da tarefa e não é trivial de definir. Propomos uma arquitetura hierárquica com dois modelos sequênciais. O mecanismo de alto nível guia o controlador de baixo nível através da tarefa selecionando sub-objetivos para este último alcançar. Ao substituir a sequência de retornos pelo método de seleção de sub-objetivos, alcançamos um modelo totalmente independente da tarefa. Validámos o nosso método em várias tarefas dos benchmarks OpenAI Gym, D4RL e RoboMimic. O nosso método supera os baselines em vinte e três de trinta e uma configurações de horizontes e frequências de recompensa variados, sem conhecimento prévio da tarefa, mostrando as vantagens da abordagem do modelo hierárquico para aprendizagem a partir de demonstrações usando um modelo de sequência. Avaliámos também o método numa tarefa de alcance no robô físico UR3, onde o HDT superou os baselines em número de posições alcançadas. O método proposto neste capítulo melhorou o estado-da-arte de modelos sequênciais em AD, não só em termos de desempenho mas removendo a necessidade de configurações especificas para cada tarefa. No mesmo capítulo utilizámos a arquitetura Mamba para melhorar o método HDT, bem como o antigo estado-da-arte de modelos sequênciais em AD. Os métodos propostos com a arquitetura Mamba foram comparados com os métodos anteriores baseados em Transformers em várias tarefas e treinados com diversos conjuntos de dados. Os resultados das experiências mostram que esta subsituição de arquitetura não só melhora o desempenho de ambos os modelos, como também reduz o tempo de inferência. Adicionalmente, os modelos com arquitetura Mamba reduzem a complexidade da arquitetura pois não requerem tantos elementos como os Transformers.

O quinto capítulo propõe uma nova abordagem de AD aplicada à aprendizagem de dança. Propomos o método MDLT, que modela a aprendizagem de política de dança como a tradução de linguagem músical para linguagem de dança. Para atingir este objetivo propomos utilizar dois modelos: um modelo Transformer, e um modelo Mamba. Devido aos resultados destas arquiteturas em problemas de tradução de linguagem. O método foi aplicado ao braço robótico UR3 presente no laboratório. Para isso foram extraídas as características musicais e as correspondentes poses de braço de um conjunto de dados

de coreografias. Os modelos foram treinado usando dois conjuntos de coreografias. Adicionalmente, os modelos foram avaliados em coreografias de diversos géneros músicais e também no conjunto de coreografias completo. O método foi avaliado usando como métricas o erro das poses geradas com as esperadas bem como a Fréchet Inception Distance (FID). Os resultados mostram um FID de 0.51% e um erro de pose médio de 25 graus quando treinado em todos os géneros músicais. Adicionalmente, o erro diminui para uma média de 17.8 graus quando treinado num único género músical. Embora o modelo não consiga extrapolar de forma perfeita para coreografias novas, demonstra claras indicações de aprender a linguagem de dança e a respetiva tradução a partir da música. Mais, é ainda notável quando se considera que, devido às caracteristicas artísticas de dança, para uma única música existem diversas coreografias aceitáveis. O método proposto neste capítulo serve o objetivo de aplicar AD a problemas incomuns.

Por último, os principais resultados deste trabalho de investigação são resumidos no capítulo sete. Além disso, são também apontados, com base no trabalho desenvolvido e nas contribuições desta tese, aqueles que acreditamos ser os passos futuros mais interessantes para a área de AD.

Abstract

With the fast improvement of machine learning, Reinforcement Learning (RL) has been used to automate human tasks in different areas. However, training such agents is difficult and restricted to expert users. Moreover, it is mostly limited to simulation environments due to the high cost and safety concerns of interactions in the real world. Demonstration Learning is a paradigm in which an agent learns to perform a task by imitating the behavior of an expert shown in demonstrations. It is a relatively recent area in machine learning, but it is gaining significant traction due to having tremendous potential for learning complex behaviors from demonstrations. Learning from demonstration accelerates the learning process by improving sample efficiency, while also reducing the effort of the programmer. Due to learning without interacting with the environment, demonstration learning can allow the automation of a wide range of real world applications such as robotics and healthcare.

Demonstration learning methods still struggle with a plethora of problems. The estimated policy is reliant on the coverage of the data set which can be difficult to collect. Direct imitation through behavior cloning learns the distribution of the data set. However, this is often not enough and the methods may struggle to generalize to unseen scenarios. If the agent visits out-of-distribution cases, not only will it not know what to do, but the consequences in the real world can be catastrophic. Because of this, offline RL methods try to specifically reduce the distributional shift.

In this thesis, we focused on proposing novel methods to tackle some of the open problems in demonstration learning. We start by introducing the fundamental concepts, methodologies, and algorithms that underpin the proposed methods in this thesis. Then, we provide a comprehensive study of the state-of-the-art of Demonstration Learning methods. This study allowed us to understand existing methods and expose the open problems which motivate this thesis. We then developed five methods that push improve upon the state-of-the-art and solve different problems. The first method proposes to tackle the context problem, where policies are restricted to the context in which they were trained. We propose a method to learn context-invariant image representations with contrastive learning, by making use of a multi-view demonstration data set. We show that these representations can be used in lieu of the original images to learn a policy with standard reinforcement learning algorithms. This work also contributed with benchmark environment and a demonstration data set. Next, we tackled the potential of combining reinforcement learning with demonstration learning to cover the weaknesses of both paradigms. Specifically, we developed a method to improve the safety of reinforcement learning agents during their learning process. The proposed method makes use of a demonstration data set with safe and unsafe trajectories. Before each interaction, the method evaluates the trajectory and stops it if deems it unsafe. The method was used to augment state-of-the-art reinforcement learning methods, and it reduced the crash rate significantly which also

resulted in a slight increase in performance. In the following work, we acknowledged the significant strides made in sequence modelling and their impact in a plethora of machine learning problems. We noticed that these methods had recently been applied to demonstration learning. However, the state-of-the-art method was reliant on task knowledge and user interaction to perform. We proposed a hierarchical method which identifies important states in each demonstration, and uses them to guide the sequence model. The result is a method that is task and user independent but also achieves better performance than the previous state-of-the-art. Next, we made use of the novel Mamba architecture to improve upon the previous sequence modelling method. By replacing the Transformer architecture with the Mamba, we proposed two methods that reduce the complexity, and inference time while also improving the performance. Finally, we apply demonstration learning to under-explored applications. Specifically, we apply demonstration learning to teach an agent to dance to music. We describe the insight of modelling the task of learning to dance as a translation task, where the agent learns to translate from the language of music to the language of dance. We used the previous experience resulted from the two sequence modelling methods to propose two variants: using the Transformer or the Mamba architectures. The method modifies the standard sequence modelling architecture to process sequences of audio features and translate them to dance poses. Results show that the method can translate diverse and unseen music to high-quality dance motions coherent within the genre.

Results obtained by the proposed methods advance the state-of-the-art in Demonstration Learning and provide solutions to open problems in the field. All the proposed methods were evaluated against state-of-the-art baselines and evaluated on several tasks and diverse data sets, improving the performance and tackling their respective problems.

Keywords

Demonstration Learning, Imitation Learning, Learning from Demonstrations, Offline Reinforcement Learning, Machine Learning, Deep Learning, Behavior Cloning

Contents

List of Figures

List of Tables

Acronyms

ACO	Ant Colony Optimization
AI	Artificial Intelligence
AJE	Average Joint Error
ALE	Arcade Learning Environment
BC	Behavior Cloning
CL	Contrastive Learning
CLfD	Contrastive Learning from Demonstrations
CMC	Contrastive Multiview Coding
CNN	Convolutional Neural Network
CQL	Conservative Q-Learning
DM	Decision Mamba
DT	Decision Transformer
DTW	Dynamic Time Warping
DDPG	Deep Deterministic Policy Gradient
DL	Demonstration Learning
DOPE	Deep Off-Policy Evaluation
EA	Evolutionary Algorithms
EMA	Exponential Moving Average
FID	Fréchet Inception Distance
GAIL	Generative Adversarial Imitation Learning
GAN	Generative Adversarial Networks
GMM	Gaussian mixture model
GPU	Graphics Processing Unit
HDM	Hierarchical Decision Mamba
HDT	Hierarchical Decision Transformer

HMM	Hidden Markov Model
ICU	Intensive Care Unit
IRL	Inverse Reinforcement Learning
KL	Kullback-Leibler
LSTM	Long-Short Term Memory
LWPR	Locally Weighted Projection Regression
LWR	Locally Weighted Regression
MDLT	Music to Dance as Language Translation
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-Layered Perceptron
NLP	Natural Language Processing
NN	Neural Network
ODE	Ordinary Differential Equation
OPE	Off-Policy Evaluation
PPL	Preference-Based Policy Learning
POMDP	Partially-Observable Markov Decision Process
PSO	Particle Swarm Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RWRL	Real-World Reinforcement Learning
SAC	Soft Actor Critic
SSSM	Structured State Space Model
SSM	State Space Model
TCN	Time Contrastive Networks
TL	Transfer Learning
TT	Trajectory Transformer
TD3	Twin-Delayed Deep Deterministic Policy Gradient
UR3	Universal Robots 3
ZOH	Zero-Order Hold

Chapter 1

Introduction

The evolution of Machine Learning (ML) algorithms continuously increases the number of tasks being automated. One of many ML paradigms, Reinforcement Learning (RL), provides the estimation of a function which allows the agent to perform the desired task from learning through trial and error interactions with the environment. This paradigm allows for the estimation of policy functions that can perform complex behaviors that would otherwise be difficult for a programmer to define. However, the trial and error nature of the paradigm makes it unsuitable for real world applications where the errors can have serious consequences. Furthermore, RL is extremely data inefficient requiring large numbers of interactions which only scales with the complexity of the task and exacerbates the aforementioned safety problem.

In recent years, a new machine learning paradigm was created to tackle the disadvantages of RL. Demonstration Learning (DL) allows the agent to learn from interaction data without the agent having to interact with the system. Here, an expert demonstrator first performs the task. The interactions of the demonstrator with the environment are recorded in the form of a data set for the agent to learn from. Since the agent does not interact with the environment during policy learning, the paradigm is safer than RL.

DL can be structured as a three-step pipeline, starting with the creation of the demonstration data set, followed by function estimation using the data set, and the estimated function can be further optimized to make up for the suboptimal data set. There are many design choices in the data set creation process, from choosing the demonstrator, to the demonstration technique, to the data representation. Although the most common application of DL is to learn the policy function, DL can also be employed to learn the reward function with Inverse Reinforcement Learning (IRL) or the transition function with model learning. The main disadvantage of DL is its reliance on the quality and coverage of the demonstration data set. Because the state and action spaces of real-world applications are often large, the creation of a complete data set is unfeasible. Hence, DL methods either accept the limited performance or try to generalize beyond the data set. In the first case, methods try to avoid out-of-distribution states, keeping the agent safe. These methods specifically try to reduce the distributional shift between the distribution of the data set and the estimated function. On the contrary, methods that try to generalize beyond the data set often employ RL using the starting DL function as a safer start than a random function.

DL expands from the RL paradigm commonly defined as a Markov Decision Process (MDP), formulated by the tuple $< S, A, P, \Delta_0, R, \lambda >$ [8]. S is the set of the possible environment states, $s \in S$, and Δ_0 denotes the initial state distribution. At each state, the agent can choose an action $a \in A$ from the set of possible actions. Acting transitions the agent to a new state of the environment, this transition is computed by the state transition function $P(s_{t+1} \mid s_t, a_t) : S \times A \to S$. The policy $\pi : S \to A$ is a function that selects an action given a state of the environment. The correct selection of the action for any given state is what allows the agent to perform the task. After interacting with the environment, the agent receives a reward $r_t = R(s_t, a_t, s_{t+1})$, which indicates the quality of the interaction. In RL, the policy is optimized to maximize the expected future rewards $\mathbb{E}[\sum_{t=0}^{\infty} \lambda R(s_t, a_t, s_{t+1})]$, where λ is the discount factor.

In DL, the agent has access to a data set of N demonstrations $D_{demo} = \{\tau_i, i \in [0, N[\}$. Each demonstration is the sequence of transitions performed by the demonstrator $\tau_i = (s_t, a_t, s_{t+1}, t \in [0, L[)$, where L is the length of the sequence. The policy is estimated from the behaviors shown in the demonstration data set. The simplest form of DL is through Behavior Cloning (BC). In BC the agent is encouraged to directly imitate the demonstrator, by selecting the actions the demonstrator took for every single state in the data set. A common approach is to maximize the likelihood of actions in the demonstration, $\max \mathbb{E}_{(s,a)\sim D}\, log\pi(a \mid s)$.

Even though RL, and more recently DL research is very active and state-of-the-art methods perform well in specific settings, there are still significant limitations and unsolved issues that prevent their use in general settings:

- DL methods are reliant on the quality and coverage of the demonstration data set. There is still a limited number of benchmark data sets and environments to train and evaluate DL methods.

- Because of the difficulty of creating a data set that covers the entire state and action spaces, DL methods try to generalize past the demonstrated behaviors to unseen scenarios.

- In cases where generalization is not possible, methods try to prevent the agent from visiting out-of-distribution states to avoid catastrophic consequences, and try to specifically reduce the distributional shift.

- Context is the task specific and agnostic information in which a policy relies on to work. RL usually trains policies under one context as it is difficult to create multiple task environments. Additionally, in DL the demonstration data set also tends to cover a single context. Hence, if the context information ever changes, the policy will likely under-perform or stop working altogether. Training multi-context policies remains an open and under-studied problem.

- Both DL and RL have advantages and disadvantages. Combining the best of both worlds to obtain quality policies can lead to better results than the limitation to one paradigm, however this is still an open problem.
- DL methods, like most ML paradigms, relies on the careful specification of hyper-parameter values. However, their ideal values are hard to estimate and are usually updated by trial and error.
- ML research has focused on sequence models which revolutionized many applications particularly in NLP. Due to the sequential nature of an MDP they have recently been applied to RL and DL. Given the impact of these models on other ML areas, the potential for RL and DL applications remains an open question.
- DL has tremendous potential for estimating quality policies from pre-existing data sets. However, the range of applications to which DL methods have been applied is limited.
- All the previous issues compound to the larger issue which is the difficulty of training and/or deploying DL policies to real-world scenarios and restricting both DL and RL to simulation environments.

These are the open problems which are tackled by the proposed methods in this thesis. However, there are more open problems in DL than the listed ones, these are identified in the state-of-the-art review in Chapter 3. These include multi-agent DL, which aims to train co-operative policies from demonstration data sets, and multi-goal DL where the agent must adapt its policy to changes to the task's objective. All the research in DL tries to culminate into a general learning framework, which can be applied to learn a policy for any task in general settings. This is because the overall idea behind DL is to reduce complexity and programming knowledge and improve the practicality of policy training, eventually culminating into policy learning from simple and easy to perform demonstrations.

The general objective of this thesis is to study the state-of-the-art of DL and develop novel methods that improve it by solving or mitigating the aforementioned open problems. The objectives and expected results for this thesis are the following:

- Extensive study of the state-of-the-art in DL:
  - Study of the pipeline: Generating the demonstration dataset, training the policy, and further optimization.
  - Analysis of the available demonstration techniques for generating the data set.
  - Comparison of DL with other machine learning paradigms and consequent understanding of its advantages and limitations.
  - Study of its main applications and identify the open problems in DL.
- Creation of a task environment and respective multi-view demonstration data set to mitigate the limited supply of such benchmarks.

- Development and evaluation of a DL algorithm that generates a context-invariant policy, without requiring extra data samples or user interaction.
- Design and develop new methods to augment RL with DL to improve the safety of RL agents during policy learning.
- Development of several sequence modelling DL methods that improve the state-of-the-art in performance and inference time, but also remove the need for hyper-parameter specification which relies on user interaction and task knowledge.
- Extend the use of DL to novel under-researched applications, thus extending the reach of the paradigm and further proving its potential.

1.1 Main Contributions

The subsequent paragraphs offer a concise overview detailing the primary contributions achieved during the development of this thesis and its associated research endeavors, aimed at advancing the state-of-the-art in DL.

The first contribution of this thesis is an extensive review of the state-of-the-art in DL. This review includes a study of DL methods, its phases, the different design choices in each phase, and the respective challenges. Then the main applications, advantages and disadvantages of the paradigm are described. The review concludes with the identification of the open problems in the field. The resulting study is presented in chapter 3.

The next three contributions are a novel method for training context-independent policies, the CLfD, a DL simulation environment for pick-and-place task, and a multi-view demonstration data set. The CLfD methods trains an encoder using contrastive learning and a multi-view demonstration data set to convert observations into view-invariant representations. We show that these representations contain information to re-align the multi-view frames of an unseen video and that they can be used in lieu of image observations to train a view-point invariant RL policy using any off-the-shelf algorithm. This work can be expanded to learn a policy invariant to other task-specific information by augmenting the demonstration data set. This work has been published in the International Conference on Robotic Computing (IRC) 2022 [9]. These contributions are included in chapter 4.

The fifth contribution of this work proposes to combine RL with DL to improve the safety of RL agents during policy learning. This method makes use of a demonstration data set with successful and unsuccessful demonstrations. We propose to use the DTW method to compare the current trajectory with the demonstrations. If the current trajectory aligns better with the unsuccessful demonstrations, the episode is terminated early to prevent the agent from crashing. We validate the approach by augmenting two state-of-the-art RL methods with our method. DEFENDER significantly increases the safety of the agent while maintaining or improving the performance of the standalone algorithm. This

work was published in the European Symposium on Artificial Neural Networks (ESANN) 2023 [10].

The sixth contribution of this thesis is the improvement of the state-of-the-art sequence modelling methods in DL. The novel method HDT, formulates sequence modelling as a hierarchical problem where the high-level model defines sub-goals for the low-level model to reach. We propose a method to identify the sub-goals from the demonstration data set before training. This sub-goal sequence replaces the need for a sequence of returns, which the previous state-of-the-art relies on to perform. We perform ablations that show that the state-of-the-art relies on the careful definition of the sequence of returns, which is non-trivial and is task-specific. The HDT not only removes the need for task-knowledge and user interaction, but it also improves the performance across a wide range of tasks and demonstration data sets. This work was published in the International Conference on Intelligent Robots and Systems (IROS) 2023 [11]

The seventh and eight contribution of this work expanded upon the previous HDT method and further improved the state-of-the-art sequence modelling methods in DL. The Transformer architecture of DT and HDT was replaced with the Mamba architecture resulting in the novel methods DM and HDM, respectively. This change in architecture reduced the complexity of the models as well as the inference time. Furthermore, the Mamba architecture removes the reliance of DM on the sequence of returns to perform, making it task and user independent without the need for the hierarchical framework. The hierarchical framework of HDM also contributes with a further increase in performance, resulting in the state-of-the-art sequence modelling in DL to date. This work is currently under review at a conference [12].

The final contributions of this work are the creation of a method with two variants for music-to-dance generation. We provide the insight of treating music to dance generation as a language translation task and propose two sequence modelling methods, a Transformer and a Mamba variant. We describe the pipeline for converting dance poses present in standard music to dance data sets to robot joint angles and synchronizing the poses with the music features. The method then trains the sequence model to learn the mapping from the language of music to the language of dance. We validate this approach on two music to dance data sets and show that despite the difficulty of generalization to unseen choreographies due to the inherent human creativity, the models are able to achieve low errors on unseen music and dance pairs. This work is currently under review at a journal [13].

All the contributions of this research have been made publicly available, and the code is available on GitHub to help fellow scholars, and to promote continuous advances. During this work, six methods were developed, with MDLT providing two variants. Additionally, a task environment and demonstration data set were created for CLfD, which are also

distributed within the GitHub repository [1] [2] [3] [4] [5] [6].

1.2 Thesis Outline

The remainder of this thesis document is organized as follows: Chapter 2 introduces the fundamental concepts, methodologies, and algorithms that are used by the proposed methods in this thesis. Chapter 3 provides an extensive review of the state-of-the-art of DL. Chapter 4 presents two methods to tackle problems of policy learning with RL. The first method CLfD proposes a solution to the context problem by using contrastive learning to obtain viewpoint-invariant state representations from demonstrations, and then use these representations in lieu of the observations to train a view-point invariant RL policy. The second method proposes to improve the safety of policy learning methods by combining the benefits of both RL and DL paradigms. We proposed the method DEFENDER that can be integrated with standalone RL algorithm and improves the safety of the agents during learning, by comparing the current trajectory with safe and unsafe demonstrations, and terminating it if it aligns better with unsafe demonstrations. The proposed method significantly improves the safety of the agent, while maintaining or improving performance. Chapter 5 presents two sequence modelling methods. The first method, HDT, improves upon the state-of-the-art DL sequence modelling, the DT. By employing a hierarchical model, we replace the user specified sequence of DT, making the model user and task independent while also improving the performance across a variety of tasks and data sets. Then, we expand upon the previous sequence modelling method, by replacing the Transformer with the Mamba architecture, resulting in two models: HDM and DM. This substitution simplifies the methods, removes the need for user interaction, reduces the inference time and further improves performance across many tasks and data sets. Chapter 6 applies DL to new applications. Specifically, we propose to model the task of learning to dance as a translation task from the language of audio to the language of dance. We propose two variants of the method, one using the Transformer, and the second using the Mamba architecture. The method is evaluated on varied data sets and is able to generalize to unseen music and dance pairs, while remaining coherent within the music genre. Lastly, chapter 7 serves as a summary to the developed work, drawing the conclusions of this thesis, and describing potential avenues for future exploration.

[1] https://github.com/meowatthemoon/CLfD
[2] https://github.com/meowatthemoon/hdt
[3] https://github.com/meowatthemoon/HierarchicalDecisionMamba
[4] https://github.com/meowatthemoon/DecisionMamba
[5] https://github.com/meowatthemoon/DEFENDER
[6] https://github.com/meowatthemoon/MDLT

Chapter 2

Background

2.1 Introduction

In this chapter, we introduce and elaborate on the fundamental concepts, methodologies, and algorithms that underpin the proposed methods in this thesis. By establishing a solid understanding of these fundamentals, we aim to provide the necessary context for methods discussed in subsequent chapters.

We begin with an exploration of the sequence modelling architectures used in this thesis: the Transformers, and the Structured State Space Model (SSSM)s. Each of these concepts is used throughout the thesis, and understanding their core principles is crucial for comprehending the methods in which they are used.

Then, we delve into pre-existing algorithms that are used by our proposed methods. We start with the Dynamic Time Warping (DTW), an algorithm for time series analysis and pattern recognition. Next, we examine state-of-the-art RL algorithms: Deep Deterministic Policy Gradient (DDPG), Soft Actor Critic (SAC) and Twin-Delayed Deep Deterministic Policy Gradient (TD3). Finally, we discuss the Decision Transformer (DT) algorithm, the previous state-of-the-art sequence modeling demonstration learning algorithm.

2.2 Transformers

Transformers are a type of sequence modeling architecture that revolutionized many NLP tasks due to their ability of processing data in parallel and their attention mechanism. Transformers are encoder-decoder based architectures where the encoder receives the input data and returns its embedding representation. The decoder receives the output of the encoder as well as the right-shifted outputs, and then outputs the prediction of the Transformer model. The Transformer architecture is represented in Fig. 2.1.

The input of the Transformer is a sequence of tokens. The size of the sequence, which the Transformer is trained to receive, is named the context length. In NLP tasks, the tokens are words. However, the Transformer can receive a wide variety of data types [14, 15]. In the case of word tokens, the Transformer can not understand word representations. Therefore, the words have to be converted into numbers. All the words that make up the vocabulary which the Transformer must understand are assigned an index number. Then, before passing the words to the Transformer, these are first replaced with their index numbers.

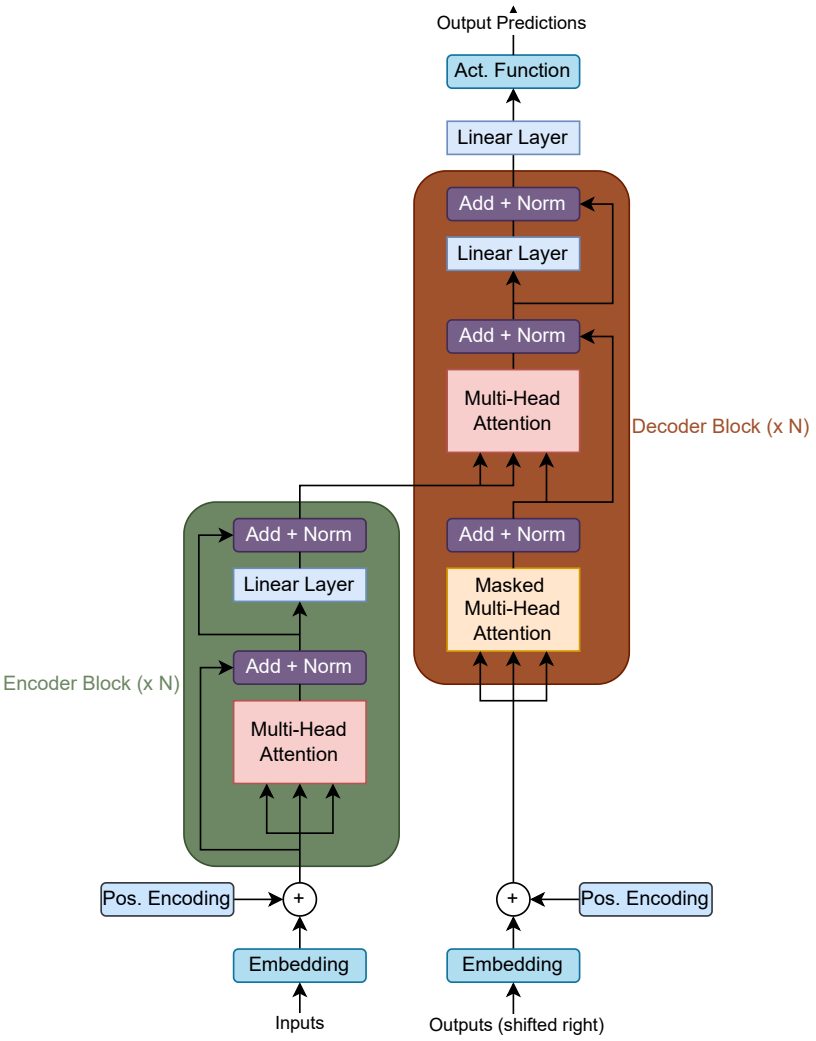


Figure 2.1: The standard Transformer architecture. The decoder and encoder blocks can be sequentially stacked, where the output of the previous block is the input of the next block.

However, forwarding these index numbers directly is impractical. This is because the Transformer would struggle to differentiate words that are very different but have similar indexes. Because of this, the inputs first pass through the embedding layer that converts the indexes into floating point vector representations. During training, the model learns the mapping of token indexes to embedding, where each dimension in the embedding captures some linguistic feature about the word in such a way that contributes to the model's success.

The next component is the positional encoding. An LSTM processes each embedding sequentially. This causes them to be very slow. However, because they process the embeddings sequentially, they know the order of the embeddings. On the other hand, the Transformer processes all the embeddings at once. This makes the model much faster than

LSTMs. However, the downside is that they lose the ordering information. To provide order information to the Transformer, positional embeddings are created and summed to each of the token embeddings. The values of the positional embeddings differ between methods. For instance, the DT forwards the index of the token to an embedding layer to obtain the positional embedding. The most common approach is to assign the values of the positional embeddings using the following formulas, alternating for every token in the sequence:

$$PE(token_{index}, 2i) = \sin\left(\frac{token_{index}}{1000^{\frac{2i}{d}}}\right) \quad (2.1)$$

$$PE(token_{index}, 2i+1) = \cos\left(\frac{token_{index}}{1000^{\frac{2i}{d}}}\right) \quad (2.2)$$

where $token_{index}$ is the index of the token in the sequence, d is the embedding size, and i is the dimension inside the embedding vector.

The main component of a Transformer [16] is the multi-head attention-layer which creates attention scores from input query **Q**, key **K** and value **V**. The attention module repeats its computations multiple times in parallel, by splitting the inputs passing them through N-heads.

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{M}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T + \mathbf{M}}{\sqrt{D}}\right)\mathbf{V} \quad (2.3)$$

where D is the number of channels in the attention layer and **M** is the attention mask. All head calculations are then combined together to produce a final Attention score. The mask parameter determines which other elements in the sequence can each element see to determine the attention score. For example, in translation tasks, the mask of the encoder is a full attention mask where only the padded elements are hidden from the attention computation.

In the encoder, the previous embeddings are repeated to form the three inputs of the attention layer. The output of the attention layer is added with the previous embedding, through a skip connection, and then the result is normalized across the batch. Then, the previous output embeddings pass through a linear layer, followed by another addition with skip connection and batch normalization. The resulting embedding vectors are the output of the encoder block. The encoder blocks can be stacked one after another, where the output of an encoder block is the input of the next encoder block. Eventually, the output of the last encoder block is the input key and value of the multi-head attention layer of the decoder blocks.

The input of the decoder is the right shifted outputs. These pass through an embedding layer the result is summed with the positional embeddings, which are frequently the same embeddings used on the encoder's side. The embeddings are passed to a masked multi-head attention layer. The difference is that the mask is a causal mask, a upper triangular look-ahead mask. This is such that each element can only look at past elements. Otherwise, the decoder would be able to see what it should predict during training. The remaining elements of the decoder correspond to the same sequence of layers of an encoder block. The difference is that the key and value inputs of the multi-head attention layer are the outputs of the encoder, while the queries correspond to the decoder's embeddings. The output of the decoder passes through a final linear layer and activation to convert the embeddings into the output dimensions.

2.3 Structured State Space Sequence Models

Transformers revolutionized NLP tasks due to their parallelization capabilities and attention mechanism. However, the attention layer calculates attention scores of all input embeddings with each other. This means that the memory and compute time scale quadratically with the context length, which can be expensive.

Contrarily, SSSMs scale linearly with the context length. SSSMs and Mamba are built upon the concept of continuous systems that maps a 1-D function $x(t) \to y(t) \in \mathbb{R}$ through a hidden state $h(t) \in \mathbb{R}^N$. This process can be represented as a linear Ordinary Differential Equation (ODE):

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t), \quad y(t) = \mathbf{C}h'(t) \quad (2.4)$$

where, $\mathbf{A} \in \mathbb{R}^{N \times N}$ serves as the evolution parameter, while $\mathbf{B} \in \mathbb{R}^{N \times 1}$ and $\mathbf{C} \in \mathbb{R}^{N \times 1}$ act as the projection parameters.

S4 models adapt continuous systems for deep learning applications through discretisation of the true continuous function. They introduce a timescale parameter, Δ which determines the precision after the discretisation, and then convert the continuous parameters $\mathbf{A}$ and $\mathbf{B}$ into discrete parameters $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$. There are different ways to perform discretisation, but the original methods choose to use the Zero-Order Hold (ZOH) method:

$$\bar{\mathbf{A}} = \exp(\Delta\mathbf{A}), \quad \bar{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(\exp(\Delta\mathbf{A}) - \mathbf{I}).\Delta\mathbf{B} \quad (2.5)$$

After the discretisation, the models can be rewritten as:

$$h'(t) = \bar{\mathbf{A}}h(t) + \bar{\mathbf{B}}x(t), \quad y(t) = \mathbf{C}h'(t) \quad (2.6)$$

Lastly, the models compute the output through a global convolution:

$$\bar{\mathbf{K}} = (\mathbf{C}\bar{\mathbf{B}}, \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}, ..., \mathbf{C}\bar{\mathbf{A}}^{K-1}\bar{\mathbf{B}}), \quad y(t) = x * \bar{\mathbf{K}} \quad (2.7)$$

where $\bar{\mathbf{K}} \in \mathbb{R}^K$ represents a structured convolutional kernel, and K denotes the length of the input sequence x.

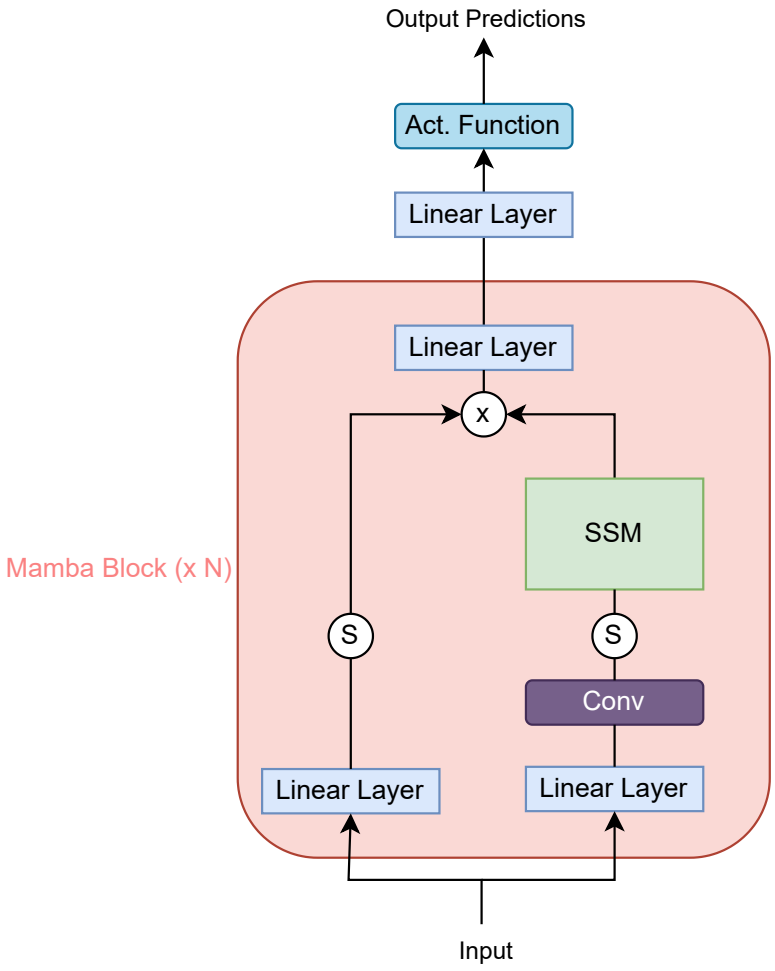


Figure 2.2: The standard Mamba block architecture. The Mamba blocks can be stacked where the output of one block becomes the input of the next in the sequence.

The Mamba architecture is represented in Fig. 2.2. Like the Transformer model, the Mamba architecture can be viewed as a sequence of stacked Mamba blocks. The inputs tokens first pass through a linear layer that is used to increase the dimensionality of the embeddings, usually doubling the dimensionality. Then the embeddings pass through the convolutional layer and are activated by a Swish activation function. The result of the activation is passed to an SSM layer. Finally, Mamba does a gated multiplication of the

output of the SSM with the output original input passed through a linear layer and activated with Swish. The author's intuition behind this operation is that the multiplication is a measure of similarity between the output of the SSM which contains information about the previous tokens and the embedding of the current token. Lastly, a linear layer reduces the dimensionality back. The output of the Mamba block becomes the input of the next block in the sequence. Like in Transformers, the output of the final block can be passed through a linear layer to convert the embeddings to the desired output dimensionality.

2.4 Dynamic Time Warping

The DTW [17] measures the similarity between two sequences of temporal data. We will use the DTW over other distance metrics that require the sequences to have the same length or require padding the sequence to equal the length of the other. Additionally, one-to-one distance metrics are naive, and require the similar regions on both sequences to be aligned. Therefore, they fail to compare sequences that are the same but one is delayed when compared to the other. The naive matching of one-to-one distance metrics is represented in Fig. 2.3.

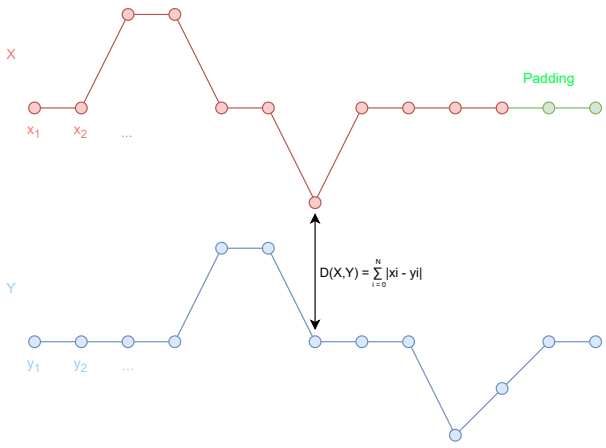


Figure 2.3: Example of a naive distance metric comparing two sequences, where the second is delayed compared to the first. The distance metric will erroneously output a large distance between the two sequences.

Contrarily, DTW allows for distortions in time and variations in speed, and does not require padding, making it particularly useful for comparing sequences with different lengths, speeds, or underlying shapes. Instead of a one-to-one comparison, the DTW matches unaligned points, by finding which points in both sequences correspond to each other. The matching process of the DTW for the example data sequences is represented in Fig. 2.4.

The DTW algorithm receives two sequences of data $x_{1:N}$ and $y_{1:M}$, where N and M are the respective sequence lengths. A cost matrix $D \in \mathbb{R}^{(N+1)\times(M+1)}$ is initialized to have the

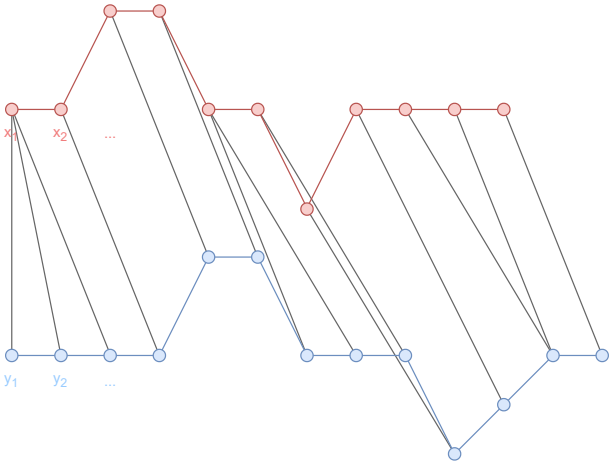


Figure 2.4: Example of DTW matching unaligned points in two sequences of data.

values of the first row and first column set to infinity, the first cell of the matrix set to zero, and the remaining values unfilled. Formally, $D_{0:N,0} = \infty$, $D_{0,0:M} = \infty$, and $D_{0,0} = 0$. The initialization of the cost matrix for the example data sequences is represented in Fig. 2.5.

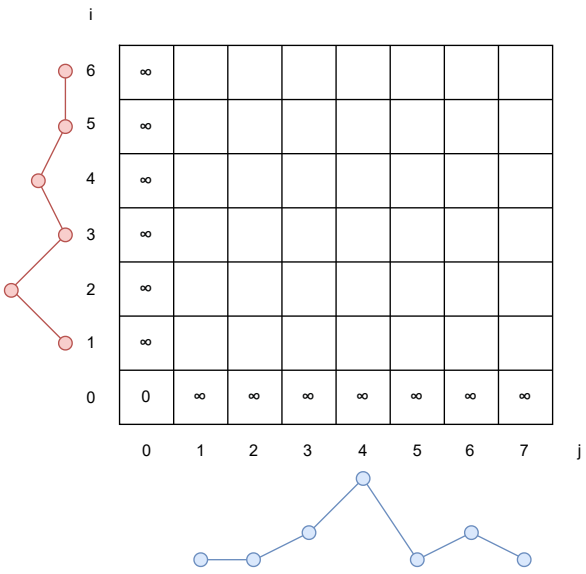


Figure 2.5: Initialization of DTW's cost matrix.

The rest of the matrix is obtained iteratively, row by row $i : N$ and column by column $j : M$. Where the value of the cell at row i and column j is obtained by the following formula: $D_{i,j} = d(x_i, y_j) + min(D_{i-1,j}, D_{i-1,j-1}, D_{i,j-1})$. The iterative process is represented in Fig. 2.6.

Finally, the alignment cost is determined by the value of the last cell, $D_{N,M}$. This value represents how much the two sequences differ from one another. The lower the alignment

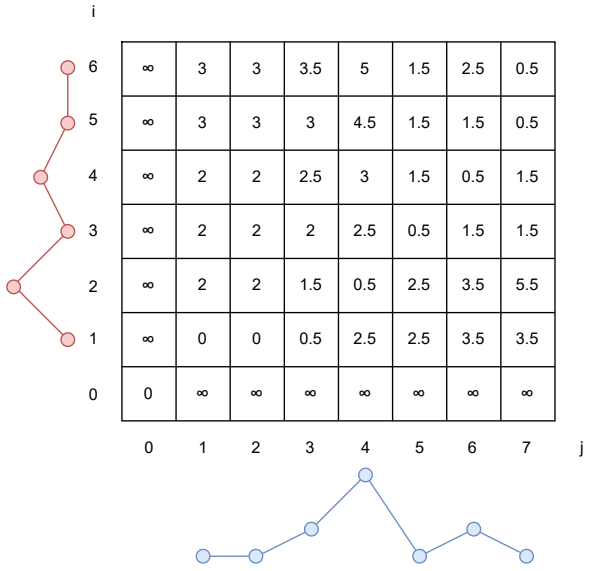


Figure 2.6: Cost matrix filled fully using the cost formula iteratively.

cost, the similar the sequences. Additionally, DTW allows to align the points of the two sequences. This alignment is done using the cost matrix by tracing the lowest cost path from $D_{N,M}$ to $D_{0,0}$. The alignment path for the previous example is shown in Fig. 2.7.

According to Fig. 2.7, the cell $\{3, 5\}$ is part of the path. Therefore, the third point of the first sequence is matched to the fifth point of the second sequence.

In practice we use FastDTW [18], an approximation to DTW with reduced time and memory costs. The algorithm finds the optimal warping path which minimizes the cumulative distance between corresponding points on the two sequences. We will use the alignment cost of the path to measure the similarity between two demonstrations.

2.5 Deep Deterministic Policy Gradient

DDPG interleaves learning an approximation to the optimal state-action value function $Q^*(s, a)$ and the optimal policy π^*. It is an off-policy algorithm, which means it stores previous transitions which were performed by a different policy, and stores them in a replay buffer $B = \{(s_i, a_i, r_i, s'_i, d_i), i \in B\}$, where s_i, a_i, r_i, s'_i, d_i are the initial state, action, reward, new state and d_i indicates whether state s'_i is a terminal state. The transitions in the replay buffer are then used to update the state-action value function which is then used to update the policy.

The Bellman equation defining the optimal action-value function is:

$$Q^*(s, a) = \mathbb{E}_{(s')\sim P}[r(s, a) + \gamma \max_{a'} Q * (s', a')] \quad (2.8)$$

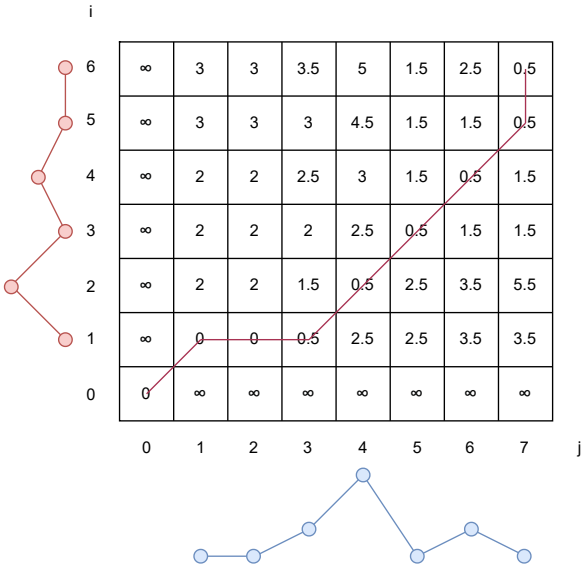


Figure 2.7: Alignment path of the two sequences obtained by tracing the lowest cost path from the last cell to the initial cell of the matrix.

The approximation to the optimal state-action value function is done through a neural network with parameters ϕ. The parameters of this neural network are updated using a batch of transitions in the replay buffer using mean-squared Bellman error (MSBE) as the loss function which determines how close the current function is to the optimal state-action function:

$$L(\phi, B) = \mathbb{E}_{(s,a,r,s',d)\sim B}[(Q_\phi(s,a) - (r + \gamma(1-d)\max_{a'} Q_\phi(s',a')))^2] \quad (2.9)$$

The loss function is used to update the parameters of the state-action value function, however, the Bellman term on the loss function depends on the same parameters that are being optimized. This causes the optimization to be unstable. To reduce this issue, the DDPG algorithm uses a second network, called the target network whose parameters ϕ_{target} are similar to ϕ but are time-delayed. In practice, the target network is a partial copy of the main network on iteration. This process is called Exponential Moving Average (EMA): $\phi_{target} \leftarrow \rho\phi_{target} + (1-\rho)\phi$, where ρ is a hyper-parameter with a value between 0 and 1 which determines the portion of the main network to be copied over to the target.

Computing the maximum over actions in the target is a challenge in continuous action spaces. DDPG deals with this issue using a target policy network to compute an action which approximately maximizes $Q_{\phi_{target}}$. Assuming the policy network π has parameters θ, the target policy network $\pi_{\theta_{target}}$ is obtained using the EMA of the parameters of the policy network. With these changes, the loss function for the estimation of the state-action

value function is:

$$L(\phi, B) = \mathbb{E}_{(s,a,r,s',d)\sim B}[(Q_\phi(s,a) - (r + \gamma(1-d)\max_{a'} Q_{\phi_{target}}(s', \pi_{\theta_{target}}(s'))))^2] \quad (2.10)$$

The estimated state-action value function is used to estimate the policy. The goal of the policy is to select the action for the current state that maximizes the state-action value function. Hence, the parameters of the policy are updated through gradient ascent using the output of the state-action value function, when using the state batch sampled from the replay buffer and the action selected by the current policy for these states:

$$\max_\theta \mathbb{E}_{s\sim B}[Q_\phi(s, \pi(s))] \quad (2.11)$$

Lastly, to perform exploration, DDPG adds the Ornstein-Uhlenbeck noise [19] to the action sampled by the policy. The known problem with the DDPG algorithm is that the state-action value function in DDPG often overestimates its values. This then enables the policy to wrongly exploit the state-action values, which causes the policy learning to wrongly converge to a local minimum.

2.6 Twin-Delayed Deep Deterministic Policy Gradient

The TD3 algorithm improves upon the DDPG algorithm in regards to state-action value overestimation, which leads to policy failure, with three key changes. Firstly, the TD3 algorithm learns two state-action value functions instead of a single one. Then, the algorithm uses the lowest output of the two value functions in the Bellman error loss functions. This is a conservative selection of the prediction of the true state-action value, which reduces the overestimation. Formally, both state-action value functions are updated using the same target, calculated using the minimum of both outputs:

$$L(\phi_{i=1,2}, B) = \mathbb{E}_{(s,a,r,s',d)\sim B}[(Q_{\phi_i}(s,a) - (r + \gamma(1-d)\min_{j=1,2} Q_{\phi_{j,target}}(s', a'(s'))))^2] \quad (2.12)$$

Secondly, the TD3 algorithm updates the policy in the same way as the DDPG by selecting one of the state-action value functions. However, in the TD3 algorithm, the policy is updated less frequently than the state-action value functions. The original TD3 paper updates the policy for every two state-action value function updates. The idea is to obtain a more accurate prediction of the state-action values before updating the policy.

Lastly, the TD3 algorithm adds noise to the target action, to prevent the policy from exploiting the state-action value functions. Specifically, the target actions are changed to: $a'(s') = clip(\pi_{target}(s') + \varepsilon, a_{Low}, a_{High})$, where ε is a noise sampled from a normal distribution $\varepsilon \sim \mathcal{N}(\imath, \sigma)$, and clip is a function that keeps the noisy action within the action range of the environment $[a_{Low}, a_{High}]$.

2.7 Soft Actor Critic

Both the DDPG and TD3 algorithms optimize a deterministic policy, that is, the output of the policy function is an action. Contrarily, SAC is algorithm that optimizes a stochastic policy. In this case the output of the policy function is the mean and standard deviations of a distribution. The main feature of the SAC algorithm is entropy regularization, which measures the randomness in the policy. The policy is trained to maximize a trade-off between expected return and entropy. Similarly to the exploration-exploitation trade-off, increasing the entropy results in more exploration, prevents the policy from converging to a local minimum and can also accelerate learning later on.

The entropy H of a distribution P is computed by:

$$H(P) = \mathbb{E}_{x\sim P}[-\log(P(x))] \quad (2.13)$$

In the SAC algorithm, the agent gets an additional reward proportional to the estimated entropy of the policy at the time:

$$\pi^* = arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t) + \alpha H(\pi(s_t)))] \quad (2.14)$$

where α is a hyper-parameter that controls the importance of the entropy in the optimization. There are SAC implementations that keep this hyper-parameter to a fixed value while others vary it during training.

Similarly to TD3, SAC learns a policy and two state-action value functions simultaneously. These state-action value functions are named the critics. The loss function for updating the state-action value functions is similar to TD3, where the minimum output of the functions is used as the prediction for the value of state-action pair. However, this loss function is augmented with entropy contribution:

$$L(\phi_{i=1,2}, B) = \mathbb{E}_{(s,a,r,s',d)\sim B}[(Q_{\phi_i}(s,a) - y(r,s',d))^2] \quad (2.15)$$

where y is the target state-action value defined by:

$$y(r,s',d) = r + \gamma(1-d) \min_{j=1,2} Q_{\phi_{j,target}}(s', a'(s')) - \alpha \log \pi_\theta(a' \mid s') \quad (2.16)$$

and the new action is sampled from the policy $a' \sim \pi_\theta(s')$.

The policy is updated in a similar way to the previous algorithms, where the differences are the added entropy component, the state-action value is obtained using the minimum

of both functions instead of selecting one like in TD3, and the next action is obtained from the current policy:

$$\max_{\theta} \mathbb{E}_{s\sim B}[min_{i=1,2} Q_{\phi_i}(s, a') - \alpha \log \pi_\theta(a' \mid s)] \quad (2.17)$$

2.8 Decision Transformer

The DT formulates RL as a sequence objective problem and applies the Transformer architecture because of their success in a wide range of applications. Specifically, the DT employs the GPT 2 [20] architecture. Instead of processing a single state observation, the policy selects the next action, based on a sequence of states, actions and returns-to-go (RTG): $(rtg_1, s_1, a_1, ..., rtg_N, s_N, a_N)$. During training, the RTG are the remaining cumulative return obtained in the trajectory after time step t: $rtg_t = \sum_{t'=t}^{T} r_{t'}$. However, during deployment, the full trajectory is not known before execution to determine the initial value of the sequence. Instead, this initial value of desired returns must be specified by the user, which is task-specific and affects the performance [11]. We performed ablation studies in Sec. 5.2.5, and the results show that the DT relies on the sequence returns-to-go to learn and that the value of the desired accumulated returns impacts the performance of the agent. However, determining a good value for the desired accumulated returns is done through trial and error. The DT sets the value to the maximum accumulated rewards found in the data set. The results in Sec. 5.2.5 show that this is not optimal, because in some tasks, setting a lower value resulted in increased performance. Additionally, in tasks with sparse rewards, where the environment returns a reward of zero, the value of returns-to-go is not changed for concurrent transitions. The DT struggles to learn the task when the value of returns-to-go is static.

The DT also requires a causal mask, which is a binary vector with as many elements as the length of the sequence. Each element in the mask determines if the corresponding tokens should be hidden from the Transformer to produce the output. Additionally, the Transformers require information about the relative position of the tokens in the sequence. For the DT, the positional encoding is done by creating a sequence of time steps, passing this sequence through an embedding layer and summing the result to each of the three token sequences. The DT is trained using the L2 loss, in a similar manner to BC, to predict the demonstrator's action:

$$\mathbb{L}(s_{t:t+K}, a_{t:t+K}, rtg_{t:t+K}) = \|a_{t+K} - \pi_\phi(s_{t:t+K}, a_{t:t+K-1}, rtg_{t:t+K})\| \quad (2.18)$$

2.9 Conclusions

In this chapter, we have introduced and elaborated on the fundamental concepts, methodologies, and algorithms that form the foundation of the proposed methods in this thesis.

By establishing a solid understanding of these fundamentals, we have aimed to provide the necessary context for the advanced methods discussed in the subsequent chapters.

Chapter 3

Related Work

3.1 Introduction

The demand for intelligent systems that mimic human behavior increases. Future directions in Artificial Intelligence (AI) focus on replacing humans with machines that replicate the desired behavior more consistently. Notable examples include self-driving vehicles [21] and surgical robots [22]. This progress is driven by continuous advancements in the area of AI, with increasingly complex problems being solved each year.

Emulating human behavior implies replicating a sequence of actions that a human would take in various situations. The human behavior can be recorded in the form of demonstration data sets, which are then used to train models to replicate the behavior by selecting the appropriate action based on the current state of the agent and the environment. Solving this problem entails learning the correct mapping between states and actions. This mapping is known in computer science as a policy, a function that selects an action based on the current state. Traditional programming approaches specify the action for every possible state. Moreover, such algorithms do not scale well to high-dimensional environments or continuous action spaces, because specifying the ideal action for all possible state conditions is tedious and computationally impractical. Additionally, these approaches require expertise in the specific area as well as programming knowledge, making them costly.

Because of this, ML techniques offer a solution to automatically learn policies from data, with RL being a common method. In RL, the agent learns the policy through trial-and-error interactions with the environment. After each interaction, the agent receives feedback and adjusts its policy accordingly. The agent often learns super-human policies [23], often achieving the task's goals by performing actions that would be impossible or unlikely for a human. While this behavior can be advantageous for maximizing performance, it can be disadvantageous if the goal is for the agent to behave naturally. A clear drawback of reinforcement learning approaches is their need for a large number of interactions. Because the agent learns from failure and attempts random actions, the learning process can be unsafe in real-world environments, posing risks to the agent and its surroundings. This limitation hinders the application of RL in fields like robotics and healthcare. Additionally, RL is highly data-inefficient, requiring many interactions to converge due to the need for exploration. Despite the significant progress and potential of deep RL, its applications remain largely confined to video games and simulations.

For these reasons, DL is an appealing alternative to RL. In [24], the authors proposed an off-policy RL algorithm that can learn to play Atari games from image data. Two years later, AlphaGo [25] trained the first computer agent capable of beating a professional Go player. In such cases, the agent has access to a data set of interactions with the environment performed by an expert teacher. The agent learns the policy from the demonstrated state-action pairs present in the data set. The goal of DL is to learn complex policies, akin to RL, but from recorded data, similar to supervised learning. By providing examples of successful actions, the need for the agent to try incorrect actions during an exploration phase is avoided. Consequently, the agent remains safe during the learning process because it does not have to interact with the environment directly. Additionally, demonstrations ensure that the agent learns the desired behavior exhibited in the data set, allowing it to avoid local minimum and converge faster than with RL.

The primary disadvantage of DL is that the agent is entirely dependent on the demonstrations for learning. The demonstrations must cover the state and action spaces for the agent to learn the task effectively. Depending on the task, this coverage can be difficult and expensive due to the size of both spaces. Demonstration learning methods attempt to generalize to unseen states. Although demonstration learning methods attempt to generalize to unseen states, the data is not identically distributed (i.i.d.), making generalization challenging. If the agent encounters states outside the distribution of the data set, it is likely to fail, with potentially serious consequences in real-world applications. Therefore, the demonstration learning methods try to mitigate the distributional shift between the learned distribution and the data set's distribution, which is still an open problem. Additionally, recording demonstrations performed by a human requires capturing the state-action pairs, showcasing good behavior, and covering various scenarios. Demonstrations can suffer from sensor noise, inaccuracies, or inconsistencies in the demonstrator's actions. Consequently, DL approaches avoid directly copying the demonstrated behavior and instead attempt to generalize to non-demonstrated trajectories.

The use of demonstration data sets is a key differentiator among DL methods. Utilizing demonstrations reduces the programming overhead, making these methods accessible to non-experts. As a result, interest in this area has grown exponentially due to its significant potential. Once a policy is learned from demonstrations, it can be further refined through online interactions via RL, with the advantage that the initial policy is safer than a randomly initialized RL policy. In [23], an agent is pre-trained on demonstration data such that the convergence and its online interactions are safer. Later, [26] proposed an algorithm to learn solely from demonstrations, causing the field to gain traction. Demonstration learning is not limited to policy learning, the demonstrations can be used to learn a dynamics model, which allows the agent to collect new transitions and learn through online RL without having to interact with the environment. Alternatively, IRL can utilize demonstrations to learn a reward function, which is often challenging to design in high-dimensional RL applications.

The paradigm of teaching robots through demonstrations emerged in the 1980s and has since been considered the future of robotics [27, 28]. Over the years, this approach has been used to teach a wide range of tasks to various types of robots. Applications include aerial and ground navigation [29], video games [23], and controlling different kinds of robots, from manipulators [30] to humanoids [31]. The advent of deep learning has exponentially increased research interest in this area over the past two decades, leading to a significant rise in publications.

This rapid growth has resulted in numerous surveys on the topic. Early surveys reviewed the initial history of the paradigm and early attempts to teach robots from demonstrations. One such survey provided an overview of demonstration learning and defined the problem using four core questions for the field: how, what, when, and whom to imitate [32]. In a subsequent survey [1], the authors discussed various design choices and proposed a categorization for the field. Later, [33] focused on reviewing artificial intelligence methods used to estimate policies. Another survey [34] reviewed recent research and development, with a focus on demonstrating behaviors to assembly robots and extracting manipulation features. [35] provided a comprehensive review of inverse reinforcement learning. In [36], the authors gave an overview of various machine-learning methods, highlighting their advantages and disadvantages. Following this, [37] and [38] reviewed offline reinforcement learning methods. The timeline of these published surveys is represented in Fig. 3.1.

Despite the surveys mentioned above aiming to standardize terminology, the terms used in recent publications remain diverse. Terms such as DL, Learning from Demonstrations, Imitation Learning, BC, and Offline RL are commonly used to describe the same paradigm. For consistency, this chapter will use the term ”DL” to refer to this paradigm.

This chapter provides an overview of the steps required to learn from demonstrations and the various methods employed by researchers at each step. The reviewed literature encompasses a wide range of applications. The chapter explains each step in a general manner, making it applicable to most tasks.

The following sections are organized as follows: Section 3.2 presents a formal definition of the DL problem. Section 3.3 discusses methods for collecting demonstration data and creating data sets. Section 3.4 explains the learning methods available from the demonstration data. In Section 3.6, we present the benchmarks available to evaluate DL methods. Section 3.7 lists the main applications of DL, followed by a discussion of the advantages and disadvantages of the paradigm in Section 3.8. Finally, Section 3.9 explores future research directions in DL, concluding with a summary in Section 3.10.

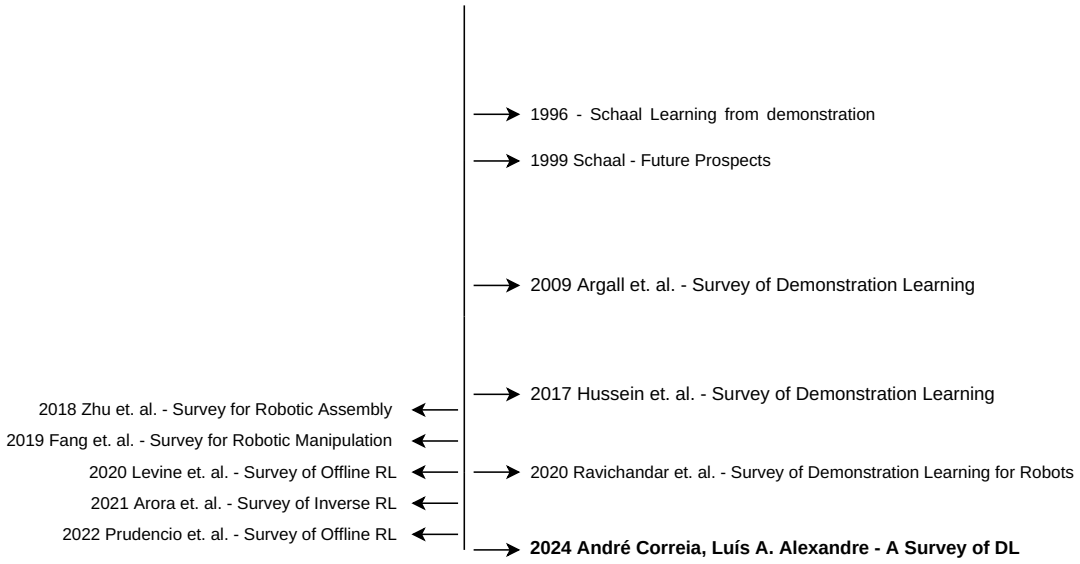


Figure 3.1: Timeline of surveys. General DL surveys are presented on the right side, while surveys specific to a sub-area or application are presented on the left side.

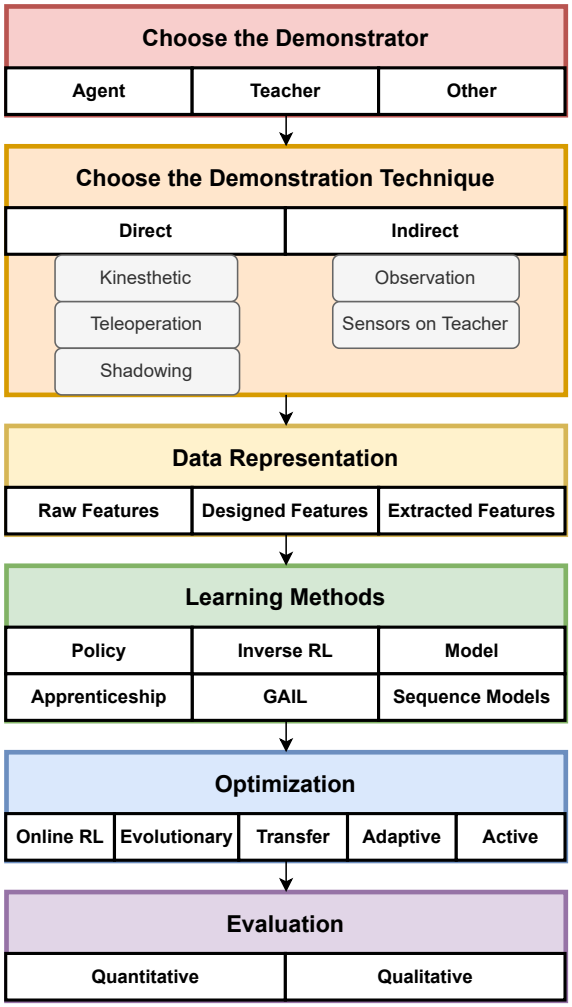


Figure 3.2: DL flowchart.

3.2 Problem Definition

This section explains the DL problem and several relevant concepts. The overall sequence of steps in DL is illustrated in figure 3.2. The first step involves creating the demonstration data set, which requires selecting the demonstrator, the demonstration technique, and the data representation. The demonstrations can be generated by the agent itself using teleoperation, kinesthetic, or shadowing demonstration techniques. These techniques are categorized as direct demonstrations because the agent itself performs them. Alternatively, a teacher, either a human or a robot not acting as the agent, can perform the demonstrations. These are known as indirect demonstrations, as the learning agent does not perform them. In these techniques, demonstrations are recorded via observations or sensors on the teacher, with the captured information mapped to a format the learning agent can use.

Next, the agent learns through DL using the demonstration data set. The learning methods can be categorized by their objective, most commonly to learn a policy function. However, DL can also be used to learn a reward function through IRL or a dynamics model, both of which can then be applied in standard RL applications. Additionally, DL can be used to learn a generator, or a discriminator through Generative Adversarial Imitation Learning (GAIL), a sequence model, and seamlessly integrated with apprenticeship learning.

Subsequently, the learned model can be further enhanced using optimization techniques such as online interactions with RL and active learning. The evaluation of DL methods is usually quantitative, though some applications may require qualitative assessment. Each of these steps is discussed in more detail in their respective sections.

An agent is an entity that autonomously interacts within an environment toward achieving or optimizing a goal [39]. It receives information from the environment through its sensors and interacts with the environment using its actuators, based on its policy.

DL is a mixture of supervised learning and RL. In supervised learning, the agent receives the labeled training data and learns an approximation to the function that produced the data. In RL, the agent must collect interaction data to learn from, by interacting with the environment through trial and error. In DL, the training data is a set of environment interactions collected beforehand by a teacher executing a task. The goal of DL is to learn a task from demonstrated examples performed by an expert demonstrator and recorded in a data set.

DL expands from the RL paradigm commonly defined as an MDP, formulated by the tuple $< S, A, P, \Delta_0, R, \lambda >$ [8]. An MDP is a mathematical formulation that enables the creation of theoretical statements and proofs in RL, where S is the set of the possible environment states, $s \in S$, and Δ_0 denotes the initial state distribution. At each state, the

agent can choose an action $a \in A$ from the set of possible actions. Acting changes the state of the environment. The mapping between states through the actions is defined by the state transition function $P(s_{t+1} \mid s_t, a_t) : S \times A \to S$. The Markov property determines that the transition function completely defines the dynamics of the environment. That is, the probability of state s_{t+1}, only depends on the current state s_t and selected action a_t, regardless of past transitions. The policy $\pi : S \to A$ is a function that selects an action given a state of the environment. A more common definition is to formulate the policy as a probability distribution $\pi(a_t \mid s_t)$, where the policy returns the probability of taking action a_t given the agent is at the current state s_t. The correct selection of the action for any given state is what allows the agent to perform the task. After interacting with the environment, the agent receives a reward $r_t = R(s_t, a_t, s_{t+1})$, which indicates the quality of the interaction. In RL, the policy is optimized to maximize the expected future rewards $\mathbb{E}[\sum_{t=0}^{\infty} \lambda R(s_t, a_t, s_{t+1})]$. A trajectory is a sequence of $H+1$ states and H actions and rewards $\tau = (s_0, a_0, r_0, ..., s_H)$, where H is the episode's horizon, which may be infinite in the case of non-episodic environments. With these definitions, the probability density function for a given trajectory τ under the policy π is $\rho_\pi(\tau) = \Delta_0(s_0) \prod_{t=0}^{H-1} \pi(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t)$. Lastly, λ is the discount factor.

In some settings, we do not have access to the full state information of the environment and have to work with observations $o_t \in O$. This formulation is named Partially-Observable Markov Decision Process (POMDP), defined by the tuple $< S, A, O, P, \Delta_0, E, R, \lambda >$, where O is the set of observations and $E(o_t \mid s_t)$ is the function that maps states to observations. To overcome the limitations of learning from observations, methods combine consecutive past observations to supply the policy with time-varying information, such as velocity and direction.

One of the main reasons behind the success of ML methods is the usage of large data sets. However, in RL the data set is collected during training and can be expensive and unsafe to collect in the real-world.

In DL, the agent has access to a data set of N demonstrations $D_{demo} = \{\tau_i, i \in \{0, ..., N-1\}\}$. Each demonstration is the sequence of visited states and the respective actions chosen by the expert demonstrator $\tau_i = (s_t, a_t, s_{t+1}, t \in \{0, ..., L-1\})$, where L is the length of the sequence. The policy is estimated from the behaviors shown in the demonstration data set. The agent learns by increasingly better imitating the behavior of the teacher represented in the demonstrations. Therefore, the behavior of the agent should converge to a working and intended behavior.

BC is the family of methods where the policy is trained to output the demonstrated action for a given state. Hence, the problem becomes a classification problem for discrete action spaces or a regression problem for continuous action spaces. However, the quality of the learned behavior is limited to the ones present in the demonstrations. Because of

this, demonstrations can be used to formulate a reward function. This family of methods is called IRL. Here, the agent is rewarded for how similar the action is to the one in the data set for a given state. Alternatively, the demonstrations can include the environment rewards in addition to the states and actions: $\tau_i = (s_t, a_t, r_t, s_{t+1}, t \in \{0, ..., L-1\})$. This family of methods is named offline RL because the agent has access to the interaction data in an offline manner. Here the ideal policy π^* is estimated by maximizing the expected accumulated reward for all trajectories τ: $\pi^* = \arg\max_\pi \mathbb{E}_{\tau \sim \rho_\pi(.)}[R_\tau]$, where $R_\tau = \sum_{t=0}^{N-1} \gamma^t r_t$ is the discounted accumulated reward of trajectory τ with N transitions. Most methods estimate a state-value function to optimize the policy: $V^\pi(s_t) = \mathbb{E}_{\tau \sim \rho_\pi(.|s_t)}[R_\tau]$, which maps a state to the expected return when starting from that state. Similarly, an action-value function $Q^\pi(s_t, a_t)$, maps state-action pairs to the expected return starting from state s_t, and using action a_t.

Under offline RL, the goal is to use the data sets to generalize instead of naive imitation learning by finding the good parts of the demonstrated behavior. Even if the data set has bad behaviors, finding the good parts would result in an improvement over the demonstrations. Though distinguishing bad from good behaviors is difficult, offline RL accounts for long-term consequences of immediate actions through the value-function, unlike BC.

In the following section, we will delve into the process of creating a demonstration data set, exploring each step and the various options available. We will begin by discussing the selection of the demonstrator, followed by an examination of different demonstration techniques and data representation methods. Additionally, we will address the primary limitations that affect demonstration data sets, their potential consequences, and possible solutions.

3.3 Demonstration Data Set

The initial step in DL is creating the demonstration data set. This data set comprises a series of demonstrations, each represented as a sequence of state-action pairs. As stated previously, the data set can include extra information such as the environment's rewards for each interaction. The developer has a plethora of design options for the system. This section outlines the various choices for designing the demonstration data set, discussing their impacts on the final design, and comparing their advantages and disadvantages.

The first step is selecting the demonstrator. The next step involves choosing the demonstration technique, which depends on the type of demonstrator chosen. Afterward, defining how the data will be stored is crucial. The data must be usable by the learning agent, ideally mapping directly to state-action pairs usable by the learner. However, this direct mapping is not always feasible, and conversion mechanisms may be required. For example, learning from images requires the extraction of features, which may be manually-designed or learned. The demonstrated images are captured from the point of view of

Figure 3.3: Record and Embodiment Mapping as per [1].

the teacher and may not be directly usable by the learning agent. Challenges that arise from the differences between the contexts of the demonstrator and the learner are named correspondence issues by [40]. All these steps are explored in the following sections.

3.3.1 Choosing the Demonstrator

The demonstrator is the agent responsible for demonstrating the task. Two choices have to be made regarding the demonstrator: selecting who controls the demonstration and who executes the demonstration. The task can be demonstrated by either a human or a robot different from the learning agent, who controls the demonstration. However, through shadowing or teleoperation, a human can control the learning agent, allowing it to execute the task. The demonstration techniques will be explored in the following sections. In these cases, a human teacher controls the demonstration, while the learning agent executes the demonstration.

The choice of the demonstrator is critical to the success of the system, as it influences the algorithms that can be used. If the learning agent performs the demonstrations itself, for example, through teleoperation, the learner's state-action spaces will automatically align with those in the data set. However, if a different agent executes the demonstrations, the state and action spaces in the demonstrations will likely need to be mapped to the learner's corresponding spaces.

3.3.2 Demonstrator and Learner Matching

This subsection addresses the alignment of state-action pairs from the teacher, as represented in the demonstration data set, with the learner's state-action pairs, enabling the learner to perform the task. In [1], the authors defined two types of mappings: record mapping and embodiment mapping. While these terms are not widely used in the literature, we adopt them here to help classify the different demonstration techniques.

Record mapping corresponds to the mapping between the teacher's demonstrated state-action pairs and the recorded state-action pairs in the data set, denoted as $(s_{recorded}, a_{recorded}) = m_R(s_{teacher}, a_{teacher})$. Embodiment mapping corresponds to the mapping between the state-action pairs recorded in the data set and the state-action pairs performed by the learning agent, denoted as $(s_{agent}, a_{agent}) = m_E(s_{recorded}, a_{recorded})$. These mappings are represented in figure 3.3. Each mapping corresponds to applying a conversion function to the input to produce the output. Importantly, neither mapping alters the information in the demonstration data set. Instead, they change the data from the context of the teacher to the context of the agent.

Improving the Robustness of Demonstration Learning

Ideally, the data is recorded by the learning agent, eliminating the need for any mapping. In such cases, these mappings are equivalent to using the identity function $I(s, a)$. However, there are scenarios where the demonstrator's setting cannot be directly recorded or directly applied to the learner. In these instances, one or both conversion mappings must be created. For any given problem, each additional mapping introduces a potential source of errors. Consequently, the more mappings that are applied, the more challenging it becomes to accurately translate and reproduce the teacher's original behavior.

As an example, consider a human teacher demonstrating a task to a robot using their own body. A camera records these demonstrations as a series of images. An implicit record mapping is applied to convert the raw data captured by the camera into these images, as the images do not fully capture the entire environment. Additionally, the specific actions performed by the teacher between frames are unknown to the robot. The robot can not infer what action was taken by the teacher to transition from one frame to the next. Therefore, an embodiment mapping is necessary to generate the state-action pairs that the robot can use to understand and replicate the demonstrated task.

3.3.3 Choosing the Demonstration Technique

In this section, we explore various techniques for acquiring demonstrations. According to [1], demonstration techniques are categorized into two groups: demonstration and imitation, based on their need for embodiment mapping. However, we adopt a simpler categorization proposed in [41], which divides demonstration techniques into indirect and direct demonstrations. In direct demonstration techniques, the learning agent itself performs the demonstration. Conversely, in indirect demonstration techniques, an external agent demonstrates the task.

In the direct category, no embodiment mapping is needed because the demonstration is performed directly by the target agent. Conversely, indirect techniques require embodiment mapping since the demonstration is not performed on the learning agent. Furthermore, within each category, approaches are grouped based on whether they require record mapping. This categorization is presented in Table 3.1.

The demonstration techniques involve choosing how the demonstration data is provided to the learner. The most common approach is to have the data available beforehand, allowing the learner to derive a policy from a pre-existing data set. Alternatively, the data can become available incrementally during training, leading to ongoing policy updates. These incremental approaches are typically used to refine the policy based on its performance during training. For instance, in [42], the agent displays its confidence score for the selected action in a given state. The teacher can then choose to intervene and demonstrate the correct action or accept the agent's action.

Table 3.1: Categorization of the demonstration techniques.

	Direct			Indirect	
	Teleoperation	Kinesthetic	Shadowing	Observation	Sensors on Teacher
Mapping	No	No	Record	Embodiment and Record	Embodiment
Demonstrator	Learner	Learner	Learner	Not the learner	Not the learner
Data recorded	Learner's internal state	Learner's internal state	Learner's internal state	Visual	Sensor

[43] discusses techniques that enable a robot to refine its existing model, focusing primarily on interactive and active learning approaches. Interactive task learning, proposed in [44], involves the agent actively learning a task through natural interactions with a human instructor. The concept of modeling verbal and non-verbal cues used by the teacher during the teaching process to enhance learning is explored in [45]. Additionally, the idea of asking for help during the learning process is examined in [46], where a data-driven method was developed to estimate the human's beliefs after receiving a request and to create better requests that guide people toward providing useful help. We provide a categorization of the available demonstration techniques in Table 3.1.

3.3.4 Direct Demonstration

Direct demonstration consists of techniques where the embodiment mapping is unnecessary because the learning agent performs the demonstrations. Hence, there is no need to convert the state-action pairs from the demonstrator's space to the learner's space, as they are inherently the same. However, record mapping may still be required if the state-action pairs performed by the demonstrator cannot be directly recorded in the data set.

3.3.4.1 Teleoperation

Teleoperation is the most direct method for transferring the teacher's behavior to the learner. In this setting, the human teacher operates the learning agent or an agent structurally identical to the learner. This agent can be a physical or simulated robot or a simulated agent, such as characters in video games. The state-action pairs of the demonstration are recorded directly from the learning agent's sensors. Because the agent performing the task is structurally identical to the learner, and the state-action pairs are extracted directly from the agent's sensors, neither embodiment mapping nor record mapping is required.

The main advantage of teleoperation is that it can be easily used in simulation environments and video games, unlike the other approaches. Additionally, teleoperation simplifies data collection, facilitating the development of new methods and benchmarks. However, a significant downside is that it requires the agent to be manually controllable, which limits its applicability to certain problems. This control can be executed through various interfaces, such as joysticks, graphical user interfaces, or virtual-reality interfaces. Another disadvantage is that not all users possess the necessary skills to teleoperate the agent effectively without extensive training. This need for technical proficiency undermines one of the key advantages of DL, which is its accessibility to a broader range of users.

Demonstrations using teleoperation have been applied to a wide variety of applications. For example, in [47] and [29], data from a robot helicopter flight, controlled with a joystick, was recorded and used to train an autonomous agent through RL. For manipulation tasks, [48] demonstrated how a robotic arm could be trained to change rolls on a paper roll holder from demonstrations. In [29], a humanoid robot was teleoperated using Virtual Reality technology, which translated the operator's arm and hand motions into those of the robot to generate demonstration data and develop a manipulation policy. Teleoperation has also been used in simulated environments. For instance, in [49], used teleoperation to transfer human skills in Robosoccer to a robot through DL. Additionally, [50] involved teleoperating a PR2 robot to touch a red cube on a table to create demonstration data, which was then used to train the robot to associate movements with labels and perform a sequence of trajectories based on this labeled data.

3.3.4.2 Kinesthetic

Kinesthetic teaching serves as an alternative to teleoperation when an external mechanism for controlling the agent is not available. In this approach, the teacher physically manipulates the agent by moving its joints through the correct positions that enable the agent to perform the task. Alternatively, the teacher can provide instructions through speech, where the robot is told specifically what to do. Similar to teleoperation, demonstration data in kinesthetic teaching is captured directly from the agent's sensors, so there is no need for any mapping. However, the quality of the demonstrations heavily relies on the capabilities of the human teacher. Even with expert demonstrators, the data obtained through kinesthetic teaching often requires post-processing techniques to refine the demonstrations.

The applications of kinesthetic demonstrations are similar to those of teleoperation. However, kinesthetic demonstrations are generally restricted to physical agents and are primarily used for manipulation tasks. For example, a learning method for collaborative and assistive robots based on imitation learning through kinesthetic demonstrations was applied to a robotic arm in various assistive scenarios [51]. To extract features from each state for kinesthetic teaching, a system that captures desired behaviors in the joint space was developed in [52]. Additionally, [53] explores how kinesthetic teaching can be used to capture demonstrations for modeling reward functions in manipulation tasks.

3.3.4.3 Shadowing

Demonstrations performed through shadowing are performed by the learning agent. Since the demonstration data is captured through the agent's sensors, there is no need for embodiment mapping. However, the agent performs the task by copying the movements of the teacher through some form of tracking. There is a record mapping that converts the actions performed by the teacher to the actions of the learning agent.

Shadowing has been effectively applied to both humanoid robots and navigation tasks. In [54], a humanoid robot learns to imitate a human demonstrator's arm gestures and is tested on a turn-taking gesture game. For navigation, a mobile robot learns routes demonstrated by a teacher in [55].

3.3.5 Indirect Demonstration

Indirect demonstration encompasses techniques where an embodiment mapping is necessary. The demonstration data is not captured directly from the learning agent's sensors because a different agent demonstrates the task. Hence, the agent can not directly apply the state-action pairs in the demonstration data set.

3.3.5.1 Observation

In indirect demonstration by observation, a teacher performs the task while sensors external to the learner capture the demonstrations. These sensors are often cameras or a camera system, and they may be complemented by additional sensors on the teacher. During this process, the learning agent remains a passive observer. The main advantage of this technique is its straightforward data collection process. However, it requires both record mapping and embodiment mapping to transform the captured data into usable state-action pairs for the agent. Since only image representations of the teacher's actions are recorded rather than the actual state-action pairs, record mapping is needed to convert these images into state-action pairs. Additionally, because the learning agent does not perform the task, embodiment mapping is necessary to translate the demonstration data into a format that the agent can use for learning. This is often accomplished through ML techniques, such as feature extraction or automated conversion from the teacher's context to the learner's context.

While indirect demonstration by observation is the simplest method for task demonstration, it necessitates both record mapping and embodiment mapping, making it less favorable compared to other techniques when they are viable options. This approach is particularly effective in settings with high degrees of freedom, where other demonstration techniques might be difficult to perform. However, it comes with challenges such as camera-related issues such as occlusions, blurriness, and noise-that can affect data quality and require extra post-processing steps.

DL through observation is the most versatile and widely applicable demonstration technique. It has been employed in various applications, such as teaching a robot to perform assembly tasks from demonstrations in [56], learning house chores in both real-world and simulated environments as explored in [57], and manipulating a piece of cloth to create different shapes in [30]. Early works teach a robotic arm to balance a pole using demonstrations [58]. In [59], a robot demonstrates a task and transfers its skill through DL to a different robot. Moreover, DL from observation is frequently combined with other information sources beyond just camera data. For instance, [60] demonstrates how a robot

learns to grasp objects by integrating visual observations with data from a force-sensing glove.

3.3.5.2 Sensors on Teacher

Sensors on a teacher is a demonstration technique where the demonstrations are recorded from sensors on the teacher such as wearable devices [61]. In this approach, record mapping is not required because the data is captured directly from the teacher's sensors. However, because the learning agent did not perform the task, the recorded data can not be directly used by the agent. This means that an embodiment mapping is required to convert the teacher's state-action pairs to the context of the learning agent. Because of this requirement, these approaches are used when the learning agent can not be controlled to perform the task itself. Otherwise, teleoperation, kinesthetic, or shadowing techniques should be preferred. Human teachers are commonly used to demonstrate a task using their own bodies to train a humanoid robot. The advantage of using sensors on a teacher over passive observation lies in the precision of the data collection. Sensors provide detailed and accurate data about the teacher's actions, whereas passive observation requires indirect methods to infer and map the teacher's actions for the learner.

In [31], the data from sensors placed on a human is used to derive joint angles for a humanoid robot, enabling it to learn and perform reaching and drawing movements with one arm as well as tennis swings. This approach has also been applied in [62] to teach a biped robot to replicate human-like walking patterns. Additionally, [63] describes the use of a custom glove to capture hand position and tactile information for recording demonstration data performed directly by a human. The data was then used to obtain object model representations and optimize the policy to perform the task.

3.3.6 Data Representation

The demonstration data needs to be recorded in a structured way. The structure is dependent on what technique was used for recording the demonstration and will determine which algorithms can be used to train the policy. The state representation is called a feature vector, which may encompass various types of information, including the agent's state, the environment, and individual objects. Due to the high dimensionality of the environment, it is often impractical to represent it in its entirety, and it may contain redundant or irrelevant information for the learning task. Hence, the selection of features needs to be adequate and efficient to convey enough relevant information to estimate a quality policy. The actions performed by the teacher are also normally included in the demonstrations. However, some approaches overcome the unavailability of actions in the data set and learn to infer the action that caused a state transition [64]. Additionally, in scenarios where the goal is to maximize a reward function, rewards for each state-action transition can also be part of the data set. Lastly, supplementary information such as episode termination indicators and safety constraint violations may be included [65, 66].

3.3.6.1 Raw Features

Raw features are the direct outputs from sensors collected during the demonstration and stored in the data set without any additional processing. Hence, these features are used in cases where there is no record mapping. However, the features may inherit noise from the sensors and may need to be pre-processed. If these features capture all the necessary information for the task, they can be used directly for training without further processing.

Such features can be easily obtained in virtual environments such as simulators or video games. For example, in [23], the agent learns to play a series of Atari games using state-action observations that are direct screenshots from the game. However, in the real-world, these features are often not readily available and typically require additional sensors to be collected.

3.3.6.2 Manually Designed Features

Manually designed features are extracted using custom-designed functions tailored to the specific problem at hand. These functions transform raw sensor data from the demonstration into a more efficient structure for training the agent. These functions filter out irrelevant and redundant information, often reducing dimensionality and focusing on the most useful features for the learning task.

In [67], a robot is trained to imitate human movement from observations. The authors define key points on the human demonstrator's body, and the agent learns to detect motions by identifying changes in the key points' locations. In [68], object positions are tracked and used as features of the demonstrations. For video games, [69] obtains screenshots of the Mario Bros game and divides them into binary cells, each signifying if the respective cell contains objects.

3.3.6.3 Extracted Features

Extracted features are obtained from a learned function designed to process raw sensor data from demonstrations and identify relevant information for learning. Unlike manually crafted features created by experts through careful problem analysis, these functions are generated automatically through a model specifically trained for feature extraction. Typically implemented as neural networks, these models function as black boxes, extracting features in ways that are not explicitly understood by the programmer. Automatic feature extraction models are particularly useful when task-specific features can not be identified through expert evaluation. Therefore, they offer the advantage of minimizing the need for task-specific knowledge and have broader applicability, as they can be trained to extract features for a variety of tasks. Consequently, they allow for the creation of a more streamlined DL pipeline.

Improving the Robustness of Demonstration Learning

In [70], deep learning techniques are employed to automatically extract features for training an agent to play a set of Atari games. Similarly, [71] demonstrates the use of features automatically extracted from observations to train a robot for various manipulation tasks. In [72], an encoder is utilized to extract features from state observations. Here, the encoder is trained simultaneously with the policy in an off-policy manner, improving sample efficiency.

3.3.6.4 Time as a Feature

In this approach, demonstrations consist of time-action pairs. Such features can be applied to tasks where it can be assumed that the environment's state depends solely on time. Hence the agent can learn the ideal action for a specific time interval, and its behavior will always be optimal. It is assumed that there is a time loop synchronized with a fixed sequence of environmental states. The overhead of designing such features and algorithms is reduced tremendously. Furthermore, the efficiency of such policies will be high while the aforementioned time synchronization requirements remain the same. The main restriction of such features is their applicability, since rarely does the state of the environment depend solely on time.

Other limitations of time-based policies include issues with robustness and task dependency. Because these policies rely heavily on a fixed time structure, they are highly sensitive to changes in the environment that can disrupt time synchronization. Additionally, time-action pairs are typically specific to a particular task, making it challenging to adapt or scale these policies for similar tasks. To address these limitations, [3, 73] explore methods for synchronizing different demonstration sequences. In their approach, frames from various demonstration sequences with the same time index are expected to exhibit similar behavior, representing the same point in the task. The model is then trained to extract features from these frames, aiming to generate consistent features for frames with the same timestamp while differentiating between features from frames at different timestamps.

3.3.7 Data set Limitations

The performance of the policy is directly dependent on the quality of the information provided by the demonstration data set. In this section, we explore how various limitations of the data set can affect the agent's performance and outline important properties to consider when designing or utilizing a demonstration data set.

3.3.7.1 Incomplete Data

The demonstration data set represents only a subset of the full MDP's distribution. The larger the distribution sample, the easier it is to generalize and tackle the curse of dimensionality problem. It also reduces the likelihood of encountering out-of-distribution states and dealing with the problem of distributional shift. However, collecting real-world demonstrations that adequately cover the entire MDP is often challenging and sometimes

impossible, particularly in continuous state and action spaces. If certain states are missing from the demonstrations, the learner can not estimate the optimal actions for those states during policy estimation. This section presents the ways DL approaches tackle missing data points in the data set. Human-generated demonstrations often produce narrow distributions, which can exacerbate issues related to out-of-distribution states. To mitigate these challenges, it is crucial to manage distributional shifts by ensuring that the agent does not venture beyond the data set's distribution. Additionally, another common issue is the non-inclusion or sparsity of reward signals in the data set. Creating a comprehensive reward function is often difficult, especially in complex state and action spaces. Consequently, it is sometimes easier to work with sparse or even absent rewards, though this approach can make the learning problem significantly more difficult.

The simplest idea to deal with limited data is to obtain new demonstrations. In such approaches, as the learner interacts with the system, it may encounter novel states and request a demonstration from the teacher for that given state. For instance, [74] proposes a confident execution approach, which focuses on learning relevant parts of the task where the agent identifies the need to request demonstrations. In this approach, the agent must decide between requesting a demonstration and executing actions autonomously. As the agent learns the task, it increases its autonomy, reducing both the teacher's training time and workload. In an alternative approach, [42] addresses this problem by having the agent provide the teacher with a confidence score for its chosen action. The teacher can then decide whether to intervene and provide a demonstration or to accept the agent's action [75, 66].

The previous approach requires additional overhead to identify missing states in the data set and demands extra commitment from the teacher during the learning process. The alternative approach corresponds to generalizing using the available data. One way to generalize is to create new data from the existing set. Data augmentation is often used in ML to enlarge the data set and improve generalization to unseen data. Such techniques can be applied to state representations to generate unseen data points. In [76, 77], different data augmentation schemes are compared and applied to off-the-shelf RL algorithms. However, naively applying data augmentation to RL can cause new problems. The authors of [78] identify pitfalls for naively applying transformations to RL algorithms and then teach how to properly use them. Additionally, [79] addresses the instability problem in RL by estimating the Q-values from an ensemble of agents. Another approach is to perform stitching, which involves combining portions of different unsuccessful trajectories to solve a task.

Another approach is to use transfer learning methods and learn from data of other tasks. In [80], it is shown that in certain conditions, the challenge of learning from few demonstrations for a given task can be mitigated by using demonstrations of other, related tasks. Even when rewards for the host task are either unusable or unavailable, they can be set to

zero to ensure that the learning process remains effective. Furthermore, performance can be enhanced through the application of re-weighting methods to adjust the significance of transitions from these other tasks.

Alternatively, to generate new data without additional effort from the teacher, the learner can directly interact with the environment. In such approaches, the learner is pre-trained on the available demonstration data and is then fine-tuned with RL to collect further data. Since the agent starts with a solid foundation from pre-training, it is more competent and safer during exploration compared to an agent trained from scratch through RL alone. However, this method requires careful balancing between exploring new data and exploiting existing data for effective policy estimation. Additionally, it requires creating an exploration policy and a reward function that gives feedback based on the agent's actions in various states.

In [23], the agent is pre-trained on demonstration data before interacting with the environment. Then, the agent's policy is updated using both the demonstration data and the exploration data. In [81], a different approach is proposed where two policies are learned simultaneously. The primary policy executes the task, while a secondary policy enforces constraints to prevent the primary policy from taking actions that could lead to harmful outcomes. Another approach is created in [82], where a second agent is trained to make the learning of the main agent as difficult as possible. This adversarial setup creates more difficult conditions for the main agent during training, resulting in a policy that is more robust and resilient to various challenges.

Some methods explore the state space by maximizing the entropy of the visited state distribution [83, 84]. In [83], a policy is trained to explore the state space while estimating the representations. The state space is clustered, and the policy is rewarded based on the distance between the visited states and the nearest cluster, thereby encouraging exploration of less-visited regions. In [84] a world model is estimated in conjunction with an exploration policy. Here, the policy is rewarded for maximizing the variance of the predictions of an ensemble of networks, which promotes exploration by favoring actions that lead to unpredictable outcomes. In [85, 86], address the challenge of safe exploration by constraining the policy to ensure that it adheres to pre-defined safety restrictions.

3.3.7.2 Inadequate Data

Most DL approaches assume that the quality of the data in the demonstration data set is optimal. However, this often is not the case. The demonstrated behavior can be suboptimal, which in some cases can be intended if the goal is for the policy's behavior to appear human. Additionally, demonstration data can suffer from various issues such as sensor noise, blurriness, and occlusions. Furthermore, the data may be redundant or unevenly distributed, which can affect the learning process. To address these challenges, [87] introduces two algorithms designed to handle demonstration data corrupted by noise.

These algorithms extract the idea of the expert demonstrator using Instrumental Variable Regression techniques from econometrics.

Another significant issue is data ambiguity, which occurs when there is inconsistency in the teacher's choices, resulting in different actions being selected for the same state across various demonstrations. This inconsistency means that a single state is mapped to multiple different actions in the data set.

Additionally, the data may contain unsuccessful demonstrations. When these demonstrations are appropriately labeled as unsuccessful or provide reward information that the policy aims to maximize, they can enhance the policy's robustness by guiding the agent to avoid certain state-action pairs. For example, if the policy is trained to learn from failed attempts, it can better understand what actions to avoid in similar situations. However, if unsuccessful demonstrations are mistakenly treated as successful, as often happens in BC approaches, they can degrade the quality of the learned policy. In short, the quality of the learned policy is directly affected by the quality of the data.

Some approaches leverage sub-optimal demonstrations to enhance generalization and achieve smoother behaviors in the learned policies. For example, [88], demonstrates how repeated demonstrations are used to encourage such behavior and smooth the policy. In another approach, [89] explores how data from multiple teachers can be used strategically to challenge the learning agent, resulting in a more robust policy. Some approaches identify inadequate demonstrations and choose to remove them from the data set before training the policy. For instance, [90] presents techniques for diagnosing and addressing sources of inadequacy in the demonstration data. Alternatively, [91] adopts a more nuanced approach by incorporating both successful and failed demonstrations. They separate the two types of demonstrations into clusters using an adapted version of Gaussian mixture model (GMM). They then perform regression using the Gaussian components from the cluster of successful demonstrations to generate improved trajectories for the learning agent.

Other solutions address inadequate data by seeking additional demonstrations from the teacher, as discussed in Section 3.3.7.1. Another effective strategy involves using RL to manage poor-quality data by collecting new interaction data. In this approach, the learner is first pre-trained on the available demonstration data and then fine-tuned through exploration-based methods. During this fine-tuning phase, the learner interacts with the environment and adjusts its policy based on feedback obtained either through a standard reward function or direct guidance from the teacher. For example, in [92], the authors propose a method where sub-optimal demonstrations are employed to constrain an RL algorithm. In contrast, [93] highlights the value of failed demonstrations, proposing a method that leverages these failures to train a policy. Their approach focuses on teaching the agent to avoid repeating unsuccessful behaviors observed in the failed demonstrations. In [94], an algorithm is proposed for learning policies from partially observable

state environments. Alternatively, [95] choose to estimate the quality of demonstrations by estimating the competence of the demonstrator and filtering the transitions based on the competence level.

3.4 Learning from Demonstrations

In this section, we explain the different methods available in the literature for using the demonstration data set. In general, the DL methods learn a policy or a world model. However, the demonstration data sets have also been used to learn other types of models which we will discuss.

3.4.1 Learning Problems

DL relies on the data set to learn the models. As discussed in Section 3.3.7, collecting the perfect data set is unfeasible for most applications, resulting in limitations that affect the learning method. To counter the limitations of the demonstration data sets, the methods should aim to generalize to regions outside the demonstrated regions in the data set. However, if the data set does not contain transitions that correspond to high-value decisions, such as those with high associated rewards, discovering these regions may be impossible. [37] argues that this challenge is insurmountable and that methods should assume that the data set contains sufficient information for developing a suitable model. To address imperfect demonstrations, methods should filter out poor demonstrations. Moreover, the method should learn to extract the beneficial parts of the demonstration, avoid the detrimental parts, and potentially combine parts from multiple demonstrations. Naive imitation in a self-supervised manner through BC risks copying bad behaviors. Therefore, methods that filter out poor demonstrations can achieve a better policy than the one represented by the data set.

Another problem with DL is its inherent paradox. DL combines supervised learning with the transition data of RL. To improve upon the policy of the data set, the goal is to answer what are the sequences of actions that generate the maximum reward. However, supervised learning methods assume that the data is i.i.d. The model should obtain good performance as long as the data it encounters comes from the same distribution as the one it was trained on. However, in DL, the goal is often to mimic or improve upon the behavior observed in the data set.

All these problems could be alleviated by interacting with the environment and testing uncertainties that the method may have. This is why DL is often followed by RL to refine the policy with online interactions. Pure DL is difficult because the agent can not collect additional transitions and explore new regions. Technically, any off-policy method for online RL could be used to learn a model from the demonstration data set. However, these methods were created with the assumption that the agent could interact with the environment to correct existing errors. DL estimates a model to perform a task defined

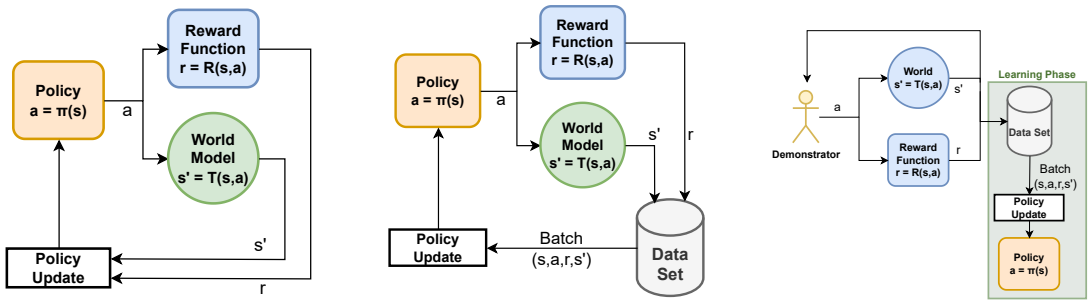


Figure 3.4: Differences between on-policy RL, off-policy RL, and DL.

by the state-action distribution Δ_{task}. DL methods estimate the model using the demonstration data set which contains a set of transitions. The data set also has an associated distribution $\Delta_{demo} \subset \Delta_{task}$, which is a subset of the task's distribution. However, during deployment, the agent will likely encounter regions outside the distribution of the data set, $(s, a) \notin \Delta_{demo}$. The prediction of the model for such regions will result in larger mistakes than for in-distribution regions. Furthermore, these mistakes will accumulate and the agent will continue to diverge from the learned distribution. This snowball effect is known as the 'distributional shift'. Most policy learning methods in offline RL address the distributional shift in various ways. Some use BC to restrict the distribution to that of the dataset, which heavily limits generalization. Others propose to punish the distributional shift in the training loss by an estimation of uncertainty. Others constrain the agent to specific regions by making conservative estimates of future rewards for the Bellman update, by learning a lower bound estimation of the true value function. In [69], the authors proved that even with optimal action labels, the compound errors from distributional shift accumulate to a quadratic error in the best-case scenario. However, this error would scale linearly, if the agent was allowed to collect additional transitions. DL methods struggle to balance generalization and avoiding the distributional shift.

3.4.2 Policy Learning

Policy learning from demonstrations involves learning the correct mapping from states to actions from the demonstration data set. The teacher demonstrated a policy $\pi_{teacher}$ and the demonstrated state-action pairs in the data set are examples of the correct mapping. The closer the estimated mapping function is to the original mapping in the data set, the better the agent will reproduce the teacher's behavior. The agent can collect additional transitions by interacting with the environment with RL, receiving a reward, and adjusting the policy accordingly. By maintaining a history of past interactions, the agent can continuously update its policy. This approach is known as off-policy RL because the policy is updated with data collected by a previous policy. In DL, the agent learns from recorded data from the start, through the demonstration data set. The differences between the three settings are summarized in Fig. 3.4.

3.4.2.1 Behavior Cloning

BC is the simplest method for deriving a policy from a demonstration data set. In this approach, the policy is trained to directly imitate the teacher's actions for all states in the data set. The problem corresponds to either a classification or regression problem, for discrete or continuous action spaces, respectively. Formally, the policy is trained to minimize the error between its predicted action and the ground truth action for all state-action pairs in the data set:
$\pi_\theta^* = \arg\min_\theta(\pi_\theta(s) - a), \forall(s,a) \in D_{demo}$.

Another approach is to maximize the likelihood of actions in the demonstration:
$\max \mathbb{E}_{(s,a)\sim\mathbb{D}}\, log\pi(a|s)$.

However, because BC naively copies the data set, it is more reliant on the quality and size of the demonstration data set than other alternatives. The data set corresponds to a sub-set distribution $\Delta_{demo}(s|a)$ of the real distribution of states over actions for a given task, $\Delta(s|a)$. BC guarantees the agent's performance, as long as it only encounters states present in the demonstration data set. However, no such guarantees exist if the agent encounters an unseen state. In [95], the authors address the susceptibility of BC to the quality of demonstrations by estimating the competence of the demonstrator and filtering the transitions based on the competence level.

3.4.2.2 Offline RL

Sometimes, direct imitation through BC is not adequate to reproduce the desired behavior and solve the task due to errors in the demonstration or poor generalization. The term offline RL is often used interchangeably with DL to describe various methods. In this context, we use offline RL to refer specifically to methods that apply RL techniques to a data set of demonstrations.

In offline RL, the agent has access to the rewards attributed by the environment to each transition. The policy is trained to maximize the expected accumulated reward $J(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma R(s_t, a_t)]$, where γ is the discount factor.

In general, all RL algorithms follow the same basic train loop. The agent observes the current environment state $s \in S$, then interacts with the environment by selecting an action from its policy $a_t \sim \pi(s_t)$. This interaction changes the state of the environment to s_{t+1}, and the agent receives a reward $r_t = R(s_t, a_t)$. This process repeats for multiple interactions. The agent stores the transitions (s_t, a_t, s_{t+1}, r_t) in its memory and uses them to update the policy. In offline RL, the memory is provided by the demonstration data set D_{demo}.

Due to the limitations of BC, some approaches pre-train the agent on demonstration data and then optimize it to learn the remaining state-action space using online RL [23]. However, online RL is dangerous as some actions can lead the agent to catastrophic states which are unrecoverable in real-world scenarios. Because of this, some approaches choose to employ pure offline RL and apply regulators to reduce the impacts of distributional shift [96] or prevent the agent from going out of distribution [97].

One way to optimize the policy, parameterized by weights θ, for the Bellman objective is to estimate the gradient: $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{(s_t,a_t,s_{t+1},r_t)\in D_{demo}}[\sum_{t=0}^{H} \gamma^t \nabla_\theta \log \pi_\theta(a_t \mid s_t) Q^\pi(s_t, a_t)]$, where $Q^\pi(s_t, a_t)$ is the state-action value function.

Alternatively, we can use dynamic programming methods by first estimating the state or state-action value functions, and then using them to optimize the policy. The state value function $V^\pi(s_t)$ returns the estimated expected accumulated reward that can be obtained by starting at state s_t: $V^\pi(s_t) = \mathbb{E}_{(s_t,a_t,s_{t+1},r_t)\in D_{demo}}[\sum_{t'=t}^{H} \gamma^{t'-t} R(s_t, a_t)]$.

The state-action value function $Q^pi(s_t, a_t)$ is similar and returns an estimation of the expected accumulated reward that can be obtained by starting at state s_t and performing action a_t: $Q^\pi(s_t, a_t) = \mathbb{E}_{(s_t,a_t,s_{t+1},r_t)\in D_{demo}}[\sum_{t'=t}^{H} \gamma^{t'-t} R(s_t, a_t)]$.

From these definitions, we can reformulate them into a recursive form: $Q^\pi(s_t, a_t) = R(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim P(s_{t+1}|s_t,a_t), a_{t+1}\sim\pi(a_{t+1}|s_{t+1})}[Q^\pi(s_{t+1}, a_{t+1})]$.

The algorithms that estimate the policy based on dynamic programming are mainly split into two families: Q-learning and Actor-Critic methods.

In Q-learning, the policy is obtained directly by estimating the state-action value function, and selecting the action that maximizes the expected accumulated reward: $\pi(a_t \mid s_t) = \arg\max_{a_t} Q(s_t, a_t)$. The Q-learning objective is defined by $Q_\theta(a_t, s_t) = R(s_t, a_t) + \gamma \mathbb{E}_{(s_{t+1}|s_t,a_t)}[\max_{a_{t+1}} Q_\theta(s_{t+1}, at + 1)]$.

Actor-critic algorithms are a mixture of policy gradients and dynamic programming because they use a policy, the actor, like policy gradients, but also use a value function, the critic, like dynamic programming. Actor-critic algorithms learn the state-action value for the current policy $\pi_\theta(s_t)$: $Q^\pi(s_t, a_t) = R(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim P(s_{t+1}|s_t,a_t), a_{t+1}|\pi(s_{t+1})}[Q^\pi(s_{t+1}, a_{t+1})]$.

In early research, a set of algorithms were explored to fasten and improve RL. The authors in [98], compared eight RL frameworks, including pre-training with demonstration data, for performing a task of playing a 2D game. The authors concluded that pre-training the policy on demonstration data prevents the learner from falling in a local minimum and increases its scores. Furthermore, the improvements are more noticeable with the increase in the difficulty of the task. One notable example of the success of pre-training is

the development of a RL agent capable of playing the game "Go" at a level that rivals the best human players [25]. In this application, the initial policy's weights are obtained from training on demonstration data, which are then refined through exploration using RL. In [23], the authors tackled the issue of RL being impractical to real-world issues, due to the risks associated with trial-and-error learning and the potential for severe consequences. Their approach involved pre-training the policy on demonstration data, which resulted in higher rewards during the initial learning iterations compared to standard RL methods. This demonstrates that pre-training on demonstration data leads to safer and more effective exploration strategies.

Additionally, a policy trained from online interactions can then demonstrate correct interactions. These demonstrations can then be used to transfer the knowledge to another agent. This approach bypasses the need for a human teacher by learning the policy entirely through trial and error, and then recording the successful interactions as demonstrations. It is especially valuable in situations where real-time policy updates are not possible [70].

3.4.2.3 Classification

In this context, classification refers to the process of assigning a specific action class to a given input state. This approach is used when the action domain is discrete, consisting of a finite set of predefined individual actions. For example, in a 2D platformer game where the agent can only walk left or right or jump, the inputs are categorized into these three actions. The policy's performance is evaluated by how often it attributes the correct action for any given input state.

Formally, the policy is a classifier $\pi(s)$, used to predict the action class a of an observation s. Where $a \in A, A = \{a_1, ...a_n\}$ is a finite set of actions.

Classification methods can be applied to different levels of complexity ranging from low-level actions to complex behaviors. For instance, Bayesian networks were used in [99] for navigating an environment and avoiding obstacles. In [100], the authors created mapped representations of the environment and used a k-nearest neighbors algorithm to select the robot's actions. In [74], a GMM was used for classifying actions in navigational problems. Additionally, [101] evaluated four different classifiers for cooperative tasks in robot soccer.

More recently, neural networks have become the preferred classifiers due to their role as universal approximators, capable of modeling complex functions. For example, [102] employs Recurrent Neural Networks (RNNs) to train a robotic arm on manipulation tasks using demonstration data collected in a simulation environment. The RNN learns to predict trajectories in real-time, considering both the current position of the end-effector and the objects in the environment. Neural network classifiers are particularly effective for

video games, where the number of possible actions is finite. In [23], the policy is represented by a neural network classifier with 18 neurons in the final layer, each corresponding to one of the 18 possible actions.

Some works have proposed ways of discretizing continuous action space [103]. This allows any discrete RL algorithm to be applied to the continuous state problem.

3.4.2.4 Regression

In this context, regression involves selecting a set of scalar values that define an action based on a given input state. These techniques are used when the action domain is continuous. For example, in the control of a robotic arm, each action can be defined by the robot's joint angles. The policy's performance is evaluated by comparing the estimated action values with the ground-truth action values in the data set for the given state.

Formally, the policy is a regressor $\pi(s)$ which maps a state $s \in S$ to actions $a \in A$, where each action is defined by a finite set of continuous values $a = \{a_1, ..., a_n | a_k \in \mathbb{R}\}$. Typically, regression approaches are applied to low-level motions and not high-level behaviors because high-level behaviors are a combination of low-level motions and are more likely to be discretized.

A traditional regression technique is Locally Weighted Regression (LWR), which is well-suited for learning trajectories composed of sequences of continuous values. In [104], a robotic arm is trained to execute a trajectory enabling it to perform manipulation tasks. Locally Weighted Projection Regression (LWPR) extends the previous approach to cope and scale with the input data's dimensionality and redundancy. [42] uses LWPR to teach a robot to perform basic soccer actions.

Similar to classification tasks, recent works commonly use neural networks for regression due to their ability to represent any function. In [3, 73], a robotic arm is trained from demonstrations to perform manipulation tasks. The action is determined by a neural network, where the number and values of the last layer's neurons correspond to the number of joints of the robot.

Other approaches specify the type of task the agent can perform within the network. In [105], a robotic arm is trained to pick and place blocks. The network that outputs the actions has four neurons: the first two specify the position and rotation of the end-effector for picking up a block, while the other two are for placing the block. Therefore, the two groups of neurons are used separately.

In [106], an algorithm is proposed to convert discrete actions in the demonstration data set into continuous ones. An encoder is trained for this purpose to promote behavioral and data-distributional relations in the features. Then, an off-the-shelf algorithm can be

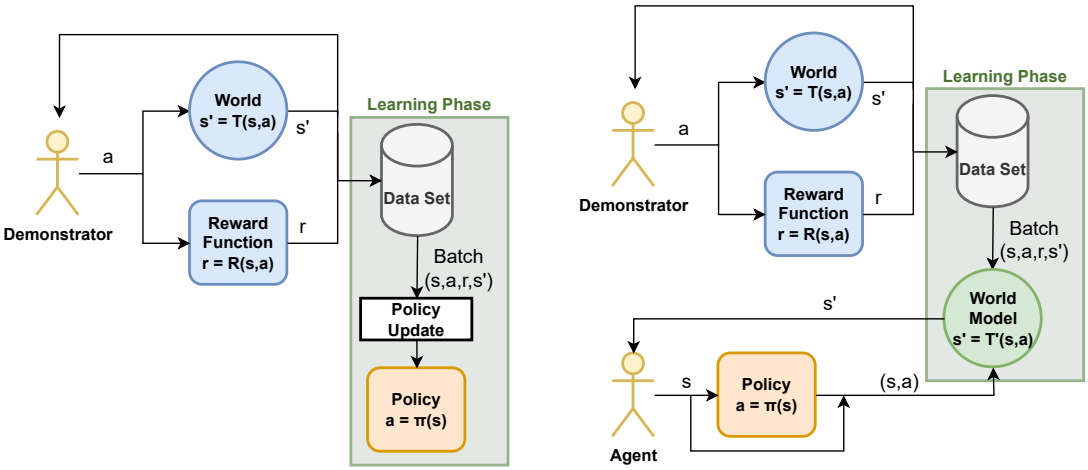


Figure 3.5: Differences between policy learning and model learning from demonstrations.

used to train a policy using the new data set. However, because the policy outputs feature embeddings, the actions can't be directly applied to the task environment. Therefore, the output of the policy is discretized by finding the action whose embedding is closer to the output of the policy.

3.4.3 Model Learning

Model-based methods learn the dynamics of the environment by estimating the transition function $\psi(s_t, a_t) \sim P(s_{t+1} \mid s_t, a_t)$. This estimated transition function serves as a proxy for the real environment, allowing the agent to collect new transitions without directly interacting with the environment, remaining safe during the learning process. In standard RL, the agent must interact with the environment to collect transition data that represent the dynamics. In DL, the transition function can be estimated from the demonstration data set. These differences are represented in Fig. 3.5. The functions are typically estimated through standard supervised regression, using the states and actions as inputs and the next states as the desired output: $\mathbb{L}_\psi(s_t, a_t, s_{t+1}) = \|s_{t+1} - \psi(s_t, a_t)\|$. Model-based learning methods from standard RL can be used to learn from demonstrations [107, 84]. Standard online learning algorithms can be applied with minimal modification to train a model from demonstrated data.

However, because the policy learns from transitions simulated by the model, its performance is dependent on the quality of the estimated model, which in turn relies on the quality and coverage of the data distribution in the data set. In standard RL, the models can correct mistakes in the estimations by collecting new transitions. Similarly to policy learning, if the model is estimated solely from demonstrations, it can suffer from the distributional shift problem. In fact, the model can suffer from distribution shift regarding the true state distribution, and the true action distribution.

The distributional shift can cause the model to be exploited by the policy. The policy is optimized to maximize the expected accumulated rewards and may use the model to pro-

duce out-of-distribution states. Because these states are out-of-distribution, the model's predicted values are likely incorrect and may have an associated higher reward than the true state in the real MDP. Consequently, the policy learns to maximize erroneous transitions, leading to a worse performance once deployed to the real MDP.

A theoretical analysis presented in [107] formulates the bounds on the error between the learned policy and the policy in the data set, attributing these errors to distributional shifts in both the policy and the model. The methods for reducing the distributional shift in policy learning can also be applied to model learning. The main way to reduce the distributional shift problem in model learning is by learning an auxiliary model $\mathbb{U}(s, a) : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}$, that punishes the reward function such that the agent avoids states outside the distribution: $R'(s, a) = R(s, a) + \mathbb{U}(s, a)$. Model-learning methods usually measure the uncertainty using an ensemble of models. In [108], the method only punishes the reward function if the disagreement of the ensemble is above a threshold. Alternatively, [109] adopts a pessimistic approach by selecting the maximum prediction uncertainty of the ensemble. In both approaches, the policy is penalized for visiting states where the model is likely to be incorrect. However, measuring uncertainty is challenging and often unreliable. To address this, [110] proposes learning the policy by generating a new data set from transitions produced by each of the models of an ensemble to counter uncertainty. Another approach to reduce the distributional shift without quantifying uncertainty is to use a regularizing term. For example, in [111] a model-based version of the Conservative Q-Learning (CQL) [96] algorithm is proposed.

Instead of learning the policy within the model, the learned model can be used to evaluate the policy without interacting with the environment. For instance, [112, 113, 114] utilize the model to estimate the expected return of the trajectories generated by the policy. In [115], a model is trained from demonstrations to estimate a task from images, which is particularly challenging due to their high dimensionality.

3.4.4 Inverse Reinforcement Learning

Reward functions map a state transition to a reward value based on the quality of the interaction: $R(s, a) : S \times A \rightarrow \mathbb{R}$. They define the objectives for the agent by guiding the learning process to maximize the expected accumulated reward. Traditionally, the reward functions are handcrafted by the programmer, which involves designing a function that assigns a reward value to each state-action pair. However, this task becomes increasingly challenging in high-dimensional domains, where covering the entire state space can be difficult and often results in sparse rewards. Transitions where the agent receives no feedback, through a reward of zero, hinder the convergence of the policy to an optimal one and sometimes may prevent convergence altogether. The requirement to create a reward function that covers the entire task, limits the applicability of learning algorithms to problems where a reward function can be easily specified.

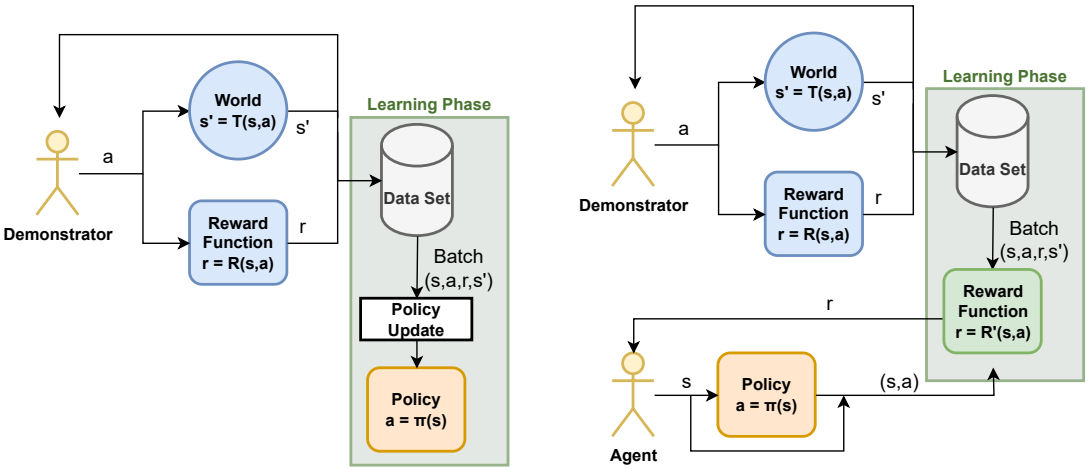


Figure 3.6: Differences between policy learning and reward learning from demonstrations.

An alternative is IRL [116], also known as reward shaping. In IRL, the demonstration data set is used to infer a reward function, which is then employed to train a policy using standard online RL methods to maximize the expected accumulated reward defined by this function. IRL thus broadens the applicability of task learning models and reduces the manual effort required by programmers when demonstrations of the task are available. The differences between policy learning and reward learning from demonstrations are illustrated in Fig. 3.6. [39] points out that the reward function is more transferable than a policy. While minor changes to the task can make a policy ineffective, such changes have a far less significant impact on the reward function. Typically, the learned reward function only needs to be extended to accommodate new states, rather than completely redesigned.

DL assumes that the teacher follows a policy $\pi_{teacher}$ which is maximizing a reward function $R_{teacher}(s, a)$ when demonstrating a skill. The idea of IRL is to estimate the underlying reward function from the demonstrations. Formally, we have an MDP without the reward function, $MDP\backslash_R$, and a demonstration data set with N demonstrated trajectories $D_{demo} = \{\tau_i\}^N$, where each trajectory is a sequence of L state-action pairs $\tau_i = \{(s_j, a_j)\}^L$. The goal is to create an estimate $\hat{R}$ of the reward function that best describes the demonstrated behavior. In essence, IRL inverts the RL problem: rather than learning an optimal policy from demonstrations, potentially using the logged reward (s, a, r), IRL seeks to explain the demonstrated behavior by estimating the corresponding reward function.

IRL should estimate a reward function that generalizes from the demonstrated behavior. Hence, like other DL methods, IRL seeks to address the question: What happens if the agent were to perform a trajectory different from those in the data set? This is important because if we want the learning agent to improve upon the behavior seen in the data set, the agent must execute a trajectory that is different than the ones in the data set. However, most ML algorithms assume that the data is i.i.d. Consequently, addressing this question is challenging due to the problem of distributional shift.

Additionally, there can be many solutions to the reward function that describe the same behavior resulting in ambiguity. Some of these solutions, such as one that always returns the same reward, might accurately describe the observed behavior but be practically unusable. Due to this ambiguity, it is important to determine how to measure the performance of the estimated reward function. If the true reward function is available, we can directly measure the error between the predicted rewards for each state-action pair and the ground truth rewards. Alternatively, we can estimate a value function from the learned reward function and compare it to the real value function. However, the true reward function is often not accessible, which is precisely why IRL is employed in the first place. A more general way of measuring performance is to estimate a policy from the learned reward function, and assess its performance using the demonstration data set. The limitation of this method is the problem of how to evaluate the policy. Interacting with the environment is not possible because the true reward function is not available. Hence the only policy evaluation metric is to compare the policy predictions with the actions of the demonstrator for every state in the data set. However, this comparison is limited because even if the policy is only wrong in a single state, it can still result in compound errors at deployment. As a result, no single metric in IRL fully captures the performance of the learned reward function without access to the ground-truth. Furthermore, the design of the reward function's structure and the choice of its parameters are non-trivial. Using too many parameters can lead to overfitting and hinder generalization, while too few parameters may prevent the policy from effectively converging.

To obtain a unique reward function, IRL methods define additional optimization goals, the most common of which are maximum margin and maximum entropy. In the maximum margin setting, the reward function is the one that maximizes the difference between the best policy and all other policies. For example, [117] employs a maximum-margin-based IRL method to develop a policy for navigating rough terrain. In contrast, the maximum-entropy approach aims to find a distribution of policies that maximizes the entropy subject to certain constraints, such as feature matching, to ensure that the task's goals are reached. For instance, [118] uses the maximum entropy framework to learn a reward function for a driving task where there are multiple routes for the same destination in the demonstrations. The approach is later expanded to use deep learning in [119] for a table tennis task.

For discrete action spaces, IRL can be formulated as a classification problem, where for each state-action pair, the action is seen as the label for the state. The direct way to obtain a reward function is by estimating the action-value function which we explain in Section 3.2. This approach was used by [120] and later by [121]. However, this method assumes that the demonstrated state-action pairs are optimal.

Another approach to estimate the reward function involves assigning higher rewards to states encountered in the demonstrations, or similar states, than to states not found in the data set. For instance, in [53], the authors use demonstrations to estimate a Hidden

Markov Model (HMM) which determines the associated reward for each state. Similarly, [58] applies this approach to the task of balancing a pole using a robotic arm. In [122], the authors explore three different methods to parameterize reward functions from demonstrations and applied them to reaching, picking, and placing tasks. In [3, 73], the reward is proportional to how close the images captured by the learning agent at a certain timestamp are to the respective frame of the demonstration video.

Some approaches employ an actor-critic algorithm, where a third-party critic defines the reward function and provides feedback on the actor's actions. For example, [81] describes a method where the critic's policy is trained simultaneously with the actor's policy. Initially, the critic imposes constraints to guide the actor's behavior, but as the learning process progresses and the actor's competency improves, the decision-making authority gradually shifts from the critic to the actor.

Reward functions obtained with IRL, can encourage the achievement of sub-goals or milestones during the task execution, that are represented in the demonstrations. In [123, 124], the authors explore the intersection between RL and DL. Their results in two simulated domains show that reward-shaping methods can be more sample-efficient and robust against sub-optimal and inconsistent demonstrations than transfer learning algorithms.

3.4.5 Other Learning Methods

In this section, we discuss methods that complement or refine the previous learning methods to achieve greater accuracy, generalization, or robustness. Learning from demonstrations alone may not be sufficient to learn the task for all scenarios due to the limitations of the data set discussed in Section 3.3.7. Interacting with the environment allows the agent to collect extra data that it may use to refine the model.

The data set is unlikely to have a demonstration for all the possible environment states, particularly in high-dimensional spaces such as those encountered in real-world tasks. Hence, during the learning process, the agent must aim to generalize beyond the provided demonstrations. However, the agent may still not be able to generalize due to either limitation of the data set or the learning method. Generalizing in DL is especially difficult because the demonstrations are sequences of interactions where each action depends on the history of previous interactions, violating the i.i.d. assumption of supervised learning [75]. During training, the agent learns a sub-set distribution of the real task distribution. As a result, the agent learns from a subset of the real task's distribution, and during inference, it may encounter out-of-distribution states where it has no prior knowledge, potentially leading to unsafe actions. As discussed previously, some methods try to reduce this issue by explicitly reducing the distributional shift. However, constraining the agent to only operate within the demonstrated state distribution can limit the agent's performance. Consequently, methods that refine the agent's model through additional interactions with the

environment, allow the agent to learn missing information. Such an agent is safer than a random agent with no task knowledge performing random interactions.

3.4.5.1 Reinforcement Learning

RL models the problem as an MDP, as does DL. Instead of learning from a pre-existing data set of environment interactions, the RL agent interacts with the environment using its current policy and receives rewards based on these interactions. RL starts with a random policy and tunes its parameters toward maximizing the expected accumulated rewards. This approach can also refine parameters of a policy learned from demonstrations. An agent initialized with a demonstration-based policy is safer and converges faster, avoiding the risk of local minimum. This advantage was shown in [23] to learn Atari games. By exploring and learning from new state space regions, the agent enhances its generalization capabilities and robustness. However, when encountering unknown states, the agent may make mistakes, which can have catastrophic real-world consequences. Hence, such algorithms should be equipped with safety mechanisms. RL can also be used to train a policy from scratch in an online manner, which can then be used to generate the demonstration data set to train and evaluate the DL methods. This approach is limited to environments where online RL is possible and primarily serves to automatically generate demonstrations to evaluate the performance of DL methods. If online RL is viable, there is no need to train a secondary policy with DL. However, training a second policy can be beneficial if the RL agent does not act in real-time [70].

One of the most well-known application occurred when [25] trained an agent to play 'Go' to the extent of beating human experts. In this case, the agent is initially trained using demonstrations and then further refined through RL. In [125], Recurrent Neural Network (RNN)s are used to deal with POMDPs by incorporating past information to guide decision-making. Here, the agent is trained with RL, and demonstrations are used to determine which memories to retain.

3.4.5.2 Evolutionary Algorithms

Like RL, optimization algorithms can be employed to learn or refine a policy to replicate a behavior. Evolutionary Algorithms (EA), inspired by natural animal behaviors, are popular optimization methods used to find solutions to various problems.

EAs can be used to generate trajectories, with the most common being Particle Swarm Optimization (PSO) [126] and Ant Colony Optimization (ACO) [127]. These algorithms, inspired by the behaviors of birds and ants respectively, aim to find optimal solutions within a search space. They have been extended with demonstrations to improve the learning process. In [49], EAs are used to optimize agents in a soccer simulation. The possible solutions, represented as chromosomes consisting of if-then rules, were derived from demonstration data. These solutions were then evaluated using a performance-measuring

function, with the best-performing ones progressing to the next generation. In [128], PSO was utilized to find optimal behaviors, where demonstrations defined the initial behavior. Each particle modified its behavior by observing better-performing particles, with performance assessed by a fitness function. Additionally, in [129], Preference-Based Policy Learning (PPL) was employed to teach a robot to navigate.

3.4.5.3 Transfer Learning

Transfer Learning (TL) is a paradigm that leverages knowledge acquired from training on one task to facilitate learning a second task. Instead of training the second task from scratch, the knowledge from the first task can serve as a starting point, an optimization step, or, in rare cases, to perform the task completely. Formally, given a task T_s learned in the MDP domain D_s, the idea is to improve the learning of the goal task T_g in the MDP domain D_g using the knowledge of the previous task. TL is beneficial because it reduces the need to gather new samples; they can either be directly used by the new task, or the knowledge gained from training a policy on the original data can then be used to train the new task.

In DL, the demonstrations recorded for one task can be used to learn a second task. For instance, [130] employ transfer learning to enhance reward shaping (IRL). Reward shaping depends on prior knowledge, and transfer learning can leverage the knowledge of a policy learned for one task to shape rewards for a similar task. Additionally, [131] discovered that even if the agent overfits on the previous task, it can still adjust its weights sufficiently to recover and converge to the optimal weights of the second task. Their approach involves using graphs to identify previously encountered games and applying the relevant knowledge to the current game.

Alternatively, policies learned for a task can advise a learner on another similarity task. This knowledge can be transferred in the form of useful feature representations and specific parameter values. Moreover, an existing policy can serve as a foundation for developing a new policy for a different task. In [132], TL was used to learn a new soccer skill after learning a different one in a simulation. The experiments demonstrated that TL reduces the convergence time and achieves better performance.

3.4.5.4 Adaptive Learning

DL algorithms must consider that demonstration data sets are often incomplete, missing regions of the state space. A common approach to address this issue is to use pessimistic or conservative methods, which keep the policy close to the regions covered by the data set and avoid significantly different behaviors. However, these approaches can lead to sub-optimal estimations due to their strong restrictions. Additionally, agents can become stuck in certain states and repeat the same action over and over. To address this, policies should be adaptable, correcting poor choices. In [133], the authors train

uncertainty-adaptive policies that incorporate a belief parameter, estimated from the interaction history using an ensemble of networks. After a failed interaction, the history changes, altering the belief value. Consequently, this new belief value prompts the agent to select a different action, preventing it from getting stuck in states.

3.4.5.5 Active Learning

Active learning is a paradigm where the learning agent can query an expert for guidance, which is particularly useful in DL when the demonstration dataset is limited. When the agent encounters a state not represented in the demonstration data set, it might fail to choose the correct action, potentially leading to unsafe outcomes. Active learning addresses this issue by enabling the agent to request additional demonstrations from the teacher.

The approach to update the policy using both demonstration data and the teacher responses requires selecting which option to choose from at any given state. This can be achieved through a confidence score, as demonstrated in [42]. If the confidence for a given state-action pair is low, the learning agent queries the teacher for guidance. The learning agent progressively increases its confidence scores while obtaining a generalized policy, reducing the need for teacher queries over time. However, the main drawback of this approach is the additional investment required from the teacher, which may be unfeasible in some cases. In [134], the authors use active learning in human-robot cooperative tasks. For successful cooperation, the robot must be able to adapt its behavior to complement the human counterpart. Active learning is used after each round of interactions, with expert feedback provided via a graphical interface. The expert's feedback is provided by a graphical interface, recorded, and added to a database. This feedback is recorded, added to a database, and subsequently used to update the robot's policy. Results indicate that the robot's policy converged more smoothly using this method, particularly in tasks such as standing-up and assisted walking.

3.4.5.6 Generative Adversarial Imitation Learning (GAIL)

The authors of [64] introduced a model-free DL method called GAIL, which adapts the Generative Adversarial Networks (GAN) framework to the DL paradigm. In GAILs, the reward function is learned from the demonstration data and then used in RL for learning the policy. GANs consist of two neural networks: a generator and a discriminator. The generator creates synthetic data points, while the discriminator has to distinguish between the synthetic data and real data from the data set. The discriminator is trained to correctly identify whether each data point is real or generated, and the accuracy of these classifications adjusts the weights of both networks. The generator seeks to deceive the discriminator into misclassifying generated data as real, and it is rewarded when it succeeds. Conversely, the discriminator aims to avoid being fooled and is rewarded for accurately classifying data.

Improving the Robustness of Demonstration Learning

In GAILs, the learned policy π serves as the generator in the adversarial setup, while the discriminator D_ϕ is responsible for determining whether a given state-action pair originates from the demonstration data set or is produced by π. The goal of π is to improve its behavior to more closely replicate the demonstration data, thereby generating trajectories that deceive D_ϕ. To achieve this, D_ϕ is trained as a binary classifier to differentiate between real state-action pairs from the demonstration data and fake pairs generated by π. The generator π is rewarded for successfully confusing D_ϕ, and treating this reward as if it were an external analytically-unknown reward from the environment through RL.

In [135], the authors present two algorithms: one designed for offline GAIL and another for online GAIL. These algorithms improve upon existing state-of-the-art methods by enhancing the efficiency and effectiveness of the GAIL framework. In [136], a discriminator is trained to distinguish between two data sets with significant differences in quality. The discriminator is then used as a filter for the policy to avoid learning from sub-optimal data. In [137], a policy is trained to perform multiple small skills, where each skill is represented by a discriminator, a replay buffer, and a demonstration buffer. Each discriminator learns to differentiate between descriptors built from a pair of consecutive states sampled from either the replay or demonstration buffer. The reward is higher when the policy fools the discriminator into thinking the consecutive states were demonstrated.

Similar to GANs, GAIL suffer from severe sample inefficiency, which hinders the agent's ability to learn effectively from a limited number of interactions with the environment. This challenge has been addressed in subsequent research, such as in [138]. In [139] and [140], a discriminator is used to distinguish between generated and demonstration state-action pairs to learn multiple similar tasks at once. This method facilitates the generalization of the learned policy to additional, contextually similar tasks.

Similar to GAILs, [141] introduces a zero-sum game framework where a second player acts as an antagonist to perturb the transition probabilities of the protagonist. In this setup, the antagonist operates with a perturbation budget designed to optimize the protagonist's policy against the worst-case alpha percentile of transitions, thus providing safety guarantees. More recently, [142] proposes to use adversaries in place of the critic in actor-critic algorithms to improve sample efficiency.

3.4.5.7 Embedding Space

Learning from visual states requires applying a function $f(s)$ that extracts a set of N values, known as features, from the observations: $f(s) : S \rightarrow \mathbb{R}^N$, where N is the dimension of the embedding space. With deep learning, these features, and the corresponding embedding space, are typically estimated by applying a set of convolutions and sub-sampling operations to the input images. In DL, the states from the demonstration data can be used to explicitly learn an embedding space to extract features with specific characteristics.

Contrastive learning is a self-supervised learning paradigm that compares different images sharing a common signal to learn robust representations. It has been applied to multiple ML fields, with a notable example being image classification [143]. In such applications, contrastive learning creates an embedding space where images from the same class are pulled closer together, while images from different classes are pushed apart. A linear classifier [144] is then trained on top of the embedding space for a few epochs to classify images based on these learned features. In [9], this approach is adapted to DL, where demonstrations captured from multiple camera view points are used to learn a viewpoint-invariant embedding space. This learned embedding space facilitates the development of a viewpoint-invariant policy, thereby enhancing the policy's robustness to changes in camera position and different perspectives.

In [145], the representation learning part is decoupled from policy estimation, where the embedding space is estimated by contrasting images that appear close to each other in a sequence of frames. In [146], the different views are obtained through transformations applied to the original image. Alternatively, in [147] the representations are obtained by contrasting the similarity between the sequence of actions required to reach each contrasting state. In [3, 73], an embedding space for view-invariant features is estimated from a multi-view data set through triplet learning. Similarly, in [148] the authors train an encoder to estimate similar features for concurrent frames of multi-view synchronized videos. However, the criterion here is cycle consistency, where for two views, a data point is cycle-consistent if the nearest neighbor of its nearest neighbor is the point itself.

Siamese networks [149, 144] have been paired with contrastive learning, where each network is responsible for extracting features from different views of the same data. In [149], the different views are obtained through data augmentation and applied to motion simulation tasks. Other forms of obtaining different views from a single image are by using different image channels. For example, [4] converts images to the Lab color space, where the L and ab components are treated as two views of the image. Then contrastive loss is applied to learn an embedding space. Their findings indicate that increasing the number of views, such as by using additional channels or data augmentations, improves the quality of the learned features.

In [150] and [83], exploration is performed to estimate an embedding space without task-specific returns. In this approach, the embedding space is initially learned through exploration, and later refined for specific tasks using the reward functions of those tasks. The core idea is to encourage the embeddings to capture meaningful skills by maximizing the mutual information between state transitions and the associated embedding. This is achieved through a contrastive loss that approximates the lower bound of the mutual information, ensuring that the learned embeddings are informative about state transitions. For exploration, the agent is trained to maximize rewards that are proportional to the entropy of state transitions, which fosters exploration of diverse state regions. This two-

phase process allows the agent to develop a robust embedding space through exploration and then fine-tune it for particular tasks to optimize performance.

In [151], the authors use bisimulation to generate an embedding space where functionally identical states from different tasks are mapped to the same embedding. Using this embedding space, the learning agent is trained to adapt to different tasks that are functionally identical to previously learned tasks.

3.4.5.8 Sequence models

Sequence models learn from a series of transitions instead of a single transition, leveraging the history of past interactions to make more informed decisions By considering the sequential nature of data, these models utilize past experiences to improve their predictions and reduce the likelihood of deviating from the intended distribution of actions. In sequence models, the objective is to optimize the model over entire trajectories $\pi(\tau)$, by finding the best distribution of actions over these trajectories.

Sequence modeling with deep networks has evolved from LSTMs architectures to Transformer architectures with self-attention [16]. The latter have revolutionized many NLP tasks. Recently, they have been applied to RL by re-formulating it as a sequence modeling problem [14, 152]. These treat RL as a supervised learning paradigm that predicts action sequences from trajectories and task specification (e.g., target goal or returns), instead of traditionally learning Q-functions or policy gradients. In the DT [14], the agent is conditioned on past trajectories and the accumulated reward to be collected in the future, the returns-to-go (RTG). While the DT has demonstrated success across various tasks, its reliance on a fixed RTG sequence can limit its effectiveness in stochastic environments where the reward structure is not predetermined and must be specified by the user [11]. This requirement for manual reward specification can be challenging and may impact performance. In contrast, the Trajectory Transformer (TT) [152] employs the Transformer model both as a policy and as a model of the environment.

Subsequent research has proposed several methods to address issues with the DT. In [153], online learning is used to train the Transformer. Alternatively, [154] pre-trains the Transformer on large corpus of text which in turn increases performance on seemingly unrelated tasks. To address the DT's dependency on the RTG, several alternative methods have been proposed. One notable approach is presented in [155], where a value function is trained using the demonstration data set to replace the RTG sequence with state value predictions. While this method mitigates issues related to RTG sequences and environmental stochasticity, it introduces a dependency on the quality of the value function, which is constrained by the available demonstration data. Other works target the stochasticity problem of the DT. The method in [156], aims to estimate environmental stochasticity using a Transformer model to aid policy learning of the main Transformer.

While Transformers have achieved remarkable success due to their self-attention mechanism, their scalability is constrained by quadratic scaling relative to the size of the context window. In contrast, SSSMs [157] have gained attention for their linear scalability with respect to the sequence length. Notably, the Mamba architecture [158] merges the context-dependent reasoning of Transformers with the linear scalability of SSSMs through its selection mechanism. Mamba has demonstrated superiority over Transformers in numerous sequence processing tasks [159]. Like the Transformers before it, Mamba has potential to impact DL applications.

3.4.6 Multi Agent

Cooperation between agents is useful for robotic tasks and has been explored in RL. In RL, multi-agent systems often explore how agents can collaborate to achieve shared goals. Despite its importance, it has not received the same attention in DL. The shift from single-agent to multi-agent settings introduces significant complexity, as it requires not only learning individual policies but also managing interactions among multiple agents.

The state space must be expanded to include the status of all agents, as each agent's decisions are interdependent on the states of the others. Consequently, the reward function in these systems must also reflect the collective or individual goals of the agents. In a cooperative setting, the reward function is designed to encourage agents to work together to maximize a shared cumulative reward. Conversely, in a competitive setting, the reward function is structured so that one agent aims to maximize its own reward while strategically minimizing the rewards of other agents.

In [49], the team of robots works together to prevent the opposing team from scoring in a soccer game. In this approach, all robots share a common policy, which is updated collectively based on the actions of any individual agent. This method, while effective, essentially mirrors single-agent learning techniques. Alternatively, in [101] each of the agents learns different roles separately that in the end complement each other. Despite these advances, true multi-agent cooperative learning remains an open problem.

3.4.7 Learning Modifications

In this section, we will discuss modifications that have been employed by demonstration learning algorithms to tackle the problems that plague them.

Constraints serve as loss terms designed to impose specific characteristics on the learned model. These terms are either distribution constraints or action constraints. Most commonly, these constraints are employed to ensure that the model remains within the data distribution of the demonstration data set, thus mitigating distributional shift and its negative impacts. Constraint methods can be divided into two categories: direct and indirect.

Improving the Robustness of Demonstration Learning

Direct methods estimate the policy of the data set through BC π_{demo} and use it to constrain the learned policy π_θ, such that the divergence between the distributions of the two policies is below a threshold ε: $|\Delta_{\pi_\theta} - \Delta_{\pi_{demo}}| < \varepsilon$. However, a major drawback of direct methods is their dependence on the quality of the behavior policy. Estimating the behavior policy is difficult due to its reliance on the quality of the demonstration data set. Then, an incorrect behavior policy can cause methods that use it to constrain the learning process to fail. For example, a behavior policy that was learned from sub-optimal or incorrect demonstrations will constrain the policy learning method on such states, causing the policy to be too pessimistic which is undesirable. Furthermore, if more demonstrations become available, direct constraints require re-estimating the behavior policy. In [26], the algorithm estimates the behavior policy using a parametric generative model and constrains the learning policy to make sure it only chooses actions that the behavior policy would choose. Later in [160], the authors argue that since constraining the distribution does not take into account the quality of the actions, action constraining is superior. In [161], the authors applied a value penalty in the state-value function to improve performance.

Contrarily, indirect methods avoid the need to estimate a behavior policy and instead modify the learning objective and use samples from the data set. The most common approach is to minimize the Kullback-Leibler (KL) divergence between the distribution of the learned model and the distribution of the demonstration data set. In [162, 163], the algorithms estimate advantage functions to constrain the policy to reduce variance and increase sample efficiency. In [164], the authors add a regularizer based on BC by penalizing the difference between actions from the learned policy and the data set.

Alternatively, instead of imposing constraints, other methods can incentivize the model to have specific behaviors independent of the demonstrated data set. If the regularization term is $\mathbb{T}$, the learning objective is adjusted to incorporate the regularization term: $J'(\pi) = J(\pi) + \mathbb{T}$. Examples of regularization terms include penalizing the weights of the networks [23], and state entropy [83], to avoid over-fitting. In [165], an entropy regularization term is proposed to control the stochasticity of the policy and promote exploration, preventing premature convergence, improving robustness and stability. In [96], the method learns a lower bound of the true Q-function by adding a regularization term in its estimation.

Next, we can relax the constraints and regularization based on how much we trust the model. For instance, if we estimate the uncertainty of the model, we can reduce safety constraints in low-uncertainty state regions: $J'(\pi) = J(\pi) + \sigma\mathbb{T}$, where σ is a function which weights how much to emphasize the regularizing term. Entropy estimation methods can be applied as regularization terms such as clustering the state space in [83] or ensembles of models [166].

Table 3.2: Distinction between evaluation methods.

	Quantitative	Qualitative
Goal	Performance	Believability
Judgement	Objective	Subjective
Metric	Distance to goal	Subjective Analysis

3.5 Evaluation

Like other ML paradigms, DL methods require evaluation using specific metrics. These metrics are categorized into quantitative and qualitative, as outlined in Table 3.2. DL inherits metrics from RL and supervised learning, with common performance metrics including success rate, accumulated reward, and classification or regression error on a demonstration test set. However, some applications prioritize human-like behavior. Additionally, DL faces challenges in specifying hyper-parameter values.

Experiments are typically conducted on specific robots or simulators tailored to the method being tested. Due to the limited number of benchmarks, evaluations can be challenging. DL can use RL benchmark simulation environments, some of which include demonstration data sets. Even if demonstration data sets are not provided, a policy can be trained through RL to learn the simulation task and generate the necessary data sets. However, real-world benchmarking remains complex due to required hardware, varied backgrounds, and safety concerns. As a result, custom task environments and demonstration data sets are often created for each individual method, though some real-world demonstration data sets do exist. This section elaborates on the evaluation processes in DL.

3.5.1 Quantitative Evaluation

Quantitative evaluation metrics are specific to tasks where the performance of a policy can be directly measured. These approaches are divided into two categories: online and off-policy evaluation, with online evaluation being more common. After training the policy, a set of N online rollouts $\tau_1, ..., \tau_N$ are performed on the environment, and a metric is applied to these rollouts. A rollout is the sequence of transitions generated by selecting actions from the current policy and obtaining the next state and reward from the transition and reward functions, respectively: $\tau_i = (s_0, a_0, r_0), ..., (s_H, a_H, r_H)$, where H is the length of the trajectory. The most common measurement of online performance is the average success rate of the policy at performing the task $J(\pi) = \frac{\sum_{i=0}^{N} \mathbb{G}(\tau_i)}{N}$, where $\mathbb{G}$ is 1 if the trajectory completed the task and 0 otherwise. For example, in [104], the success is determined by whether the ball falls inside the cup or outside.

If a reward function is available, the performance of the model can be measured by the accumulated rewards of a rollout [23]: $J(\pi) = \sum_{t=0}^{H} R(s_t, a_t, s_{t+1})$. Similarly, the average or maximum accumulated rewards over multiple rollouts can also be used as performance metrics. In video games, a similar approach involves obtaining the score directly from the

environment and using it as an evaluation metric. For example, in [69], performance can be evaluated by the distance traveled.

Alternatively, if the goal is to closely imitate the teacher, the performance measurement can be the distance between the agent's actions and the teacher's actions [167]. This can be quantified through classification or regression error for discrete or continuous action spaces, respectively: $J(\pi) = \sum_{(s_i,a_i)\in D_{demo}} ||\pi(s_i) - a_i||$.

Time can also be used as an evaluation metric. This can include the time taken to execute a task or the time to converge during learning, such as the number of training steps. For instance, in [132], RoboCup soccer agents were simulated with the goal of keeping the ball away from the opposing team. Thus, the duration for which the ball is kept away from the enemy team can serve as a performance metric.

Another way to evaluate a policy's performance online is by measuring the safety of the method. Safety metrics can include the length of the episode (the number of transitions) or defining a set of safety constraints and counting the number of times the agent violated the constraints. This can be done by associating certain states with violation occurrences or by defining violations separately from the state space and checking for any violations after the agent executes an action. In [81], an advisor agent aims to prevent the main agent from violating constraints that could cause damage during learning. The challenge with online evaluation is that it relies on interactions with the environment, which can be dangerous. This risk makes it problematic to evaluate a policy while learning, because it might be too dangerous to deploy.

DL inherits challenges from ML, particularly in determining the optimal hyper-parameter values. Identifying these ideal values before or early in the training process saves time and computing resources by reducing the need for repeated experiments with different value sets. Furthermore, selecting appropriate hyper-parameters can help prevent dangerous interactions after deployment.

Off-Policy Evaluation (OPE) involves evaluating a policy using past experience, which can come from demonstration data sets, memories of online interactions, or a combination of both. In [168], various OPE methods are reviewed for selecting hyper-parameter values. Most DL methods do not use OPE and evaluate performance on a set of predefined hyper-parameters. The choice of these parameters is often influenced by state-of-the-art practices, small ablation studies, past methods, or random selection. Alternatively, some approaches train the model using multiple sets of hyper-parameter values to find better configurations.

OPE methods rely on a dataset of past interactions, D_{demo}, and an optimization objective, $J(\pi)$. Some OPE methods use the transition function $P(s_{t+1} \mid s_t, a_t)$ and the reward

function $R(s_t, a_t, s_{t+1})$ to evaluate the policy. If these are not available for the current problem, model-based methods estimate a dynamics model $\psi(s_t, a_t) \sim P(s_{t+1} \mid s_t, a_t)$ and a reward function through IRL $\hat{R}(s_t, a_t, s_{t+1})$. Using the transition function, reward function, and the actions selected by the current policy π, we can calculate the expected return: $J(\pi) = \mathbb{E}_{a_t \sim \pi(s_t), s_{t+1} \sim P(s_t, a_t)}[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1})]$.

Alternatively, instead of relying on a transition and reward function, we can estimate a state-action value function $Q(s, a)$ by minimizing the Bellman error using the dataset D_{demo} and the current policy π. In this approach, the OPE objective is to evaluate the expected accumulated rewards as given by the state-action value function, expressed as: $J(\pi) = \mathbb{E}_{(s,a) \sim D_{demo}}[Q(s, a)]$.

An extension of this evaluation method involves weighting the importance of each reward using importance sampling [169]. In this approach, the weights are derived by first estimating a policy from the demonstration data set through BC, denoted as π_{BC}. A common weighting scheme involves calculating the ratio of the product of the probabilities of the actions under the current policy to the product of the probabilities under the behavior policy, given by: $w = \frac{\prod_{t=0}^{H} \pi(a_t|s_t)}{\prod_{t=0}^{H} \pi_{BC}(a_t|s_t)}$. The weights can be used to regulate the original objective as such: $J(\pi) = \mathbb{E}_{a_t \sim \pi(s_t), s_{t+1} \sim P(s_t, a_t)}[w \sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1})]$.

3.5.2 Qualitative Evaluation

Qualitative metrics are used for tasks where the agent's behavior is more important than the performance it achieves. As previously mentioned, some applications require the agent to simulate human-like behavior, which can be challenging to quantify objectively. One approach to evaluating this believability is to involve multiple judges, each assessing and scoring the agent's behavior based on their own analysis. This method leverages diverse perspectives to gauge how convincingly the agent mimics human actions.

In [170], the authors explored various methods for generating controllers that best replicate human behavior. Although these methods were evaluated using the "Super Mario Bros" game, they are applicable to a range of other tasks. The evaluation was conducted qualitatively, where users were shown pairs of gameplay sequences-one performed by a trained controller and the other by a human. The users were asked to identify which gameplay was performed by a human for each pair.

3.6 Benchmarks

The demonstration data set is often gathered from rollouts of a policy that was trained using an RL algorithm. Alternatively, a human demonstrator can interact with the environment and generate demonstrations. Hence, online RL benchmarks can be employed for DL in these scenarios. We summarize the available benchmarks in Table 3.3.

Table 3.3: Summary of the available benchmarks for demonstration learning methods. RLU stands for RL Unplugged, and Sim. for Simulation.

Benchmark	Data Set	Action Space	State Space	Dynamics	Observability	Type
AI Habitat	Included	Discrete	Continuous	Deterministic	Full	Sim.
Adroit	D4RL	Continuous	Continuous	Deterministic	Full	Sim.
ALE	No	Discrete	Visual/Cont.	Stochastic	Full	Sim.
Atari	RLU	Discrete	Visual/Cont.	Stochastic	Full	Sim.
BSuite	No	Discrete	Continuous	Deterministic	Full	Sim.
DM Control	RLU	Continuous	Continuous	Both	Full	Sim.
DM Lab	No	Continuous	Visual/Cont.	Deterministic	Partial	Sim.
DM Locomotion	RLU	Continuous	Visual	Deterministic	Both	Sim.
Google Research Football	No	Discrete	Continuous	Deterministic	Full	Sim.
Gym-MuJoCo	D4RL	Continuous	Continuous	Deterministic	Full	Sim.
Gym-Retro	No	Discrete	Visual/Cont.	Stochastic	Full	Sim.
Meta-World	No	Continuous	Continuous	Stochastic	Full	Sim.
MineRL	Included	Both	Visual	Deterministic	Partial	Sim.
RWRL	No	Continuous	Continuous	Deterministic	Full	Sim.
RoboTurk	Included	Continuous	Continuous	Stochastic	Both	Real/Sim.

AI Habitat is a simulation platform designed for the research and development of embodied agents in an efficient 3D environment, with the goal of transferring learned skills to real-world applications. Another benchmark is BSuite [171], which offers a diverse set of experiments to evaluate the capabilities of learning methods. This library automates the evaluation process across nine varied environments. DeepMind's Control Suite [172] is a benchmark that provides a collection of RL environments built on the MuJoCo simulator. It includes tasks for controlling a variety of agents, such as Acrobot, Ball-in-Cup, Cart-Pole, Cheetah, Finger, Fish, Hopper, Humanoid, Manipulator, Pendulum, Point-Mass, Reacher, Swimmer, and Walker, with state spaces that are non-visual. DeepMind Lab [173] is a 3D learning environment built on the Quake III Arena game. It challenges agents with visual observations and complex 3D navigation and puzzle-solving tasks. One of the most widely used visual benchmarks is OpenAI's Gym, which includes environments for training agents to perform various tasks such as walking with a Humanoid agent, similar to DeepMind's control suite, using MuJoCo. Gym additionally offers environments for classic Atari games. Gym Retro extends the OpenAI Gym framework by providing environments for over 1000 classic games. Google Research Football [174] introduces a novel RL environment with a physics-based 3D football simulation. This benchmark allows agents to control either a single player or an entire team, making it suitable for multi-agent and multi-task learning scenarios.

Meta-World [175] is a benchmark designed for meta-RL and multi-task learning, featuring 50 distinct robotic manipulation tasks. Meta-learning algorithms can acquire new skills more quickly, by leveraging prior experience to learn how to learn. The Real-World Reinforcement Learning (RWRL) Suite [176] identifies nine challenges that prevent RL agents from being applied to the real-world. It also describes a framework and a set of environments to evaluate the method's potential applicability to the real-world.

In practical applications, such as the real-world, we do not have access to a policy. In such scenarios, the data might come from non-Markovian agents, such as humans, which

differs significantly from the data generated by online RL policies Consequently, the data sets generated by the online RL policies are not representative of practical applications. The field is continuously developing a novel benchmarks with varying properties to address these challenges. The demonstration data sets and environments should take into account the properties explained in Section 3.3. Continuous action and state spaces are generally more challenging than discrete ones, since it is impractical to explore every possible state and action. Therefore, an agent that learns in such domains is obligated to generalize beyond the visited data. Furthermore, visual observations are considered more challenging than non-visual observations. Another common real-world issue is the occlusion of state representations, which results in partially observable states and turns the problem into a POMDP. This scenario is significantly harder and closely mirrors real-world situations where complete state information is rarely available. Lastly, real-world environments are often stochastic rather than deterministic. Thus, environments that incorporate stochastic transition functions are generally more desirable for evaluating methods.

The D4RL benchmark [5] provides demonstration data sets for OpenAI Gym's MuJoCo tasks. These data sets come from human demonstrations and vary in quality levels such as random, medium, and expert. The RL Unplugged benchmark [177] provides demonstration data set for four different suites: Arcade Learning Environment (ALE) [178], DeepMind Control Suite [172], DeepMind Locomotion [179], and the Real-World Reinforcement Learning Suite [176]. The disadvantage is that the data sets of RL Unplugged are obtained from online RL policies. This means they lack the desirable non-Markovian properties which arise from human-generated data.

MineRL [180] is a benchmark built on top of the game Minecraft. The benchmark includes a wide array of tasks, including finding a cave, creating a pen, making a waterfall, building a house, and locating a diamond. These tasks are characterized by their sequential nature, which presents significant challenges for learning algorithms, such as sparse rewards and long horizons. The benchmark provides a large demonstration learning data set with over 60 million frames of recorded human gameplay.

For real robots, RoboTurk [181] provides the largest collection of demonstration data sets of a variety of real-world robots performing diverse manipulation tasks. The data set is increased through crowd-sourcing. It also provides a framework for generating demonstrations for a personal robot controlled through teleoperation using a phone.

The Deep Off-Policy Evaluation (DOPE) benchmark [182] offers a standardized framework for the evaluation and comparison of various OPE algorithms. Because it is not a benchmark to evaluate demonstration learning methods, it is not included in Table 3.3. This benchmark is structured into two main suites: DOPE RL Unplugged and DOPE D4RL. DOPE allows the selection of different learning objectives, such as ensuring the

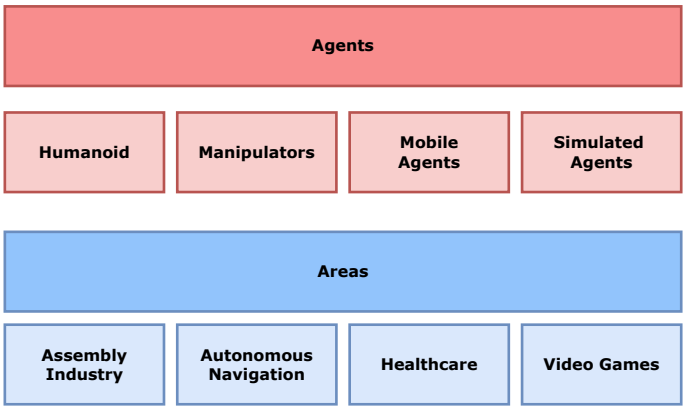


Figure 3.7: Agents and areas DL methods can be applied to.

estimated value of a policy is as close as possible to the true value, selecting the best possible policy from a set of policies, and hyper-parameter tuning with early stopping.

In Table 3.4, we categorize various demonstration learning methods. The table organizes the methods based on key attributes: the demonstration technique used to obtain the data set; the type of input data used to train the method; the learning objective (e.g., policy learning); the type of inference type (classification, regression, or both); the metrics used to evaluate the method; and the tasks the method was applied to.

3.7 Applications

DL approaches have been successfully applied to a wide range of domains, as illustrated in Fig. 3.7. This section provides a detailed exploration of these applications, highlighting a selection of relevant examples. The primary applications of DL are robotics and simulation environments. RL can learn complex skills but at the cost of interaction with the environment which can be dangerous or expensive in real-world scenarios. Moreover, the sample inefficiency of RL requires millions of interactions even for simple tasks in the real-world. For this reason, DL can be critical for applying policy learning algorithms to real-world settings.

3.7.1 Assistive Robots

DL can be applied to assistive robots which aim to help humans in their daily tasks. This application presents unique challenges, because these robots must be capable of adapting to a wide range of scenarios that the untrained human may require. In [134], the authors adapt the training process to account for the dynamic and evolving behavior of humans. Robots may be used to interact with humans [66] or help with social or mental problems [217]. Assistive robots likely have to imitate how a human would assist, which is something that DL can provide over RL.

3.7.2 Autonomous Navigation

Self-driving cars have been a central focus of ML research in recent years. The goal is to use the data captured from a wide range of sensors to control the vehicle safely in complex environments. Given the immense complexity and high dimensionality of the problem, RL represents a viable solution by learning from interactions. However, the unsafe characteristics of RL represent an obstacle, as the consequences of unsafe driving are catastrophic. DL avoids this problem since the agent does not have to interact with the environment to learn.

Early research in [218] proposes a method for learning to fly an aircraft from demonstrations recorded via teleoperation using IRL. DL has also been applied successfully to autonomous aerial navigation, such as drones for their potential for the delivery of goods, and helicopters. In [47] and in [29], data of a robot helicopter flight, using a joystick, was recorded and used to train an autonomous helicopter agent through RL and apprenticeship learning, respectively. With advances in sensors for capturing data, DL has been increasingly used to learn control policies. Beyond aerial applications, DL has also been employed for the locomotion of bipedal and quadrupedal robots. The demonstrations for training such robots are obtained either by teleoperation or observation. Some works have applied DL to driving tasks [183, 75, 118]. [219] collected a dataset for navigation tasks using a camera mounted on a robot. In [74], the authors evaluated active learning techniques based on demonstrations in both simulated and real-world environments. Similarly, [100] utilized a mobile robot to gather environmental data and select appropriate algorithms from a pre-defined database based on situational context. Full autonomous control of a vehicle in the real-world is such a complex problem that it likely can not be solved using a single field of ML. Nonetheless, DL offers tremendous potential to learn skills from data, without having to interact with the dangerous environment of driving in the real-world.

RobotCar [220] and BDD100K [221] are large data sets containing real-life driving demonstrations in the form of videos. Demonstrations have been used towards autonomous driving in [222, 201, 21, 187]. The safety of DL can be used to employ RL methods to autonomous driving. For example, in [205, 65, 85] the agent can not violate safety constraints learned from demonstrations.

3.7.3 Dance

Audio processing for dance generation encompasses various methodologies that leverage distinct techniques for feature extraction and sequence modelling. The most common practice is to use the Librosa [223] library to extract audio features, though various differ in their choices of sampling rate and the types of features used to represent audio vectors. Ablations in [224] determined the more useful audio features for dance generation and highlighted the importance of feature normalization in the process. In contrast, the

method in [225] deviates from using Librosa, opting instead for features extracted from the Jukebox [226] pre-trained model.

Early attempts at automatic audio-to-dance conversion using deep learning, utilized LSTM networks for sequence modelling [227] and realized a sole LSTM could not converge. Instead of using LSTMs, in [228], the authors train two embedding spaces, where dances or music from the same style or genre, respectively, are mapped to similar embedding vectors. A CNN then receives style and rhythm embedding to generate dance.

GANs [229] have also been employed for training generators conditioned on music to produce quality dances. In [230] the method incorporates stability concerns by augmenting the discriminator to penalize unstable poses. The method in [231] also utilizes GANs but structures the architectures with LSTM layers to incorporate sequentially.

The advent of Transformer models [16] revolutionized sequence modelling, offering parallelized input processing and a masked attention mechanism for inter-sequence element contributions. The method in [15] combines adversarial learning with Transformer models. It employs two Transformer models, one conditioned on music to predict the correct pose on the beats, another conditioned on consecutive poses to predict the motion curve parameters between the two poses. This paper also introduced the PhantomDance data set. In [232] the authors present AIST++ data set, an enhanced version of AIST with annotations of the dancer's poses. Additionally, they employed a three-Transformer framework that predicts the remaining half of the choreography given an audio sequence and initial choreography. The authors of [233] deal with the dimensionality and precision dance pose generation issues by discretizing the poses into two finite codebooks. These codebooks are estimated using variational auto-encoders. A Transformer model then learns the mapping between the audio sequences to a pose in each codebook.

3.7.4 Manipulators

DL has been applied to manipulators in manufacturing applications since the 1980s. Training manipulators through demonstrations, usually through kinesthetic teaching, is more efficient than hard-coding their behavior. One prominent application of this technology is in assistive and healthcare robotics, where robots are designed to help humans with various tasks. ML methods are used for such tasks because the robot must perform effectively in diverse environments. Training a policy to generalize to new scenarios is more practical than coding every possible situation manually. Additionally, since assistive robots operate in close proximity to humans, there is a greater emphasis on safety, which is enhanced through the convergence and stability guarantees of learned policies compared to hard-coded behaviors. Furthermore, effective collaboration requires the desired robotic movements to be similar to those of humans. Moreover, effective collaboration with humans necessitates that robotic movements closely mimic human actions, making DL a good approach for developing such behaviors in the policies.

Several studies have utilized demonstrations for learning robotic grasping [202, 196]. Examples of DL methods applied to such robots include handing over objects to humans in [234], where the authors encode coordination from demonstration data to enable humans to transfer control of objects to other entities. Additionally, a model-based algorithm [215] learns a variety of manipulation skills.

Policies for learning manipulation tasks have the potential to replace humans in healthcare. However, this raises significant safety concerns, as exploration required by RL is impractical [235]. In [22], the authors train multiple robotic arms to perform surgical tasks using trajectories learned from demonstrations. Later, [236] applies a model-based method for the treatment of lung cancer. The MIMIC-III dataset [237], which contains 60,000 ICU records, has been used for drug recommendations [210] and sepsis treatment [212] by leveraging demonstrated treatment processes from the data set. The automation of healthcare through DL has the potential to optimize procedures and improve outcomes.

3.7.5 Humanoid Robots

Humanoid robots are perhaps the most obvious application of DL. These are robots with a structure similar to humans whose goal is to perform tasks typically carried out by people. Since the required tasks are already performed by humans, DL is a fitting approach. Demonstrations can be captured using a sensor suit and converted into joint values for the robot. The tasks vary from utilizing only part of the robot [238] to engaging the entire humanoid body [239, 240].

In [31], a humanoid robot was trained from human demonstrations captured by sensors placed directly on the human body to perform reaching and drawing movements with one arm, as well as tennis swings. In [54], a humanoid robot learned to imitate human arm gestures and was tested in a turn-taking gesture game. A human teleoperated a humanoid robot in [29], where Virtual Reality technology was used to convert the operator's arm and hand motions into those of the robot to learn a manipulation policy. In [3], an embedding space was learned from demonstrations, allowing the humanoid robot to replicate human movements projected into this space. In human-robot interaction, beyond learning task control, the robot must also learn where to focus its attention.

3.7.6 Video Games

ML methods, particularly RL, have been successfully applied to video games. Key applications include creating agents to learn and master games or generating controllers for non-playable characters. However, video games often present challenges such as sparse rewards and high-dimensional spaces, which can hinder policy convergence during exploration. Additionally, non-playable characters often need to exhibit human-like behavior [241]. Therefore, DL serves as a bridge to reach these goals that RL struggles to achieve.

DL has been applied to racing games such as Mario Bros in [170] and [69]. The ALE [178] offers a platform for benchmarking algorithms on Atari games. Approaches such as [23] have used this environment to evaluate their method and compare them to other state-of-the-art methods for Atari games. While these examples involve relatively simple problems with small state and action spaces, they demonstrate potential that can be extended to more complex games in the future.

The variety of genres and the growing complexity of video game domains present challenges increasingly akin to real-world difficulties. Methods often rely on simulation to validate their correctness due to safety concerns associated with real-world deployment. However, creating effective simulators is challenging and often yields inconclusive results, as the real-world remains more complex. Video games offer a diverse array of problems for DL methods to tackle. Addressing progressively difficult problems in video games provides a pathway for validation that is closer to real-world scenarios.

3.8 Advantages and Disadvantages of DL

Different DL algorithms bring different advantages and disadvantages, which are summarized in Table 3.5. This section will explore the overall advantages and disadvantages of DL methods.

3.8.1 Advantages

The most significant advantage of DL over other paradigms is its potential to eliminate the need for expert programming. Although the field has not yet reached this goal, the aim is to train an agent to perform tasks through demonstrations alone. This approach enhances adaptability by enabling agents to learn behaviors from demonstrations rather than being programmed on static instructions. Demonstrating a manipulation task is often much easier than detailing every step a robot must take to complete it. Additionally, DL allows agents to learn from various external sources, including humans or other agents with different types of hardware.

Additionally, the paradigm is highly data efficient, especially compared to RL which is severely data-inefficient. Several approaches use a small number of demonstrations. In contrast, RL rely on trial-and-error interactions to learn optimal policies, resulting in numerous failed attempts. DL allows the agent to learn the task by providing the correct choices for the input states, thus solving high-dimensionality problems and effectively addressing the curse of dimensionality that plagues ML problems suffer from. This property allows the paradigm to solve high-dimensionality problems and effectively addresses the curse of dimensionality ML problems suffer from. Furthermore, RL agents' policies can converge to local minima with suboptimal performance. In DL, agents learn the behavior demonstrated to them, offering performance guarantees for the policies learned. Then,

DL can be combined with other ML fields, such as RL, to further enhance the learned methods.

Lastly, one of the main limitations of RL is that it is difficult to apply it to real-world problems. The RL agents learn through trial-and-error interactions, which is ideal for simulation environments. However, in the real-world, errors can have serious consequences, such as collisions that may severely damage a robot, hindering or preventing learning altogether. DL is extremely valuable in applications where interaction is impractical, expensive, or dangerous. DL leverages demonstration data sets to provide interaction data, allowing the agent to learn without directly interacting with the environment. This approach ensures much safer learning, making it applicable to real-world problems. Even when it is safe to interact with the environment, using demonstrations can speed up convergence and improve generalization in complex domains. After learning a policy from demonstrations, the policy can still be improved with online interactions. Since the policy already possesses task knowledge, it interacts with the environment more safely compared to a random policy in RL.

3.8.2 Disadvantages

While DL is appealing and addresses many issues associated with RL, it presents challenges when working with limited data. Technically, any off-policy RL algorithm can be used to learn from a demonstration data set. However, these algorithms were originally designed for online RL, where the agent interacts with the environment and can correct its mistakes. As a result, these methods often fall short when applied to limited data.

The disadvantages of DL are primarily related to the data set, as discussed in Section 3.3.7. The quality of the policy is directly dependent on the quality of the demonstrations within the data set. Sub-optimal or incorrect demonstrations can hinder or even prevent the policy from converging to adequate behavior. To address this issue, existing solutions aim to identify and filter out these sub-optimal demonstrations.

Additionally, collecting demonstrations is expensive, and the data set may be insufficient, often lacking coverage of all possible state-action pairs. Creating the environment with the sensors, demonstrating, collecting the data, and creating adequate embodiment and record mappings for the data set, are all non-trivial steps. Then, the distributional shift discussed in previous sections can severely hinder the performance of the model. The demonstration data set represents only a subset of the real MDP. Unless the data set is optimal the agent can not generalize to cover the entire problem. Hence, when queried in out-of-distribution states, it will likely fail. Restricting the agent to in-distribution states to avoid queries for actions it cannot generalize to from limited data is challenging, and most methods must address this issue explicitly.

3.9 Future Directions

In this section we will discuss the open research problems in the field of DL.

3.9.1 Benchmarking

The goal of DL is to teach a real-world agent a task from demonstrations. However, most methods are evaluated in simulation environments. There are few standardized benchmarking environments and data sets, particularly for real-world environments. Standardization in any research area is beneficial as it facilitates the comparison of different methods for the same problem, and DL is no exception. The lack of a real-world DL benchmark is an open issue in the field, limiting its progression and applicability to real-world tasks.

Current DL methods are often evaluated using a limited range of RL environments, frequently creating their own demonstration data set. A robust method should be able to generalize across various environments and perform tasks with high repeatability. Additionally, the method should be computationally efficient and, in some cases, capable of producing behavior that appears human-like.

Although some environments provide corresponding demonstration data sets, methods that generate their own data sets create uncertainty about the method's quality. The number of demonstrations in the data set significantly impacts the method's performance. In real-world scenarios, collecting demonstrations is challenging, resulting in smaller data sets. However, some methods utilize large numbers of demonstrations, offering greater coverage of the state-action space, which is a distinct advantage over methods with smaller data sets. Measuring and comparing the quality of demonstrations across different data sets is difficult. It is reasonable to assume that two different data sets will vary in quality. Consequently, the performance of a method is directly proportional to the quality of the demonstrations used.

Existing stochastic environments are very limited. Additionally, non-stationary environments, which are very common in the real-world, are also scarce, with the notable exception of the RWRL suite. Additionally, there is a lack of available multi-agent environments and data sets. To address these gaps, future research should focus on creating more standardized environments, demonstration data sets, and evaluation metrics, and encourage their usage.

3.9.2 Context Problem

The context encompasses the set of characteristics that define the environment, such as background objects, illumination, and camera positioning. Most DL methods are designed to learn a policy for a single, fixed context. When any of these contextual elements change, they form a new context, and the policy trained for the original context is often

unable to perform effectively in the new one. This issue is frequently overlooked because evaluation is typically conducted in simulation or controlled environments where the context remains consistent with the training conditions, allowing the policy to succeed. However, this limitation severely affects the scalability of the policy for real-world applications, where context changes can be drastic. Some works have tackled the problem of learning from demonstrations captured from multiple contexts [242]. In [57, 105], a translation of the recorded observations from the demonstrator's context to the context of the learner is used. However, their approach only works for simple tasks which can be represented solely by the first and the final observations. Recently [140] encoded a demonstration and provided the embedding as an extra signal for the policy to perform in a novel context. However, this requires a demonstration for the given context to be available and user interaction to define which demonstration to encode.

Unsupervised learning has been used to learn policies from unlabelled visual demonstrations. Previous methods have made use of multiple modalities to obtain rich embeddings. In [148] an encoder is trained to estimate similar features for concurrent frames of multi-view synchronized videos. Here the criterion is cycle-consistency, where for two views, a data point is cycle-consistent if the nearest neighbour of its nearest neighbour is the point itself.

Triplet learning has been used to estimate an embedding space where similar observations are closer in space than time-distant ones. This method was applied to DL using Time Contrastive Networks (TCN) [3], where synchronized multi-view demonstration videos are used to estimate an embedding space where observations from different viewpoints but identical timestamps generate identical features. Later, in [73] the encoder was enhanced with multiple levels of attention to further focus on viewpoint invariant features. In [214], they explicitly disentangle state and viewpoint features from the feature vector using an encoder-decoder architecture combined with a permutation loss. The main drawback of the previous methods is how computationally intensive they are to train because they rely on organizing the data set into triplets (sets of anchor, positive and negative images). Sampling a negative image for an anchor-positive pair is not trivial. On the one hand, sampling every possible combination leads to an extensive data set to process and will include weak negatives (the anchor-negative distance is easily greater than the anchor-positive distance). On the other hand, sampling hard negatives (anchor-negative distance is close to the anchor-positive distance) is a complex task that comes with downsides such as the frequent evaluation and re-sampling of negatives [243].

CL compares different images sharing a common signal to learn representations in a self-supervised fashion. It has been applied to multiple ML fields, most notably image classification [143], where the embedding space is robustly obtained in a self-supervised manner by bringing images from the same class closer in the space. Siamese networks [149] have been paired with CL where each network outputs the features of a different

view. In [149], the different views are obtained through data augmentation, and applied to motion simulation tasks.

Other ways to obtain different views from a single image are by using different channels. In Contrastive Multiview Coding (CMC) [4], the images are converted into the Lab color space, where the L and ab components are treated as two views of the image. Then contrastive loss is applied to learn an embedding space. Additionally, they show that increasing the number of views leads to better features.

3.9.3 Goal Specification

DL methods focus on learning one specific task at a time. Pick-and-place tasks, for example, involve picking up an object from one location and placing it somewhere else. In the literature, picking a different type of object or placing it in a different location is often considered a distinct task from the original [140], despite both being variations of the same fundamental task. As a result, the effort required to train an agent to handle multiple goals scales linearly with the number of distinct goals, as a separate policy must be developed for each one. This approach overlooks the fact that these tasks share redundancy and common information. Future research should aim to develop policies that can scale and adapt to different goals specified by the input, rather than requiring a new policy for each individual goal.

3.9.4 General DL Framework

The end goal of DL is to enable the training of an agent without the need for expert programming knowledge. Ideally, an agent should be able to learn the task solely through demonstrations. However, current approaches still require the design of algorithms for feature extraction, reward specification based on the type of agent and task, and policy derivation algorithm. A learning framework that could be universally applied to any task would allow a teacher to train an agent solely by demonstrating the task.

3.9.5 General Feature Extraction

All DL approaches rely on the quality of the data set. In high-dimensional environments, images are the most practical state representations. Learning from images requires the extraction of quality features, which currently vary depending on the specific task. Creating a general feature extractor framework would eliminate the need for engineered feature extraction frameworks for each task. This is one of the open problems preventing the creation of a general DL algorithm.

3.9.6 Generalization

The policies should be able to select the correct action for any possible state. However, since policies are estimated from a demonstration data set, they are inherently limited

to the state-action distribution present in that data set. Generalization is the ability of a policy to make correct decisions in unseen scenarios based on previous experiences. To improve generalization, several techniques are employed: some approaches expand the data set through methods such as data augmentation [78, 77], while others fine-tune the policy using online data [188]. Additionally, some methods restrict the policy to in-distribution states to prevent it from being queried for actions it has not been trained on [65].

3.9.7 Hyper-parameter Selection

Hyper-parameters are a critical aspect of most ML approaches and must be specified in advance. Examples include determining the number of hidden layers in a neural network and the number of neurons in each layer. The ideal values for the hyper-parameters are often difficult to estimate and are usually determined through trial and error. As noted by [244], many studies present results based on carefully tuned hyper-parameters for a limited set of tasks, which are insufficient for evaluating the robustness of methods. Instead, methods should be trained multiple times with varying hyper-parameter configurations, and the resulting policies should be assessed using the proposed metric, 'Expected Validation Performance'.

Performing policy rollouts in real-world settings is generally challenging due to costs and safety concerns, particularly when evaluating policies that are still being trained. Evaluating a policy before completing its training to determine the optimal hyper-parameter values would allow for adjustments without incurring the full cost of training a complete policy, thereby saving time and resources. This is another factor limiting DL to simulators. Off-policy methods aim to evaluate a policy from past interactions, without requiring new interaction with the environment. For example, [245] conducted a study evaluating thirty-three different OPE methods by measuring the distance between the estimated policy value compared to the true policy value. The study concluded that evaluation with the state-action function outperforms the remaining methods. To help accelerate the development of OPE methods, [182] propose the DOPE benchmark. The available options are using inaccurate OPE methods or training the model for a fixed number of steps and adjusting the values based on the early results. While hyper-parameter selection is a common challenge across various machine learning paradigms, DL needs to overcome this problem to truly reduce the requirement of expert programming. We believe that developing general, accurate, and efficient OPE methods is crucial for the growth of the field and for expanding its applications to real-world scenarios.

3.9.8 Long-Horizon Tasks

Offline RL relies on a reward function to estimate policies. More frequent rewards make it easier to develop effective policies, but creating a reward function that provides feedback for every step is often challenging, especially for complex tasks. As a result, DL has been restricted to simple tasks that complete within a few steps and require minimal interac-

tions. Long-horizon tasks are more common in real-world scenarios, which consist of either a series of small interactions or a single extended and complex interaction. These tasks require a greater number of steps, leading to a more extensive state space for the agent to explore and learn from. Although learning to handle long-horizon tasks remains an open challenge, some methods have achieved partial success in addressing these complex problems.

The method in [193] detects intersections among demonstrations in the data set at certain states to generalize new trajectories. [197, 213] demonstrate that complex tasks can be decomposed into sub-tasks, which are learned from the demonstration data set. The agent receives a reward upon completing each sub-task. [206] estimates a state distribution to ensure that these sub-goals correspond to reachable states. A particularly promising approach involves using goal-conditioned policies across multiple layers of hierarchy in RL [246]. Additionally, some methods generate sequences of sub-goals through a divide-and-conquer approach [247].

Another approach is to learn an embedding space of skills from unstructured demonstrations [211] and then train the policy on the embedding space. In [211], skills are defined as useful sequences of actions, and an embedding space of these skills is learned from unstructured demonstrations. They then train a RL policy on the embedding space using a hierarchical model where the high-level component generates embeddings of skills which the decoder then converts into sequences of actions. A particularly promising approach was proposed, using goal-conditioned policies at multiple layers of hierarchy for RL [246]. Alternatively, other methods generate sequences of sub-goals with a divide-and-conquer approach [247].

Hierarchical approaches learn a high-level planner and a low-level controller [194, 213]. The high-level planner identifies a sequence of sub-goal states that guide the agent towards the main task goal, while the low-level controller is conditioned to achieve these sub-goals. This extra guidance helps the agent learn in sparse reward environments and long horizon tasks. Conditioning RL and DL approaches on goal observations improves sample efficiency. In [206] the high-level policy is encouraged to predict intermediate states between the current state and the goal state.

RUDDER [208] redistributes the reward by identifying key steps in the demonstrations and increasing the reward of the respective transition. However, RUDDER uses LSTMs to predict the associated reward and models require many demonstrations to generalize well. Because of this, Align-RUDDER replaces the LSTM model with a profile model adapted from bio-informatics. This model can be estimated with very few sequences. In Align-RUDDER, a sequence is aligned with the profile, providing an alignment metric that indicates how well the sequence matches the model. The reward for a transition is the difference between the alignment values of the sequence with and without the transition.

3.9.9 Multi-Agent DL

Most DL research is focused on single-agent learning, while real-world applications often require agents to interact and cooperate with one another. Although there has been some exploration of multi-agent cooperation and competition, as discussed in the relevant section, the applications of the methods are limited. Hence, multi-agent DL remains an open problem.

3.9.10 Learning from Sub-Optimal Data

DL heavily relies on the quality of the demonstration data set. In many cases, the data is sub-optimal due to various factors. Sensor data often contains noise and errors, which can increase the distributional shift between the estimated models and the demonstrated behavior. Some works address inadequate data through identification and removal of poor samples [48] or by correcting the data [204]. Collecting demonstrations is generally expensive, resulting in small data sets. To mitigate this, some methods use demonstrations from other tasks, enhancing the policy's generalization and adaptability. For instance, [190] gathers data sets from multiple tasks sharing the same state and action spaces but differing in reward functions and dynamics. An encoder is trained to encode trajectories into an embedding space, and the policy is conditioned on a task embedding alongside the state. However, using data sets from other tasks can increase the distributional shift. To address this, [186] proposes a method to learn policies from multiple task data sets without causing a distributional shift, by relabeling transitions from the other tasks' data sets.

Additionally, the estimated distribution of the policy may differ significantly from the demonstrator's distribution. A larger distributional shift typically results in poorer generalization of the policy. In [94], the authors address this mismatch by deriving a tractable bound on the distributional shift between the offline data set and the learned policy. This bound is then used as an extra regularization parameter in the optimization step.

Offline RL requires reward specification which is expensive in the real-world scenarios. Beyond IRL, unsupervised techniques offer a promising direction by leveraging unlabeled data. For instance, [80] demonstrates that a policy can be effectively learned by combining large sets of unlabeled data with a smaller set of labeled data. Additionally, [192] introduces a downstream reward labeling method applied to unlabeled datasets.

3.9.11 Safety Concerns

RL has gained significant attention in recent years due to its diverse applications. However, its trial-and-error nature can pose safety risks. In contrast, DL allows the agent to learn policies from a dataset without interacting with the environment, remaining safe while learning. In offline RL, the goal is to maximize the expected accumulated rewards, while in DL the goal is to align action choices with those of the demonstrator. However,

these goals don't take into account safety concerns. Additionally, the limitations of the data set can result in sub-par policies and the agent can still reach an unsafe state if it exists the distribution of the data set. The demonstrator is unlikely to cover all potential safety-critical scenarios, and the reward function often does not prioritize safety. Safety is paramount in real-world applications, particularly in human-robot interaction settings [248, 134].

One approach to safe RL is to maintain the agent within a safe distribution of states. For example, [249] proposes learning a manifold that captures natural variations in the environment and uses a secondary policy to guide the agent back into the distribution of visited states. The method in [250] introduces an advantage-based mechanism to determine when the recovery policy should intervene. Existing RL algorithms address safety by specifying constraints that the agent cannot violate during execution [86, 65, 85, 203]. For instance, [251] proposes learning a barrier function that constrains the agent's policy to remain within a set of states that do not violate constraints. The method proposed in [189] initially finds a reward-optimal policy, which is then projected onto the feasible set of policies that satisfy the cost constraints. However, these constraints are often task-specific and agent-specific, making it impractical to specify all the constraints for every situation. Alternatively, [252] proposes a zero-sum game where a second player perturbs the agent's transition probabilities to optimize the worst-case transitions to produce a more robust policy. Methods in [66, 184] require access to sets of safe and unsafe states to train the agent safely.

Active learning can address uncertainty. The DAgger framework trains a student policy by allowing it to query an expert policy [75], where it learns from data generated by both itself and the expert. Before each interaction with the environment, a decision rule decides whether to use the student's or the expert's policy. Since selecting the student's action early on can be unsafe, SafeDAgger [209] introduces a criterion that only allows the student to act if the difference between its action and the expert's action is below a threshold. EnsembleDAgger [191] further refines this approach by requiring the prediction of an ensemble of networks to be below a threshold for the student to act. However, these algorithms depend on the availability of an expert policy.

Some methods leverage expert demonstration data sets to combine the advantages of RL and DL. For instance, [97] proposes learning a Lyapunov function to ensure the agent's policy remains within the distribution of states from the data set. In [253], the authors use a pre-existing expert policy to filter the agent's action if it differs from the expert's.

3.10 Conclusion

We have presented a comprehensive survey on DL. DL reduces the programming overhead by teaching the agent tasks through demonstrations. The paradigm comprises two

main phases. The first phase involves collecting and building the demonstration data set, with key considerations including selecting the demonstrator, recording the demonstrations, representing the data, and addressing common data set limitations. The second phase focuses on extracting the behavior from the data set and training the agent accordingly.

The main challenge in DL is the generalization to unseen scenarios. Demonstrations only cover a subset of the distribution, and direct imitation through BC learns this limited distribution. When the agent encounters out-of-distribution cases, it may not know how to respond, potentially leading to catastrophic real-world consequences. To address this, offline RL methods aim to reduce the distributional shift. Various strategies and mitigations were discussed, including model learning methods that enable the agent to generate new transitions without interacting with the environment. Additionally, methods for refining the behavior were explored, such as trial-and-error interactions using RL, skill transfer through transfer learning, active teacher querying, and optimizing behavior with EAs.

DL is predominantly used across various types of robots and video games, with potential applications extending to diverse domains such as healthcare and industry. The main advantages of DL include reduced reliance on expert programming, greater data efficiency than RL, and enhanced safety during the learning process. Despite these benefits, DL requires a good framework to create high-quality demonstration data sets and accurately estimate the policy.

DL is a promising area within the broader field of ML. This survey has identified several key research challenges, highlighting that the development of robust benchmarks and high-quality data sets are crucial for the field's advancement. As noted in [37] much of the success of deep learning is due to large data sets. Even recent applications still utilize relatively old methods paired with improved models and large data sets. As explained in this chapter, many of the problems of current DL methods stem from the limitations of the data sets. Creating more, better and larger data sets or pipelines for generating such data sets is the fundamental direction for the success of the field.

Based on the comprehensive review presented in this chapter, we start tackling the open problems in DL. In the next chapter, we focus on combining the advantages of RL with DL. Specifically, we tackle the context problem and the safety problem. We proposed two methods to tackle these problems. The first method learns context-invariant policies by using a multi-context demonstration data set to learn context-invariant features that then replace the state observation in policy learning. The second method improves the safety of RL agents during the learning process, by making use of existing demonstrations and a method to prevent the agent from performing catastrophic actions.

Table 3.4: Categorization of Demonstration Learning papers.

Name		Year	Technique	Data	Learned Goal	Inference	Evaluation	Benchmark
Abeel et al. 2004	[183]	2004	Teleoperation	Raw Data	IRL	Regression	Acc. Reward	Grid World Car Driving Simulation
ABPS	[184]	2021	N/A	N/A	Policy Learning	Classification	Acc. Reward	Custom Grid World
Align-RUDDER	[185]	2022	N/A	Raw Data	Policy Learning, IRL	Regression	Succ. Rate	Minecraft
AOG	[61]	2017	Sensors on Teacher	Sensor data, Image	Classification	Classification	Succ. Rate	Water bottle opening
APE-V	[133]	2022	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward, Succ. Rate	D4RL, Procgen Mazes
APID	[92]	2013	Teleoperation	Raw Data	Policy Learning	Regression	Time, Acc. Reward	Path Finding
AQuaDem	[103]	2021	Teleoperation	Raw Data	Policy Learning	Classification	Acc. Reward, Succ. Rate	D4RL
ARC	[147]	2018	N/A	Image	Policy Learning, IRL	Regression	Acc. Reward	Navigation, Robot Pushing
ATAC	[142]	2022	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	D4RL
ATC	[145]	2021	Observation	Image	Policy Learning	Regression	Acc. Reward, Train Time	DM control, Atari, DM Lab
AT-Net	[73]	2020	Teleoperation	Image	Policy Learning	Regression	Alignment Error, Accuracy, Succ. Rate	Manipulation
AWAC	[163]	2020	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward, Succ. Rate	Simulated, Real Robot Manipulation
AWR	[162]	2019	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	OpenAI Gym, Simulated Robot
BCQ	[26]	2019	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	OpenAI Gym MuJoCo tasks
BEAR	[160]	2019	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	OpenAI Gym MuJoCo
BRAC	[161]	2019	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	OpenAI Gym MuJoCo
BREMEN	[110]	2020	Teleoperation	Raw Data	Model-Based Policy Learning	Regression	Acc. Reward, KL Divergence	OpenAI Gym MuJoCo
CDS	[186]	2021	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward, KL Divergence	MuJoCo, Meta-world
ChauffeurNet	[187]	2018	Teleoperation	Sensor data	Policy Learning	Regression	Distance Error	CARLA driving simulator
CIC	[150]	2022	N/A	N/A	Policy Learning	Regression	Acc. Reward	Mujoco, Simulated Jaco Robot
CLfD	[9]	2022	Observation	Image	Policy Learning	Regression	Alignment Error, Success Rate, Acc. Reward	Simulated Panda Manipulation
Coarse-to-Fine IL	[188]	2021	Observation	Image	Policy Learning	Regression	Error, Succ. Rate	Target Reaching
Codevilla et al., 2018	[21]	2018	Teleoperation	Sensor data	Policy Learning	Regression	Succ. Rate, Missed turns, Interventions, Infractions	Real Truck Driving
Confidence-Based LfD	[74]	2007	Teleoperation	Raw Data	GMM	Classification	Accuracy, Collision Rate	Custom Simulation
Context-Aware Translation	[57]	2018	Observation	Image	Policy Learning,IRL	Regression	Error, Succ. Rate	MuJoCo Manipulation
COPO	[189]	2022	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward, Cost, Violations	Walk-Around-Grid, Bipedal Walker
COMBO	[111]	2021	Teleoperation	Raw Data	Model-Based Policy Learning	Regression	Acc. Reward, Model Error	D4RL
CORRO	[190]	2022	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	Mujoco
CQL	[96]	2020	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward, Value Error	D4RL
Cross-Context IL	[105]	2020	Observation	Image	Policy Learning	Regression	Distance, Succ. Rate	Manipulation
CSI	[121]	2013	Teleoperation	Raw Data	IRL	Both	Acc. Reward	Mountain Car, Driving Simulator
CVaR	[141]	2022	N/A	N/A	Policy Learning	Regression	Acc. Reward	Custom Grid World
CVPO	[65]	2022	N/A	N/A	Policy Learning	Regression	Cost, Acc. Reward	Custom Simulation Tasks
DAgger	[69]	2011	Teleoperation	Raw Data	Policy Learning	Classification	Falls, Distance Traveled	Super Tux Kart, Super Mario Bros.
Dalal et al., 2018	[85]	2018	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward, Violations	Custom MuJoCo tasks
DIS	[123]	2016	Teleoperation	Raw Data	Policy Learning, IRL	Regression	Acc. Reward	Maze, Mario AI
DQfD	[23]	2018	Teleoperation	Image	Policy Learning	Classification	Acc. Reward	ALE
DQN	[24]	2015	N/A	N/A	Policy Learning	Classification	Acc. Reward	Atari
Dogged Learning	[42]	2007	Teleoperation	Raw Data, Image	Other Predictive Learning	Both	Student visual inspection, Error	Ball seeking, head mirroring tail
DoubIL/ResiduIL	[87]	2022	Teleoperation	Raw Data	Policy Learning	Regression	Loss	OpenAI Gym
DT	[14]	2021	Teleoperation	Raw Data	Sequence Model	Regression	Acc. Reward	Atari, OpenAI Gym
DWBC	[136]	2022	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward, Discriminator Accuracy	D4RL
EnsembleDAgger	[191]	2019	N/A	N/A	Policy Learning	Regression	Acc. Reward	OpenAI Gym
ExORL	[192]	2022	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	DeepMind control suite
GAIL	[64]	2016	Teleoperation	Raw Data	GAIL	Regression	Acc. Reward	MuJoCo
GCB	[151]	2022	Observation	Image	Policy Learning	Regression	Succ. Rate	Pybullet
Gradient-Based IRL	[122]	2021	Observation	Image	Model Learning, IRL	Regression	Train Time, Distance	Teaching
GTI	[193]	2021	Teleoperation	Raw Data	Policy Learning	Regression	Succ. Rate, Generalization	Manipulation
Guo et al., 2022	[94]	2022	Observation	Image	Policy Learning	Regression	N/A	N/A
HAMMER	[68]	2005	Observation	Image	Bayesian Belief	Regression	N/A	N/A
Hayes et al., 2014	[56]	2014	Observation	Image	Active learning	Regression	Execution Paths	Lego Montage
HDT	[11]	2022	Teleoperation	Raw Data	Sequence Model	Regression	Acc. Reward	UR3 Reaching, D4RL
IRIS	[194]	2020	Teleoperation	Raw Data	Policy Learning	Regression	Succ. Rate, Acc. Reward, Traj. Length	Graph Reach, RoboTurk, Robosuite
Ijspeert et al., 2002	[31]	2002	Sensors on Teacher	Sensor Data	LWR	Regression	Error	Tennis Swings
ILEED	[95]	2022	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	Simulated Minigrid tasks
LazyDAgger	[195]	2021	N/A	N/A	Policy Learning	Regression	Acc. Reward, Interventions	MuJoCo
Levine et al., 2016	[71]	2016	N/A	Image	Policy Learning	Regression	Distance, Succ. Rate, Error	Manipulation
Levine et al. 2018	[196]	2018	Observation	Image	Policy Learning	Regression	Failure Rate	Manipulation Task
LDM	[97]	2022	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward, Succ. Rate	OpenAI Gym MuJoCo, SimGlucose
LTSD	[197]	2019	Teleoperation	Raw Data	Policy Learning, IRL	Regression	Acc. Reward	BiMGame, AntTarget, AntMaze
LfMD	[30]	2015	Teleoperation	Point Cloud	Other	N/A	Succ. Rate	Pick Place, Towel Folding
Maeda et al., 2017	[51]	2017	Kinesthetic	Sensor Data	Policy Learning	Regression	Distance Error	Manipulation Tasks
MAGIC	[113]	2016	N/A	N/A	Policy Learning (OPE)	Regression	Regression Error	ModelFail/Win, Maze, Mountain Car, Cart
Max. Ent. IRL	[118]	2008	Sensors	GPS Data	IRL	N/A	Matching	Path Following
MBPO	[107]	2019	N/A	N/A	Policy Learning (OPE)	Regression	Acc. Reward, Regression Error	MuJoCo
MERLION	[106]	2021	Teleoperation	Raw Data	Policy Learning	Classification	Acc. Reward	Maze, Dialogue, Recommendation
MIR	[198]	2021	Observation	Image	Policy Learning	Regression	Succ. Rate	MuJoCo, Manipulation
MOPO	[109]	2020	Teleoperation	Raw Data	Model-Based Policy Learning	Regression	Acc. Reward	D4RL
MOReL	[108]	2020	Teleoperation	Raw Data	Model-Based Policy Learning	Regression	Acc. Reward	OpenAI Gym MuJoCo
Motion2Vec	[199]	2021	Kinesthetic, Observation	Robot Data, Image	HMM	Regression	Loss, Segmentation Accuracy, Noise	Pick-and-Place, Suturing
MRDR	[112]	2018	N/A	N/A	Policy Learning (OPE)	Regression	Regression Error	ModelFail/Win, Maze, Mountain Car, Cart
Mülling et al., 2013	[119]	2013	Kinesthetic	Raw Data	LWR	Regression	Cost, Succ. Rate, Acc. Reward	Tennis Swings
Multi-AMP	[137]	2022	Observation	Image	Policy Learning	Regression	Stand Duration	4 Legged Robot Movements
MVP	[200]	2022	Observation	Image	Policy Learning	Regression	Succ. Rate	PixMC
ODT	[153]	2022	N/A	Raw Data	Policy Learning	Regression	Acc. Reward	D4RL
Pan et al., 2017	[201]	2017	Teleoperation	Sensor data, Image	Policy Learning	Regression	Speed, Succ. rate	AutoRally
PC-GMM	[91]	2020	Teleoperation	Raw Data	GMM, GMR, RL	Regression	Succ. Rate, Error	Peg-in-Hole
PEMIRL	[139]	2019	Teleoperation	Raw Data	IRL	Regression	Acc. Reward	MuJoCo
Pi2-ES-Cov	[53]	2020	Kinesthetic	features from point cloud	Policy Learning	Regression	Succ. Rate	Simulated, Real Robot Manipulation
Pinto et al., 2016	[202]	2016	Observation	Image	Policy Learning	Regression	Accuracy	Manipulation Task
PTS	[130]	2015	N/A	N/A	IRL	Regression	Score	Mounting Car 3d, Cart Pole, Mario
Recovery RL	[203]	2021	Teleoperation	Raw Data	Policy Learning	Regression	Succ. Rate, Violations	Simulated task, Real Robot
REPaIR	[204]	2020	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward, AUC, Accuracy	Robot Reaching
Rhinehart et al., 2018	[205]	2018	Teleoperation	Raw Data	Model-Based Policy Learning	Regression	Succ. Rate, Classification Metrics	CARLA driving simulator
RLfD	[124]	2015	Teleoperation	Raw Data	Policy Learning, IRL	Regression	Acc. Reward	Mario AI, Cart Pole
RIS	[206]	2021	N/A	N/A	Policy Learning	Regression	Acc. Reward, Distance, Succ. Rate	Navigation, Manipulation
RPL	[207]	2019	Teleoperation	Raw Data	Policy Learning	Regression	Succ. Rate	MuJoCo
RUDDER	[208]	2019	N/A	Raw Data	Policy Learning, IRL	Regression	Acc. Reward	Atari
Ruppel et al., 2020	[63]	2020	Sensors on Teacher	Sensor Data	Sequence Model	Regression	Succ. Rate, Error	Manipulation
S4RL	[76]	2022	Teleoperation	Image	Policy Learning	Regression	Acc. Reward	D4RL, MetaWorld, RoboSuite
SafeDAgger	[209]	2017	Teleoperation	Raw Data	Policy Learning	Regression	Score, Safety, loss	TORCS Driving Car
SAILR	[66]	2021	N/A	N/A	Policy Learning	Regression	Acc. Reward, Constraints	Point Robot, OpenAI Gym MuJoCo
SAM	[138]	2019	Teleoperation	Raw Data	GAIL	Regression	Acc. Reward	MuJoCo
SCIRL	[120]	2012	Teleoperation	Raw Data	IRL	Classification	Acc. Reward	Highway
Sepsis Treatment	[210]	2017	N/A	N/A	Policy Learning	Classification	Mortality	MIMIC-3.
Silver et al., 2010	[117]	2010	Kinesthetic	Raw Data	LWR	Regression	Cost, Loss, Distance, Speed	Navigation
SMILe	[75]	2010	Teleoperation	Raw Data	Policy Learning	Regression	Falls, Distance	Mario Bros
SPiRL	[211]	2020	Teleoperation	Raw Data	Policy Learning	Regression	Succ. Rate	D4RL
SQUIRL	[140]	2020	Observation	Image	Policy Learning, IRL	Regression	Succ. Rate	Pick-Carry-Drop, Pick-Place
SRL-RNN	[212]	2018	N/A	N/A	Sequence Model	Both	Estimated Mortality	MIMIC-3.
SWIRL	[213]	2019	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	RCCar, Acrobot
SWITCH	[114]	2017	N/A	N/A	Policy Learning (OPE)	Regression	Regression Error	UCI data sets
TCC	[148]	2019	Observation	Image	Representation Learning	N/A	Classification Accuracy, Kendall's Tau	Pouring and Penn action data sets
TCN	[3]	2018	Observation	Image	Policy Learning	Regression	Alignment Error, Classification Error, Acc. Reward	Pouring
TD3+BC	[164]	2021	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	D4RL
TPIL	[214]	2021	Observation	Image	Policy Learning	Regression	Alignment Error, Loss	PyBullet, Minecraft
UCT	[70]	2014	Teleoperation	Image	Regressor/Classifier	Both	Score	ALE
UDS	[80]	2022	Teleoperation	Raw Data	Policy Learning	Regression	Acc. Reward	D4RL
visual MPC	[215]	2018	Observation	Image	Model-Based policy learning	Regression	Regression Error, Succ. Rate,	Manipulation Task
Weak Label LfD	[50]	2020	Teleoperation	Sensor Data, Image	Classifier	Classification	Effort	Manipulation
Yang et al., 2022	[216]	2022	Teleoperation	Raw Data	Policy Learning, GAN	Regression	Acc. Reward	D4RL

Table 3.5: Summary of the advantages and disadvantages of demonstration learning methods.

Advantages	**Diverse and flexible:** can be combined with other paradigms. **Data efficiency:** faster convergence from fewer data over reinforcement learning. **Performance guarantees:** the method will at least be as good as the demonstrations. **Reduction of programming load:** no need to program the model's decision for every case. **Safety:** the agent learns without interacting with the environment. **Simplification:** demonstrate the task and estimate a policy with a SOA method.
Disadvantages	**Creation of the data set:** difficult to create large numbers of high quality demonstrations. **Distributional shift:** generalize from the data set or remain inside its distribution. **Quality of the data set:** performance relies heavily on the demonstrated behavior.

Chapter 4

Tackling RL Policy Learning Problems with DL

4.1 Introduction

RL has gained significant attention in recent years due to its wide range of applications and flexibility. However, RL policy learning has a few disadvantages that restrict it to simulation environments and prevent its application to real world scenarios. The main disadvantage is its trial-and-error nature that can pose safety risks. Failed interactions with a real world environment can lead to serious consequences that prevent policy learning altogether.

Another problem with policy learning is the context problem. We refer to context as the set of characteristics that define the environment. For example, the background objects, the illumination, and the camera position. Most policies are estimated under a single specific context, that is, under a fixed camera position, fixed background objects with their fixed positions, and a fixed level of illumination, amongst other characteristics. Changing any of these elements will likely occur in a real-world environment, which results in a different context. Then, because the context is not the same as the one where the policy was trained to perform, the policy will likely not be able to perform the task under this new context. This problem is often ignored because the policy is trained and evaluated in simulation or controlled real-world environments, allowing for the context to remain unchanged. However, this problem severely limits the scalability of the policy to real applications where the context likely changes.

DL learns the policy using the demonstration data set. This means that the agent remains safe during the learning process, because it does not interact with the environment during training. However, the policies estimated with DL generally obtain worse performance than RL policies. This is because the demonstration data set is sub-optimal due to the difficulty of collecting a data set that covers the entire state and action spaces, especially in complex environments. In this chapter, we aim to augment RL's policy learning with the demonstrations of DL to tackle the two aforementioned problems without hindering the performance of the estimated policy. We propose two novel methods: CLfD to tackle the context problem and DEFENDER to tackle the safety problem. The next two sections are dedicated to each method individually.

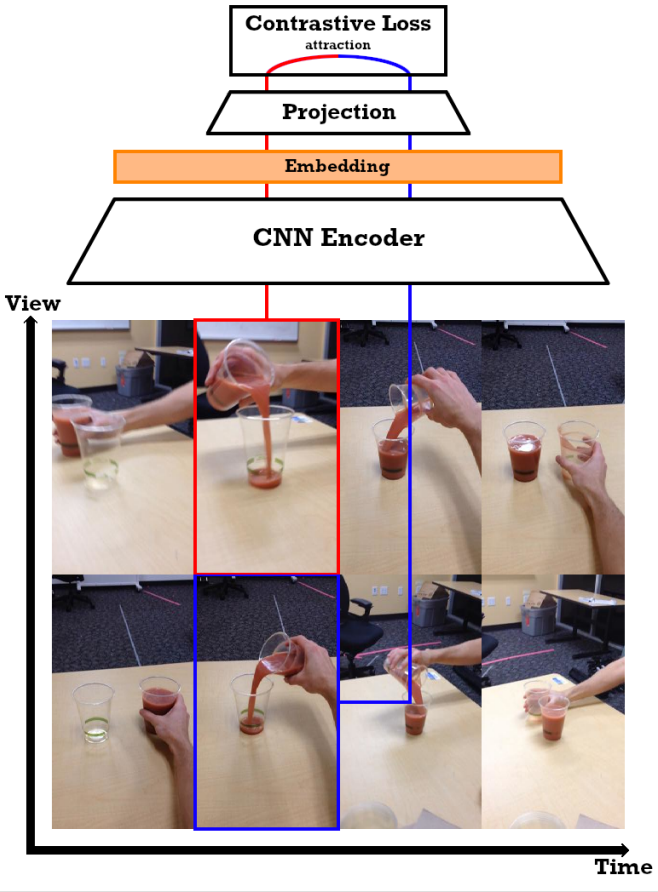


Figure 4.1: CLfD framework: The data set of synchronized multi-view video pairs generates a set of anchor-positive image pairs. The CNN encoder generates the respective feature embedding pairs. A projection network maps the embeddings to the learning space where the loss is applied. The networks' parameters are updated to maximize the agreement between the pair of features using the contrastive loss. The embeddings can replace the images for viewpoint invariant representations in various tasks.

4.2 Multi-View Contrastive Learning from Demonstrations

RL algorithms have enabled human-robot collaboration. Although these methods can be used to obtain a working policy and are computationally efficient, the set of visited environment states present in the demonstrations is small, and they do not generalize to scenarios unseen during the demonstration phase. Because of this, DL approaches are frequently coupled with RL to extend the agent's knowledge of the environment and consequently increase its generalization capability.

However, such policies are trained under a static domain and will likely struggle to perform in other domains. Minor changes to the positions of the cameras, illumination, and objects in the scene change the content of the images representing the state of the task. Ideally, robots would extract task-relevant attributes from the images while ignoring the irrelevant noise information also present in the images. Existing solutions include supervised learning for tasks where humans can easily specify labels, such as object classification. Examples of possible attributes could be background, containers, and viewpoint. However, most robotic tasks are too complex, and it becomes impractical to label every at-

tribute for every state in the demonstration. Other solutions make use of metric learning, which is computationally complex and makes the learning and inference processes slow [254]. We alleviate these issues through the means of self-supervised and Contrastive Learning (CL) [255]. We train an encoder on multi-viewpoint video demonstrations of the robotic task, where time is the supervision signal. CL encodes the state images into viewpoint-invariant representations.

This chapter presents a DL framework for extracting image embeddings that enhance task-specific information while suppressing task-irrelevant noise. We show that the representations can be applied to learn robotic tasks through standard RL algorithms using a single demonstration to provide the reward signal. The proposed framework does not require any prior knowledge about the task beforehand. We validate our method on the publicly available Multi-View Pouring data set and a custom Pick and Place simulation environment and data sets. The proposed model reaches equal performance while requiring less training time when compared with other baselines. Additionally, ablation studies were performed on different components of the method to identify the best configuration and the contributions of the individual components.

The main contributions made by this work are the multi-view Contrastive Learning from Demonstrations (CLfD) framework, which learns viewpoint invariant representations from demonstrations, the Pick and Place task environment in CoppeliaSim and the respective multi-view demonstration data set for bench-marking RL and DL methods.

4.2.1 Proposed Approach

Algorithm 1 CLfD algorithm

Input: Contrastive data set $D = \{x_a, x_p\}$, batch size N, training iterations I, constant τ, encoder f, projector g.
for $i = 1$ **to** I **do**
  for $batch$ $\{x_{a_{t:N}}, x_{p_{t:N}}\}$ **in** D **do**
    $h_{a_{t:N}} = f(x_{a_{t:N}})$
    $h_{p_{t:N}} = f(x_{p_{t:N}})$
    $z_{a_{t:N}} = g(h_{a_{t:N}})$
    $z_{p_{t:N}} = g(h_{p_{t:N}})$
    for $i \in \{1, ..., 2N\}$ and $j \in \{1, ..., 2N\}$ **do**
      $s_{i,j} = z_i^T z_j / (\|z_i\| \|z_j\|)$
    end for
    $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
    Update f and g to minimize $\mathcal{L}$ using Adam optimizer.
  end for
end for
return f

Our method applies CL [255] to multi-view DL and is represented in Fig. 4.1. CLfD learns viewpoint-independent state representations by maximizing the agreement between synchronized frames from different viewpoints in the embedding space using the contrastive loss. We assume to have access to a demonstration data set where each demonstration is captured from varying viewpoints and the frames of the different views are synchronized. Frames from the same demonstration and timestamp but different viewpoints represent the same world state, we call these anchor-positive pairs. We want to obtain representations that capture task-relevant features and ignore task-irrelevant ones. We do this by encouraging the features from anchor-positive pairs to be close in the embedding space through CL. Our framework is described in Algorithm 1 and is composed of three modules.

A deep CNN encoder extracts the frames' embedding vectors. The framework is flexible and allows for changes to the network without requiring changes to the remaining modules, as long as it ends with a fully connected layer outputting $1 \times N$ feature vectors, where N is the number of features each embedding vector should contain. We use vectors with 32 features. We tested several network architectures to act as the encoder. We present the results in section 4.2.2. The authors of SimCLR [143] found that using an additional small nonlinear projection neural network was more beneficial than applying the contrastive loss directly to the features. Therefore, we also used an MLP with a single hidden layer, with N neurons, as the projection network to project the features obtained from the encoder to the space where the contrastive loss is applied. We project the features to 64-dimensional space.

Lastly, we apply the contrastive loss introduced in SimCLR [143] to the output of the projection head. Originally the authors of SimCLR applied a set of transformations to generate a positive pair from the anchor image. The multi-view data set directly provides the pairs. We use this loss because it applies to contrastive prediction tasks without a prior sampling of negative examples. Given a batch of anchor-positive pairs of size N, $\{x_{ki}, x_{kj}\}$ where $0 \leq k < N$, the contrastive loss aims to pair each anchor element x_i with its respective positive pair x_j in the full set of elements $\{x_{1\ldots 2N}\}$. Instead of explicitly sampling negative examples such as in triplet learning, for each anchor image x_i, the remaining $2(N-1)$ images in the batch of size N pairs act as negative examples relative to the anchor. Not having to explicitly sample negative pairs corresponds to a significant complexity reduction and performance advantage over methods that require it.

The loss function for a pair of anchor-positive examples (x_a, x_p) is [143]:

$$\ell(x_a, x_p) = -log \frac{exp(sim(z_a, z_p)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq a]} exp(sim(z_a, z_k)/\tau)} \quad (4.1)$$

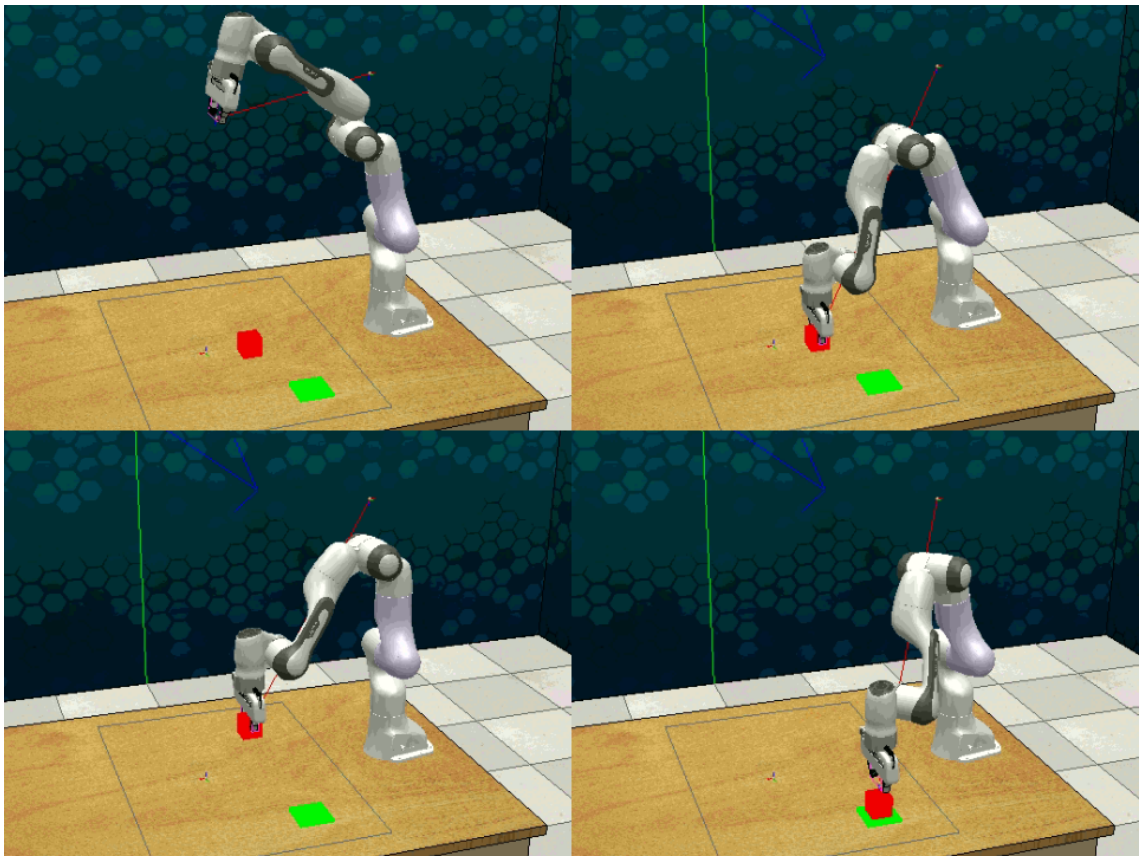


Figure 4.2: The pick and place task environment in CoppeliaSim [2]. The 7-DOF simulated Panda arm must first place its end-effector near the red box and close the gripper to pick it up. Then the robot must move its end-effector above the goal position marked by a green plane and open the gripper, causing the box to fall on top of the plane. Images are simultaneously captured from 5 viewpoints: first-person (camera attached to the gripper), front, top, overhead, and right-side viewpoints. We use the environment to capture the demonstration data set and to train the DDPG agent. In each new demonstration, the locations of the box and the green plane are randomly changed.

where $\mathbb{1}_{[k \neq a]} \in \{0, 1\}$ is 1 if $k \neq a$ otherwise is zero, τ is a temperature hyper-parameter which we set to 0.5, and $sim(a, b)$ is the cosine similarity between two vectors. Lastly, the final loss $\mathcal{L}$ used to update the parameters of the networks is the normalized temperature-scaled cross-entropy loss (*NT-Xent*) [143] computed using the pair losses across the batch.

4.2.1.1 Pick and Place Multi-View Data Set

We created a custom Pick and Place task environment in CoppeliaSim [2]. The environment is composed of a 7-DOF Panda arm on top of a table. The arm must pick up the red box and place it on the goal location marked by the green plane. Additionally, there are five fixed cameras capturing observations from different viewpoints: one attached to the robot, one in front, another above, one behind, and one on the right side of the robot. The environment is represented in Fig. 4.3.

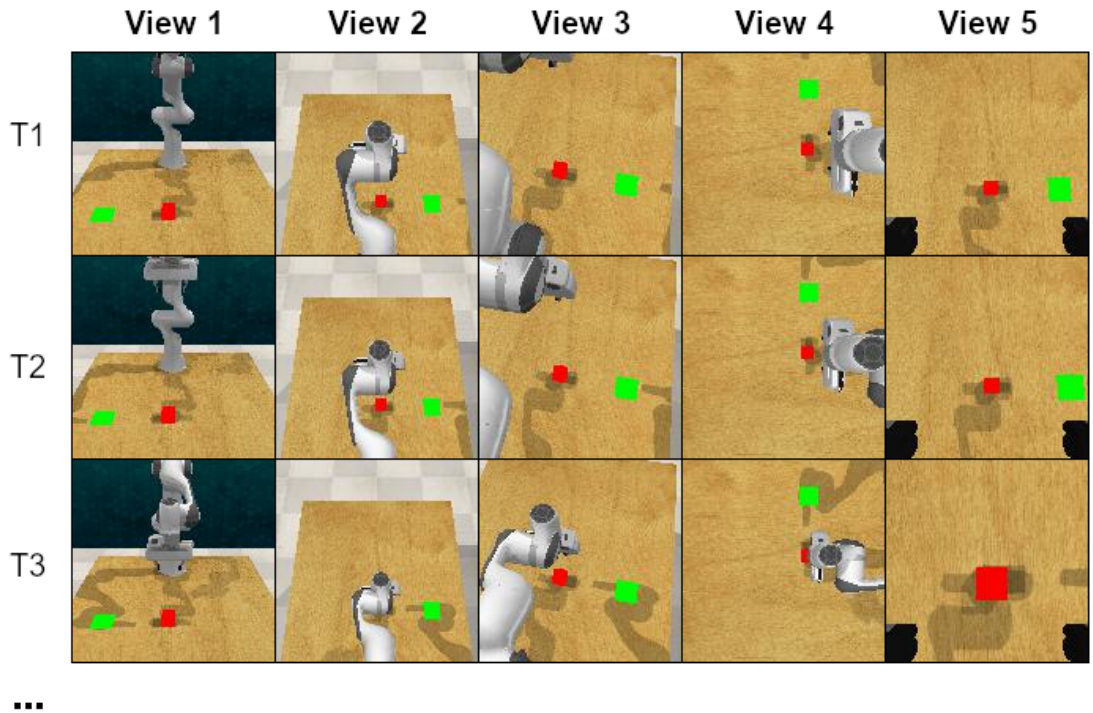


Figure 4.3: Examples of stored views in the custom demonstration data set for the Pick and Place task simulated in CoppeliaSim [2], captured from 3 timestamps. Each demonstration is captured from 5 different fixed viewpoints.

Using this environment, we then created a custom multi-view demonstration data set for the pick-and-place task. The demonstrations are performed automatically by having access to the position of the objects inside the simulation. With them, we obtain the joint paths to pick and to place the box, using inverse-kinematics. We perform the path, and after each transition, we obtain the images from each camera, as well as the robot's current joint angles and velocities and store them in the data set. The initial positions of the box and stack are randomized in each demonstration.

The generated data set is composed of 150 demonstrations, a realistic size, smaller than what is used in the literature [3, 140, 73], due to the difficulty of obtaining real demonstrations. We select 100 demonstrations for training, 25 for validation, and the remaining 25 for testing. Unlike [73], we do not include failed demonstrations in our data set. Some examples of views stored in the data set are presented in Fig. 4.3. This data set and our code are available at https://tinyurl.com/mvclfdemos.

4.2.1.2 Intuition towards using CL for Demonstration Learning

Triplet learning has been successfully combined with DL to estimate a viewpoint invariant space by pulling anchor-positive images closer while repelling anchor-negative images in the space. The embeddings can then be applied to pose demonstration and RL tasks [3]. However, it is a slow and complex method. It requires three images and the sampling and frequent re-sampling of negative examples for each anchor-positive pair.

CL has been applied to more traditional ML approaches, such as object classification [143]. Typically the positive image is obtained from the anchor image through transformations. Alternatively, in [4] the images are converted into Lab color space and the anchor-positive images correspond to each channel. Then, the embedding space is estimated by pulling the features from the generated images closer to the original image or another image generated from a different set of transformations.

We avoid the downsides of triplet learning by replacing it with CL. Using synchronized video pairs, we avoid the need to generate fake images and since both images are authentic, we ensure they are contrasting in the viewpoint, causing the model to converge much faster and the embedding space to be viewpoint invariant. Additionally, by using CL, we reduce the number of images from three to two compared to triplet learning. Consequently, removing the need to sample negative examples reduces complexity and fastens the learning process. We use the (*NT-Xent*) contrastive loss to repel the features from images outside the anchor-positive pair in the embedding space, similarly to triplet learning.

4.2.1.3 Using the features for RL

We show that the features obtained from our model can be used to learn a policy through RL. The agent learns to pick up a box and place it on top of a plane using the embedding obtained from the encoder as the state representations. The agent learns the task using a single demonstration for guidance. The demonstration used for RL was not used during training.

The RL problem is a standard MDP, defined by the tuple $< S, A, R, T, \gamma >$ [8], where S is the set of states, A is the set of actions, R is the reward function, T is the state transition function, and γ is the discount factor. Each state is composed of the embedding vector representation of the state image obtained from the encoder along with the robot's joint angles and velocities and binary gripper state. This ensures representation of both the environment and robot states. The actions are the joint velocities and gripper state.

For multi-stage tasks, such as the Pick and Place task, we treat each stage as an individual task and train a policy for each stage to speed up convergence. The reward at timestep t is defined by how close the environment's state embeddings are to the last demonstration state embeddings for the respective stage:

$$R(t) = -\|f(s_{et}) - f(s_{dg})\| \quad (4.2)$$

where $f(s_{et})$ is the embedding of environment's current state, while $f(s_{dg})$ is the last state embedding for the stage g in the demonstration. We set the discount factor γ to 0.99. We use an off-the-shelf RL algorithm, DDPG [256], to estimate the policy because it is compatible with continuous action spaces and enhances it with HER [257] to speed up convergence.

Table 4.1: Comparison between the CLfD method with the TCN [3] and CMC [4] baselines using the alignment error percentage metric, varying the encoding architecture and data set. Models were trained for 1000 epochs and using a batch size of 50.

Data Set	Encoder	Alignment Error [%]		
		TCN	CMC	CLfD
PickPlace	Resnet18	24.98	**12.79**	16.82
	TCN	15.35	4.45	**1.90**
Pouring	Resnet18	27.20	27.80	**23.28**
	TCN	26.74	24.57	**19.84**

4.2.2 Experiments

In this section, we define the methods used to evaluate the framework, which helps to understand the different design choices performed. We performed the experiments on a machine equipped with an Intel i7 980 with 3.33 GHz and 6 cores, 24 GB of RAM, and an Nvidia GTX 1080Ti GPU. We train the models using the Pytorch framework [258] and use its builtin Resnet18 implementation. Use CMC [4] and TCN [3] baseline implementations provided in their respective works.

4.2.2.1 Alignment Error

We compare the performance of our method with the TCN [3] and CMC [4] baselines using the alignment error metric, proposed in [3]. For the TCN baseline, the negative pairs are randomly sampled. For the CMC baseline, we don't perform the conversion into the Lab color space because we can directly obtain the anchor-positive pairs from the data set. The alignment error metric measures how well the model aligns the frames of synchronized video pairs based on their embeddings. Simultaneously, it evaluates how well the model approximates embeddings from concurrent frames and repels non-concurrent embeddings. We test two different encoders on the two data sets. The alignment error between two synchronized videos is:

$$AE(v_1, v_2) = \sum_{i=1}^{N} \min_j (\|v_{1i} - v_{2j}\|)/N \quad (4.3)$$

where v_{ki} is the i-th frame of video v_k, and N is the total number of frames in each video. First, we evaluate the framework on the publicly available Multi-View Pouring data set [3], which contains 235 synchronized video pairs of a person demonstrating pouring liquids from one container to another. In each pair, one video is captured from a static viewpoint while the other is moved around the task, capturing different viewpoints. Additionally, we used the custom demonstration data set for the pick-and-place task simulated in CoppeliaSim [2].

The methods are trained on the data sets for 1000 epochs using batches of 32. All encoders are pre-trained on the ImageNet [259] data set. We calculated the alignment error of the model using the validation set every 25 epochs and present the lowest value obtained throughout training. We compare the performance of the Resnet18 encoder with

Table 4.2: Classification accuracy, for determining the stage of the demonstration, is obtained from evaluating an MLP with two fully connected layers on the Pick or Place classification data set's test set. The MLP is trained over the features obtained from a TCN encoder trained using the CLfD algorithm. The accuracy is calculated for camera angles seen and unseen camera viewpoints during the training of the TCN encoder. The MLP obtains near-perfect accuracy proving that the features are viewpoint independent.

ViewPoints	Accuracy [%]
Seen	100
Seen + Unseen	99.41
Unseen	98.69

the TCN encoder from the baseline [3]. It is clear from Table 4.1 that CLfD outperforms the TCN baseline in all four setups. With the custom Pick-and-Place data set, CLfD reduces the alignment error by over 8% using Resnet18 encoder and 13.45% using the TCN encoder compared with the TCN baseline. For the real-world multi-view pouring data set, CLfD reduces the alignment error by nearly 4% using the Resnet18 encoder and by 6.9% using the TCN encoder.

CLfD outperforms the CMC baseline in three out of four setups. CLfD shows an alignment error lower than CMC by 2.55% when using TCN encoder, for the Pick and Place data set, and by 4.52% and 4.73%, when using Resnet18 and TCN encoders respectively, for the Pouring data set. CMC does outperform our method in the Pick and Place data set when using Resnet18 encoder. However, we obtain a smaller alignment error by using the TCN encoder. In fact, the TCN encoder outperforms Resnet18 in all setups, proving that it can also be used in CL to encode viewpoint invariant features.

Lastly, we obtain an alignment error near the value of 18.8% for the multi-view Pouring data set, which is the one presented by the baseline in [3]. However, this TCN baseline encoder is trained on the multi-view pouring data set for 397 thousand iterations while sampling negative examples. Therefore, our model converges in far fewer epochs, with faster training and without a specialized algorithm for sampling the data.

4.2.2.2 Stage Classification

We use a variant of the labelled classification error as seen in [3, 73], which requires labelling different features for each frame. Instead, we classify the stage of the task from the embeddings. The stage classification metric measures how well the model can disentangle stage-related attributes from the features, proving that they contain valuable information to be used in a real robotic task. We reorganized the Pick-and-Place demonstration data set into a classification data set. The new data set is organized into two classes, pick and place. Each class contains the exact number of images, totalling 4000 examples. The images are evenly distributed amongst the five cameras. A TCN encoder is first trained on the multi-view Pick-and-Place data set using the CLfD algorithm. The encoder generates the features from the images of the classification data set. The features are forwarded to an MLP with two fully connected layers. The MLP is trained for 500 epochs. We evalu-

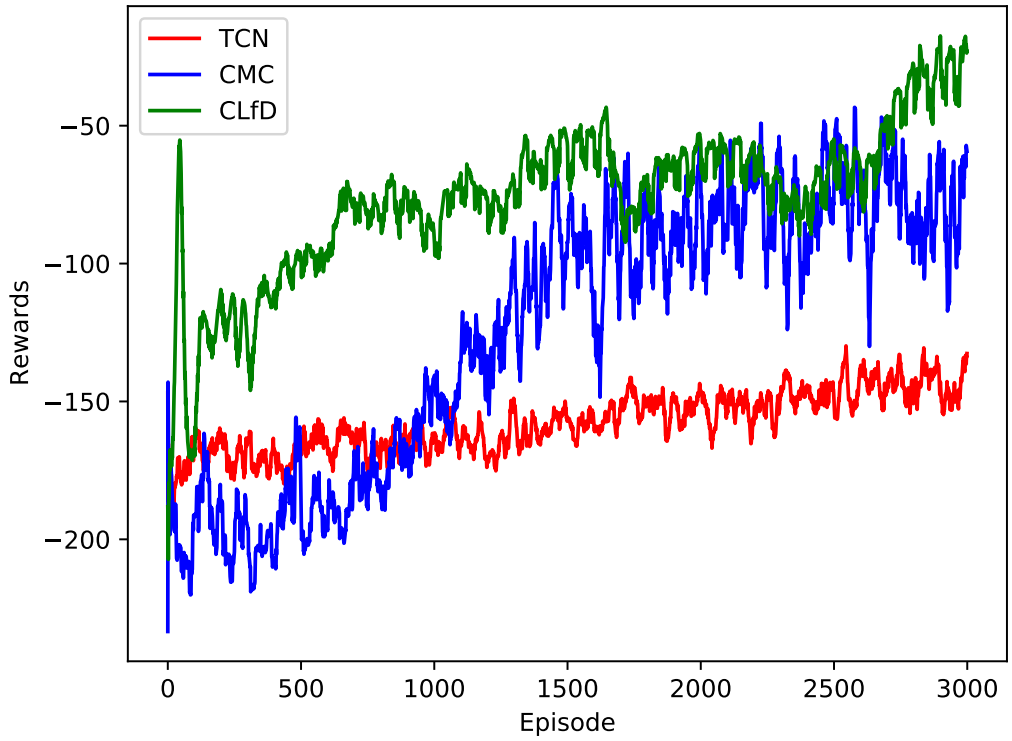


Figure 4.4: Accumulated reward by the DDPG agent over 3000 episodes when trained using CLfD, CMC and TCN encoders with TCN network [3] as the backbone for learning the Pick stage of the Pick and Place task in the simulated environment.

ate the accuracy of the MLP for predicting the correct stage from the features using 200 unseen test images.

We also test whether the features are viewpoint independent by comparing the MLP's performance when trained on viewpoints not seen by the encoder during its training. The encoder was trained on images captured by three specific cameras. We refer to these as seen viewpoints. The images captured by the remaining two cameras are from unseen viewpoints. Table 4.2 shows that the MLP obtains near-perfect results even for images captured from unseen viewpoints during the training of the encoder. These results show that the encoded features are viewpoint independent and therefore can be used in real robotic tasks to encode the state image into a feature vector regardless of the viewpoint from where the image was captured.

4.2.2.3 Reinforcement Learning

Lastly, we determine whether the embedded features can replace the state as viewpoint invariant representations for RL tasks as well as if they can be used to provide rewards, by evaluating the accumulated rewards over time. The RL agent described in section 4.2.1.3 is trained using the CLfD encoder with the TCN architecture as the backbone, trained on the Pick-and-Place data set. The agent must solve the Pick and Place task. Because it is

a two-stage task and we're using the DDPG algorithm with sparse rewards, we split each stage of the task into individual tasks for faster convergence. The agent must learn a policy for each stage. The agent is trained for 3000 episodes, after which the agent learns each stage. The accumulated reward after each episode, for the pick stage, is calculated and shown in Fig. 4.4. An identical graph is obtained for the place stage.

We compare the performance of the agent using the CLfD encoder's features with the performance obtained when using the features obtained from the encoder trained with the TCN and CMC baselines. The agent is not able to learn the task when using the features of the baselines. Particularly, the TCN agent still performs random actions after the 3000 episodes. CMC gets close to picking up the box and to placing the box but fails by a small misplacement of the gripper. The results show that: the performance of TCN [3] is dependent on the efficient sampling of negative pairs and that CLfD's features can be used to learn this type of task with a smaller computational effort. This further cements that the proposed framework is able to learn an embedding space capable of representing the environment's state and that such representations can be used directly as a replacement of the state image for the agent's input as well as for guiding the agent through the learning process by providing the rewards.

4.3 DEFENDER: DTW-Based Episode Filtering Using Demonstrations for Enhancing RL Safety

RL [8] enables agents to learn how to behave in an environment through trial and error, but safety concerns have limited RL deployments to simulations. Ensuring safety in unknown environments remains a challenge in RL, particularly in safety-critical domains such as healthcare and autonomous driving. To address these issues, safe RL studies the RL problem subject to certain constraints, with the agent aiming to maximize task reward while limiting constraint violations. However, deploying agents with complete knowledge of the environment to perform their tasks is unrealistic. Alternatively, in DL the agent learns from expert demonstrations without direct environment interaction. However, the quality of the data set plays a crucial role in the performance. Due to the difficulty of collecting a demonstration data set that covers the entire state space, pure DL policies often underperform compared to RL policies.

In this chapter, we propose a novel task-agnostic algorithm that enhances existing RL algorithms by promoting safety during interactions with the environment. Our algorithm uses a small data set of good and bad demonstrations to filter unsafe actions, terminate episodes with unsafe trajectories, and encourage the agent to explore different trajectories. We conduct ablation studies to evaluate filtering strategies and demonstrate the utility of our method on four tasks from OpenAI's MuJoCo environment. Our contributions in this work are: (1) enhancing RL algorithms with safety filtering, (2) performing ablation

studies on filtering strategies, (3) demonstrating the utility of our method on OpenAI's MuJoCo tasks, and (4) providing the code implementation:
https://github.com/meowatthemoon/DEFENDER.

4.3.1 Proposed Approach

Algorithm 2 RL loop with DEFENDER enhancement

Input: Policy π; Memory β; Dynamics θ; Constant R_{task}; Alignment cost functions: $Safe, Unsafe$; Task environment Env.
while EPISODE not DONE **do**
    $a \leftarrow \pi(s)$
    $\tau \cup$ s or (s, a)
    if $Safe(\tau) < Unsafe(\tau)$ **then**
        $s', r, done = Env(a)$
    else
        $s', r, done = \theta(s, a), R_{task}, True$
    end if
    $\beta \leftarrow (s, a, r, s')$
    Optimize π and θ
end while
return π

We propose DEFENDER: DTW-Based Episode Filtering Using Demonstrations for Enhancing RL Safety, a method to improve the safety of any RL algorithm during learning by leveraging limited demonstration data sets without task-specific constraints. The algorithm keeps track of the current trajectory which is either the sequence of states or state-action pairs. We performed ablation studies to evaluate the type of trajectory which results in better performance. We assume access to a demonstration data set $D_{demo} = \{\tau_i\}^N$ containing N demonstrations, where $N > 1$ is potentially a small number that would not suffice for demonstration learning. The data set must contain both safe and unsafe trajectories. Unsafe trajectories terminated due to reaching unsafe states. Safe trajectories do not have to be perfect but must avoid unsafe states.

DEFENDER measures the alignment cost of the current trajectory at every step with each demonstration from both groups using the DTW algorithm. If the trajectory aligns better with the unsafe group the episode is terminated and the agent is discouraged from re-sampling this trajectory. If the trajectory is terminated, the agent must be encouraged not to sample the same action that would have caused termination for the state. Otherwise, the agent will try again. A negative reward is assigned to discourage sampling the same action again. We use the lowest reward from the reward function of the respective task.

RL algorithms use the next state to perform temporal learning. Since the filtered action is not performed, the agent does not transition to the next state. We propose predicting the

next state using a dynamics model $p_\theta(s_{t+1} \mid s_t, a_t)$. The dynamics model is a simple fully-connected feed-forward network with two hidden layers parameterized by θ trained with L2 loss to imitate the true transition function of the MDP. DEFENDER is summarized in Algorithm 2.

4.3.2 Filtering Strategies

We evaluated different filtering strategies for DEFENDER. One compares the current trajectory with the complete demonstration. A second compares the trajectory with the demonstration using an equal length window. For instance, if the current trajectory has length L, we compare the trajectory with the last L transitions of the demonstration. We also test using a fixed window of 5 and 10 transitions applied to only the trajectory, the demonstration and both. After computing the cost between the trajectory and each demonstration, we obtain a set of values for each group. We test selecting the minimum, maximum and average value from each set. This results in 24 methods to compare the trajectory with a group of demonstrations. We can use one method to compare the trajectory with the safe demonstrations and another method to compare it with the unsafe demonstrations, resulting in 576 filtering strategies.

To determine the best filtering strategy, we trained a SAC agent on three tasks and saved the transitions of the episodes. We simulated how the agent would perform the episodes with each strategy. For each episode, we measured its length with the filter active and divide it by the length of the episode without the filter. We also determined if the filter prevents a crash. At the end, we obtained the average episode length percentages and multiplied it by the safe episode rate. We ranked the strategies by the average score for the three tasks, and selected the top 5 strategies for both state and state-action trajectories.

4.3.3 Experiments

In this section, we evaluate the effectiveness of our method in enhancing the safety of the underlying RL algorithms, SAC [165] and TD3 [260]. We selected three tasks from OpenAI Gym: Hopper, Inverted Double-Pendulum, and Walker2d. The episode horizon for these tasks is 1000 steps, but an episode can end early if the agent reaches an unsafe state, leading to a crash if the transition was not filtered. We measured the performance and safety of RL agents by accumulated reward and crash rate, respectively. Agents were trained for 5000 episodes, and each experiment was repeated three times for seed dependency. We used a learning rate of 3^{-4}, batch size of 256, and all the networks have 2 hidden layers with 256 neurons each. We used 50 demonstrations for both safe and unsafe groups, obtained by training a SAC agent and generating a demonstration after every episode. These tasks are non-visual, however DEFENDER can be applied to visual tasks by utilizing an image encoder to convert images into features.

We trained SAC and TD3 agents with and without our algorithm, using the top 5 filtering strategies from the ablation study for both state and state-action trajectories. Results are

Table 4.3: Performance, safety and computation time of SAC and TD3 agents enhanced with our algorithm using different filters for state trajectories.

Algorithm		Hopper			InvertedDoublePendulum			Walker2d		
		Acc Reward	% Crash	% Time	Acc Reward	% Crash	% Time	Acc Reward	% Crash	% Time
SAC		3474±85.0	64±6.0	100	**9359±0.2**	14±2.7	100	3326±254.6	40±4.0	100
MinDemoW5	MinDemoW10	50±4.0	3±0.2	143	9357±1.5	**1±0.2**	144	4149±106.3	**3±0.6**	157
MeanBothW5	MeanBothW5	3557±50.8	9±0.3	**119**	9357±0.7	21±0.6	**121**	4178±53.7	8±0.3	**122**
MeanBothW10	MeanBothW10	3500±16.4	**8±0.7**	138	9357±0.3	20±0.8	141	4215±76.9	12±0.5	150
MeanDemoW5	MeanDemoW10	**3600±11.9**	11±0.8	143	9357±0.6	22±1.5	144	**4281±23.2**	13±0.1	156
MinDemoW10	MinDemoW10	1±0.0	0±0.0	152	9356±1.5	**1±0.0**	157	3998±105.1	**3±0.4**	168
TD3		2561±1311.6	93±9.9	100	**9355±0.2**	47±7.9	100	4783±230.9	35±8.2	100
MinDemoW5	MinDemoW10	49±9.9	**8±1.7**	198	9353±0.4	**2±0.2**	184	4978±119.1	**6±2.5**	221
MeanBothW5	MeanBothW5	3652±67.4	16±1.9	**143**	9352±1.4	62±3.3	**139**	5122±30.3	13±4.1	**147**
MeanBothW10	MeanBothW10	3690±55.2	14±1.0	188	9352±1.1	55±0.5	177	5159±102.0	15±3.0	204
MeanDemoW5	MeanDemoW10	**3821±47.1**	15±0.2	198	9351±0.5	58±1.5	184	**5263±138.4**	18±2.0	218
MinDemoW10	MinDemoW10	1±0.0	0±0.0	219	9350±0.9	**3±0.5**	208	4571±67.5	**5±1.6**	242

Table 4.4: Performance, safety and computation time of SAC and TD3 agents enhanced with our algorithm using different filters for state-action trajectories.

Algorithm		Hopper			InvertedDoublePendulum			Walker2d		
		Acc Reward	% Crash	% Time	Acc Reward	% Crash	% Time	Acc Reward	% Crash	% Time
SAC		3474±85.0	64±6.0	100	**9359±0.2**	**14±2.7**	100	3326±254.6	40±4.0	100
MeanBothW5	MeanBothW5	3475±17.7	9±0.8	**119**	9357±0.5	21±1.4	**120**	4207±32.8	**11±1.0**	**123**
MeanBothW10	MeanBothW10	3486±17.0	**8±0.1**	141	9356±0.2	21±1.0	141	**4261±13.0**	14±0.4	155
MinTrajW5	MinTrajW10	1±0.0	0±0.0	1119	8±0.0	0±0.0	728	-4±0.0	0±0.0	1642
MinBoth	MinTrajW10	**3602±0.0**	53±8.4	602	201±4.3	1±0.1	196	3878±0.0	24±4.3	824
MinBoth	MinTrajW5	3443±13.1	11±0.2	460	189±19.2	1±0.2	179	3944±91.7	29±9.2	621
TD3		2561±1311.6	93±9.9	100	**9355±0.2**	**47±7.9**	100	4783±230.9	35±8.2	100
MeanBothW5	MeanBothW5	3645±29.8	**14±1.1**	**143**	9351±0.9	59±12.8	**139**	5196±55.0	17±2.4	**149**
MeanBothW10	MeanBothW10	3651±35.1	15±1.5	193	9351±0.3	58±5.2	178	**5253±78.2**	17±1.9	216
MinTrajW5	MinTrajW10	1±0.0	0±0.0	2442	8±0.0	0±0.0	1297	-4±0.0	0±0.0	3345
MinBoth	MinTrajW10	3759±0.0	60±0.3	1253	204±7.5	0±0.0	282	4991±0.0	**13±2.2**	1623
MinBoth	MinTrajW5	**3849±0.0**	68±0.3	928	167±0.0	0±0.0	251	5220±37.3	**13±3.4**	1196

shown in Tables 4.3 and 4.4. Our algorithm incurs extra computational cost which we present in the tables. Overall, both SAC and TD3 agents vary consistently across filtering strategies for the same task. Results show that state trajectories are preferred over state-action trajectories. Not only are they less computationally expensive, but they consistently lead to far fewer crashes for the same task and filtering strategy. Some filtering strategies are too strong and prevent the agent from interacting with the environment and learning the task. Those that allow the agent to interact with the environment lead the agent to the same performance in the case of InvertedDoublePendulum and to higher performance in the case of Hopper and Walker. More importantly, the crash rate is significantly decreased with some exceptions in the InvertedDoublePendulum task. With an appropriate filtering strategy, DEFENDER is able to significantly reduce the crash rate. However, selecting a good filtering strategy for the task may require some trial and error.

Table 4.5: Performance and safety of SAC agent with DEFENDER using and MeanDemoW5 and MeanDemoW10 filters, varying number of demonstrations.

#Demonstrations	Hopper		InvertedDoublePendulum		Walker2d	
	Acc. Reward	% Crash	Acc. Reward	% Crash	Acc. Reward	% Crash
10	3541±25.2	16±0.8	**9356±0.2**	**22±2.1**	4225±46.2	14±0.9
20	3529±13.5	13±1.1	**9356±0.8**	**22±1.7**	4259±26.6	13±1.1
50	**3600±11.9**	**11±0.8**	**9357±0.6**	**22±1.5**	**4281±23.2**	**13±0.1**

Lastly, we evaluated the impact of the number of demonstrations on the performance by training an agent with DEFENDER using 'MeanDemoW5' for the safe set, and 'MeanDemoW10' for the unsafe set as the filtering strategy, varying the number of demonstrations. Results are shown in Table 4.5. DEFENDER is able to increase the agent's safety with small data sets, and it can be further improved by increasing the demonstration data set size.

4.3.4 Conclusions

This chapter introduced two methods that combine RL with DL to tackle the limitations of RL policy learning. Specifically, we tackled the context problem and the safety problem. The context problem of RL is that policies trained under one context fail at inference time if the context information changes. The safety problem is the main factor restricting RL to simulation environments and preventing it from being applied to real-world applications. This is because the policy is estimated using trial and error interactions with the environment. However, failed interactions can lead to catastrophic consequences in the real world, hindering the learning.

First, we presented a framework that uses contrastive learning to obtain viewpoint-invariant state representations from demonstrations. The representations are obtained by contrasting semantically aligned frames from different viewpoints. The semantic alignment is ensured by synchronizing video demonstrations, as is done in [3, 73].

We carefully studied different backbones for the encoders and showed the effects of different design choices. We showed that our framework can correctly align frames between two viewpoints and with fewer training iterations than triplet learning while being lighter to compute and more straightforward to organize the data. We also created a DL data set that can be used to explore other approaches and compare them against our proposal. Lastly, we showed that the representations contain information that can be used for classification purposes and provide a reward function within a RL algorithm to learn a manipulation task. This method can be extended to obtain policies invariant to other types of contextual information such as illumination and background objects through the creation of a demonstration data set where the same state information is captured using multiple images where this information varies.

Next, we introduced the second method in this chapter, the DEFENDER algorithm. This proposed algorithm can be integrated with any RL algorithm for improving the safety of agents during learning. Ablation studies helped identify effective filtering strategies, which we evaluated on state-of-the-art RL algorithms and multiple tasks. The results demonstrate significant enhancement in the safety of the learning agent while maintaining or improving performance. However, there is room for improvement, as the filtering strategy may be overly protective for certain tasks, requiring many iterations to select an

appropriate strategy. Our work highlights the potential of using demonstrations for task-agnostic enhancement of RL algorithm safety.

In the next chapter, we focus on improving the sequence modelling methods in DL by removing the need for a user specified hyper-parameter.

Chapter 5

Hierarchical Sequence Models

5.1 Introduction

RL [8] has showcased remarkable prowess across a broad spectrum of robotic tasks, ranging from object manipulation like pushing and grasping to complex navigational challenges such as path finding and locomotion [261, 262, 206]. However, the inherent trial-and-error nature of online learning, crucial for estimating policies in large state and action spaces, poses a substantial cost in real-world applications. Furthermore, the process of learning a policy through online RL requires the crafting of a tailored reward function specifically designed to guide exploration. This requirement adds another layer of complexity to the problem, due to the difficulty of devising a function that comprehensively covers the entire state and action spaces.

DL emerges as a promising solution to unlock RL's full potential by circumventing both the need for active engagement with an online environment and the creation of a reward function. By leveraging pre-existing demonstration data sets, DL offers a pathway to effective and generalizable policy learning [96]. Specifically, BC formulates the problem of learning the task as a supervised learning problem to imitate the policies represented in data set.

Because the demonstrator was optimizing a reward function, the data set can be used in lieu of a reward function. However, the performance of the agent is dependent on the quality of the data set which is often sub-optimal, unstructured, and diverse due to the difficulty of collecting data. One way to utilize unstructured prior experience is to identify key states that contributed to the success of the trajectory. Hierarchical models are used to solve this issue, where a high-level model identifies sub-goals that the low-level model must reach, guiding it to the final goal and consequently solving the task.

Sequence models have revolutionized many NLP tasks. Specifically, Transformer models have sparked a paradigm shift across various ML domains and have initially been applied to RL in the form of DT [14]. DTs cast RL problems into sequence modelling, empowering agents to assimilate a series of past interactions rather than a single observation, allowing them to make more informed decisions. However, the reward sequence is what guides the agent through the task. The performance of these models relies on the specification of the value of desired accumulated rewards as well as dense reward functions. However, the value of the desired accumulated rewards is non-trivial to determine and can not be arbitrarily high.

In this chapter we propose two sequence modelling methods that improve upon the state-of-the-art DT. First, we propose the HDT method that replaces the need for user interaction and task knowledge by identifying sub-goal states in the data set that guide the agent through the task. We show that these sub-goals can replace the sequence of returns-to-go at guiding the agent, causing the model to be task and user independent while improving performance. Then, we propose a second method that replaces the Transformer architecture of HDT and DT with the Mamba architecture. This change in architecture further improves task performance and inference time, while simplifying the overall model architectures.

5.2 Hierarchical Decision Transformer

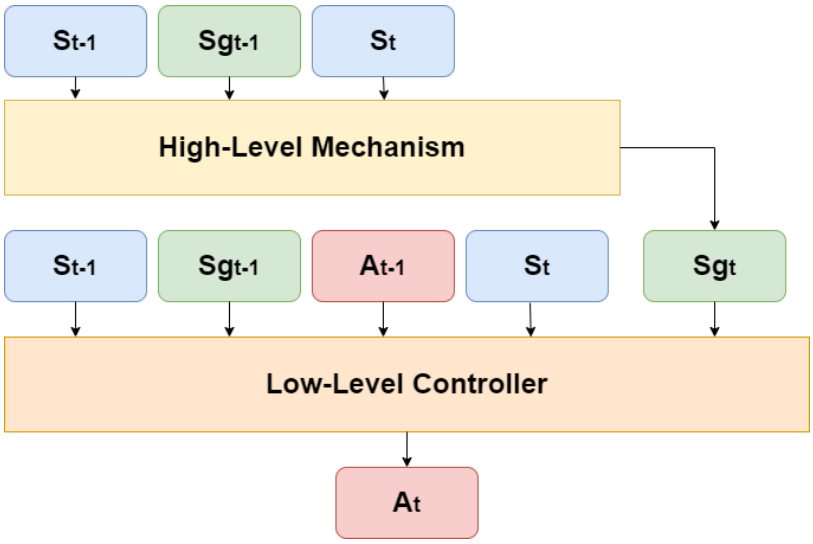


Figure 5.1: The HDT framework. The high-level mechanism guides the low-level controller through the task by selecting sub-goal states, based on the history of sub-goals and states. The low-level controller is conditioned on the history of past states, sub-goals, and actions to select the appropriate action. By reaching each sub-goal, the controller gets closer to completing the task.

We propose the Hierarchical Decision Transformer (HDT), a dual decision transformer architecture that scales transformers in DL to tasks with long episodes and removes the need for the specification of desired rewards by a user. We propose a method for selecting sub-goal states from the demonstration data set. The high-level transformer predicts key sub-goal states, identified from the demonstration data set, based on the sequence of prior states. The low-level transformer is conditioned on the sequence of sub-goals to perform the task. By successfully reaching each sub-goal, the agent completes the task. Unlike the state-of-the-art DT [14], the HDT does not require any prior task knowledge and user interaction. We conduct experiments on several tasks from D4RL, OpenAI Gym, and RoboMimic benchmarks. Our results show that the performance of the DT is dependent on the careful specification of desired rewards. We show that the HDT successfully replaces the need for specifying desired rewards with the added benefit of an increase in

performance, particularly in long tasks or environments with sparse rewards, when compared with the state-of-the-art.

Summary of Contributions:

1. We present the HDT, a dual transformer framework that enables offline learning from a large set of diverse and sub-optimal demonstrations by selectively selecting sub-goal states from the data set.

2. We present a method for sampling sub-goal states from a trajectory. Experiments show that the sequence of sub-goals can guide the agent through the task in place of the sequence of returns-to-go.

3. We evaluate the HDT on ten environments and data sets from the D4RL [5], OpenAI Gym [6] and RoboMimic [7] benchmarks. We also validate the HDT on a reaching task using a physical robot. Our method outperforms the original DT baseline, especially in tasks with longer episodes, proving that the sub-goal sequence can replace the reward function. The code can be found at https://github.com/meowatthemoon/HDT.

Algorithm 3 HDT training algorithm.

Input: Demonstration data set $D = \{\tau_1, ..., \tau_N\}$, batch size B, token size K, training iterations I, high-level mechanism π_ϕ, low-level controller π_θ, sub-goal selection method f.
for $i = 1$ **to** I **do**
  $\mathbf{s}, \mathbf{a}, \mathbf{t}, \mathbf{sg}, \mathbf{mask} \leftarrow$ Sample B sequences from D.
  $sg' \leftarrow \pi_\phi(\mathbf{s}, \mathbf{sg}, \mathbf{t}, \mathbf{mask})$, obtain sub-goal predictions.
  Update ϕ by minimizing $L_\phi(sg, sg')$.
  $a' \leftarrow \pi_\theta(\mathbf{s}, \mathbf{sg}, \mathbf{a}, \mathbf{t}, \mathbf{mask})$, obtain action predictions.
  Update θ by minimizing $L_\theta(a, a')$.
end for
return π_θ, π_ϕ

5.2.1 Overview

We present the HDT, represented in Fig. 5.1. The HDT is a hierarchical BC algorithm that improves the performance of transformer methods in DL, improving their robustness to tasks with longer episodes and/or sparse rewards, without requiring task knowledge or user interaction currently present in the state-of-the-art.

Here we first provide an overview of the HDT and the motivation of each component. We split the decision-making process of the agent into a high-level mechanism that sets sub-goal states for a low-level controller to try and reach. We use the GPT 2 [20] transformer architecture, the same used by the DT [14], for both the high-level mechanism and the low-level controller. The high-level mechanism is conditioned on the state sequence and aims to produce sub-goal states for the low-level controller to reach, guiding

it through the task. The low-level controller is conditioned on the sequence of sub-goals produced by the high-level model to predict the correct action.

The algorithm learns solely from demonstration data present in a data set. The data set is first processed, where for each state, a sub-goal state is selected from the trajectory using the sub-goal selection algorithm we present. The original transition is then augmented with the selected sub-goal state. Then, both the high-level mechanism and low-level controller are trained simultaneously using the sampled batches of sequences. The high-level mechanism receives sequences of the previous states and sub-goals and tries to predict the next sub-goal state in the sequence. The low-level controller receives sequences of the previous states, sub-goals, and actions, and tries to predict the next action in the sequence. Because both models are transformers, they also receive the sequence of time steps and the mask for positional encoding and to hide the future tokens from the decoder, respectively.

The complete training loop is provided in Algorithm 3. We encourage training both models simultaneously to avoid repeating steps, such as batch sampling. However, because both models do not rely on data produced by the other to learn, they can be trained sequentially. Next, we further explain each component.

5.2.2 Sub-Goal Selection

Our method requires sequences of states, actions, time steps, and sub-goals. To obtain these sequences, it processes the demonstration data set before training. The sequences of states, actions, and time steps can be directly extracted from the data set. However, the sub-goal sequences are not explicitly present in the data set and must be inferred. We define sub-goals sg in regards to the current state s_t as later states in the trajectory, which are highly valuable for the agent to reach. These states should mark milestones in the trajectory which when reached sequentially, would make it highly probable that the agent successfully performs the task.

Using this definition, the sub-goals guide the agent through the task, meaning they have the same purpose as the returns-to-go. Therefore, we replace the returns-to-go sequences with the sub-goal sequences. This way, the user does not need to iterate over the value of the desired returns for each task in order for the method to perform. Additionally, it replaces null rewards in tasks with sparse rewards, consequently improving the learning process. We perform experiments to show that the original DT is dependent on the returns-to-go sequences to learn and that the value of the desired returns is important.

By our definition, a sub-goal state sg has a high value for the success of the trajectory. We can model this behavior by finding a future state s_j in the trajectory with high values of accumulated rewards since the current state $W(s_j) = \sum_{k=i+1}^{j} r_k$. However, this would always prioritize the last states of the trajectory. To encourage selecting states close to

the current state, we divide the accumulated rewards by the distance between the states: $W(s_j) = \sum_{k=i+1}^{j} \frac{r_k}{j-i}$.

The weighted average of the accumulated rewards in a sub-trajectory identifies sub-goal states that have correctly completed a part of the task resulting in a significant reward, without always selecting far away states, by punishing the length of the sub-trajectory. For each state-action pair, we select the state with the highest associated weight to be the sub-goal. The result is a data set of M trajectories $D = \{\tau_1, ..., \tau_M\}$, where each trajectory $\tau_j = \{(s_i, a_i, sg_i), i \in N\}$, where N is the length of the trajectory.

5.2.3 Low-Level Controller

The low-level controller is a goal-conditioned transformer π_θ. Similar to DT [14], we model the demonstration trajectories as sequences of tokens for the transformer to learn from. The low-level controller receives sequences of states, actions, and sub-goals and tries to predict the next action. During training, sequences of size K are randomly sampled in a batch from the enhanced data set, where K indicates the number of tokens in a sequence. The low-level transformer is trained to minimize the BC loss:

$$\underset{\theta}{\mathbb{L}}(s_{t:t+K}, a_{t:t+K}, sg_{t:t+K}) = \|a_{t:t+K} - \pi_\theta(s_{t:t+K}, a_{t:t+K}, sg_{t:t+K})\| \quad (5.1)$$

In practice, this corresponds to calculating the mean squared error between the generated action and the action in the data set. For each sequence, the model also receives the respective token time steps for positional encoding and a mask to hide the future tokens from the decoder. We do not include them in the formulas for simplicity. At test time, the sequences are built by keeping a history of past states, actions, and time steps from the environment and past sub-goals selected from the high-level mechanism. Hence, the new sequence model does not require any task knowledge to be provided by a user.

5.2.4 High-Level Mechanism

The high-level mechanism chooses sub-goal states for the low-level controller to try and reach. It is a state-conditioned transformer π_ϕ. The high-level mechanism receives the sequence of states, past sub-goals, time steps, and a mask. The mechanism aims to predict the next sub-goal state which will help the low-level controller succeed in the task. During training, the high-level mechanism is trained to minimize its own BC loss:

$$\underset{\phi}{\mathbb{L}}(s_{t:t+K}, sg_{t:t+K}) = \|sg_{t:t+K} - \pi_\phi(s_{t:t+K}, sg_{t:t+K})\| \quad (5.2)$$

In practice, this corresponds to calculating the mean squared error between the generated sub-goal state and the ground-truth sub-goal state in the data set.

Table 5.1: Maximum accumulated returns of the original DT and of a DT variant without the desired returns input sequence trained for 100 thousand iterations.

Task	Data Set	Desired Return	Original DT	DT w/o Desired Returns
Half-Cheetah	medium	2655	**5095**	47
		5309	**5092**	47
		10000	**4953**	47
Hopper	medium	1611	**2303**	300
		3222	**2557**	300
		10000	**2036**	300
Kitchen	complete	2	**2.4**	1.0
		4	**2.5**	1.0
		10000	**2.1**	1.0
Walker-2D	medium	2113	**3619**	149
		4227	**3711**	149
		10000	**3100**	149

5.2.5 Experiments

In this section, we evaluate the performance of the HDT and compare it with the state-of-the-art method, DT [14], and BC baseline. We evaluate the models on ten tasks from OpenAI Gym [6], D4RL [5] and RoboMimic [7] benchmarks. The tasks range from short episodes to long episodes and from frequent rewards to sparse reward settings. We train the algorithms on the different data sets provided by the benchmarks for each task. For the DT, we first find the maximum accumulated returns collected by a trajectory in the demonstration data set. We then set the desired returns to this value and also to half, following the original tests of the DT. We train each model for 100 thousand epochs, using batch sizes of 64, a learning rate of $1e^{-4}$, and a sequence length of 20 tokens. To reduce the impact of outliers and because the episodes are seed-dependent, for every one thousand epochs we validate the model on 100 episodes and calculate the average accumulated rewards. In the tables, we present the highest average accumulated rewards seen throughout the 100 thousand epochs.

5.2.5.1 Does the DT rely on desired returns?

One of the motivations of our work is to remove the requirement of specifying the value of the desired returns in transformer models in DL, which is present in the state-of-the-art, the DT. We first need to know whether the DT truly depends on the returns-to-go sequence to perform or if they can simply be removed from the model's input. Table 5.1 shows the accumulated returns obtained by the original DT model and by a DT model variant without the desired returns sequence. For the original DT, we set the value of the desired returns for the task to the maximum value of accumulated rewards present in the data set and to half of this value. We also set the value of the desired return to an arbitrarily high value, for example 10000.

The results in Table 5.1 show that the DT model is dependent on the returns-to-go sequence to perform. Removing them prevents DT from learning the task. Additionally,

Table 5.2: Maximum accumulated returns of the HDT and of a HDT variant including the desired returns input sequence trained for 100 thousand iterations.

Task	Data Set	Desired Return	HDT	HDT with Desired Returns
Half-Cheetah	medium	2655	**5205**	5004
		5309	**5205**	5002
Hopper	medium	1611	**3071**	1213
		3222	**3071**	1198
Kitchen	complete	2	**2.6**	2.1
		4	**2.6**	1.4
Walker 2D	medium	2113	**3879**	3645
		4227	**3879**	3516

it also shows that the value of the desired returns is not obvious. Because the DT only reaches the desired return of half of the maximum value on the kitchen task. Then, when doubling the value of the desired returns, it only slightly increases the performance. Lastly, when using an arbitrarily high value, the DT underperforms. In conclusion, the value of the desired returns is crucial for the performance of the DT model while being non-trivial to determine. This means that the state-of-the-art transformer model in DL relies on task knowledge and user interaction to perform.

5.2.5.2 Does the HDT replace the need for desired returns?

Next, we test whether the high-level mechanism, by selecting sub-goals, replaces the need for the sequence of desired returns, in regards to guiding the agent through the task. We test this hypothesis by adding the sequences of desired returns as an extra input to the low-level controller and compare the performance with our base method without this extra input. Similarly to the previous experiment, we test the performance by setting the value of the desired returns to the maximum accumulated reward present in the data set and to half of this value. Table 5.2 shows the accumulated returns obtained by our base HDT model and by the HDT model variant with the extra sequence of desired returns.

The results show that the HDT performs slightly better without the desired returns sequence. This means that the high-level mechanism is able to output sub-goal states from the history of past states. Moreover, such generated sequences are capable of guiding the low-level controller through the task successfully replacing the need for the sequence returns-to-go. Consequently, the HDT removes the requirement for external knowledge about the specific task present in DT.

5.2.5.3 Baseline Comparison

Next, we compare the performance of the HDT on ten tasks against the DT and the BC baselines and present the results in Table 5.3. For BC, we used an MLP with two fully connected layers. Because the three methods clone the behavior present in the data set, their performance is limited by the quality of the demonstrations. Hence, we include the average accumulated rewards collected by the demonstrator in the data set. The HDT

Table 5.3: Maximum accumulated returns of the HDT compared to the DT and BC baselines. We test the models on ten tasks from D4RL [5], OpenAI Gym [6], and RoboMimic [7] benchmarks and varying demonstration data sets.

Task	Data set	Data Avg.	BC	DT	HDT
Ant	expert	4621	5261	5319	**5365**
	medium	3051	3952	3943	**3993**
	replay	390	3244	**3624**	3325
Door	cloned	304	-53	**758**	275
	expert	2915	-53	3053	**3056**
	human	796	363	**459**	338
Half-Cheetah	expert	10656	10549	10984	**11092**
	medium	4770	5198	5095	**5205**
	replay	3093	0	**4567**	4412
Hammer	cloned	786	-202	118	**775**
	expert	12393	16399	16492	**16507**
	human	3072	50	-68	**3115**
Hopper	expert	3511	3467	3626	**3654**
	medium	1422	1969	2557	**3071**
	replay	467	1017	**2762**	1169
Kitchen	complete	325	0.8	2.5	**2.6**
	mixed	302	0.6	2.1	**2.7**
	partial	255	0.6	**2.7**	**2.7**
Maze 2D	large	2	126	31	**145**
	medium	4	37	68	**188**
	open	6	72	57	**75**
	umaze	7	49	54	**228**
Pen	cloned	3334	1561	3716	**3961**
	expert	3297	2540	4551	**4696**
	human	6326	867	3606	**3897**
Relocate	cloned	1223	3	50	**58**
	expert	4329	3730	**4697**	4624
	human	3674	**232**	35	35
Walker 2D	expert	4921	4261	5063	**5065**
	medium	2852	3723	3711	**3879**
	replay	896	876	**3627**	3325
		Count:	1	8	23

outperforms the baselines in twenty three out of thirty one settings. This means that our method successfully improved upon the original DT in raw performance while also removing its dependency on desired reward specification, making it task-independent. Particularly, the Maze 2D constitutes a task with sparse rewards where on average 90% of the transitions receive no reward. Similarly, in the kitchen task, the agent only receives a reward after completing each of the four sub-tasks. However, the data sets for the Kitchen task are augmented to include extra rewards. The HDT vastly outperforms the baselines in the Maze 2D task and also outperforms the baselines in the kitchen task. This shows that the sequence of sub-goals provides a stronger guiding signal for the model than the sequence of returns-to-go. This means that the HDT is more robust to tasks with sparse rewards. For tasks with longer horizons, such as half-cheetah and walker 2D, the HDT also surpasses the two baselines. The discrepancy between the transformer methods and BC is more prominent in these types of tasks. We conclude that our method offers superior performance overall, and is more robust to both long and sparse-reward tasks.

Table 5.4: Number of goal positions achieved by the HDT and DT on UR3 reaching task, varying the number of train demonstrations.

Train Demos	10	20	30	40	50	60	70	80
HDT	1.4 ±1.6	**2.7 ±2.3**	**2.3 ±2.3**	**3.6 ±2.2**	**4.4 ±1.3**	**4.5 ±1.5**	**5 ±0**	**5 ±0**
DT	**2.1 ±1.2**	0.3 ±0.6	1 ±2	3.2 ±2.1	2.6 ±2.4	1.5 ±2.3	3 ±2.4	4.5 ±1.5

5.2.5.4 UR3 Reaching Task

We evaluate the HDT by performing an experiment on a physical UR3 robot performing a reaching task. We defined five positions that the robot must reach in any sequence. A state is defined by the six joint angles and a one hot-encoded vector representing the goal positions reached. An action is the translation of the angle of each joint, and the reward is the negative absolute error between the angles of the current joints and the nearest goal position. We generated 100 demonstrations, each starting the robot at a random position and teleoperating the robot through the sequence of goal positions. 80 of the demonstrations are used for training the methods, and 10 for validation. We also evaluate the performance of the algorithms when trained with fewer demonstrations. The remaining 10 demonstrations are used for setting the robot's initial position during the test phase. During testing, we run the algorithms for a maximum of 200 steps and measure the number of goal positions reached. The results are presented in Table 5.4. Results show that the HDT reaches the five goal positions on all 10 initial positions when trained with 70 demonstrations. The DT is never able to consistently reach the five positions. Moreover, the HDT consistently outperforms the DT when trained with the same number of demonstrations. This shows that the HDT can be applicable to tasks using a physical robot and improves the performance over the DT, especially in long-horizon tasks, while not requiring human intervention.

5.3 Decision Mamba Architectures

Recently, SSSMs [157], have garnered attention for their linear scalability with sequence length, outperforming Transformers in a wide range of domains. We identify that the evolutionary parameter of SSSMs provides the guidance signal that the sequence of rewards offers the DT. For this reason we propose to replace the Transformer architecture of both the DT and the HDT resulting in the Decision Mamba (DM) and the Hierarchical Decision Mamba (HDM) models. Specifically for the DM, we show that it can provide the sequence modelling advantages of the original the DT without requiring the sequence of rewards which requires task knowledge and user intervention as well as causing it to fail in stochastic environments.

We conduct experiments on seven tasks from D4RL benchmark. Our results show that the performance of the DM is no longer dependent on the sequence of rewards, requiring no user intervention or task knowledge. Unlike the other methods, the DM does not require access to the reward function. Both DM and HDM outperform their Transformer state-of-the-art predecessors. The architectures of DM and HDM are represented in Fig.

5.2.

Summary of Contributions:

1. We present the HDM and DM, the Mamba successors of HDT and DT, respectively. The models are smaller, faster and more accurate than their Transformer predecessors.

2. We show that DM does not require the sequence of rewards to perform, unlike its predecessor DT. This removes the need for a reward function, task knowledge to craft an adequate desired reward value and user intervention.

3. We evaluate the HDM and DM on seven environments and data sets from the D4RL [5] benchmark. Our methods outperform the original Transformer baselines, proving the viability of the Mamba architecture for imitation learning. Recent advancements in imitation learning have been largely fueled by the integration of sequence models, which provide a structured flow of information to effectively mimic task behaviors. Currently, DT and subsequently, the HDT, presented Transformer-based approaches to learn task policies. Recently, the Mamba architecture has shown to outperform Transformers across various task domains. In this work, we introduce two novel methods, DM and HDM, aimed at enhancing the performance of the Transformer models. Through extensive experimentation across diverse environments such as OpenAI Gym and D4RL, leveraging varying demonstration data sets, we demonstrate the superiority of Mamba models over their Transformer counterparts in a majority of tasks. Results show that DM outperforms other methods in most settings.

5.3.1 Decision Mamba

We assume to have access to a demonstration data set D composed of trajectories with states, actions and rewards. Similarly to the DT, we condition the model on the sequences of states actions and RTG, each sequence with a context length of K. However, we perform experiments that show that the DM is also able to perform without the sequence of RTG, while improving the performance. During batch data sampling, a demonstration trajectory is randomly selected, followed by the choice of a trajectory index $t \in [0, T-1]$, where T is the length of the trajectory. The $K-1$ states, actions and RTG preceding t compose the three sequences. We also sample a right-shifted sequence of actions. This sequence corresponds to the ground-truth actions that the model should predict. The sequences are then padded with zero-filled vectors to the right, if their length is smaller than the context length. Because of this padding, we create a binary padding mask. This is a sequence of length K with zeros in indexes where the previous sequences were padded. This padding mask is solely used to prevent the loss function from considering padded values, and is not used by the DM model or during inference.

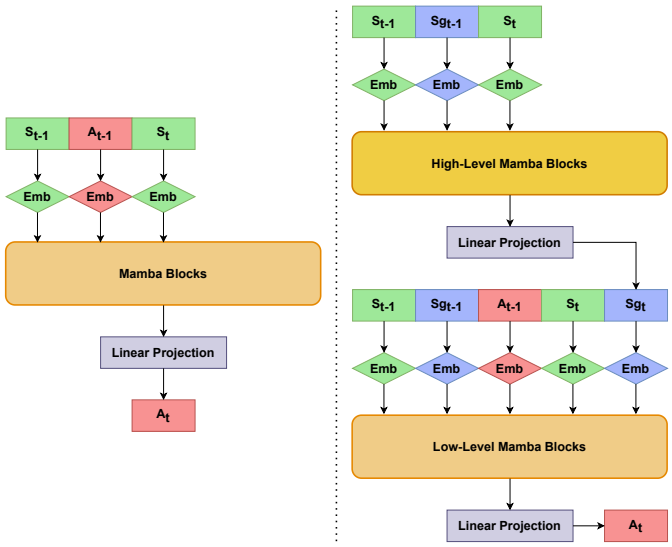


Figure 5.2: The DM architecture on the left and the HDM architecture on the right side. The DM is conditioned on the sequence of past states and actions to predict the correct action. The HDM is composed of two modules. The high-level mechanism guides the low-level controller through the task by selecting sub-goal states, based on the history of sub-goals and states. The low-level controller is conditioned on the history of past states, sub-goals, and actions to select the appropriate action.

Each of the three input sequences pass through individual linear layers, converting them to the embedding dimension used by the model. The resulting embeddings are joined to form the sequential input of the model in an identical way to what is done in the DT. The difference is that the DM does not require the causal mask of the Transformer, and Mamba provides sequential information. Therefore, we can skip a significant number of steps. We don't need to sample an additional sequence time steps from the data set, that would be used to generate a positional encoding, and this encoding would then be summed to the three embeddings.

The DM employs a Mamba architecture with repeating Mamba layers each receiving the embeddings of the previous layer and outputting new embedding vectors. We perform tests varying the number of layers, the embedding size, and the context length. There wasn't an overall configuration that resulted in superior performance across all the tasks. In the experiments section, we use 6 Mamba layers, an embedding size of 128, and a context length of 20. The embeddings returned by the final Mamba layer pass through a feed-forward layer to project them to the action space of the task. Subsequently, a hyperbolic tangent activation function normalizes the values between -1 and 1, which are then multiplied by the respective action range of the task.

The DM's objective is to predict the corresponding ground-truth actions in the data set after being conditioned on the sequence of past states, actions and optionally RTG. Because of this, the model's objective is the L2 loss between the predicted action sequence and the ground-truth action sequence.

Table 5.5: Maximum accumulated returns of the DT, HDT, DM, DM with RTG, and HDM methods using 6 layers, an embedding size of 128 and context length of 20. We test the models on seven tasks from the D4RL [5] benchmark and vary the demonstration data sets.

Task	Data Set	DT			HDT	DM w/ R			DM wo/ R	HDM
		Half	Max	10k		Half	Max	10k		
Ant	expert	2722.84 ± 9.27	2731.52 ± 3.91	2733.84 ± 9.3	2731.83 ± 8.38	1853.28 ± 7.54	2735.34 ± 11.45	2742.30 ±1 14.37	2746.57 ± 10.69	**2749.2 ± 11.62**
	medium	787.54 ± 22.87	801.41 ± 14.61	800.91 ± 17.32	787.23 ± 33.33	760.23 ± 21.73	782.97 ± 13.83	803.63 ± 14.76	**821.03 ± 18.21**	820.39 ± 23.41
	medium-expert	1525.61 ± 724.43	2718.68 ± 11.66	2713.55 ± 15.06	2171.27 ± 792.37	1769.70 ± 18.46	**2742.88 ± 13.64**	2734.04 ± 15.92	2738.41 ± 8.42	2730.01 ± 12.45
	medium-replay	650.23 ± 63.67	732.89 ± 13.76	**815.53 ± 42.72**	712.61 ± 39.5	639.22 ± 20.49	694.40 ± 13.69	716.67 ± 37.54	716.85 ± 33.65	728.2 ± 48.64
Antmaze	large-diverse	0.02 ± 0.04	0.05 ± 0.05	0.05 ± 0.05	0.02 ± 0.04	0.0 ± 0.01	0.0 ± 0.01	0.0 ± 0.01	**0.08 ± 0.13**	0.05 ± 0.05
	medium-diverse	0.2 ± 0.17	0.18 ± 0.19	0.2 ± 0.17	0.12 ± 0.04	0.0 ± 0.05	0.1 ± 0.05	**0.5 ± 0.05**	0.2 ± 0.17	0.15 ± 0.05
	umaze	0.92 ± 0.04	0.92 ± 0.04	0.92 ± 0.04	0.52 ± 0.25	0.8 ± 0.04	0.8 ± 0.04	0.9 ± 0.04	0.95 ± 0.05	**1.0 ± 0.0**
	umaze-diverse	0.92 ± 0.04	0.95 ± 0.05	0.9 ± 0.07	0.50 ± 0.25	0.9 ± 0.04	0.9 ± 0.07	0.9 ± 0.05	0.92 ± 0.04	**1.0 ± 0.0**
HalfCheetah	expert	1516.19 ± 25.85	1530.86 ± 21.28	1497.99 ± 48.14	1479.88 ± 22.09	1025.66 ± 23.74	1570.21 ± 27.80	1573.29 ± 46.12	**1585.85 ± 19.24**	1538.32 ± 50.83
	medium	655.96 ± 8.55	658.54 ± 2.02	667.08 ± 16.17	669.73 ± 15.74	597.75 ± 7.69	676.78 ± 3.14	664.15 ± 15.39	677.84 ± 13.6	**679.03 ± 14.74**
	medium-expert	1432.58 ± 158.2	1572.42 ± 68.04	1619.32 ± 43.02	1103.51 ± 141.89	1097.76 ± 68.47	1541.82 ± 58.53	1566.91 ± 83.36	**1636.29 ± 27.28**	1541.19 ± 73.06
	medium-replay	806.58 ± 175.57	1074.72 ± 11.14	**1100.29 ± 15.2**	946.68 ± 91.13	607.06 ± 110.68	749.42 ± 13.93	916.15 ± 67.71	862.53 ± 35.02	826.06 ± 96.8
Hopper	expert	2129.4 ± 116.64	2234.25 ± 54.94	2239.86 ± 58.55	2072.71 ± 69.58	1211.95 ± 123.71	2183.71 ± 58.71	2218.42 ± 64.48	**2260.68 ± 22.36**	2230.41 ± 65.08
	medium	1144.2 ± 100.69	1241.6 ± 188.58	1252.95 ± 99.69	1175.3 ± 63.39	1101.02 ± 97.47	1299.97 ± 143.74	1289.51 ± 89.36	**1383.51 ± 127.34**	1231.19 ± 108.38
	medium-expert	2056.43 ± 166.75	**2270.67 ± 62.89**	2259.2 ± 117.52	1549.4 ± 112.42	1128.34 ± 128.57	1776.79 ± 86.91	1914.85 ± 105.72	2026.48 ± 180.21	1948.04 ± 342.99
	medium-replay	636.21 ± 91.45	564.7 ± 99.69	648.13 ± 172.74	378.57 ± 176.15	822.5 ± 86.30	1065.73 ± 90.73	1175.59 ± 95.76	**1199.42 ± 192.37**	770.51 ± 95.49
Kitchen	complete	2.52 ± 0.18	2.53 ± 0.13	2.42 ± 0.38	2.7 ± 0.23	**3.1 +0.16**	2.5 ± 0.09	3.0 ± 0.02	2.58 ± 0.36	2.2 ± 0.14
	mixed	2.28 ± 0.27	2.55 ± 0.21	2.15 ± 0.34	2.28 ± 0.13	2.5 ± 0.26	2.7 ± 0.18	2.5 ± 0.37	**2.8 ± 0.2**	2.65 ± 0.09
	partial	2.65 ± 0.27	3.0 ± 0.07	2.08 ± 0.51	2.42 ± 0.44	2.1 ± 0.25	3.0 ± 0.03	3.0 ± 0.02	**3.02 ± 0.04**	2.55 ± 0.43
Maze2D	large	106.72 ± 17.23	103.7 ± 15.03	103.45 ± 19.85	34.1 ± 8.97	32.7 ± 16.05	69.9 ± 14.08	**110.8 ± 17.74**	79.65 ± 20.43	93.7 ± 71.77
	medium	40.07 ± 9.19	33.72 ± 13.3	**154.0 ± 63.45**	33.72 ± 7.87	37.8 ± 5.34	56.3 ± 5.35	111.9 ± 45.34	114.1 ± 75.94	145.23 ± 20.81
	open	16.62 ± 0.48	17.17 ± 1.49	15.85 ± 3.61	19.08 ± 2.17	25.7 ± 1.64	30.2 ± 8.63	**32.4 ± 4.56**	32.33 ± 2.05	26.67 ± 0.76
	umaze	58.68 ± 20.14	59.3 ± 20.61	61.2 ± 19.74	37.72 ± 11.7	58.8 ± 17.03	86.9 ± 10.47	**186.1 ± 16.43**	110.25 ± 51.04	31.13 ± 2.58
Walker2D	expert	1863.25 ± 11.68	1867.61 ± 9.43	1877.92 ± 5.12	1869.07 ± 8.54	1088.01 ± 14.72	1866.81 ± 4.72	**1881.46 ± 9.46**	1874.51 ± 10.86	1861.86 ± 7.2
	medium	1070.96 ± 61.03	1081.54 ± 42.68	**1175.83 ± 58.49**	1106.63 ± 24.09	932.2 ± 40.81	1008.24 ± 54.91	1088.64 ± 43.09	1111.94 ± 32.08	1093.42 ± 36.83
	medium-expert	1310.26 ± 352.94	**1883.23 ± 19.77**	1880.87 ± 24.98	1153.5 ± 54.57	1055.61 ± 35.89	1861.45 ± 23.23	1866.47 ± 18.67	1871.34 ± 4.88	1843.48 ± 11.49
	medium-replay	1129.27 ± 129.82	1335.64 ± 10.87	**1377.9 ± 21.19**	1197.66 ± 51.96	716.23 ± 78.38	1302.54 ± 12.94	1330.53 ± 34.71	1310.88 ± 32.54	1237.97 ± 63.21

5.3.2 Hierarchical Decision Mamba

For the HDM, we follow the same data processing pipeline of the HDT and replace the sequences of RTG with sequences of sub-goals. The HDM is also composed of two models: the high-level mechanism and low-level controller. Similarly to the DM, during batch sampling we sample padded sequences of states, actions and sub-goals of length K from the data set and create the padding mask. The high-level mechanism receives the sequences of states and sub-goals and predicts new sub-goals, while the low-level controller receives the sequences of states, actions and sub-goals and predicts new actions. Therefore, we also sample right-shifted sequence of actions and right-shifted sequence of sub-goals, as the ground-truth for the low-level, and high-level model respectively.

The sequences of states and sub-goals each pass through a linear layer, converting them to the embedding dimension used by the model. They are then joined and passed to the Mamba layers of the high-level mechanism. The output vectors of the final Mamba layer are passed to a linear layer which projects them to the state space of the task. Then, a hyperbolic tangent activation function normalizes the values between -1 and 1. Following the HDT, the states of the data set are normalized between -1 and 1. The high-level mechanism is trained using the L2 loss between the predicted sub-goal sequence and the ground-truth sub-goal sequence.

The training of the low-level controller is similar to the training of the DM. Instead of receiving a sequence of RTG, it receives a sequence of sub-goals. The low-level controller is trained using the L2 loss between the predicted sequence of actions and the ground-truth action sequence.

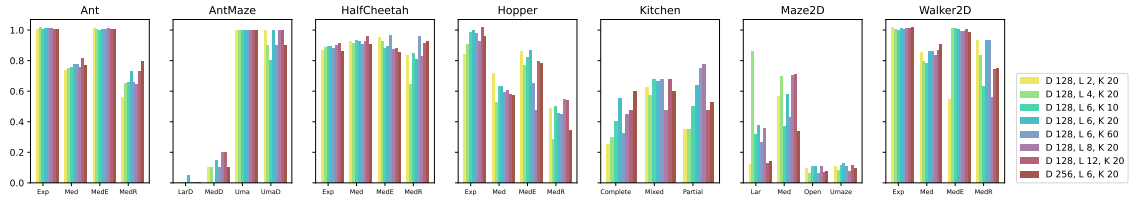


Figure 5.3: Comparison of the performance of the HDM varying architecture configuration across the 7 D4RL tasks, for different demonstration data sets. The scale of the bar graphs is the maximum reward present in the respective data set. L is the number of layers, D is the embedding size, and K is the context length.

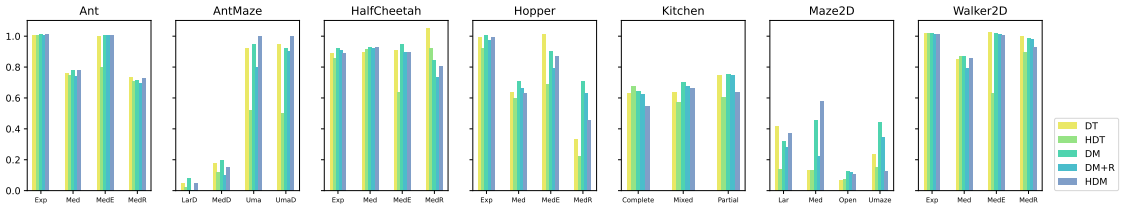


Figure 5.4: Comparison of the 5 methods across the 7 D4RL tasks, for different the demonstration data sets. The scale of the bar graphs is the maximum reward present in the respective data set. All models have 6 layers, an embedding size of 128 and use context length of 20. The values of the DM and the DT are obtained by using the maximum reward of the data set as the desired reward.

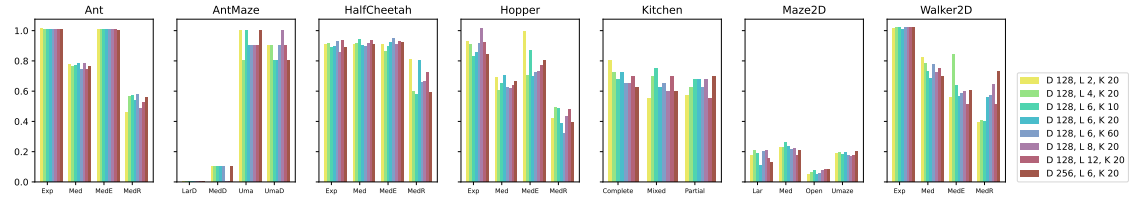


Figure 5.5: Comparison of the performance of the DM varying architecture configuration across the 7 D4RL tasks, for different demonstration data sets. The scale of the bar graphs is the maximum reward present in the respective data set. L is the number of layers, D is the embedding size, and K is the context length.

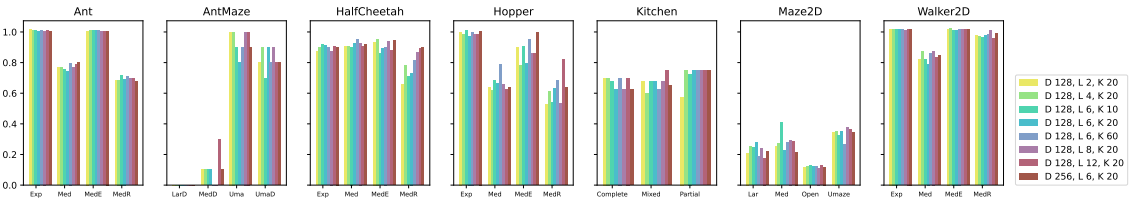


Figure 5.6: Comparison of the performance of the DM with the sequence of RTG, varying architecture configuration across the 7 D4RL tasks, for different demonstration data sets. The scale of the bar graphs is the maximum reward present in the respective data set. L is the number of layers, D is the embedding size, and K is the context length. The values are obtained by using the maximum reward of the data set as the desired reward.

5.3.3 Experiments

In this section, we assess the performance of the proposed DM and HDM models, comparing them with Transformer-based methods, the DT and the HDT. Our evaluation spans over seven distinct tasks sourced from the D4RL benchmark, selected precisely for its provision of a diverse set of tasks, each accompanied by multiple demonstration data

sets of varying quality. As the efficacy of these methods relies heavily on their capacity to approximate policies represented in the demonstration data set, the resulting performance of the methods is intrinsically linked to the quality of these data sets. Consequently, we conduct our training procedures using the diverse data sets provided by the benchmark for each respective task.

We train each model for 1 million epochs, using batch sizes of 16, and a learning rate of $1e^{-4}$. To mitigate the influence of outliers and the inherent seed-dependency of episodes, we adopt a validation strategy wherein, every one thousand epochs, we validate the model on 100 episodes, and compute the average accumulated rewards. Due to the seed dependency, we repeat each experiment across 4 different random seeds. The presented results are the average values across the 4 seeds of the highest accumulated rewards seen throughout the 1 million epochs.

We also varied the number of Mamba and Transformer layers, the embedding size inside the model, and the length of the token sequence. Results for the DM, the DM with RTG, and the HDM are shown in Fig. 5.3, Fig. 5.5 and Fig. 5.6, respectively. The values of the DM with RTG were obtained using a desired reward equal to the maximum reward present in the respective demonstration data set. However, results were unclear, as there wasn't a best performing configuration for any of the models. Different configurations resulted in better or worse performance for the models on the different tasks and data sets. To fairly compare the models using the same architecture, we chose the architecture that better represented the average results across the set of architectures. This architecture is composed of 6 layers (Transformer or Mamba), an embedding size of 128 and a sequence length of 20. For the execution of the DT and DM using the sequence of RTG, we initially identify the maximum accumulated returns attained by a trajectory in the demonstration data set. Subsequently, during validation, we set the desired returns to this maximum value, half of it, and a larger value—specifically, 10k. The results are presented in Table 5.5.

One of the driving motivations behind the development of the HDT was to alleviate the necessity of manually specifying the desired RTG, a notable challenge encountered in the evaluation and deployment of the DTs. To determine whether the DM inherits this drawback from the DT, we evaluate whether it also relies on the RTG sequence to guide the model, or if the sequence can be simply removed from the model's input. Table 5.5 presents the accumulated returns achieved by the DM model without the desired returns sequence, compared with a variant of the DM model with this additional sequence.

The results depicted in the table highlight that the DM does not require the sequence of RTG for effective performance. Additionally, results also show that the DM without the sequence of RTG does seem to reach a slightly better performance than with the sequence. This is also true across different architectures as shown in Fig. 5.5. Notably, in the DT,

Table 5.6: Average and STD time for a single training iteration, and to perform inference of an episode, across the different D4RL tasks, using a batch size of 16, of each of the 5 methods, configured with 6 layers, an embedding size of 128 and sequence length 20.

	Train Time (ms)					Inference Time (ms)				
	DT	**HDT**	**DM wo/ R**	**DM w/ R**	**HDM**	**DT**	**HDT**	**DM wo/ R**	**DM w/ R**	**HDM**
Ant	15 ± 11	20 ± 11	18 ± 102	18 ± 97	26 ± 101	5 ± 7	7 ± 1	3 ± 4	3 ± 4	5 ± 8
Antmaze	8 ± 0	14 ± 0	7 ± 0	8 ± 0	15 ± 1	4 ± 5	6 ± 10	3 ± 3	3 ± 4	5 ± 5
HalfCheetah	14 ± 1	19 ± 1	14 ± 1	15 ± 1	22 ± 1	5 ± 6	7 ± 13	3 ± 4	3 ± 5	5 ± 8
Hopper	12 ± 1	17 ± 1	12 ± 1	12 ± 1	20 ± 1	5 ± 6	8 ± 12	3 ± 4	3 ± 4	5 ± 7
Kitchen	8 ± 0	14 ± 0	9 ± 0	9 ± 0	17 ± 1	6 ± 3	9 ± 6	5 ± 3	5 ± 3	6 ± 4
Maze2d	8 ± 0	14 ± 0	8 ± 0	8 ± 0	15 ± 1	3 ± 3	6 ± 6	3 ± 3	3 ± 3	4 ± 4
Walker2d	13 ± 1	20 ± 1	14 ± 1	14 ± 1	22 ± 1	5 ± 7	6 ± 12	3 ± 4	3 ± 5	5 ± 8

eliminating this sequence impedes the DT's ability to learn the task entirely as shown in [11]. This indicates that the evolutionary parameter of the Mamba architecture successfully replaces the need for RTG. Since the DM without the sequence of rewards achieves higher performance without requiring additional user interaction, we can conclude that the DM should be used without the reward sequence.

Lastly, we compare the new proposed Mamba methods with their Transformer predecessors. According to the results in Table 5.5, the DM without rewards outperforms the DT in 15 out of the 27 settings. Additionally, the HDM outperforms the HDT in 22 out of the 27 settings. The superiority of the Mamba models exists even while comparing to the DT with the ideal desired reward for each task. When comparing the DM to the DT using a fixed desired reward of 10k, the DM outperforms the DT in 17 out of the 27 settings. These results show that the proposed Mamba methods improve upon the Transformer predecessors in the D4RL benchmark. Overall, the DM without rewards is the best performing model of the set. Although, the HDM and the DT are still very competitive, it is worth noting that HDM requires two models and pre-processing the data set, while DT requires user interaction and task knowledge. Moreover, unlike the other models, DM can be applied to tasks without a reward function.

5.3.4 Time Comparison

Mamba models have outpaced Transformers in terms of speed in other applications. Also, HDT and HDM require the training of two models, and the extra computational cost may not be worth the performance benefits. Because of this, we compare the time required to perform a training iteration and the inference step using the different methods across the 7 task environments available in the D4RL benchmark. We use a batch size of 16, an embedding size of 128, a sequence length of 20, and 6 layers per model. Specifically, for the HDT and the HDM, we employ 6 layers for both the high-level and low-level models, and we measure the time to train both models. For training, we measure the time it takes for a gradient calculation and update. For inference, we measure the time it takes to build the sequences, obtain an action from the model and perform the transition. To ensure statistical robustness, we repeat both these steps 1000 times for each model, presenting the average and standard deviation time to perform a training iteration, and an inference step in Table 5.6. Results show that during training there's not much difference between

the Mamba methods and their Transformer predecessors. As expected the HDT and the HDM take close to double the time to train due to having double the models than the DT and the DM, respectively. Adding rewards to the DM does not increase the training time significantly. At inference time however, the Mamba methods are faster than the Transformer methods. In addition to the increase in performance, this computational boost further shows the benefits of the Mamba methods compared to the Transformer baselines.

5.4 Conclusion

In this chapter, we introduced two methods that improve upon the state-of-the-art sequence modelling in DL, the DT. We show that the DT relies on the sequence of returns-to-go to perform. This sequence requires precise specification of the value of the desired accumulated rewards, which is non-trivial to determine.

We first proposed the HDT method, which reformulates the problem in a hierarchical manner, where the sequence of returns-to-go is replaced with a sequence of sub-goals proposed by a high level model. By replacing the sequence of returns-to-go with a sub-goal selection method, we achieved a fully task-independent model that outperforms the DT and BC baselines in various tasks. Notably, the HDT is more robust to longer episodes and sparse rewards, making it suitable for real-world applications.

Our experimental results demonstrate the effectiveness of the HDT on three benchmarks and a reaching task with a real-world robot. The HDT achieved higher accumulated rewards compared to the DT and BC in most tasks and reached a higher number of positions in the reaching task. Future work will explore different architectures for both models to further improve their performance. Our findings suggest that using a hierarchical transformer model with a sub-goal selection method can enhance the performance of learning methods and facilitate their application to complex tasks.

Next, we improved upon this sequence modelling method by using the Mamba architecture instead of the Transformer. We introduced the DM and the HDM models, direct evolutions of the DT and the HDT, respectively, by leveraging the potent Mamba architecture. We show that the DM does not rely on the sequence of RTG to perform. Through our evaluation across seven diverse tasks within the D4RL benchmark and varying the demonstration data set, we demonstrate the superiority of these Mamba models over their Transformer-based predecessors in the majority of cases. Specifically, the DM and HDM outperform their transformer counterparts in most settings, while also being simpler and faster to train and perform inference. Lastly, the DM outperforms the baselines in most tasks, while not requiring a reward function, and being task-independent by not requiring user specified values. This advancement underscores the viability of Mamba architectures

in BC sequence modelling, and opens the way to keep solving more complex sequence modelling problems through DL.

In the next chapter, we apply demonstration learning to teach an agent to dance to music, which is an under-explored application. We will treat the music to dance generation as a language translation task, and will employ sequence models to learn the mapping between the two languages.

Chapter 6

Music to Dance as Language Translation using Sequence Models

6.1 Introduction

Dance is an aesthetic performing art, that serves as a medium to convey intricate emotional nuances through choreography closely intertwined with music. The task of automatically generating choreographies from music has garnered significant interest within the research community, promising practical applications in various real-world domains like games and films [232, 263, 264]. The current methodologies involving sequence posing, hand animation, or motion capture are not only laborious but also expensive. However, the challenge of dance generation lies in balancing the inherent human creativity in a choreography with the need for the movements to remain coherent with the rhythm and genre of the music.

The work in this chapter is grounded in two fundamental concepts that underpin our research. Firstly, we recognize the temporal coherence between the music piece and the dance choreography, emphasizing that treating each component separately is not feasible. Secondly, we capitalize on the understanding that music and dance constitute unique languages with an inherent correspondence. In light of this, we propose to conceptualize the automatic dance generation challenge as a translation problem—akin to converting from the source language of music to the target language of dancing. Embracing this perspective enables us to leverage the capabilities of sequence models [16, 158], specifically designed for processing sequential data, and well-established in achieving success in language translation.

We introduce our method as Music to Dance as Language Translation (MDLT). MDLT leverages a pre-existing data set of music-dance pairs and learns the mapping between both languages. The initial stages involve extracting musical features and joint angles from the data set. We propose to variants: MDLT-T using the Transformer architecture and MDLT-M using the Mamba architecture. The model is conditioned on the sequence of musical features and predicts the subsequent pose in the choreography. This predicted pose is incorporated into the choreography sequence, and the musical sequence advances in time. This iterative process continues until the culmination of the song.

We conducted a series of experiments using a subset of the AIST++ [232] data set and the entire PhantomDance [15] data set. Our method's performance was assessed across distinct music genres as well as collectively across all genres. Because the evaluation was

performed using unseen music data, we show that our method learns to translate music into dance in a coherent and adaptive manner. The contributions of this work can be summarized as follows:

- We propose a new perspective to model the music conditioned dance generation as a translation task, where an agent learns the mapping between the music and dance languages from music-dance pairs in a data set. To our knowledge, this is the first study to propose this perspective and demonstrate its viability.
- We propose two variants: MDLT-T employs a modified Transformer architecture and MDLT-M employs the Mamba architecture.
- We compare the performance the two variants on a section of the AIST++ data set and the full PhantomDance data set.
- Experiments with a UR3 show that MDLT, can learn the mapping from audio to dance of different music pieces from different genres. Code and models are available at github.com/meowatthemoon/MDLT.

6.2 Translation

In language modelling, the objective is to estimate the likelihood of a word given a preceding sequence of words. For a given source sentence $X = (x_1, x_2, ..., X_{N_s})$ and a corresponding target sentence $Y = (y_1, y_2, ..., y_{N_t})$, the translation model aims to sequentially predict the words in the target sentence. Here, N_s and N_t denote the lengths of the source and target sentences, respectively. More formally, the model should provide estimates of the conditional distribution $p(y_i \mid X, y_{1:i-1})$.

In the context of our problem, we conceptualize the sequences of audio features as the source language and the sequences of dance poses as the target language. The goal is to estimate the parameters of a sequence model that best capture the mapping between these two languages.

6.3 Data Preparation

MDLT learns the mapping between music pieces and dance choreographies. To enable the model to interpret music, audio features are initially extracted from the music pieces, forming feature vectors that serve as the model's input. Then, because in this work we apply the method to teach a UR3 robot to dance, the human poses in the choreographies must be translated into the corresponding joint angles of the robot. Subsequently, the sequences of audio features and joint angles need to be synchronized into pairs, allowing the model to learn the mapping between music and the corresponding joint angles. The subsequent sections explain each data processing step in more detail.

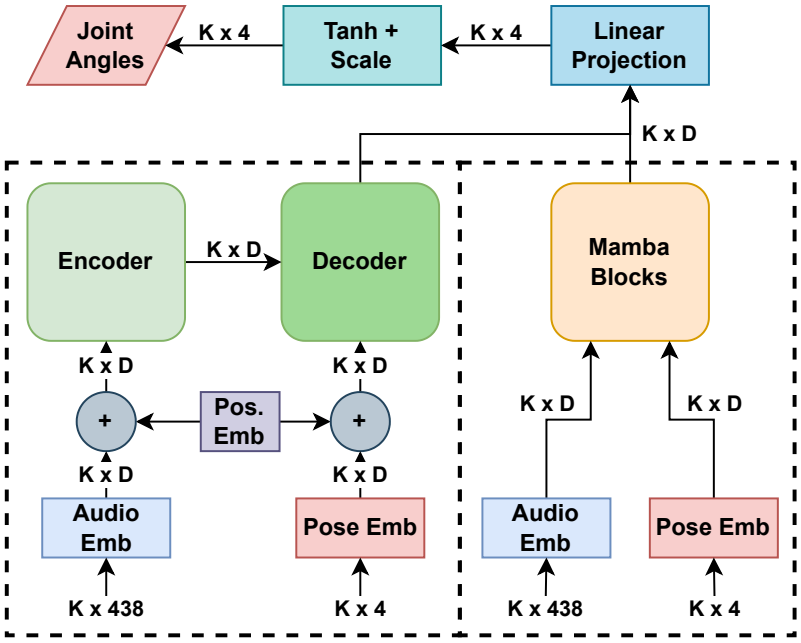


Figure 6.1: Architecture of MDLT model variants. The audio features and poses first pass through their respective embedding layers. In the case of the Transformer variant (dashed left block) these embeddings are augmented with positional encoding. The encoder of the Transformer is conditioned on the audio features. The decoder of the Transformer is conditioned on the output of the encoder as well as the poses. The Mamba variant (dashed right block) receives both the audio and pose vectors. The embeddings of the final Mamba or Decoder block are projected to the pose dimensions. Finally, these values are activated using tanh and scaled to the joint angle range to produce the next poses. Only one of the dashed blocks is used.

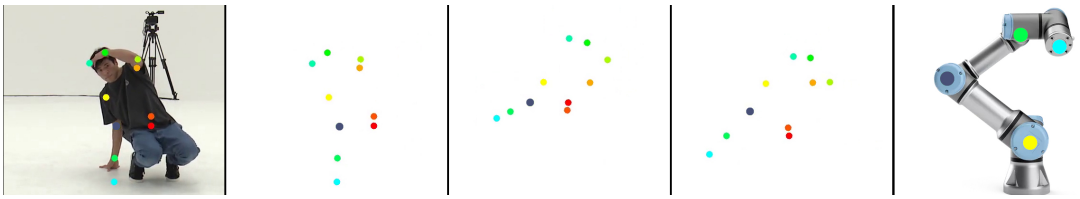


Figure 6.2: Joint angle extraction from keypoints: First we obtain the arm and pelvis keypoints. Then we align the shoulders with the ground plane. Next, we align the spine vertically. Lastly, we extract the joint angles from the angles between the arm vectors.

6.3.1 Data sets

We use the AIST++ [232] and PhantomDance [15] data sets. Both provide music-to-dance pairs. AIST++ was built by reconstructing 3D poses from 2D multi-view videos sourced from the original AIST Dance Database. Comprising 1408 sequences executed by multiple dancers across 10 diverse dance genres. While multiple choreographies are available for each music piece in the data set, our focus on estimating the mapping from music to dance requires a one-to-one correspondence between audio and poses. Consequently, we assign a single choreography to each music piece. We amalgamate the music pieces from the original train, test, and validation splits, totalling 60 pieces with 6 pieces per genre. In the experiments section, we assign custom splits of the data set.

The PhantomDance data set was built by professional animators and is composed of 260 dance videos from over 100 different subjects. Formally, the data sets can be expressed

Table 6.1: Comparison of music features used by music-to-dance generation methods. Most methods use Librosa for music feature extraction. The most commonly used features are MFCC, MFCC delta, constant-Q chromagram, tempogram, and onset strength.

Method	Chromagram	MFCC	MFCC Delta	Mel Spectogram	Onset Strength	Tempogram	Others
AI Choreographer [232]	x	x					x
Bailando [233]	x	x	x		x	x	
Choreomaster [228]				x			
DanceHat [230]	x		x		x	x	x
DanceFormer [15]	x	x					
Dance with Melody [227]	x	x					
DeepDance [231]		x	x	x			x
M2C [224]	x	x	x		x	x	

as:

$$\mathcal{D}' = \bigcup_{i=1}^{G} \{(M_{ij}, D_{ij})\}_{j=1}^{G_i} \quad (6.1)$$

where G is the set of music-genres, and G_i is the number of pairs in the genre.

6.3.2 Audio Features

We extracted musical features from the audio files, thereby transforming the music into interpretable vectors for our model. Following a prevalent approach employed in the majority of music-to-dance studies, excluding [225], we utilize the publicly available audio processing toolbox, Librosa, for feature extraction. The diversity in feature selection across various works prompted us, in synergy with the contributions of [224], to delve into different feature combinations explored by existing music-to-dance methodologies. A comprehensive overview of these features and their respective utilization is provided in Table 6.1. Drawing from this study, we used the MFCC, MFCC delta, constant-Q chromagram, tempogram, and onset strength features. These features are concatenated to form a 438-dimensional vector. Feature extraction is conducted at a rate of 60 times per second. Acknowledging the cautionary note of [224], we normalize the features across each index of the 438 dimensions, constraining the values within the range of 0.1 to 0.9.

6.3.3 Joint Angles

MDLT learns to translate music feature sequences into corresponding pose sequences. In our application, involving a UR3 robotic arm, the human poses from our data set need to be converted to corresponding joint angles. While our current application involves translating human poses into the corresponding joint angles for the UR3 robot, it's important to highlight that our method is not limited to this particular robot or joint struc-

ture. Our method can be extended to other robotic platforms by adjusting the mapping between the human joints represented in the SMPL format and the joints of the target robot. For instance, both the AIST++ and PhantomDance data sets provide poses in the SMPL format. These poses can be converted into 24 joints, and our method can then be applied to a full humanoid robot.

The UR3 is a 6-DOF robotic arm, hence we use the right arm poses within each full human pose. To convert a human arm pose to the UR3 joints, we simplify the problem and reduce the 6 joints to 4. We consider two joints connecting the shoulder to the trunk, one at the elbow connecting the arm to the forearm, and one at the wrist connecting the hand to the forearm. This choice aligns with the first four joints of the UR3. Although we could explore incorporating hand rotation around the wrist, the inherent challenge in accurately detecting this in the data set led us to prioritize avoiding potential errors that could affect convergence.

To derive joint angles, we rely on the 3D keypoint annotations. Acknowledging that the arm's position is influenced by the movement of the rest of the body, we do transformations to maintain a fixed position of the base of the arm across different poses. We compute the unit vector between the two shoulder keypoints. Then, we determine the rotation required to align this vector parallel to the floor, and apply it to all keypoints. Subsequently, we find the midpoint between the shoulders, calculate the unit vector to the pelvis point, and apply a rotation making this vector perpendicular to the floor, effectively straightening the spine and maintaining consistent arm positioning relative to the trunk.

To obtain the joint angles, we calculate unit vectors between the shoulder and elbow, elbow and wrist, and wrist and the last point of the index finger. The first two joint angles correspond to rotations between the forearm unit vector and the vertical and outward-pointing axes from the human, respectively. The third joint angle accounts for the rotation between the forearm and arm unit vectors, while the last joint angle reflects the rotation between the arm and hand unit vectors. This process is repeated for every pose in the data set. The values of the joint angles range from $-\pi$ to π. This process is represented in Fig. 6.2.

6.3.4 Synchronization

For each music-dance pair, we acquire a sequence of audio feature vectors $\{f_{t_f}\}_{t_f=1}^{T_f}$ by sampling at a rate of 60 times per second and a sequence of poses $\{p_{t_p}\}_{t_p=1}^{T_p}$ obtained from the data set annotations. Because the annotations were not generated with the exact frequency of 60 times per second, it is likely that $T_f \neq T_p$. To pair the elements from the two sequences, we will use the timestamps provided alongside the annotations $\{t_{t_p}\}_{t_p=1}^{T_p}$. We compute the timestamp of each feature vector as $t_{t_f} = t_f * 1/60$ and then associate it with the pose with the closest timestamp.

6.4 Music to Dance Translation

Automatic music-to-dance generation is often framed as a sequence modelling challenge. We propose to expand from this sequence modelling view and propose to model the music to dance generation problem as a direct translation task, where the model translates sequences from the source language of audio to the sequences of the target language of dance. We propose two variants to perform music-to-dance translation, the first uses the Transformer architecture, while the second uses the Mamba architecture.

6.4.1 Transformer

While other methods have employed Transformers for music-to-dance generation [232], they often under-utilize the Transformer's full capabilities. They frequently overlook crucial components such as the decoder, masked attention, or positional encoding. Moreover, many existing approaches rely on multiple networks trained with adversarial learning, which can be highly data-inefficient [15]. Additionally, some methods [233] require the discretization of the dance space, which may introduce limitations in expressiveness and precision. In contrast, our work applies the translation framework to music-to-dance generation, harnessing the full capabilities of a single Transformer model.

Unlike typical natural language translation tasks where the entire source language sentence conditions the encoder, we face a constraint. Due to the extensive length of music pieces, ranging up to 8000 audio features, conditioning the encoder on the entire music piece is impractical. Consequently, we condition the encoder on a sequence of size K of preceding audio feature vectors $\mathbf{s}$. During batch data sampling, a music piece is randomly selected, followed by the choice of a trajectory index $t \in [0, T-1]$, where T is the length of the music piece. The $K-1$ audio features preceding t compose the sequence. The sequence is padded with zero-filled vectors to the right, if its length $L_s < K$. This sequence is paired with a padding mask, creating the input of the encoder. This mask is a binary sequence of length K with zeros in indexes where the audio feature vector was padded, and prevents the encoder from considering the padding elements.

On the decoder's side, after selecting index t, we sample the corresponding poses to the sampled audio features, but right-shifted by one. Akin to the start token in NLP, we append a padding pose of zeros to the start of the pose sequence. This shift conditions the encoder on past poses and prevents it from seeing the ground-truth. Similar to the audio features, this pose sequence is right-padded to reach a length of K elements. The decoder also receives a causal binary masking matrix where the values under the diagonal are ones and the remaining are zero. This ensures that attention scores consider only past tokens, preventing access to future information.

Both audio feature vectors and pose vectors pass through individual linear layers, converting them to 128-dimensional vectors. To convey positional information, an embed-

ding layer processes the index t of each element in the K sequence. The vocabulary's size is set to the length of the largest music piece in the data set plus one for the start token. The resulting embeddings are summed to both the audio and pose embeddings before entering the Transformer.

We employ a Transformer architecture where the encoder and the decoder are composed of 6 blocks, 8 attention heads, a dropout rate of 0.1, and feed-forward layers with 2048 neurons, and output 128-dimensional embeddings. The embeddings returned by the decoder pass through a final feed-forward layer to project them to four joint angles. Subsequently, a hyperbolic tangent activation function normalizes the values between -1 and 1, which are then multiplied by π to scale them to the range of the joint angles.

The Transformer's objective is to translate audio sequences into corresponding poses after learning the mapping from music-dance pairs in the data set. Given input sequences of audio features **s** and right-shifted poses **a'**, the Transformer aims to output the non-shifted pose sequence **a**. The loss function is the L2 loss between the predicted joint angles and the ground-truth.

6.4.2 Mamba

The Mamba variant is much simpler than the Transformer. Mainly because it is composed of the Mamba backbone instead of an encoder or decoder which streamlines the processing. This means that the causal binary mask of the Transformer is not needed. Additionally, because Mamba keeps track of the previous state, it also does not require positional embeddings. Hence, the audio feature vectors and pose feature vectors pass through the respective embedding layers and the outputs are fed to the Mamba blocks directly. We employ a Mamba architecture composed 6 Mamba blocks, a dropout rate of 0.1, and feed-forward layers with 2048 neurons, and output 128-dimensional embeddings. The embeddings returned by the Mamba blocks are fed through the same pipeline as previously described to obtain the joint angles. The loss function is also the L2 loss between predicted joint angles and ground-truth.

6.5 Experiments

6.5.1 Experimental Setup

We apply the MDLT on the UR3, a 6-DOF robotic arm. Our experimental process begins with the application of the data processing pipeline (see Section 6.3). This process yields a refined data set D' of pairs of sequences of 438-dimensional audio features and 4-dimensional poses. The processed AIST++ data set comprises 60 pairs, organized into 10 music genres, each containing 6 pairs. The PhantomDance data set is comprised of 260 pairs. In all the experiments we compare the performance of our method using the Transformer or the Mamba architecture.

We made an ablation study to determine an adequate hyper-parameter configuration for the model with either the Transformer or Mamba variant. We varied the number of Transformer or Mamba layers, the size of the embeddings within the model, and the length of the token sequence upon which the model is conditioned. Subsequently, we evaluate the models based on the AJE observed during training, and the model's size. All models are trained using the complete AIST++ data set. The configuration that yields the lowest pair of AJE and model size for the MDLT-T variant entails 6 Transformer layers, an embedding size of 128, and a sequence length of 20. In contrast, for the MDLT-M variant, the sequence length is extended to 120. We will use these two architectures throughout the remaining experiments.

We evaluate MDLT on both the entire data set and individual music genres. In the case of single-genre experiments, our approach involves training MDLT on 5 data pairs within a specific genre and subsequently evaluating its performance on the remaining pair. To ensure robustness and generalization, we perform cross-validation across the 6 possible combinations within each genre, calculating both mean and standard deviation metrics across these variations.

For experiments encompassing the complete data set, we adopt a validation strategy by assigning one pair per genre for validation purposes, leaving the remaining 5 pairs for training. Employing a similar cross-validation methodology, we iterate this process 6 times, presenting the aggregated mean and standard deviations over these runs. This evaluation strategy enables us to gauge MDLT's performance not only across diverse genres but also its ability to generate genre-coherent music, providing valuable insights into its robustness and generalization capabilities. Lastly, we evaluate the performance of our method on the PhantomDance data set by performing cross-validation across 10 different combinations of 26 pairs assigned to the validation set.

Each experiment comprises a 250k update steps, utilizing a batch size of 16 and a learning rate of 10^{-4}. Model validation is conducted at regular intervals of 1k updates. During validation we iterate over each music in the music-dance pairs and use the model to generate a sequence of poses. This sequence of poses is then compared with the ground-truth poses using the following two metrics:

- **Average Joint Error (AJE)** : Average joint error in radians between translated poses and the ground-truth for a given music sequence $AJE(p_i^G, p_i^T) \in [0, \infty[$.
- **Fréchet Inception Distance (FID) [265]** : Evaluates how close the distribution of generated dances is to that of the ground-truth dances.

6.5.2 AIST++ Music Genre Generalisation

We investigate the performance of the two variants using the processed AIST++ data set. We evaluated on individual music genres and also using all music genres. The res-

Table 6.2: FID and AJE metrics on AIST++ data set.

AIST++	MDLT-T		MDLT-M	
Genre	AJE (rad)	FID (%)	AJE (rad)	FID (%)
All Genres	0.58 +/- 0.10	1.32 +/- 0.64	**0.57 + 0.06**	**0.96 + 0.31**
mBR	0.57 +/- 0.13	0.46 +/- 0.25	**0.53 + 0.04**	**0.40 + 0.18**
mHO	**0.25 +/- 0.03**	**0.06 +/- 0.04**	**0.25 + 0.04**	0.13 + 0.08
mJB	**0.57 +/- 0.16**	0.79 +/- 0.75	**0.57 + 0.15**	**0.61 + 0.50**
mJS	0.35 +/- 0.13	0.38 +/- 0.36	**0.34 + 0.08**	**0.22 + 0.19**
mKR	**0.58 +/- 0.27**	**0.76 +/- 0.59**	0.61 + 0.19	0.91 + 1.05
mLH	0.30 +/- 0.04	0.10 +/- 0.05	**0.29 + 0.05**	**0.06 + 0.01**
mLO	**0.45 +/- 0.18**	0.74 +/- 0.92	0.47 + 0.14	**0.51 + 0.65**
mMH	**0.33 +/- 0.10**	**0.11 +/- 0.06**	**0.33 + 0.05**	0.12 + 0.05
mPO	**0.23 +/- 0.06**	**0.16 +/- 0.11**	0.29 + 0.10	0.24 + 0.25
mWA	**0.62 +/- 0.28**	**0.75 +/- 0.96**	0.63 + 0.25	1.03 + 1.68

Table 6.3: FID and AJE metrics on PhantomDance data set.

MDLT-T		MDLT-M	
AJE	FID	AJE	FID
0.87 ± 0.02	0.39 ± 0.02	0.73 ± 0.01	0.82 ± 0.01

ults, showcasing the mean and standard deviations after cross-validation of AJE and FID on the each genre's validation set, are presented in Table 6.2. The results show small differences between the two variants with the Transformer variant offering a slightly lower average AJE and FID across the experiments. The model seems to perform the best on the 'mPO' music genre and the worst on the 'mWA'. Overall MDLT-T offers an average AJE of 0.44 radians and an average FID of 0.51% across the 11 experiments. These low values validate the model's ability to successfully learn the mapping between the audio and dance languages and consequently generalize to unseen audio pieces.

6.5.3 PhantomDance Evaluation

Finally, we assess performance of the two variants using the processed PhantomDance data set. The summarized results are presented in Table 6.3. Each value is the mean and standard deviation of cross-validation across 10 combinations of 90/10 train/validation splits. Notably, both AJE and FID metrics exhibit an increase compared to the previous results obtained from the AIST++ data set. This is attributed to the augmented complexity of the data set. Specifically, the choreographies within PhantomDance are lengthier and exhibit greater diversity as opposed to AIST++ where the pairs can be organized by music genre. Nevertheless, the model consistently achieves significantly lower AJE compared to a random model, along with a low FID score. This demonstrates the model's capability to learn the mapping from music to dance, thus validating our approach of framing music-to-dance generation as a language translation task.

6.6 Conclusions

We propose to model the music-to-dance generation task as a language translation problem. We describe two variants: MDLT-T and MDLT-M using the Transformer and the Mamba architecture, respectively. We researched previous music-to-dance methods which validated the selection of audio features, and presented an approach for mapping human poses from keypoints to joint angles of a robotic arm. MDLT can also be applied to humanoid agents.

Evaluation on music-dance pairs from the AIST++ and PhantomDance data sets, through AJE and FID metrics, demonstrate that MDLT can robustly and efficiently translate diverse and unseen music to high-quality dance motions coherent within the genre.

The next chapter presents the general conclusions of this thesis and discusses possible future developments.

Chapter 7

Conclusion

7.1 Conclusion

Throughout this thesis, significant strides have been made to advance the field of DL. By addressing open problems, enhancing existing methodologies, and pioneering novel applications, this research endeavors to fortify the foundations of DL. With the obtained results, we achieve the desired contributions to the field of DL of this thesis. The following paragraphs present the main conclusions of this work.

The comprehensive review of the state-of-the-art in DL facilitated the identification of several open problems: i) the main challenge in DL methods is the generalization to unseen scenarios and avoiding out-of-distribution states; ii) most policies tend to be context-specific, rendering them vulnerable to performance degradation under altered conditions; iii) fine-tuning of hyper-parameters is often imperative for achieving optimal performance; iv) the policies are reliant on the demonstration data set to learn the desired behavior, however the data set is often sub-optimal; v) although DL improves the safety of agents when compared to RL, the trained policies may still harbor safety concerns.

Next, we introduce and elaborate on the fundamental concepts, methodologies, and algorithms that underpin the proposed methods in this thesis. We begin by describing the sequence modelling architectures used in this thesis: the Transformer and the SSSMs. Then, we summarize pre-existing algorithms that are used by our proposed methods. We start with the DTW, an algorithm for time series analysis and pattern recognition. Next, we examine state-of-the-art RL algorithms: DDPG, SAC and TD3. Finally, we discuss the former state-of-the-art DL sequence modelling algorithm, the DT.

In response to these challenges, this thesis introduces several innovative methodologies designed to mitigate these obstacles and generate advancements in DL. Aware that most policies only perform on the same context in which they were trained on, we proposed a framework that uses contrastive learning to obtain viewpoint-invariant state representations from demonstrations. This method can be extended to ignore other task-irrelevant features while keeping task-relevant information. We showed that the features extracted by this framework can be used to correctly align frames between two viewpoints. Additionally, to contribute towards limited benchmarks we created also created a multi-viewpoint DL data set. Furthermore, we showed that the feature representations can replace the original state image for classification and policy learning purposes.

Then, we focused on improving the safety of policy learning methods by combining the benefits of both RL and DL. We proposed a method that can be integrated with any existing RL algorithm and improves the safety of agents during learning. The method compares the agents trajectory with safe and unsafe demonstrations, and the trajectory is terminated if it aligns better with unsafe demonstrations. Ablation studies were performed to help identify effective comparison strategies. The results demonstrate significant enhancement in the safety of the agent over the base algorithm, while maintaining or improving performance. This work highlighted the potential of using demonstrations to enhance RL algorithms.

Next, language models revolutionized many machine learning applications and eventually impacted DL. Like many methods, the state-of-the-art sequence model in DL relied on careful specification of hyper-parameters. We proposed a novel method to remove the need for a particular hyper-parameter through hierarchical learning. Our method not only successfully replaces the need for hyper-parameter specification, but it also improves performance over the state-of-the-art in a large number of tasks.

Then, we improve upon this past method using the recent advancements in sequence modelling: the Mamba architecture. We proposed to replace the Transformer methods with the Mamba architecture, reducing their complexity, and inference time, while improving their performance.

With the aim of transferring DL to new applications, we described the insight of modelling the task of learning to dance as a translation task from the language of audio to the language of dance. We researched previous music-to-dance methods to validate the selection of audio features, and presented a method for mapping human poses from keypoints to joint angles of a robotic arm. We propose two variants: using the Transformer or the Mamba architectures. The method modifies the standard sequence modelling architecture to process sequences of audio features and translate them to dance poses. Results show that the method can translate diverse and unseen music to high-quality dance motions coherent within the genre.

In summation, the contributions set forth in this thesis mark significant strides toward advancing the state-of-the-art in DL. By addressing existing challenges and introducing innovative methodologies, this research not only enhances the efficacy and safety of learned policies but also extends the applicability of DL to other domains. By publicly releasing all the code with proper documentation, we show the commitment to promote the continuous growth of the field.

7.2 Future Directions

Expanding on the progress achieved in the thesis regarding DL, numerous intriguing paths for further exploration and innovation lie ahead. This work contributed with several methods that address open problems in DL, including the context problem addressed in CLfD, creating task-independent large sequential models with HDT, improving the robustness and safety of policies with DEFENDER, and applying DL to novel applications as in MDLT. In addition to the open problems identified in 3 and based on the contributed methods, the subsequent avenues present vast potential for advancing DL further:

- CLfD showed the possibility for obtaining policies invariant to the view-point of the camera. Future directions could explore into the creation of data sets to encompass a broader range of task-irrelevant features.
- HDT used the same architecture as DT for both the high-level mechanism and the low-level controller for equitable comparison. However, future research might explore alternative architectures to enhance performance.
- Furthermore, while HDT conducted ablation studies on sub-goal sampling methods for each state from the demonstration data set, there is room for exploring different sampling methods to better condition the low-level controller and optimize performance.
- DEFENDER's filtering strategies are task-dependent and sometimes overly restrictive. Future research could investigate a universal filtering approach that facilitates learning across varied tasks.
- MDLT is conditioned on past audio sequences to predict the next pose. Future research trajectories could investigate the possibility of conditioning the encoder on future audio features to determine if this could allow the model to make a more informed decision.
- Additionally, future research could focus on refining MDLT through online RL, training the model to navigate real-world environments while avoiding collisions, beyond the mere replication of the demonstrated choreographies.

In conclusion, this work contributed to a exhaustive understanding of the DL paradigm and proposed a set of methods that expanded the boundaries of the paradigm to new domains. Through these contributions, we believe to have accomplished the objectives of this thesis and that the contributions of this work solved, or at least mitigated, a significant number of DL problems.

Bibliography

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009. xxiii, 23, 28, 29

[2] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 1321–1326. xxiv, 83, 84, 86

[3] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-contrastive networks: Self-supervised learning from video," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1134–1141. xxiv, xxvii, 35, 44, 49, 54, 66, 70, 77, 84, 86, 87, 88, 89, 93

[4] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*. Springer, 2020, pp. 776–794. xxvii, 54, 71, 85, 86

[5] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," *arXiv preprint arXiv:2004.07219*, 2020. xxvii, xxviii, 62, 97, 100, 102, 104, 106

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016. xxvii, 97, 100, 102

[7] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín, "What matters in learning from offline human demonstrations for robot manipulation," *arXiv preprint arXiv:2108.03298*, 2021. xxvii, 97, 100, 102

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018. 2, 25, 85, 89, 95

[9] A. Correia and L. A. Alexandre, "Multi-view contrastive learning from demonstrations," in *2022 Sixth IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2022, pp. 338–344. 4, 54, 77

[10] ——, "Defender: Dtw-based episode filtering using demonstrations for enhancing rl safety," in *31th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning - ESANN 2023*, 2023. 5

[11] ——, "Hierarchical decision transformer," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 1661–1666. 5, 18, 55, 77, 109

[12] ——, "Decision mamba architectures," 2024. [Online]. Available: https://arxiv.org/abs/2405.07943 5

[13] ——, "Music to dance as language translation using sequence models," 2024. [Online]. Available: https://arxiv.org/abs/2403.15569 5

[14] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 084–15 097, 2021. 7, 55, 77, 95, 96, 97, 99, 100

[15] B. Li, Y. Zhao, S. Zhelun, and L. Sheng, "Danceformer: Music conditioned 3d dance generation with parametric motion transformer," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 1272–1279. 7, 65, 113, 115, 116, 118

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017. 9, 55, 65, 113

[17] R. Bellman and R. Kalaba, "On adaptive control processes," *IRE Transactions on Automatic Control*, vol. 4, no. 2, 1959. 12

[18] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007. 14

[19] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930. 16

[20] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019. 18, 97

[21] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4693–4700. 21, 64, 77

[22] H. Wang, J. Chen, H. Y. Lau, and H. Ren, "Motion planning based on learning from demonstration for multiple-segment flexible soft robots actuated by electroactive polymers," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 391–398, 2016. 21, 66

[23] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband *et al.*, "Deep q-learning from demonstrations," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018. 21, 22, 23, 34, 37, 42, 43, 44, 50, 57, 58, 67, 77

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015. 22, 77

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016. 22, 43, 50

[26] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2052–2062. 22, 57, 77

[27] S. Schaal, "Learning from demonstration," *Advances in Neural Information Processing Systems*, vol. 9, 1996. 23

[28] ——, "Is imitation learning the route to humanoid robots?" *Trends In Cognitive Sciences*, vol. 3, 1999. 23

[29] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010. 23, 31, 64, 66

[30] A. X. Lee, A. Gupta, H. Lu, S. Levine, and P. Abbeel, "Learning from multiple demonstrations using trajectory-aware non-rigid registration with applications to deformable object manipulation," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 5265–5272. 23, 32, 77

[31] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 2. IEEE, 2002, pp. 1398–1403. 23, 33, 66, 77

[32] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Survey: Robot programming by demonstration," Springer, Tech. Rep., 2008. 23

[33] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017. 23

[34] Z. Zhu and H. Hu, "Robot learning from demonstration in robotic assembly: A survey," *Robotics*, vol. 7, no. 2, p. 17, 2018. 23

[35] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103500, 2021. 23

[36] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 297–330, 2020. 23

[37] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review," *Perspectives on Open Problems*, vol. 5, 2020. 23, 39, 76

[38] R. F. Prudencio, M. R. Maximo, and E. L. Colombini, "A survey on offline reinforcement learning: Taxonomy, review, and open problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2023. 23

[39] S. Russell, "Learning agents for uncertain environments," in *Proceedings of the eleventh annual Conference on Computational Learning Theory*, 1998, pp. 101–103. 25, 47

[40] C. L. Nehaniv, K. Dautenhahn *et al.*, "The correspondence problem," *Imitation in Animals and Artifacts*, vol. 41, 2002. 28

[41] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun, "Survey of imitation learning for robotic manipulation," *International Journal of Intelligent Robotics and Applications*, vol. 3, pp. 362–369, 2019. 29

[42] D. H. Grollman and O. C. Jenkins, "Dogged learning for robots," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. Ieee, 2007, pp. 2483–2488. 29, 36, 44, 52, 77

[43] S. Chernova and A. L. Thomaz, *Robot learning from human teachers*. Morgan & Claypool Publishers, 2014. 30

[44] J. E. Laird, K. Gluck, J. Anderson, K. D. Forbus, O. C. Jenkins, C. Lebiere, D. Salvucci, M. Scheutz, A. Thomaz, G. Trafton *et al.*, "Interactive task learning," *IEEE Intelligent Systems*, vol. 32, no. 4, pp. 6–21, 2017. 30

[45] A. Saran, E. S. Short, A. Thomaz, and S. Niekum, "Enhancing robot learning with human social cues," in *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2019, pp. 745–747. 30

[46] T. Kessler Faulkner, S. Niekum, and A. Thomaz, "Asking for help effectively via modeling of human beliefs," in *Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 2018, pp. 149–150. 30

[47] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental robotics IX: The 9th international symposium on experimental robotics*. Springer, 2006, pp. 363–372. 31, 64

[48] J. Chen and A. Zelinsky, "Programing by demonstration: Coping with suboptimal teaching actions," *The International Journal of Robotics Research*, vol. 22, no. 5, pp. 299–319, 2003. 31, 74

[49] R. Aler, O. Garcia, and J. M. Valls, "Correcting and improving imitation models of humans for robosoccer agents," in *2005 IEEE Congress on Evolutionary Computation*, vol. 3. IEEE, 2005, pp. 2402–2409. 31, 50, 56

[50] Y. Hristov and S. Ramamoorthy, "Learning from demonstration with weakly supervised disentanglement," *arXiv preprint arXiv:2006.09107*, 2020. 31, 77

[51] G. J. Maeda, G. Neumann, M. Ewerton, R. Lioutikov, O. Kroemer, and J. Peters, "Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks," *Autonomous Robots*, vol. 41, pp. 593–612, 2017. 31, 77

[52] Y. Shavit, N. Figueroa, S. S. M. Salehian, and A. Billard, "Learning augmented joint-space task-oriented dynamical systems: a linear parameter varying and synergetic control approach," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2718–2725, 2018. 31

[53] C. Eteke, D. Kebüde, and B. Akgün, "Reward learning from very few demonstrations," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 893–904, 2020. 31, 48, 77

[54] M. Ogino, H. Toichi, M. Asada, and Y. Yoshikawa, "Imitation faculty based on a simple visuo-motor mapping towards interaction rule learning with a human partner," in *Proceedings of the 4th International Conference on Development and Learning, 2005.* IEEE, 2005, pp. 148–148. 32, 66

[55] O. Akanyeti, U. Nehmzow, C. Weinrich, T. Kyriacou, and S. A. Billings, "Programming mobile robots by demonstration through system identification," in *European Conference on Mobile Robots: ECMR 2007*, 2007. 32

[56] B. Hayes and B. Scassellati, "Discovering task constraints through observation and active learning," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 4442–4449. 32, 77

[57] Y. Liu, A. Gupta, P. Abbeel, and S. Levine, "Imitation from observation: Learning to imitate behaviors from raw video via context translation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1118–1125. 32, 70, 77

[58] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *ICML*, vol. 97. Citeseer, 1997, pp. 12–20. 32, 49

[59] R. Dillmann, M. Kaiser, and A. Ude, "Acquisition of elementary robot skills from human demonstration," in *International Symposium on Intelligent Robotics Systems*. Citeseer, 1995, pp. 185–192. 32

[60] M. Lopes and J. Santos-Victor, "Visual learning by imitation with motor representations," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 3, pp. 438–449, 2005. 32

[61] M. Edmonds, F. Gao, X. Xie, H. Liu, S. Qi, Y. Zhu, B. Rothrock, and S.-C. Zhu, "Feeling the force: Integrating force and pose for fluent discovery through imitation learning to open medicine bottles," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3530–3537. 33, 77

[62] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato, "Learning from demonstration and adaptation of biped locomotion," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 79–91, 2004. 33

[63] P. Ruppel and J. Zhang, "Learning object manipulation with dexterous hand-arm systems from human demonstration," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5417–5424. 33, 77

[64] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in Neural Information Processing Systems*, vol. 29, 2016. 33, 52, 77

[65] Z. Liu, Z. Cen, V. Isenbaev, W. Liu, S. Wu, B. Li, and D. Zhao, "Constrained variational policy optimization for safe reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 13 644–13 668. 33, 64, 72, 75, 77

[66] N. C. Wagener, B. Boots, and C.-A. Cheng, "Safe reinforcement learning using advantage-based intervention," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 630–10 640. 33, 36, 63, 75, 77

[67] A. Billard and M. J. Matarić, "Learning human arm movements by imitation:: Evaluation of a biologically inspired connectionist architecture," *Robotics and Autonomous Systems*, vol. 37, no. 2-3, pp. 145–160, 2001. 34

[68] Y. Demiris and A. Dearden, "From motor babbling to hierarchical learning by imitation: a robot developmental pathway," pp. 1430–1440, 2021. 34, 77

[69] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635. 34, 40, 59, 67, 77

[70] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," *Advances in Neural Information Processing Systems*, vol. 27, 2014. 35, 43, 50, 77

[71] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016. 35, 77

[72] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving sample efficiency in model-free reinforcement learning from images," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 10 674–10 681. 35

[73] K. Ramachandruni, M. Babu, A. Majumder, S. Dutta, and S. Kumar, "Attentive task-net: Self supervised task-attention network for imitation learning using video demonstration," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4760–4766. 35, 44, 49, 54, 70, 77, 84, 87, 93

[74] S. Chernova and M. Veloso, "Confidence-based policy learning from demonstration using gaussian mixture models," in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007, pp. 1–8. 36, 43, 64, 77

[75] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 661–668. 36, 49, 64, 75, 77

[76] S. Sinha, A. Mandlekar, and A. Garg, "S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics," in *Conference on Robot Learning*. PMLR, 2022, pp. 907–917. 36, 77

[77] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 884–19 895, 2020. 36, 72

[78] N. Hansen, H. Su, and X. Wang, "Stabilizing deep q-learning with convnets and vision transformers under data augmentation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 3680–3693, 2021. 36, 72

[79] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel, "Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6131–6141. 36

[80] T. Yu, A. Kumar, Y. Chebotar, K. Hausman, C. Finn, and S. Levine, "How to leverage unlabeled data in offline reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 25 611–25 635. 36, 74, 77

[81] L. Zhu, Y. Cui, and T. Matsubara, "Dynamic actor-advisor programming for scalable safe reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10 681–10 687. 37, 49, 59

[82] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2817–2826. 37

[83] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto, "Reinforcement learning with prototypical representations," in *International Conference on Machine Learning*. PMLR, 2021, pp. 11 920–11 931. 37, 54, 57

[84] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak, "Planning to explore via self-supervised world models," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8583–8592. 37, 45

[85] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," *arXiv preprint arXiv:1801.08757*, 2018. 37, 64, 75, 77

[86] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3387–3395. 37, 75

[87] G. Swamy, S. Choudhury, D. Bagnell, and S. Wu, "Causal imitation learning under temporally correlated noise," in *International Conference on Machine Learning*. PMLR, 2022, pp. 20 877–20 890. 37, 77

[88] J. Aleotti and S. Caselli, "Robust trajectory learning and approximation for robot programming by demonstration," *Robotics and Autonomous Systems*, vol. 54, no. 5, pp. 409–413, 2006. 38

[89] P. K. Pook and D. H. Ballard, "Recognizing teleoperated manipulations," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE, 1993, pp. 578–585. 38

[90] M. Kaiser, H. Friedrich, and R. Dillmann, "Obtaining good performance from a bad teacher," in *Programming by Demonstration vs. Learning from Examples Workshop at ML*, vol. 95. Citeseer, 1995. 38

[91] M. Hamaya, F. von Drigalski, T. Matsubara, K. Tanaka, R. Lee, C. Nakashima, Y. Shibata, and Y. Ijiri, "Learning soft robotic assembly strategies from successful and failed demonstrations," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 8309–8315. 38, 77

[92] B. Kim, A.-m. Farahmand, J. Pineau, and D. Precup, "Learning from limited demonstrations," *Advances in Neural Information Processing Systems*, vol. 26, 2013. 38, 77

[93] D. H. Grollman and A. G. Billard, "Robot learning from failed demonstrations," *International Journal of Social Robotics*, vol. 4, pp. 331–342, 2012. 38

[94] H. Guo, Q. Cai, Y. Zhang, Z. Yang, and Z. Wang, "Provably efficient offline reinforcement learning for partially observable markov decision processes," in *International Conference on Machine Learning*. PMLR, 2022, pp. 8016–8038. 38, 74, 77

[95] M. Beliaev, A. Shih, S. Ermon, D. Sadigh, and R. Pedarsani, "Imitation learning by estimating expertise of demonstrators," in *International Conference on Machine Learning*. PMLR, 2022, pp. 1732–1748. 39, 41, 77

[96] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020. 42, 46, 57, 77, 95

[97] K. Kang, P. Gradu, J. J. Choi, M. Janner, C. Tomlin, and S. Levine, "Lyapunov density models: Constraining distribution shift in learning-based control," in *International Conference on Machine Learning*. PMLR, 2022, pp. 10 708–10 733. 42, 75, 77

[98] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, pp. 293–321, 1992. 42

[99] T. Inamura, M. Inaba, and H. Inoue, "Integration model of learning mechanism and dialogue strategy based on stochastic experience representation using bayesian network," in *Proceedings 9th IEEE International Workshop on Robot and Human Interactive Communication. IEEE RO-MAN 2000 (Cat. No. 00TH8499)*. IEEE, 2000, pp. 247–252. 43

[100] J. Saunders, C. L. Nehaniv, and K. Dautenhahn, "Teaching robots by moulding behavior and scaffolding the environment," in *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*, 2006, pp. 118–125. 43, 64

[101] S. Raza, S. Haider, and M.-A. Williams, "Teaching coordinated strategies to soccer robots via imitation," in *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2012, pp. 1434–1439. 43, 56

[102] R. Rahmatizadeh, P. Abolghasemi, and L. Bölöni, "Learning manipulation trajectories using recurrent neural networks," *arXiv preprint arXiv:1603.03833*, 2016. 43

[103] R. Dadashi, L. Hussenot, D. Vincent, S. Girgin, A. Raichuk, M. Geist, and O. Pietquin, "Continuous control with action quantization from demonstrations," *arXiv preprint arXiv:2110.10149*, 2021. 44, 77

[104] J. Kober and J. Peters, "Learning motor primitives for robotics," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 2112–2118. 44, 58

[105] S. Yang, W. Zhang, W. Lu, H. Wang, and Y. Li, "Cross-context visual imitation learning from demonstrations," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 5467–5473. 44, 70, 77

[106] P. Gu, M. Zhao, C. Chen, D. Li, J. Hao, and B. An, "Learning pseudometric-based action representations for offline reinforcement learning," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 7902–7918. 44, 77

[107] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019. 45, 46, 77

[108] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims, "Morel: Model-based offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 810–21 823, 2020. 46, 77

[109] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma, "Mopo: Model-based offline policy optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 129–14 142, 2020. 46, 77

[110] T. Matsushima, H. Furuta, Y. Matsuo, O. Nachum, and S. Gu, "Deployment-efficient reinforcement learning via model-based offline optimization," *arXiv preprint arXiv:2006.03647*, 2020. 46, 77

[111] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn, "Combo: Conservative offline model-based policy optimization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 954–28 967, 2021. 46, 77

[112] M. Farajtabar, Y. Chow, and M. Ghavamzadeh, "More robust doubly robust off-policy evaluation," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1447–1456. 46, 77

[113] P. Thomas and E. Brunskill, "Data-efficient off-policy policy evaluation for reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2016, pp. 2139–2148. 46, 77

[114] Y.-X. Wang, A. Agarwal, and M. Dudık, "Optimal and adaptive off-policy evaluation in contextual bandits," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3589–3597. 46, 77

[115] R. Rafailov, T. Yu, A. Rajeswaran, and C. Finn, "Offline reinforcement learning from images with latent space models," in *Learning for Dynamics and Control*. PMLR, 2021, pp. 1154–1168. 46

[116] P. R. Norvig and S. A. Intelligence, "A modern approach," *Prentice Hall Upper Saddle River, NJ, USA: Rani, M., Nayak, R., & Vyas, OP (2015). An ontology-based adaptive personalized e-learning system, assisted by software agents on cloud storage. Knowledge-Based Systems*, vol. 90, pp. 33–48, 2002. 47

[117] D. Silver, J. A. Bagnell, and A. Stentz, "Learning from demonstration for autonomous navigation in complex unstructured terrain," *The International Journal of Robotics Research*, vol. 29, no. 12, pp. 1565–1592, 2010. 48, 77

[118] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." in *AAAI*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438. 48, 64, 77

[119] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013. 48, 77

[120] E. Klein, M. Geist, B. Piot, and O. Pietquin, "Inverse reinforcement learning through structured classification," *Advances in Neural Information Processing Systems*, vol. 25, 2012. 48, 77

[121] E. Klein, B. Piot, M. Geist, and O. Pietquin, "A cascaded supervised learning approach to inverse reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part I 13*. Springer, 2013, pp. 1–16. 48, 77

[122] N. Das, S. Bechtle, T. Davchev, D. Jayaraman, A. Rai, and F. Meier, "Model-based inverse reinforcement learning from visual demonstrations," in *Conference on Robot Learning*. PMLR, 2021, pp. 1930–1942. 49, 77

[123] H. B. Suay, T. Brys, M. E. Taylor, and S. Chernova, "Learning from demonstration for shaping through inverse reinforcement learning," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 2016, pp. 429–437. 49, 77

[124] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé, "Reinforcement learning from demonstration through shaping," in *Twenty-fourth International Joint Conference on Artificial Intelligence*, 2015. 49, 77

[125] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel, "Learning deep neural network policies with continuous memory states," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 520–527. 50

[126] Y. Zhang, S. Wang, G. Ji *et al.*, "A comprehensive survey on particle swarm optimization algorithm and its applications," *Mathematical Problems in Engineering*, vol. 2015, 2015. 50

[127] C. Zhang, Z. Zhen, D. Wang, and M. Li, "Uav path planning method based on ant colony optimization," in *2010 Chinese Control and Decision Conference*. IEEE, 2010, pp. 3790–3792. 50

[128] R. Cheng and Y. Jin, "A social learning particle swarm optimization algorithm for scalable optimization," *Information Sciences*, vol. 291, pp. 43–60, 2015. 51

[129] J. C. Bongard and G. S. Hornby, "Combining fitness-based search and user modeling in evolutionary robotics," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, 2013, pp. 159–166. 51

[130] T. Brys, A. Harutyunyan, M. E. Taylor, and A. Nowé, "Policy transfer using reward shaping." in *AAMAS*, 2015, pp. 181–188. 51, 77

[131] G. Kuhlmann and P. Stone, "Graph-based domain mapping for transfer learning in general games," in *Machine Learning: ECML 2007: 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007. Proceedings 18*. Springer, 2007, pp. 188–200. 51

[132] L. Torrey, T. Walker, J. Shavlik, and R. Maclin, "Using advice to transfer knowledge acquired in one reinforcement learning task to another," in *Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16*. Springer, 2005, pp. 412–424. 51, 59

[133] D. Ghosh, A. Ajay, P. Agrawal, and S. Levine, "Offline rl policies should be trained to be adaptive," in *International Conference on Machine Learning*. PMLR, 2022, pp. 7513–7530. 51, 77

[134] S. Ikemoto, H. B. Amor, T. Minato, B. Jung, and H. Ishiguro, "Physical human-robot interaction: Mutual learning and adaptation," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 24–35, 2012. 52, 63, 75

[135] Z. Liu, Y. Zhang, Z. Fu, Z. Yang, and Z. Wang, "Learning from demonstration: Provably efficient adversarial policy imitation with linear function approximation," in *International Conference on Machine Learning*. PMLR, 2022, pp. 14 094–14 138. 53

[136] H. Xu, X. Zhan, H. Yin, and H. Qin, "Discriminator-weighted offline imitation learning from suboptimal demonstrations," in *International Conference on Machine Learning*. PMLR, 2022, pp. 24 725–24 742. 53, 77

[137] E. Vollenweider, M. Bjelonic, V. Klemm, N. Rudin, J. Lee, and M. Hutter, "Advanced skills through multiple adversarial motion priors in reinforcement learning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5120–5126. 53, 77

[138] L. Blondé and A. Kalousis, "Sample-efficient imitation learning via generative adversarial nets," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 3138–3148. 53, 77

[139] L. Yu, T. Yu, C. Finn, and S. Ermon, "Meta-inverse reinforcement learning with probabilistic context variables," *Advances in Neural Information Processing Systems*, vol. 32, 2019. 53, 77

[140] B. Wu, F. Xu, Z. He, A. Gupta, and P. K. Allen, "Squirl: Robust and efficient learning from video demonstration of long-horizon robotic manipulation tasks," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 9720–9727. 53, 70, 71, 77, 84

[141] M. Godbout, M. Heuillet, S. C. Raparthy, R. Bhati, and A. Durand, "A game-theoretic perspective on risk-sensitive reinforcement learning." in *SafeAI@ AAAI*, 2022. 53, 77

[142] C.-A. Cheng, T. Xie, N. Jiang, and A. Agarwal, "Adversarially trained actor critic for offline reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 3852–3878. 53, 77

[143] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1597–1607. 54, 70, 82, 83, 85

[144] I. Melekhov, J. Kannala, and E. Rahtu, "Siamese network features for image matching," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 378–383. 54

[145] A. Stooke, K. Lee, P. Abbeel, and M. Laskin, "Decoupling representation learning from reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 9870–9879. 54, 77

[146] M. Laskin, A. Srinivas, and P. Abbeel, "Curl: Contrastive unsupervised representations for reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5639–5650. 54

[147] D. Ghosh, A. Gupta, and S. Levine, "Learning actionable representations with goal-conditioned policies," *arXiv preprint arXiv:1811.07819*, 2018. 54, 77

[148] D. Dwibedi, Y. Aytar, J. Tompson, P. Sermanet, and A. Zisserman, "Temporal cycle-consistency learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1801–1810. 54, 70, 77

[149] G. Berseth, F. Golemo, and C. Pal, "Towards learning to imitate from a single video demonstration," *arXiv preprint arXiv:1901.07186*, 2019. 54, 70, 71

[150] M. Laskin, H. Liu, X. B. Peng, D. Yarats, A. Rajeswaran, and P. Abbeel, "Cic: Contrastive intrinsic control for unsupervised skill discovery," *arXiv preprint arXiv:2202.00161*, 2022. 54, 77

[151] P. Hansen-Estruch, A. Zhang, A. Nair, P. Yin, and S. Levine, "Bisimulation makes analogies in goal-conditioned reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 8407–8426. 55, 77

[152] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1273–1286, 2021. 55

[153] Q. Zheng, A. Zhang, and A. Grover, "Online decision transformer," in *International Conference on Machine Learning*. PMLR, 2022, pp. 27 042–27 059. 55, 77

[154] M. Reid, Y. Yamada, and S. S. Gu, "Can wikipedia help offline reinforcement learning?" *arXiv preprint arXiv:2201.12122*, 2022. 55

[155] H.-L. Hsu, A. K. Bozkurt, J. Dong, Q. Gao, V. Tarokh, and M. Pajic, "Steering decision transformers via temporal difference learning." 55

[156] A. R. Villaflor, Z. Huang, S. Pande, J. M. Dolan, and J. Schneider, "Addressing optimism bias in sequence modeling for reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 22 270–22 283. 55

[157] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021. 56, 103

[158] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023. 56, 113

[159] R. Bhirangi, C. Wang, V. Pattabiraman, C. Majidi, A. Gupta, T. Hellebrekers, and L. Pinto, "Hierarchical state space models for continuous sequence-to-sequence modeling," *arXiv preprint arXiv:2402.10211*, 2024. 56

[160] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine, "Stabilizing off-policy q-learning via bootstrapping error reduction," *Advances in Neural Information Processing Systems*, vol. 32, 2019. 57, 77

[161] Y. Wu, G. Tucker, and O. Nachum, "Behavior regularized offline reinforcement learning," *arXiv preprint arXiv:1911.11361*, 2019. 57, 77

[162] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning," *arXiv preprint arXiv:1910.00177*, 2019. 57, 77

[163] A. Nair, A. Gupta, M. Dalal, and S. Levine, "Awac: Accelerating online reinforcement learning with offline datasets," *arXiv preprint arXiv:2006.09359*, 2020. 57, 77

[164] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20 132–20 145, 2021. 57, 77

[165] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1861–1870. 57, 91

[166] R. Agarwal, D. Schuurmans, and M. Norouzi, "An optimistic perspective on offline reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 104–114. 57

[167] S. Schaal, A. Ijspeert, and A. Billard, "Computational approaches to motor learning by imitation," *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 358, no. 1431, pp. 537–547, 2003. 59

[168] T. L. Paine, C. Paduraru, A. Michi, C. Gulcehre, K. Zolna, A. Novikov, Z. Wang, and N. de Freitas, "Hyperparameter selection for offline reinforcement learning," *arXiv preprint arXiv:2007.09055*, 2020. 59

[169] R. Zhang, B. Dai, L. Li, and D. Schuurmans, "Gendice: Generalized offline estimation of stationary values," *arXiv preprint arXiv:2002.09072*, 2020. 60

[170] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis, "Imitating human playing styles in super mario bros," *Entertainment Computing*, vol. 4, no. 2, pp. 93–104, 2013. 60, 67

[171] I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvari, S. Singh *et al.*, "Behaviour suite for reinforcement learning," *arXiv preprint arXiv:1908.03568*, 2019. 61

[172] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018. 61, 62

[173] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik *et al.*, "Deepmind lab," *arXiv preprint arXiv:1612.03801*, 2016. 61

[174] K. Kurach, A. Raichuk, P. Stańczyk, M. Zając, O. Bachem, L. Espeholt, C. Riquelme, D. Vincent, M. Michalski, O. Bousquet *et al.*, "Google research football: A novel reinforcement learning environment," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 4501–4510. 61

[175] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on Robot Learning*. PMLR, 2020, pp. 1094–1100. 61

[176] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester, "An empirical investigation of the challenges of real-world reinforcement learning," *arXiv preprint arXiv:2003.11881*, 2020. 61, 62

[177] C. Gulcehre, Z. Wang, A. Novikov, T. Paine, S. Gómez, K. Zolna, R. Agarwal, J. S. Merel, D. J. Mankowitz, C. Paduraru *et al.*, "Rl unplugged: A suite of benchmarks for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7248–7259, 2020. 62

[178] 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*, vol. 47, pp. 253–279, 2013. 62, 67

[179] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa, "dm_control: Software and tasks for continuous control," *Software Impacts*, vol. 6, p. 100022, 2020. 62

[180] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov, "Minerl: A large-scale dataset of minecraft demonstrations," *arXiv preprint arXiv:1907.13440*, 2019. 62

[181] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay *et al.*, "Roboturk: A crowdsourcing platform for robotic skill learning through imitation," in *Conference on Robot Learning*. PMLR, 2018, pp. 879–893. 62

[182] J. Fu, M. Norouzi, O. Nachum, G. Tucker, Z. Wang, A. Novikov, M. Yang, M. R. Zhang, Y. Chen, A. Kumar *et al.*, "Benchmarks for deep off-policy evaluation," *arXiv preprint arXiv:2103.16596*, 2021. 62, 72

[183] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-First International Conference on Machine learning*, 2004, p. 1. 64, 77

[184] C. He, B. G. León, and F. Belardinelli, "Do androids dream of electric fences? safety-aware reinforcement learning with latent shielding," *arXiv preprint arXiv:2112.11490*, 2021. 75, 77

[185] V. P. Patil, M. Hofmarcher, M.-C. Dinu, M. Dorfer, P. M. Blies, J. Brandstetter, J. A. Arjona-Medina, and S. Hochreiter, "Align-rudder: Learning from few demonstrations by reward redistribution," *arXiv preprint arXiv:2009.14108*, 2020. 77

[186] T. Yu, A. Kumar, Y. Chebotar, K. Hausman, S. Levine, and C. Finn, "Conservative data sharing for multi-task offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 11 501–11 516, 2021. 74, 77

[187] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018. 64, 77

[188] E. Johns, "Coarse-to-fine imitation learning: Robot manipulation from a single demonstration," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4613–4619. 72, 77

[189] N. Polosky, B. C. Da Silva, M. Fiterau, and J. Jagannath, "Constrained offline policy optimization," in *International Conference on Machine Learning*. PMLR, 2022, pp. 17 801–17 810. 75, 77

[190] H. Yuan and Z. Lu, "Robust task representations for offline meta-reinforcement learning via contrastive learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 25 747–25 759. 74, 77

[191] K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer, "Ensembledagger: A bayesian approach to safe imitation learning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 5041–5048. 75, 77

[192] D. Yarats, D. Brandfonbrener, H. Liu, M. Laskin, P. Abbeel, A. Lazaric, and L. Pinto, "Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning," *arXiv preprint arXiv:2201.13425*, 2022. 74, 77

[193] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei, "Learning to generalize across long-horizon tasks from human demonstrations," *arXiv preprint arXiv:2003.06085*, 2020. 73, 77

[194] A. Mandlekar, F. Ramos, B. Boots, S. Savarese, F.-F. L. 0001, A. Garg, and D. Fox, "Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data," in *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*. IEEE, 2020, pp. 4414–4420. 73, 77

[195] R. Hoque, A. Balakrishna, C. Putterman, M. Luo, D. S. Brown, D. Seita, B. Thananjeyan, E. Novoseller, and K. Goldberg, "Lazydagger: Reducing context switching in interactive imitation learning," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 502–509. 77

[196] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018. 66, 77

[197] S. Paul, J. Vanbaar, and A. Roy-Chowdhury, "Learning from trajectories via subgoal discovery," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. 73, 77

[198] Y. Zhou, Y. Aytar, and K. Bousmalis, "Manipulator-independent representations for visual imitation," *arXiv preprint arXiv:2103.09016*, 2021. 77

[199] A. K. Tanwani, A. Yan, J. Lee, S. Calinon, and K. Goldberg, "Sequential robot imitation learning from observations," *The International Journal of Robotics Research*, vol. 40, no. 10-11, pp. 1306–1325, 2021. 77

[200] T. Xiao, I. Radosavovic, T. Darrell, and J. Malik, "Masked visual pre-training for motor control," *arXiv preprint arXiv:2203.06173*, 2022. 77

[201] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," *arXiv preprint arXiv:1709.07174*, 2017. 64, 77

[202] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3406–3413. 66, 77

[203] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J. E. Gonzalez, J. Ibarz, C. Finn, and K. Goldberg, "Recovery rl: Safe reinforcement learning with learned recovery zones," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, 2021. 75, 77

[204] T. A. K. Faulkner, E. S. Short, and A. L. Thomaz, "Interactive reinforcement learning with inaccurate feedback," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7498–7504. 74, 77

[205] N. Rhinehart, R. McAllister, and S. Levine, "Deep imitative models for flexible inference, planning, and control," *arXiv preprint arXiv:1810.06544*, 2018. 64, 77

[206] E. Chane-Sane, C. Schmid, and I. Laptev, "Goal-conditioned reinforcement learning with imagined subgoals," in *ICML*, 2021. 73, 77, 95

[207] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," *arXiv preprint arXiv:1910.11956*, 2019. 77

[208] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter, "Rudder: Return decomposition for delayed rewards," *Advances in Neural Information Processing Systems*, vol. 32, 2019. 73, 77

[209] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end simulated driving," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017. 75, 77

[210] A. Raghu, M. Komorowski, I. Ahmed, L. Celi, P. Szolovits, and M. Ghassemi, "Deep reinforcement learning for sepsis treatment," *arXiv preprint arXiv:1711.09602*, 2017. 66, 77

[211] K. Pertsch, Y. Lee, and J. J. Lim, "Accelerating reinforcement learning with learned skill priors," in *Conference on Robot Learning (CoRL)*. PMLR, 2021, pp. 188–204. 73, 77

[212] L. Wang, W. Zhang, X. He, and H. Zha, "Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2447–2456. 66, 77

[213] S. Krishnan, A. Garg, R. Liaw, B. Thananjeyan, L. Miller, F. T. Pokorny, and K. Goldberg, "Swirl: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards," *The international Journal of Robotics Research*, vol. 38, no. 2-3, pp. 126–145, 2019. 73, 77

[214] J. Shang and M. S. Ryoo, "Self-supervised disentangled representation learning for third-person imitation learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 214–221. 70, 77

[215] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control," *arXiv preprint arXiv:1812.00568*, 2018. 66, 77

[216] S. Yang, Y. Feng, S. Zhang, and M. Zhou, "Regularizing a model-based policy stationary distribution to stabilize offline reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 24 980–25 006. 77

[217] R. Bemelmans, G. J. Gelderblom, P. Jonker, and L. De Witte, "Socially assistive robots in elderly care: a systematic review into effects and effectiveness," *Journal of the American Medical Directors Association*, vol. 13, no. 2, pp. 114–120, 2012. 63

[218] C. Sammut, S. Hurst, D. Kedzier, and D. Michie, "Learning to fly," in *Machine Learning Proceedings 1992*. Elsevier, 1992, pp. 385–393. 64

[219] K. Mo, H. Li, Z. Lin, and J.-Y. Lee, "The adobeindoornav dataset: Towards deep reinforcement learning based real-world indoor robot visual navigation," *arXiv preprint arXiv:1802.08824*, 2018. 64

[220] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 year, 1000 km: The oxford robotcar dataset," *The International Journal of Robotics Research*, vol. 36, no. 1, pp. 3–15, 2017. 64

[221] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, T. Darrell *et al.*, "Bdd100k: A diverse driving video database with scalable annotation tooling," *arXiv preprint arXiv:1805.04687*, vol. 2, no. 5, p. 6, 2018. 64

[222] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016. 64

[223] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python." in *SciPy*, 2015, pp. 18–24. 64

[224] M. Marchellus and I. K. Park, "M2c: Concise music representation for 3d dance generation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 3126–3135. 64, 116

[225] J. Tseng, R. Castellon, and K. Liu, "Edge: Editable dance generation from music," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 448–458. 65, 116

[226] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv preprint arXiv:2005.00341*, 2020. 65

[227] T. Tang, J. Jia, and H. Mao, "Dance with melody: An lstm-autoencoder approach to music-oriented dance synthesis," in *Proceedings of the 26th ACM International Conference on Multimedia*, 2018, pp. 1598–1606. 65, 116

[228] K. Chen, Z. Tan, J. Lei, S.-H. Zhang, Y.-C. Guo, W. Zhang, and S.-M. Hu, "Choreomaster: choreography-oriented music-driven dance synthesis," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 4, pp. 1–13, 2021. 65, 116

[229] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018. 65

[230] B. Nie and Y. Gao, "Dancehat: Generate stable dances for humanoid robots with adversarial training," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8511–8517. 65, 116

[231] G. Sun, Y. Wong, Z. Cheng, M. S. Kankanhalli, W. Geng, and X. Li, "Deepdance: music-to-dance motion choreography with adversarial learning," *IEEE Transactions on Multimedia*, vol. 23, pp. 497–509, 2020. 65, 116

[232] R. Li, S. Yang, D. A. Ross, and A. Kanazawa, "Ai choreographer: Music conditioned 3d dance generation with aist++," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 13 401–13 412. 65, 113, 115, 116, 118

[233] L. Siyao, W. Yu, T. Gu, C. Lin, Q. Wang, C. Qian, C. C. Loy, and Z. Liu, "Bailando: 3d dance generation by actor-critic gpt with choreographic memory," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 050–11 059. 65, 116, 118

[234] K. Strabala, M. K. Lee, A. Dragan, J. Forlizzi, S. S. Srinivasa, M. Cakmak, and V. Micelli, "Toward seamless human-robot handovers," *Journal of Human-Robot Interaction*, vol. 2, no. 1, pp. 112–132, 2013. 66

[235] O. Gottesman, F. Johansson, M. Komorowski, A. Faisal, D. Sontag, F. Doshi-Velez, and L. A. Celi, "Guidelines for reinforcement learning in healthcare," *Nature medicine*, vol. 25, no. 1, pp. 16–18, 2019. 66

[236] H.-H. Tseng, Y. Luo, S. Cui, J.-T. Chien, R. K. Ten Haken, and I. E. Naqa, "Deep reinforcement learning for automated radiation adaptation in lung cancer," *Medical Physics*, vol. 44, no. 12, pp. 6690–6705, 2017. 66

[237] A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark, "Mimic-iii, a freely accessible critical care database," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016. 66

[238] D. Vogt, H. Ben Amor, E. Berger, and B. Jung, "Learning two-person interaction models for responsive synthetic humanoids," *Journal of Virtual Reality and Broadcastings*, vol. 11, no. 1, 2014. 66

[239] S. Calinon and A. Billard, "Incremental learning of gestures by imitation in a humanoid robot," in *Proceedings of the ACM/IEEE International Conference on Human-robot Interaction*, 2007, pp. 255–262. 66

[240] A. Ude, C. G. Atkeson, and M. Riley, "Programming full-body movements for humanoid robots by observation," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 93–108, 2004. 66

[241] P. Hingston, *Believable bots: can computers play like people?* Springer, 2012. 66

[242] P. Sermanet, K. Xu, and S. Levine, "Unsupervised perceptual rewards for imitation learning," *arXiv preprint arXiv:1612.06699*, 2016. 70

[243] H. Xuan, A. Stylianou, X. Liu, and R. Pless, "Hard negative examples are hard, but useful," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*. Springer, 2020, pp. 126–142. 70

[244] V. Kurenkov and S. Kolesnikov, "Showing your offline reinforcement learning work: Online evaluation budget matters," in *International Conference on Machine Learning*. PMLR, 2022, pp. 11 729–11 752. 72

[245] C. Voloshin, H. M. Le, N. Jiang, and Y. Yue, "Empirical study of off-policy policy evaluation for reinforcement learning," *arXiv preprint arXiv:1911.06854*, 2019. 72

[246] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 31, 2018. 73

[247] K. Pertsch, O. Rybkin, F. Ebert, S. Zhou, D. Jayaraman, C. Finn, and S. Levine, "Long-horizon visual planning with goal-conditioned hierarchical predictors," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17 321–17 333, 2020. 73

[248] A. De Santis, B. Siciliano, A. De Luca, and A. Bicchi, "An atlas of physical human–robot interaction," *Mechanism and Machine Theory*, vol. 43, no. 3, pp. 253–270, 2008. 75

[249] A. Reichlin, G. L. Marchetti, H. Yin, A. Ghadirzadeh, and D. Kragic, "Back to the manifold: Recovering from out-of-distribution states," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022. 75

[250] N. C. Wagener, B. Boots, and C.-A. Cheng, "Safe reinforcement learning using advantage-based intervention," in *International Conference on Machine Learning*. PMLR, 2021. 75

[251] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019. 75

[252] M. Godbout, M. Heuillet, S. C. Raparthy, R. Bhati, and A. Durand, "A game-theoretic perspective on risk-sensitive reinforcement learning." in *SafeAI@ AAAI*, 2022. 75

[253] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics*, 2011. 75

[254] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823. 81

[255] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 1735–1742. 81, 82

[256] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*. Pmlr, 2014, pp. 387–395. 85

[257] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in Neural Information Processing Systems*, vol. 30, 2017. 85

[258] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019. 86

[259] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Ieee, 2009, pp. 248–255. 86

[260] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*. PMLR, 2018. 91

[261] N. Hansen, H. Su, and X. Wang, "Stabilizing deep q-learning with convnets and vision transformers under data augmentation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 3680–3693, 2021. 95

[262] T. Yu, A. Kumar, Y. Chebotar, K. Hausman, S. Levine, and C. Finn, "Conservative data sharing for multi-task offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 11 501–11 516, 2021. 95

[263] F. Ofli, E. Erzin, Y. Yemez, and A. M. Tekalp, "Learn2dance: Learning statistical music-to-dance mappings for choreography synthesis," *IEEE Transactions on Multimedia*, vol. 14, no. 3, pp. 747–759, 2011. 113

[264] A. Aristidou, A. Yiannakidis, K. Aberman, D. Cohen-Or, A. Shamir, and Y. Chrysanthou, "Rhythm is a dancer: Music-driven motion synthesis with global structure," *IEEE Transactions on Visualization and Computer Graphics*, 2022. 113

[265] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in Neural Information Processing Systems*, vol. 30, 2017. 120