Brain-Inspired Planning for Better Generalization in Reinforcement Learning

Mingde “Harry” Zhao, School of Computer Science
McGill University, Montréal
Nov, 2025

An **updated version** of the thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Ph.D. in Computer Science

Abstract

Existing Reinforcement Learning systems encounter significant challenges when applied to real-world scenarios, primarily due to poor generalization across environments that differ from their training conditions. This thesis explores the direction of enhancing agents' zero-shot systematic generalization abilities by granting RL agents reasoning behaviors that are found to help systematic generalization in the human brain. Inspired by human conscious planning behaviors, we first introduced a top-down attention mechanism, which allows a decision-time planning agent to dynamically focus its reasoning on the most relevant aspects of the environmental state given its instantaneous intentions, a process we call "spatial abstraction". This approach significantly improves systematic generalization outside the training tasks. Subsequently, building on spatial abstraction, we developed the Skipper framework to automatically decompose complex tasks into simpler, more manageable sub-tasks. Skipper provides robustness against distributional shifts and efficacy in long-term, compositional planning by focusing on pertinent spatial and temporal elements of the environment. Finally, we identified a common failure mode and safety risk in planning agents that rely on generative models to generate state targets during planning. It is revealed that most agents blindly trust the targets they hallucinate, resulting in delusional planning behaviors. Inspired by how the human brain rejects delusional intentions, we propose learning a feasibility evaluator to enable rejecting hallucinated infeasible targets, which led to significant performance improvements in various kinds of planning agents. Finally, we suggest directions for future research, aimed at achieving general task abstraction and fully enabling abstract planning.

Table of Contents

Abstract i

Nomenclature & Indices vii

Terminologies & Abbreviations . . . vii
List of Figures . . . xiii
List of Tables . . . xiv

1 Introduction 1

1.1 Thesis Overview . . . 1
1.2 Summary of Original Contributions to Knowledge . . . 2
1.3 Collaborator Contributions Breakdown . . . 4
1.3.1 Chap. 4 - Conscious Planning: Spatially-Abstract Decision-Time Reasoning . . . 4
1.3.2 Chap. 5 - Skipper Framework: Spatio-Temporal Abstractions for Planning . . . 4
1.3.3 Chap. 6 - Rejecting Hallucinations: Addressing Delusional Planning Behaviors . . . 4
1.4 Copyright . . . 5

2 Literature Review: Reinforcement Learning 6

2.1 What is Reinforcement Learning? . . . 7
2.1.1 Learning by Trial-and-Error . . . 7
2.1.2 Reward Maximization in Sequential Decision-Making . . . 7
2.1.3 Markov Decision Processes . . . 9
2.1.4 RL Objectives in MDPs (Episodic Setting) . . . 12
2.1.5 Policies & Value Functions . . . 13
2.1.6 Optimal Policies & Optimal Value Functions . . . 14
2.2 Policy Evaluation & Policy Improvement . . . 16
2.3 Dynamic Programming (DP) . . . 17
2.3.1 DP for Value Function . . . 17
2.3.2 DP for Optimal Policy . . . 19
2.4 (Approximate) RL . . . 19
2.4.1 RL Agent Components . . . 20
2.4.2 Train & Evaluate . . . 21
2.4.3 Temporal Difference Learning . . . 21
2.4.4 Off-Policy Evaluation . . . 22

2.4.5 Q-Learning . . . 23
2.4.6 Function Approximation . . . 23
2.4.7 Semi-Gradient Methods for Learning Function Approximators . . . 24
2.4.8 Off-policy Methods with Function Approximation . . . 25
2.5 Temporal Abstraction . . . 25
2.5.1 Semi-Markov Decision Processes (SMDPs) . . . 26
2.5.2 Options: SMDPs with Policies as Macro Actions . . . 27
2.5.3 Goal Conditioned RL & Options . . . 28

3 Literature Review: Model-Based Deep RL 29
3.1 Deep RL: Neural Network based Methods . . . 30
3.1.1 DQN . . . 30
3.1.2 Distributional RL . . . 32
3.1.3 Goal-Conditioned RL: Source-Target Pairs & Hindsight Relabeling . . . 34
3.2 Deep Learning Preliminaries . . . 35
3.2.1 Generative Models: VAEs & Conditional VAEs . . . 35
3.2.2 Attention . . . 37
3.3 Model-based RL . . . 41
3.3.1 Sources of Models: Where They Come From . . . 42
3.3.2 Timing of Planning: Decision-Time *v.s.* Background . . . 43
3.3.3 Usage of Models: What They Estimate & How They Help . . . 47
3.3.4 Inspiring Developments in Model-Based RL . . . 49
3.4 Other Related Background Knowledge . . . 51
3.4.1 Consciousness in the First Sense & Conscious Planning . . . 51
3.4.2 Generalization-Focused Environments & Experimental Settings . . . 51

4 Conscious Planning: Spatially-Abstract Decision-Time Reasoning 57
4.1 Overview of This Thesis Milestone . . . 58
4.2 Discussions of Methodologies . . . 59
4.2.1 About Reconstruction, Observation-Level & State-Level Planning . . . 59
4.2.2 About Staged Training with World Models . . . 59
4.2.3 About Vectorized *v.s.* Set-based State Representations . . . 60
4.3 Methodology: Model-based RL with Set Representations . . . 60
4.3.1 State Representation Encoder . . . 61
4.3.2 (State-Action) Q-Value Estimator . . . 61
4.3.3 Transition Model . . . 61
4.3.4 Training . . . 64
4.4 Methodology: Consciousness-Inspired Bottleneck . . . 65
4.4.1 Conditional State Selection . . . 66
4.4.2 Dynamics / Reward-Termination Prediction on Bottleneck Sets . . . 66
4.4.3 Change Integration . . . 67

4.4.4 Discussion . . . 67
4.4.5 Birdseye View of Overall Design . . . 68
4.5 Research Findings: Experiments . . . 68
4.5.1 Environment / Task Description . . . 68
4.5.2 Compared Methods . . . 70
4.5.3 Agent Variants . . . 70
4.5.4 Performance Evaluation (RDS with Turn-Or-Forward Dynamics) . . . 71
4.5.5 Ablation Studies . . . 73
4.5.6 Sensitivity Studies . . . 76
4.5.7 More Experiments, Discussions & Failed Attempts . . . 77
4.5.8 Details of Baselines . . . 79
4.5.9 Summary of Experiments . . . 81
4.6 Summary . . . 81

5 Skipper Framework: Spatio-Temporal Abstractions for Planning 82
5.1 Overview of This Thesis Milestone . . . 82
5.2 Methodology: Proxy Problems . . . 83
5.2.1 Definition . . . 83
5.2.2 Potential of Proxy Problems . . . 85
5.3 Methodology: Skipper Framework - Spatially & Temporally Abstract Planning . . . 88
5.3.1 Problem 1: Edge Estimation . . . 88
5.3.2 Problem 2: Vertex Generation . . . 92
5.3.3 Skipper Framework: Assemble Everything . . . 95
5.4 Discussions of Methodologies . . . 97
5.4.1 About Proxy Problems . . . 97
5.4.2 About Task Decomposition with Selected States / Sub-Goals . . . 97
5.4.3 About Temporal Abstraction . . . 98
5.4.4 About Spatial Abstraction . . . 98
5.4.5 About Planning Estimates . . . 99
5.5 Research Findings: Experiments . . . 99
5.5.1 Generalization Performance with 50 Training Environments . . . 100
5.5.2 Scalability Studies: Number of Training Tasks . . . 101
5.5.3 Validation of Claims . . . 104
5.5.4 Ablation Studies . . . 107
5.5.5 Sensitivity Studies . . . 109
5.5.6 Summary of Experiments . . . 110
5.6 Summary . . . 111

6 Rejecting Hallucinations: Addressing Delusional Planning Behaviors 112
6.1 Overview of This Thesis Milestone . . . 112
6.2 Methodology: Target Evaluation & Relabeling . . . 114

6.2.1 Target-Assisted Planning & Targets . . . 114
6.2.2 Source-Target Pairs & Hindsight Relabeling . . . 116
6.3 Methodology: Understanding Hallucinated State Targets . . . 116
6.3.1 G.0: ∞-Feasible . . . 116
6.3.2 G.1 - Permanently Infeasible (Hallucinated) . . . 117
6.3.3 G.2 - Temporarily Infeasible (Hallucinated) . . . 117
6.3.4 Examples . . . 117
6.3.5 Non-Singleton Targets . . . 118
6.4 Methodology: Evaluating Targets Correctly and Robustly . . . 118
6.4.1 Desiderata for Evaluator . . . 119
6.4.2 Learning Rule & Architecture for Feasibility . . . 119
6.4.3 Training Data for Feasibility . . . 120
6.4.4 Computational Overhead . . . 122
6.5 Discussions of Methodologies . . . 123
6.5.1 About TAP Agents . . . 123
6.5.2 About Hindsight Relabeling . . . 124
6.5.3 About Delusions & Delusional Behaviors . . . 124
6.5.4 About Goal Mis-Generalization . . . 125
6.6 Research Findings: Experiments . . . 125
6.6.1 Rejecting Infeasible Goals in Decision-Time Planning (Exp. 1/8 - 4/8) . . . 126
6.6.2 Rejecting Destabilizing Updates in Background Planning (Exp. 5/8 - Exp. 6/8) . . . 133
6.6.3 Feasibility Convergence to Non-Singleton Targets (Exp. 7/8 & 8/8) . . . 134
6.7 Summary . . . 135

7 Discussions: Conclusions, Limitations & Future Work 137
7.1 Summary of Contributions . . . 137
7.1.1 Conscious Planning: Spatially-Abstract Decision-Time Reasoning . . . 137
7.1.2 Skipper Framework: Spatio-Temporal Abstractions for Planning . . . 139
7.1.3 Rejecting Hallucinations: Addressing Delusional Planning Behaviors . . . 140
7.2 Limitations . . . 141
7.2.1 Limitations of Work on Conscious Planning (Chap. 4) . . . 141
7.2.2 Limitations of Work on Skipper (Chap. 5) . . . 142
7.2.3 Limitations of Work on Rejecting Hallucinations (Chap. 6) . . . 143
7.3 Future Work . . . 143

8 APPENDICES 159
8.1 Auxiliaries—CP . . . 159
8.1.1 Additional Experimental Insights . . . 159
8.1.2 Experiment Configurations . . . 160
8.2 Auxiliaries—Skipper . . . 161
8.2.1 Skipper . . . 161

8.2.2 LEAP 164
8.2.3 Director 165
8.3 Auxiliaries—Delusions 166
8.3.1 Implementation of PERTASK 166
8.3.2 Implementation Details for Experiments 167
8.3.3 Applying the Evaluator on Dreamerv2 168

Nomenclature & Indices

Terminologies & Abbreviations

For the readers' convenience in cross-referencing, the rows are sorted alphabetically and some terms defined in the following list are highlighted in **bold**.

Agent	An entity that makes decisions and takes actions in an environment. Often refers to the computational methods.
Actor-Critic	A reinforcement learning algorithm with two components: one for taking action (actor) and one for value estimation (critic). See Sec. 2.2 on Page. 16.
Attention	A mechanism allowing computations to focus on parts of the input. See Sec. 3.2.2 on Page. 37.
Auxiliary	Additional tasks or information used to assist a model. We use the term "auxiliary learners" to describe components in a reinforcement learning agent that do not conduct **value estimation**. See Sec. 2.4.1 on Page. 20 and Sec. 3.1.2 on Page. 33.
Background Planning	A planning behavior that does not immediately improve an agent's next decision (Alver and Precup, 2024).
Baseline	An existing method used as a reference point for the performance of new methods. When a new agent is proposed, existing agents can serve as baselines. When a variant of an agent is proposed, the agent (in its original form) should serve as the baseline.
Behavior Policy	An agent's adopted policy when interacting with the environment. Introduced in Sec. 2.4.4 on Page. 22 to differentiate with the target policy.
Bottleneck	A point in a system that limits overall performance. Chap. 4 contributes a bottleneck mechanism limiting the number of objects that the model could **reason** with at decision time (Chap. 4 on Page. 57).
Checkpoints	Saved snapshots of model parameters, agent or environment states, containing complete information with which the previous behaviors could be resumed. Chap. 5 uses checkpoints to differentiate *the subset of states imagined by the generative model* from regular states, while also emphasizing their info-completeness (Sec. 5.2, Page. 83).
Consciousness	A capability that refers to agents that can act with awareness of the states of the world and the states of the self.
Consciousness in the 1st Sense (C1)	A capability allowing humans to act with awareness of the relevant environmental entities (Dehaene et al., 2020). C2 refers to the awareness of the self.
Decision Point	A specific state or point in time where an agent needs to make a decision.

Decision-Time The moment when an agent must make a decision about how to act next

Decision-Time Planning The behavior of planning at **decision-time** to reason about the decision that is to be made imminently (Alver and Precup, 2024).

Deep Learning A subset of machine learning using more-than-one layer of artificial neural networks to establish learning systems (Goodfellow et al., 2016).

Delusions Obviously false ideas / beliefs which an agent is unable to reject. Reflects a pathology in the learning system (Kiran and Chaudhury, 2009).

Delusional Planning Behaviors Behaviors triggered by an agent's delusions. Describes agents' often erratic actions while seeking to achieve infeasible targets (Chap. 6, Page. 112).

Discrete Action Space A set of distinct, countable actions decision-making agents can choose from.

Dynamics The rules and processes governing the state transitions of an environment.

Episode A single run or sequence of interactions between an agent and an environment, from start to finish. Often marked with an initial and a terminal state.

Environment The external world with which decision-making agents interact and draw observations, perhaps rewards. See Fig. 2.1 on Page. 9.

Estimator A model or method used to estimate certain quantities, like **value estimators** that estimate **value functions** in reinforcement learning.

Evaluator In Chap. 6, this refers to an estimator used to produce an estimate indicating whether a target is feasible, named after the belief evaluation system in the human brain (Kiran and Chaudhury, 2009). See Sec. 6.1 on Page. 112.

Generator A model or process that produces data. In Chap. 6, it refers to the component that proposes targets for the agent to reason with, named after the brain's belief formation system (Kiran and Chaudhury, 2009). See Sec. 6.1 on Page. 112.

Goal A desired outcome an agent seeks to achieve. Discussed in Chap. 2 on Page. 6.

Gridworld A grid-based environment used for evaluating agents. Agents move from one cell to another by taking navigation actions, and possibly interacts with objects located in certain cells.

Hallucination The behavior of generating nonfactual beliefs.

Intuition Describes **estimators**' inexplicable knowledge, also the ability to make fast predictions without reasoning. Correspond to System-1 (Kahneman, 2017).

KL-Divergence The Kullback–Leibler (KL) divergence, *a.k.a.* relative entropy, is a measure of the difference between two probability distributions. Formally, it is defined as: $D_{\mathrm{KL}}(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$. It can be used as a loss function to make the distributional output of a neural network closer to its **update targets**.

Loss Function A function used to measure the quality of the output of a model, which can then guide model training.

Lookup Table A table used for quick retrieval of precomputed values.

Macro Actions High-level actions that are sequences of smaller, simpler actions. See Sec. 2.5.1 on Page. 26.

Model-Based RL A type of reinforcement learning assisted by predictive / generative models. See Sec. 3.3 on Page. 41.

Optimizer In the context of training parametric models with gradient-based methods, an algorithm used to optimize an objective function.

Option A structured temporally extended action, sometimes representing a **macro-action**. See Sec. 2.5.2 on Page. 27.

Oracle A model established over ground truths that learning agents should no access to.

Planning The reasoning process of deciding what actions to take in the relative future. See Sec. 3.3.2 on Page. 43.

Policy The mechanism that defines how an agent would act given different environmental states. See Sec. 2.1.5 on Page. 13.

Priority An ordering determining which work is to be done first. Tree search algorithms use priority queues to determine which branch of the search tree is to be simulated next. See Sec. 3.3.2 on Page. 43.

Proxy Problem A simplified version of a problem used as a proxy of the full problem. Chap. 5 uses proxy problems as a way to decompose Markov decision processes. See Sec. 5.2 on Page. 83.

Reasoning The ability of an AI system to infer new information based on available knowledge. **Planning** is a form of reasoning.

Regularization Techniques used in machine learning to prevent overfitting (Goodfellow et al., 2016).

Scalability The ability of a model or algorithm to handle increased amounts of data or complexity effectively. In Chap. 5, we tested the agents' generalization abilities with increasing numbers of training task instances. See Sec. 5.5 on Page. 99.

Semi-Hard Attention A variant of **attention** mechanism, where the attention weights are continuous but can take the discrete value 0 to fully rule out some choices. See Sec. 3.2.2 on Page. 40.

Source-Target Pairs Pairs of data where one element is used as the source (input) and the other as the target (desired output) in learning. In Chap. 6, we use **source-target pairs** to differentiate how **TAP agents** learn based on two decision points, to learn the relationship between a source state and a target. See Sec. 3.1.3 on Page. 34 and Chap. 6 on Page. 112.

Spatial Abstraction A computational description of **consciousness in the 1st sense**. Refers to an agent's ability to focus on partial aspects of the state for decision-making. Spatial abstraction is a special, dynamic, intention-dependent form of state abstraction. Named intuitively to differentiate from generic state abstraction and to rhyme with temporal abstraction.

State Representation An encoding of the state of the **environment**, used by agents' policies for decision-making. See Sec. 2.1.2 on Page. 8.

System-2 A mode of computation in the human brain that involves deliberate, slow, and logical **reasoning** on-the-fly, as opposed to **intuition**-based System-1 thinking (Kahneman, 2017).

Target An abbreviation of a "state target". Chap. 6 uses targets to refer to the observations or states or sets of states generated by the model of a planning agent that can be used to guide its behaviors. Targets correspond to a set of states that agents seek to achieve. Note that a **target state** instead corresponds to a singleton target, *i.e.*, a target of a single state. See Sec. 3.1.3 on Page. 34 and Chap. 6 on Page. 112.

Target-Assisted Planning (TAP) Target-Assisted Planning (TAP) is a methodology of decision-making agents that generate **targets** during planning. See Sec. 3.1.3 on Page. 34 and Chap. 6 on Page. 112.

Top-k A technique often used in machine learning to focus on the top k results of a search or model, based on certain criteria. A top-k mechanism is used to implement **semi-hard attention**. See Sec. 3.2.2 on Page. 40.

Transition A change in the environment's state due to an agent's action. Specifically in reinforcement learning, it refers to a data structure resembling $\langle s, a, r, s' \rangle$, where s and s' are two consecutive environmental states, where the transition between them is triggered by the agent taking action a, and r is the immediate reward received. See Sec. 2.1.3 on Page. 9.

Trajectory A sequence of states, actions, and environmental feedbacks an agent experiences during an episode. See Sec. 2.1.3 on Page. 9.

Update Target The target towards which an update of an estimate is made. For example, in Sec. 2.4.3 (Page. 21), we discussed how temporal difference learning constructs its update targets.

Value A measure of the desirability of a state or state-action pair in reinforcement learning, defined as expected return. See Sec. 2.1.5 on Page. 13.

Value Estimate An approximation of the **value**, learned by a **value estimator**. See Sec. 2.4.1 on Page. 20.

Value Function A function that outputs the **value** of a given state in RL. Defined in Sec. 2.1.5 on Page. 13.

Vanilla A term often used to describe basic versions of ice-creams and algorithms.

World Model A model that represents the agent's understanding of the environment, used for planning and decision-making. Specifically, in Chap. 4, it is used to refer to a model-based planning methodology proposed in Ha and Schmidhuber (2018), which trains a model of the environment independently of any rewards, through an unsupervised exploration stage. See Sec. 4.2.2 on Page. 59.

List of Figures

1.1 Main Chapters in this Thesis (Research Methodologies & Original Findings) . . . 3

2.1 Agent-Environment Interaction in RL . . . 9

3.1 Dissection of DQN . . . 32
3.2 C51 Distributional Variant of DQN Value Estimators . . . 33
3.3 VAE Mechanisms . . . 36
3.4 Querying A Set with an Object . . . 39
3.5 Best-First Tree Search for MPC . . . 46
3.6 Rendering `RDS` Instances . . . 53
3.7 Rendering `SSM` Instances . . . 54
3.8 `SSM` State Structure . . . 55

4.1 Set-based State Representation Encoder . . . 61
4.2 Value Estimator & Set-to-Vec Architecture . . . 62
4.3 Reward-Termination Estimator based on Set Transitions . . . 62
4.4 Set-Based Dynamics Model . . . 63
4.5 Action-Conditioned Transformer Layer . . . 64
4.6 Bottleneck-Enabled Set-Based Dynamics Model . . . 65
4.7 Design of the Bottleneck Selector . . . 66
4.8 Design of the Bottleneck Integrator . . . 67
4.9 Overall Organization of Proposed Components for CP Agent . . . 69
4.10 Multitask RL Setting, with In-Distribution and OOD Tasks on `RDS` . . . 70
4.11 In-Distribution Task Performance . . . 71
4.12 OOD performance of Compared Agents under a Gradient of Difficulties . . . 72
4.13 Reasoning Addresses Generalization Gap, Bottleneck benefits OOD capability . . . 72
4.14 Visualization of Attention Selection . . . 73
4.15 Ablation Results regarding Spatial Abstraction . . . 74
4.16 In-Distribution Model Performance . . . 75
4.17 Impact of Action Regularization Loss . . . 76
4.18 Sensitivity to Bottleneck Sizes (controlled by k) . . . 76
4.19 Success Rates given Different Numbers of Planning Steps . . . 77
4.20 Scalability of OOD Performance under a Spectrum of Difficulties and World Sizes . . . 78

4.21 OOD Performance of Compared Agents in "Turn-and-Forward" Tasks . . . 78
4.22 Extended Run to Demonstrate the OOD Performance Differences of CP and WM agents . . . 80
4.23 NOSET Baseline Performance on Multitask and Monotask Settings . . . 80

5.1 A Proxy Problem on an RDS Task . . . 84
5.2 Illustration of the Spatial Abstraction Mechanism . . . 90
5.3 Simultaneously Estimating Distributions of Discounts & Distances . . . 91
5.4 Checkpoint Generator Training & Inference . . . 93
5.5 Skipper Framework . . . 96
5.6 Evaluation Performance Evolution of Compared Agents during Training . . . 101
5.7 Evaluation Performance of Agents with Different Numbers of Training Tasks . . . 102
5.8 Evaluation Performance Evolution of Skipper-once with Different Numbers of Training Tasks 102
5.9 Performance of Skipper-regen with Different Numbers of Training Tasks . . . 103
5.10 Evaluation Performance Evolution of **modelfree** with Different Numbers of Training Tasks 103
5.11 Evaluation Performance Evolution of LEAP with Different Numbers of Training Tasks . . . 103
5.12 Evaluation Performance Evolution of Director with Different Numbers of Training Tasks . . . 104
5.13 Evaluation Performance of Agents Trained on 1 Task . . . 104
5.14 Evaluation Performance of Agents Trained on 5 Tasks . . . 105
5.15 Evaluation Performance of Agents Trained on 25 Tasks . . . 105
5.16 Evaluation Performance of Agents Trained on 50 Tasks . . . 105
5.17 Evaluation Performance of Agents Trained on 100 Tasks . . . 106
5.18 Evaluation Performance of Agents Trained on ∞ Tasks . . . 106
5.19 Evaluation Performance of Agents in **Stochastic** Environments . . . 106
5.20 Evaluation Performance of Skipper-once *v.s.* Oracle Agents . . . 107
5.21 Ablation for Spatial Abstraction on Skipper-once agent . . . 108
5.22 Ablation Results on 50 Training Tasks without Uniform Initial State Distributions . . . 108
5.23 Ablation Result for the Effectiveness of Proxy Problems . . . 109
5.24 Ablation Results on 50 Training Tasks for k-medoids Vertex Pruning . . . 109
5.25 Sensitivity of Skipper-once to Number of Checkpoints in Proxy Problem . . . 110

6.1 Target Evaluator attached to a Target-Assisted Planning (TAP) Agent . . . 113
6.2 Delusional Plans in SSM . . . 117
6.3 An Example of How PERTASK Reduces E.2 Errors by Sampling Across Episodes . . . 121
6.4 Representative Atomic Hindsight Relabeling Strategies & Newly Proposed Ones . . . 123
6.5 Details of Skipper's Performance on SSM . . . 128
6.6 Hallucination Frequencies . . . 129
6.7 Evolution of OOD Performance of Skipper Variants on SSM . . . 130
6.8 LEAP on SSM . . . 131
6.9 Evolution of OOD Performance of LEAP Variants on SSM . . . 131
6.10 Skipper's Performance on RDS . . . 132
6.11 Evolution of OOD Performance of Skipper Variants on RDS . . . 132

6.12 LEAP's Performance on RDS . . . 133
6.13 Evolution of OOD Performance of LEAP Variants on RDS . . . 133
6.14 Dyna's Performance on SSM . . . 134
6.15 Dyna's Performance on RDS . . . 135
6.16 Feasibility of Non-Singleton Targets on SSM . . . 135
6.17 Feasibility of Non-Singleton Targets on RDS . . . 136

8.1 An Example for Simplified Observations for Director . . . 165
8.2 Results of Director on Tasks with Lavas *v.s.* on Tasks with Walls . . . 167
8.3 Generalization Performance of Director on Different Numbers of "Walled" Training Tasks . . . 167
8.4 Truncated λ-Returns with Rejected States for Dreamer . . . 170

List of Tables

3.1 OOD-Focused Experiment Settings for Different Chapters 56

6.1 Discussed Methods, Properties & How to use the Feasibility Evaluator 115
6.2 Categorization of Targets based on Composition, Characteristics, Risks & Delusion Mitigation Strategies 118
6.3 Detailed Comparison of Atomic Hindsight Relabeling Strategies 123

8.1 Changed Hyperparameters of Director 166

Chapter 1

Introduction

Reinforcement Learning (RL) is a methodology for learning through trial-and-error, aiming to reinforce behaviors that yield long-term benefits in sequential decision-making scenarios. Recent advances have been driven by integrating RL with artificial Neural Networks (NNs), leading to successes like mastering strategic games such as Chess and Go (Silver et al., 2016), achieving superhuman performance in pixel-based Atari games (Schrittwieser et al., 2019), etc. However, current RL systems still struggle when deployed in real-world contexts, primarily due to challenges in generalizing their learned capabilities to environments different from those in which they were trained (Igl et al., 2019; Quiñonero-Candela et al., 2022). This "generalization gap" significantly restricts both the application and academic interest in RL (Ada et al., 2024).
Recent studies suggest that this limitation stems from RL agents' inadequate reasoning capabilities when dealing with Out-Of-Distribution (OOD) scenarios (Alver and Precup, 2024; Langosco et al., 2022). Many agents rely solely on intuition-based decision-making (akin to system-1 thinking from Kahneman (2017)) (including both model-free and background planning model-based methods), or cannot make necessary longer-term plans (Kahneman, 2017; Zhao et al., 2024). These insights have spurred the development of agents capable of adaptive reasoning in novel situations, which is the central theme of this thesis.

1.1 Thesis Overview

This thesis explores enhancing generalization in RL agents by granting them reasoning capabilities inspired by human higher-level cognitive functions. The structure of this thesis is as follows:
Part I: Literature Review on Background Knowledge:

- In Chap. 2 (Page. 6), I provide an overview of the literature and introduce foundational concepts of reinforcement learning that are pertinent to the subsequent chapters.
- In Chap. 3 (Page. 29), I discuss more advanced concepts, covering deep reinforcement learning, generative modeling with deep learning, attention mechanisms, model-based reinforcement learning, and related experimental methodologies. This chapter prepares readers who have a basic understanding of reinforcement learning for the content in Part II, by establishing necessary background knowledge.

Part II: Methodology & Original Research Findings:

- Chap. 4 (Page. 57), inspired by the OOD generalization abilities facilitated by consciousness, introduces bottleneck mechanism, allowing a decision-time planning agent to dynamically focus its reasoning on the relevant aspects of the state based on its instantaneous intent. This bottleneck mechanism, which achieves "spatial abstraction", enables significant Out-Of-Distribution (OOD) systematic generalization abilities.

- Chap. 5 (Page. 82), inspired by spatial and temporal abstraction abilities in human planning, builds upon the bottleneck mechanism and proposes a framework named Skipper that automatically decomposes an overall given task into smaller and more manageable steps. This framework shows potential to be robust in distributional shifts and compositional long-term planning, by focusing attention on relevant parts of the environment (spatial) and aspects of the future (temporal);

- Chap. 6 (Page. 112) discovers a commonly shared failure mode / safety risk of planning agents which rely on the generated observations / states / goals. This failure mode resembles hallucinations and the resulting delusional behaviors in the human brain. Inspired by understanding of how human brains address delusions, we propose general solutions that enable agents to autonomously and preemptively avoid issues such as blindly trusting hallucinated targets.

The 3 main chapters on the methodologies and original research findings are in lock-steps to serve as the milestones for the thesis topic. Each main chapter serves as the basis for the upcoming chapters, *i.e.*, Chap. 5 is based on the findings of Chap. 4 and Chap. 6 is based on both Chap. 4 & Chap. 5. We present a chart for the relationship among the contributions in Part II, illustrated with key ideas and methodologies, in Fig. 1.1.
Part III: Discussions & Conclusions:

- Chap. 7 (Page. 137) provides a comprehensive scholarly discussion of all findings, including how the contributions met the objectives of the doctoral study, the impact of the contributions, their limitations, along with directions on future work.

1.2 Summary of Original Contributions to Knowledge

The following are the *short* summaries of the original contributions of this thesis. Chap. 7 expands on the detailed contributions of each main chapter.
Chap. 4 describes a decision-time planning agent that can dynamically focus on interesting partial aspects of the state for better OOD generalization, a first in the literature. The core bottleneck mechanism is a top-down attention computation inspired by conscious reasoning in humans (Dehaene et al., 2020). This work is one of the first works utilizing transformer-based architectures in computational decision-making. This work also opens up discussions about the ways in which ideas from higher-level cognitive functions in humans can be used to improve the generalization abilities of computational decision-making agents.
Chap. 5 proposes a framework that automatically decomposes an overall task into smaller and more manageable steps. Skipper utilizes a constrained form of option-based planning, which builds on the consciousness-inspired spatial abstraction mechanisms in Chap. 4 when considering each steps. Novel mechanisms are proposed to learn a problem decomposition consistent with the agent's own capabilities of handling each decomposed step. We also prove that the performance of the approach is guaranteed under practical conditions.

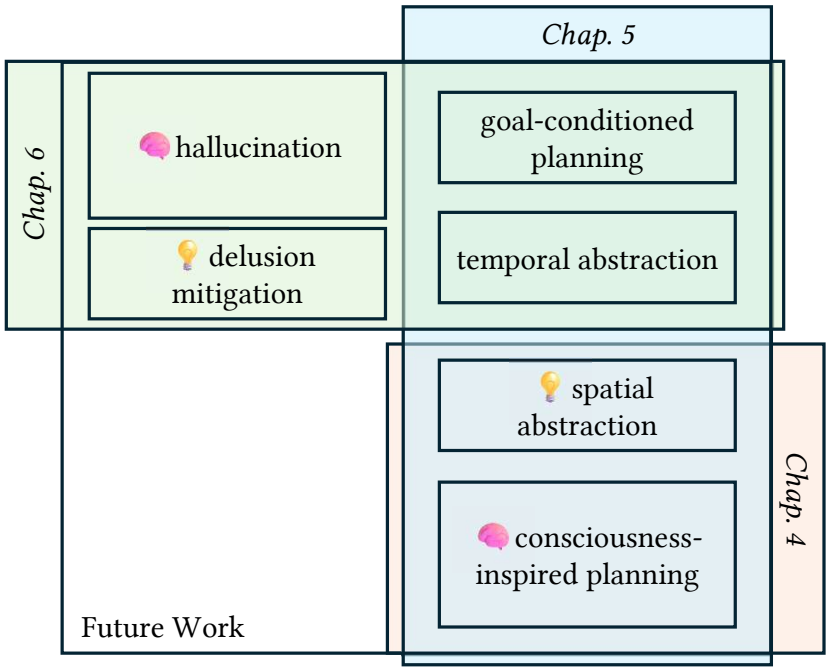


Figure 1.1: ***Main Chapters in this Thesis (Research Methodologies & Original Findings)****: Chap. 4 (Conscious Planning: Spatially-Abstract Decision-Time Reasoning) is heavily influenced by human* ***conscious planning****, and proposes a* ***spatial abstraction*** *process, which is later combined organically with* ***temporal abstraction****, to create the Skipper framework in Chap. 5 (Skipper Framework: Spatio-Temporal Abstractions for Planning). The Skipper framework embraces* ***goal-conditioned planning****, which utilizes* ***temporal abstraction****.* ***Goal-conditioned planning*** *agents, Skipper included, suffer from delusions in planning. Chap. 6 (Rejecting Hallucinations: Addressing Delusional Planning Behaviors) takes inspiration from how the human brain addresses delusional behaviors, to propose* ***delusion mitigation*** *strategies that allow* ***target-assisted planning*** *agents to preemptively and autonomously address delusional planning behaviors during training. The proposed future work encompasses all the key inspirations and contributions (Sec. 7.3, Page. 143).*

This work shows that spatially and temporally abstract planning, like that of humans, is not only viable, but its performance can also be guaranteed RL agents, showing a promising direction of option-based planning. Chap. 6 points out an important flaw in planning agents that utilize generated state targets: they blindly trust hallucinated targets. Many agents do not fully understand the targets that they can propose during planning. Inspired by the belief evaluation system in the human brain, we propose a method to reject hallucinated targets by properly learning a target feasibility evaluator. To be able to learn such an evaluator effectively, we analyze the categories of delusions that can appear in a model generating state targets, and propose a comprehensive solution that can reliably address delusional behaviors resulting from such targets. The solution is a combination of update rules, model architectures as well as hindsight relabeling strategies to solve the mismatch between the planning agents' training and behaviors. In experiments, we find that our solution significantly reduces feasibility errors and the frequency of delusional behaviors, and boosts OOD generalization performance compared to existing methods. Instead of blindly optimizing for sample efficiency, this is the first work that systematically discusses the failure modes of relevant planning agents, introducing the hallucination-delusion perspective. These ideas could be used to save future research efforts from continuing todevelop delusional / unsafe agents.

1.3 Collaborator Contributions Breakdown

For work presented in the 3 main chapters of this thesis, I assumed the primary role in formulating the methodological ideas, mathematical components and proofs, writing, implementation and experiments. My co-authors mostly participated in discussions and brainstorming sessions, providing feedback, contributing to proofreading, and improving communication.

1.3.1 Chap. 4 - Conscious Planning: Spatially-Abstract Decision-Time Reasoning

The collaborators of work presented in this chapter include myself, Zhen Liu, Sitao Luan, Shuyuan Zhang, Doina Precup and Yoshua Bengio. Some contents of this chapter are published as a conference paper at the Conference on Neural Information Processing Systems (NeurIPS) 2021 (Zhao et al., 2021).
Yoshua and I brainstormed and formulated the original abstract idea of the chapter. Then, Sitao, Doina and I discussed the high-level design to implement the ideas. I supervised Zhen and Shuyuan in developing detailed algorithmic designs, implementations as well as experiments. All collaborators proofread the accepted manuscript and contributed to its communication.

1.3.2 Chap. 5 - Skipper Framework: Spatio-Temporal Abstractions for Planning

The collaborators of work presented in this chapter include myself, Safa Alver, Harm van Seijen, Romain Laroche, Doina Precup and Yoshua Bengio. Some contents of this chapter are published as a conference paper at the International Conference on Learning Representations (ICLR) 2024 (Zhao et al., 2024).
Based on earlier brainstorming with Doina and Yoshua, Harm, Romain and I brainstormed and formulated the ideas of the chapter and identified the milestones needed. I conducted all the detailed algorithmic designs, implementations as well as experiments. Safa helped me implement a baseline in the experiments. I collaborated with Romain closely on proving the theoretical results. All collaborators proofread the accepted manuscript and contributed to the communications.

1.3.3 Chap. 6 - Rejecting Hallucinations: Addressing Delusional Planning Behaviors

The collaborators of work presented in this chapter include myself, Tristan Sylvain, Romain Laroche, Doina Precup and Yoshua Bengio. Some contents of this chapter are published as a conference paper at the International Conference on Machine Learning (ICML) 2025 (Zhao et al., 2025).
I conceived the idea of this chapter after realizing that certain delusional behaviors I observed in Chap. 5 are commonly shared among existing methods. Then, Tristan and I developed the ideas for a controlled environment to identify the causes of these behaviors. I implemented the environments, conducted the experiments, and identified the types and root causes of delusions and wrote the manuscript. Romain, Doina and I investigated the theoretical aspects of the chapter and drafted the formal definitions. All collaborators proofread the manuscript and contributed to the communications.

1.4 Copyright

Most figures used in this work are directly created by me. Some figures are original for this thesis, while some others are taken from the related conference papers that I have published as the first author, for which I hold the copyright.

Chapter 2

Literature Review: Reinforcement Learning

This chapter presents basic background knowledge about Reinforcement Learning

Contents

4.1 Overview of This Thesis Milestone . . . 58

4.2 Discussions of Methodologies . . . 59

4.2.1 About Reconstruction, Observation-Level & State-Level Planning . . . 59

4.2.2 About Staged Training with World Models . . . 59

4.2.3 About Vectorized *v.s.* Set-based State Representations . . . 60

4.3 Methodology: Model-based RL with Set Representations . . . 60

4.3.1 State Representation Encoder . . . 61

4.3.2 (State-Action) Q-Value Estimator . . . 61

4.3.3 Transition Model . . . 61

4.3.4 Training . . . 64

4.4 Methodology: Consciousness-Inspired Bottleneck . . . 65

4.4.1 Conditional State Selection . . . 66

4.4.2 Dynamics / Reward-Termination Prediction on Bottleneck Sets . . . 66

4.4.3 Change Integration . . . 67

4.4.4 Discussion . . . 67

4.4.5 Birdseye View of Overall Design . . . 68

4.5 Research Findings: Experiments . . . 68

4.5.1 Environment / Task Description . . . 68

4.5.2 Compared Methods . . . 70

4.5.3 Agent Variants . . . 70

4.5.4 Performance Evaluation (RDS with Turn-Or-Forward Dynamics) . . . 71

4.5.5 Ablation Studies . . . 73

4.5.6 Sensitivity Studies . . . 76

4.5.7 More Experiments, Discussions & Failed Attempts . . . 77
4.5.8 Details of Baselines . . . 79
4.5.9 Summary of Experiments . . . 81
4.6 Summary . . . 81

2.1 What is Reinforcement Learning?

Reinforcement Learning (RL) is a methodology aimed at addressing decision-making problems through learning from interactions with the environment, without explicit supervision. RL draws significant inspiration from the neural basis of learning and operant conditioning in neuroscience (Sutton and Barto, 2018).

2.1.1 Learning by Trial-and-Error

A distinctive feature of RL is that agents learn through trial-and-error. In essence, RL methods autonomously discover both the nature of the situation and the appropriate actions to take post-deployment, highlighting the reduced need for domain-specific knowledge.
The methodology of RL is both powerful and versatile, as trial-and-error learning can be applied to virtually any decision-making problem. It is worth noting that the study of RL extends beyond merely refining learning algorithms: the learning process can also be impeded by challenges associated with environment setup, data collection, and other preparatory activities.
RL is predominantly applied on the reward maximization scenarios in sequential decision-making (Sutton and Barto, 2018).

2.1.2 Reward Maximization in Sequential Decision-Making

Sequential Decision-Making is a concept which involves making a series of decisions and actions over time to optimize objective functions, such as maximizing cumulative rewards, or somewhat equivalently, minimizing costs. Each decision influences subsequent choices and system outcomes, taking into account the current status of the system, available actions, and the probabilistic nature of action-induced transitions (Puterman, 2014).
In a sequential decision-making problem, we name the decision-maker an *agent*, and everything else the *environment*. An agent can be an algorithm or a method, while the environment represent the "world" that the agent is in.
This thesis is particularly interested in the case of Sequential Decision-Making with the objective of maximizing cumulative scalar rewards, which is the case that RL seeks to tackle. A *reward* signal defines the objective of a sequential decision-making problem and is received after taking each action. The reward signal thus defines what are the good and bad events for the agent. Reward signals can also explicitly correspond to a given "goal", *e.g.*, a sparse terminal reward if some goal is achieved.
The generality of RL's maximization approach is backed by the so-called reward hypothesis, which posits that, "all of what we mean by goals and purposes can be well thought of as maximization of the expected

value of the cumulative sum of a received scalar signal (reward)". The readers of this thesis can use this hypothesis to specify the implicit requirements on goals and purposes under which the hypothesis holds (Bowling et al., 2023).

Particularly, we focus on *discrete-time* scenarios where the timings of decision and corresponding environment change are placed over discrete time intervals named **timesteps** defined between **decision points**. These can be formulated as follows: let the decision points (in the flow of time) be $t_0, t_1, \cdots$. At a decision point t, the agent receives an *observation* $x_t \in \mathcal{X}$ and chooses an action $a_t \in \mathcal{A}$, where $\mathcal{X}$ is the *observation space*, the collection of possible observations for the agent that is often assumed to be arbitrarily large, and $\mathcal{A}$ is the collection of all possible actions that the agent could take, named the *action space*. Possibly represented in a variety of ways, an observation is what an environment allows the agent to observe during the agent-environment interactions, and can often conceal important information regarding the environmental state. An action is a choice among all the possible ways that the environment allows an agent to interact with itself. Let t' be the next decision point after t. For the period from t to t', the agent receives feedback $r_{t \to t'}$, namely *reward*, which depends on the agent's action. The objective of an agent in this scenario is to maximize the cumulative **return** received over its lifetime, expressed as a sum of rewards, until the agent can no longer interact with the environment.

In addition to the sequential decision-making framework above, RL relies on the notion of **environment state**, or **states** for short, to ground itself in the mathematical framework of Markov Decision Processes (MDPs), to be introduced later. In the aforementioned definitions, observations are the system outcomes of the actions that are exposed to the agent, while some changes to the environment that remain hidden to the agent could still influence the future outcomes and decisions. A state is a notion unifying both the exposed parts and the hidden (Amortila et al., 2024), an identifier of the current situation of the environment, which can be hidden from the agent (and can be exposed fully as well) and may require the agent to infer through its own perceptions. The **environment state** refers to the complete representation of the current state of the environment in which the agent operates. This state includes all relevant information about the environment that can affect the agent's decision-making and future interactions. Conversely, an agent needs to infer the **agent state** from its interaction history with the environment. These agent states refer to the internal representations or understandings that the agent has of its current situation based on its observations and prior knowledge, and thus may not necessarily match the complete environment state.

When we talk about "state representations" in the later parts, we are either referring to the observations that are lossless transformations of the real environment states (emitted by the environment) or to the agent states constructed by the agents themselves. We overload the term "state" for convenience, and should be able to easily differentiate based on the context.

With the notion of states, in the RL setting, the *environment* becomes an ensemble of a reward function (a scalar feedback for the decisions), state dynamics (transition probabilities of states by actions) and optionally a termination signal (a binary feedback for ending the episode). The environment emits observations to the agent, and the agent needs to predict the current environmental state from the interaction history, which can be used as the basis for decision-making, compared to considering all interaction history.

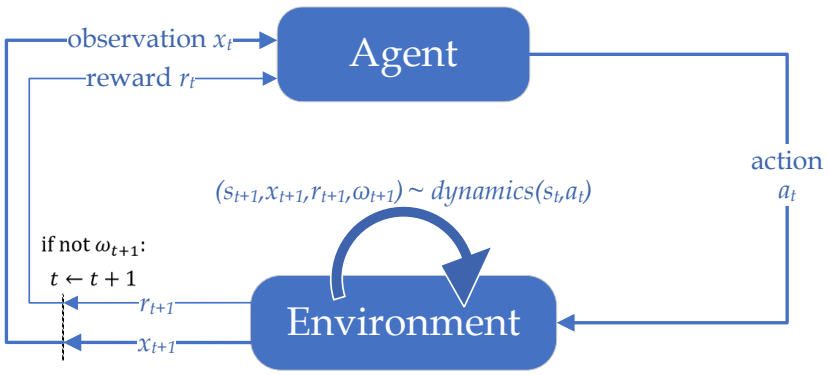


Figure 2.1: ***Agent-Environment Interaction in an RL Problem (from an agent-centric view)****:* ω_{t+1} *is the binary termination signal indicating if* s_{t+1} *is terminal, i.e., if the current episode ends and the agent can take no more actions. The environment can hide the states from the agent, only exposing the observations. The environment dynamics is only decided by the environment state* s_t *and the action* a_t*.*

2.1.3 Markov Decision Processes

Markov Decision Processes (MDPs), an abstraction of the sequential-decision making problems, are classically used as a mathematical formalization to ground RL in mathematical analyses, a connection introduced and popularized by Watkins (1989). MDPs can be seen as an environment-centric view of RL problems, contrary to that presented in Fig. 2.1.

Coinciding with RL, MDPs are developed through the viewpoints of state transitions, as the states are used to specify the starts and the ends of all transitions. Observations are often ignored in this environment-centric view of the RL problem. In an MDP, an agent is in a state and only one state at any time.

States convey some sense of "how the environment is" at a particular time and act as sufficient statistics to make optimal decisions, *s.t.* the previous interaction histories with the environment needs not be considered for decision-making, giving rise to MDP's Markovian properties. In other words, a state is a compact lossless compression of the agent-environment interaction history. With the help of the notion of states, we establish the framework of MDPs.

Formally, an MDP is a five-tuple $\langle \mathcal{S}, \mathcal{A}, p, R, p_0 \rangle$, defining the properties of the world and the objective within it. The state space $\mathcal{S}$ defines the set of all possible states. Each decision that the agent can make is known as an action $a \in \mathcal{A}$, where the action space $\mathcal{A}$ defines the set of all actions. The function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ defines the *dynamics* of the MDP, and is often recognized as the *transition probability function* or simply *transition function.* In the MDP, the rewards are specified with a function R, a marginalized descriptor based on p, established over a transition. The states that the agent start from in a finite MDP can be described using a probability distribution $p_0(s) \coloneqq \mathbb{P}\{S_0 = s\}$ for each $s \in \mathcal{S}$, named the **initial state distribution**.

In a (discrete-time) MDP, at each timestep $t \in \{0, 1, 2, \dots\}$, the agent resides in a state $S_t \in \mathcal{S}$ and on that basis selects an action, $A_t \in \mathcal{A}$. One timestep later at $t + 1$, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and finds itself in a new state S_{t+1}. Thus, p explains the joint probabilities of transitions organized in the form $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$, where s_t, a_t correspond to the state and the action the agent took at time t, while s_{t+1} correspond to the state the agent transitioned to following $\langle s_t, a_t \rangle$, and finally r_{t+1} corresponds to the reward for such transition.

Technically, in a finite MDP, where the state set $\mathcal{S}$, the action set $\mathcal{A}$ and the reward set $\mathcal{R}$ are all finite, the random variables R_t and S_t have well-defined discrete probability distributions that only depend on the preceding state and action. That is, for particular values of these random variables, $s' \in \mathcal{S}$ and $r \in \mathcal{R}$, there is a probability of those values occurring at timestep t, given particular preceding state and action:

$$p(s', r|s, a) \coloneqq \mathbb{P}\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}, \forall s, s' \in \mathcal{S}, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}$$

An episode of agent-environment interactions begins with an initial state s_0 sampled from an initial state distribution $p_0(s) \coloneqq \mathbb{P}\{S_0 = s\}$ over each state s. The following sequence defines the process of an MDP (without termination signals):

1. The agent starts from an initial state s_0, whose distribution can be described with p_0;
2. The agent selects an action (according to its policy);
3. A new state and reward are sampled according to the Markovian dynamics function p;
4. The process is repeated from Step 2 if the agent is not in a terminal state.

The MDP formulation describes what happens during agent-environment interactions with all perfect access to all the environmental dynamics. Note that in an MDP formulation, the termination signal is absorbed into the agent's knowledge about the states.
The MDP and the actions (taken by an agent) together thereby give rise to a *trajectory* like:

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, \ldots$$

An **episode** is used to describe a full trajectory from the initial state S_0 until termination in S_T.
Note that we have used the uppercase letters to denote the random variables since we have not yet observed the states, the rewards or the actions. Yet, if we have already, we would use lowercase to denote their specific instantiation. For example, at timestep t, the agent took action a_t based on state s_t and transitioned to the state s_{t+1} while receiving the reward r_{t+1}.

Markov Property

In an MDP, the probabilities given by p *completely* characterize the environment's dynamics. That is, the probability of each possible value for S_{t+1} and R_{t+1} depends only on the immediately preceding state and action S_t and A_t, not on earlier states and actions. This is best viewed a restriction not on the decision process but on the state, which means the state must include information about all aspects of the past agent-environment interaction that make a difference for the future.
The 4-argument transition function is a most general form of a transition function defined in an MDP. There are also alternative forms of the transition function, which rely on either additional assumptions of the environment or exist as marginalized expectations.
Marginalizing over rewards yields the 3-argument state-transition probability function:

$$p(s'|s, a) \coloneqq \mathbb{P}\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

where p is overloaded. However, if we assume that the state transition and the reward are jointly determined, *i.e.*, a fixed transition from one state to another always generates the same reward, then the 4-argument transition function collapse into the 3-argument version $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \times[0, 1]$.
Many other useful expected statistics can be derived from the general 4-argument p by marginalizing. These include:
Expected rewards for state-action pairs as a 2-argument function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s, a) \coloneqq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a)$$

Since the only way that the agent could interact with the environment is through the action, there is no way for the agent to optimize the transition and reward by any other means, this 2-argument expected reward function should be an appropriate choice when the agent tries to model the reward function for decisioning, through agent-environment interactions.
Expected rewards for state-action-next-state triples as a 3-argument function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:

$$r(s, a, s') \coloneqq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r | s, a)}{p(s' | s, a)}$$

where $p(s'|s, a)$ is the 3-argument transition function we derived earlier. This function can be estimated to predict the reward incurred by some certain transition, which is often used in model-based RL.
For many environments, the reward function is a signal consistent with the encouragement of reaching a certain goal, that is, a certain subset of $\mathcal{S}$. Under these circumstances, we could also use goals to represent rewards. However, it is worth noting that not all wanted behaviors can be incentivized by Markovian rewards in the context of MDPs (Abel et al., 2021).

Limitations to MDP Formulation

Implicit Pause-able Environment Assumption: MDP assumes that the environment only changes after the agent takes an action, whereas this assumption seems inappropriate in many other sequential decision-making problems, *e.g.* when the agent is planning with a complex model (Ramstedt and Pal, 2019).
Stationarity: MDPs assume a stationary environment, where the transition and reward functions do not change over time. In dynamic environments, the models may need to adapt, which can complicate learning and decision-making.
Discrepancy between State and Observations: the MDP formulation can be used to analyze the behaviors of agents, but often cannot be used to direct RL agents' learning behaviors when the environment is not fully observable, *i.e.*, when the observations do not contain full information of the corresponding state, or when it is difficult for information related to making better decisions to be extracted from observations. A way to address such discrepancy is to use the Partially-Observable MDP (POMDP) formulation, whose discussions I will skip because of the low relevance to this thesis.

2.1.4 RL Objectives in MDPs (Episodic Setting)

In MDPs, actions influence more than just the immediate rewards, essentially feedback from the environment, but also subsequent situations, or states, and through the future rewards. Thus, MDPs involve delayed reward and the need to tradeoff immediate and delayed reward.

By connecting RL with an MDP, we can formulate the objectives that RL agents seek to optimize. Note that in this thesis, we only discuss cases where trajectories have finite lengths, often recognized as the **episodic** setting. The end of trajectory is triggered when an agent transitions into a terminal state. After reaching a terminal state, the agent is taken out of the MDP.

Undiscounted Episodic Return

In the form most aligned with sequential decision-making, RL seeks to maximize the expected return G_t, which, in the simplest case, is defined as the sum of the rewards:

$$G_t \equiv R_{t+1} + R_{t+2} + \cdots + R_T \tag{2.1}$$

where T is a final timestep. The notion of *episodes* is naturally formed as the interaction sequence from the starting timestep 0 until the terminal time T, which is a random variable that normally varies and independent with each other, *i.e.*, one episode does not affect the environment dynamics of the next.

In episodic tasks, it is sometimes necessary to distinguish the (sub-)set of all non-terminal states, denoted $\mathcal{S}$ from the set of all states plus the terminal states $\mathcal{S}^+$. In the literature, sometimes "trajectory" is used loosely to denote a segment of an episode. For instance, in some control tasks, a Maximum Episode Length (MEL) would be set to make sure that the agent does not get stuck for an unacceptable amount of time in uninteresting regions of the state space (Erraqabi et al., 2022). These trajectories are segments of full episodes, since they are not terminated by transitioning into a terminal state.

Discounted Episodic Return

In the episodic setting, which is of interest of this thesis, RL agents would often employ an alternative objective additionally defined by a notion of discounting. This means, instead of RL agents trying to maximize the simple sum of episodic rewards, some try to maximize a "discounted" version of the original objective. Such discounting is often introduced to improve the convergence of policy evaluation methods, to be introduced later.

According to the discounting approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses to maximize the *expected discounted return*:

$$G_t \equiv R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

where $\gamma \in [0, 1]$ is the *discount parameter*, also recognized as the *discount rate*. Note that here the discount parameter is constant throughout the states and the episodes, but it can be a state or observation-based function as well (Zhao et al., 2020).

Let us see an example of how discounting could influence the behavior of a return-maximizing agent. Imagine a navigation task, where an agent receives a terminal reward of $+1$ upon success. By maximizing the original undiscounted objective, the agent would have no incentive to reach the goal more quickly, as it just needs to reach the goal in the end to get the same amount of reward. While, for an agent with the discounted objective, every timestep wasted is a penalty towards the objective, thus it would be incentivized to reach the goal quickly.

The fact that the discount parameters can be set to 1 gives a unified formulation for the discounted episodic tasks as well as the undiscounted. Ideally, the discount factors should come from the task itself, as its value reflects the objective of the task. However, for complicated tasks with Deep Reinforcement Learning (DRL), which is essentially using artificial neural networks for RL, it is generally observed that lowering the discount factor yields significantly more stable performance rather than using $\gamma = 1$, even if the objective includes no discounting (Mnih et al., 2015). These blur the line how we should see the discount factor, which classically should be seen as some kind of built-in characteristics of the environment yet now a parameter that could be set or learned for some purposes.

Discounting schemes, other than a constant γ-controlled exponential weighting presented above, are also investigated in the literature (Schultheis et al., 2022), *e.g.* state-based discounting (White and White, 2016; Zhao et al., 2020).

In this thesis, we focus on the discounted episodic setting.

2.1.5 Policies & Value Functions

A *policy* is a component used by an agent to decide what to do. In an MDP, a policy is a function which takes in the (environmental) state the agent is in as input and outputs decisions about actions. Formally,

Definition 1: policy

In an MDP, a policy π is a mapping from states to probabilities of selecting each possible action, *i.e.*, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$.

In an MDP, if the agent is following policy π at timestep t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$. If all probability mass is concentrated at a single action, then we call the policy **deterministic**, otherwise **stochastic**, in the sense that an action would be sampled according to the probabilities. Note that it is also common to define state-based action sets, however we will stick loyal to the simple setting of Sutton and Barto (2018) for this thesis, *i.e.*, assume that all actions are available at all times. The policies in RL are essentially stationary decision rules defined for more general Markov chains (Puterman, 2014), where "stationary" means that the decision rules are consistent for every possible states, as in we are only interested in the stationary MDPs in this thesis.

The *value function* of a state s under a policy π, denoted $v_\pi(s)$, is the expected (discounted) return if an agent starts in s and following π thereafter. For historical reasons, we often call it "V-value" to contrast with "Q-value" given by the state-action value function. Formally,

Definition 2: state-value function (V-values)

In an MDP, the *state-value function* for policy π or simply *value function* $v_\pi(s)$, given discount function $\gamma(s)$ and policy $\pi(s)$, is defined as

$$v_\pi(s) \coloneqq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \mid S_t = s\right]$$

where $\mathbb{E}_\pi$ denotes the expected value of the random variable given that the agent follows policy π and the values of the terminal states are defined as 0.

We are also interested in the state-action value function, which is more useful for control cases, *e.g.* when searching for better policies with a value estimator.

Definition 3: state-action-value function (Q-values)

In an MDP, the *state-action-value for policy* π $q_\pi(s)$, given discount function γ and policy $\pi(s)$, is defined as the expected return starting from s, taking the action a and thereafter following π:

$$q_\pi(s,a) \coloneqq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \mid S_t = s, A_t = a\right]$$

One of the key subroutines of RL is to estimate v_π or q_π from experience, as $\hat{v}_\pi$ or $\hat{q}_\pi$. This estimation can also sometimes be recognized as *policy evaluation* or *prediction*. This concept is to be introduced in detail in Sec. 2.2.

With the recursive definition in Def. 2, we can obtain the following equation for the state-value function v_π, given a policy π:

$$\begin{aligned} v_\pi(s) &\coloneqq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\,[r + \gamma \cdot \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\,[r + \gamma \cdot v_\pi(s')] \end{aligned} \tag{2.3}$$

The equation is essentially one basic form of the *Bellman equation* for v_π. Serving as a core equation of RL, it expresses the relationship between the value of one state and its successor states. With this equation and access to environment dynamics (defined as an MDP), the method of dynamic programming can be used to exactly compute the value function v_π or q_π given any valid policy π, to be introduced later.

2.1.6 Optimal Policies & Optimal Value Functions

To find the optimal policy, we must define what it means for one policy to be better than another. This is done via the value functions, which define a partial ordering over policies in an MDP.

Definition 4: Partial Order of Policies

In an MDP with certain discount γ, a policy π is defined to be *better than or equal to* a policy π' if its expected return is greater than or equal to that of π' for all states, *i.e.*, $\pi \geq \pi'$ ***iff*** $\forall s \in \mathcal{S}, v_\pi(s) \geq v_{\pi'}(s)$.

There always exists at least one policy that is better than or equal to all other policies, which is identified as the **optimal policy** π_*. Intuitively, the optimal policy is the policy with the largest value. However, less intuitively,

Fact 1: Existence of Optimal Deterministic Markovian Policies

Given a finite MDP, there always exists a deterministic Markovian policy π_* that achieves the optimal values.

Together with the policy improvement theorem (Bellman et al., 1957), to be discussed later, the fact serves as the reason why we could confidently focus on the space of deterministic Markovian policies, since an optimization trajectory towards an optimal policy always exist inside. This also justifies some RL methods without explicit policies (as a separate component), *e.g.* value-based methods that use ϵ-greedy upon deterministic Q-value estimates as policies, such as Q-learning (Sec. 2.4.5, Page. 23) (Sutton and Barto, 2018). Also, despite that there can be more than one optimal policies, we denote all of them by π_* for simplicity. In accordance with Def. 4, all optimal policies must share the same state-value function, called the *optimal state-value function* v_*, which is defined as:

Definition 5: optimal state-value (V-value) function

The optimal state-value function is shared by all optimal policies π_* and defined as:

$$v_*(s) \equiv \max_\pi v_\pi(s), \forall s \in \mathcal{S}$$

where the $\max$ operator is defined upon the policy partial orders.

Optimal policies also share the same optimal action-value function q_*, which is defined as

Definition 6: optimal state-action-value (Q-value) function

The optimal state-action-value function is shared by all optimal policies π_* and defined as:

$$q_*(s,a) \equiv \max_\pi q_\pi(s,a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

where the $\max$ operator is defined upon the policy partial orders.

q_* gives the expected return for taking action a in state s and thereafter following an optimal policy. Naturally, we can establish the following connections between Def. 5 and Def. 6:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right]$$

There are also *Bellman optimality equations*, which are specialized for optimal policies[1].
The methods for finding better policies, are recognized as *policy improvement* methods. The Bellman optimality equation conducts policy evaluation and policy improvement at the same time. Later, such equation would be used to support value iteration, a classic dynamic programming method for computing the optimal policy, to be discussed in Sec. 2.3.2 on Page. 19.

2.2 Policy Evaluation & Policy Improvement

Central to the search for better policies are two important synergetic mechanisms: **policy evaluation** and **policy improvement**, which refers to understanding how it is doing and improving behaviors to do better, respectively. These are central to both dynamic programming and (approximate) RL methods, to be introduced later.
Policy Evaluation enables the understanding of how good a policy is, and hence to help find better policies (Sutton and Barto, 2018). Policy evaluations produce either an estimate of state value function ("V") or an estimate of state-action value function ("Q"). Representative methods of policy evaluation include Monte Carlo estimation or (approximate) dynamic programming (temporal-difference methods), to be introduced later.
Policy Improvement is the process of enhancing a policy based on its evaluation, with the aim of maximizing the expected return or cumulative rewards. This step typically follows policy evaluation and is essential for iterative approaches like policy iteration.
Policy evaluation points the direction of policy improvement, *i.e.*, improving values equals to improving policies. This is supported by the policy improvement theorem (Bellman et al., 1957), which states that from any given policy, a superior policy can be obtained by choosing actions with a higher state value for any state applicable. For instance, if conveniently we have for a state s that $Q(s, a^{'})$ is larger than the current policy's action a's $Q(s, a)$, then the policy can be improved by learning to choose $a^{'}$ at s instead of a.
If a policy is parameterized, *i.e.*, not a deterministic function over the value estimates (*e.g.*, greedy policy based on the value estimates) and the policy can be itself improved by certain optimization methods, the resulting RL agent architecture is commonly named **actor-critic**, with the **actor** being the policy and the **critic** being the value estimator. When an agent with an actor-critic architecture employs a parameterized policy optimized by gradient-based optimization methods, **policy gradients** methods are often used, where the actor is the policy improved via estimated gradients established over the estimates from the critic (Konda and Tsitsiklis, 1999; Sutton and Barto, 2018). Evidently, the idea behind actor-critic[2] is essentially alternating policy evaluation and policy improvement, which is recognized as **policy iteration**. Policy iteration is a powerful and universal idea towards addressing reward maximization.
It is also worth pointing out that the statements about policy iteration above *did not necessarily* assume that policy evaluation algorithms run until convergence and converges to the correct values. This is because if the policy evaluation algorithm is guaranteed to converge, then policy improvement attempts based on the prematurely terminated value estimates could result in correct policy improvements as well, giving rise to

[1] Historically, Bellman optimality equation is often abbreviated as "Bellman Equation". While we explicitly differentiate it with the Bellman (policy evaluation) equation in Eq. 2.3

[2] I will not introduce policy gradient methods in this chapter because of its low relevance to this thesis.

Generalized Policy Iteration (GPI) (Sutton and Barto, 2018). Thus, the assumption is not necessary for the cases of dynamic programming and certain approximate RL cases, to be introduced soon.
We will introduce detailed policy evaluation and policy improvement methods in the following sections regarding dynamic programming and (approximate) RL, respectively.

2.3 Dynamic Programming (DP)

Dynamic Programming (DP) is an optimization algorithm that simplifies a complicated problem by breaking it down into simpler sub-problems in a recursive manner (Bellman et al., 1957).
In the context of RL, DP mostly refers to a family of heavily-used algorithms that, given an environment MDP and access to all state-action combinations, computationally solve interesting quantities such as the values function given a certain policy, finding the optimal policies, *etc.*. Note that DP methods should be distinguished from the approximate RL methods, which are to be introduced later and are often call "RL methods" for short, as approximate RL methods do not assume access to the environment MDP and all state-action combinations. This should be intuitive, since DP methods do not employ the core RL strategy of learning from interactions. However, because of DP methods' provable convergence in solving the interesting quantities, they often serve as the bases of developments of approximate RL methods which can scale to real-world problem-solving, to be introduced later.
DP is widely used to analytically calculate the ground truth values of relatively small sized environments, which can be used to analyze the performance of approximate RL methods which does not have the access to the environment MDP. Take the contributions of this thesis for example, we solve the optimal policies in Chap. 4 to compare the planned actions against the ground truth optimals, as well as the ground truth relationships between pairs of states in Chap. 5 to understand how well the estimators work. Despite that DP could be used to solve an optimal policy, its use is limited in practice because of the need of access to environment MDP, as well as the expensive computational cost.

2.3.1 DP for Value Function

First, we consider the method of computing the state-value function v_π for an arbitrary policy π,*i.e.*, policy evaluation via DP. This is enabled by the following:

Fact 2: Existence & Uniqueness of State-Value Function

The existence and the uniqueness of state-value function v_π are guaranteed as long as either $\gamma < 1$ or termination will be reached from any state following π.

From the Bellman equation (2.3), we have

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma \cdot v_\pi(s')\right]$$

This means, when the environment dynamics (4-argument p) is known, the Bellman equation gives a system of $|\mathcal{S}|$ linear equations with $|\mathcal{S}|$ unknowns, unsurprisingly solvable. In a compact matrix form, this linear system can be presented as:

$$\boldsymbol{v}_\pi = \boldsymbol{r}_\pi + \gamma P_\pi \boldsymbol{v}_\pi$$

where $\boldsymbol{r}_\pi$ is a $|\mathcal{S}| \times 1$ vector in which $\boldsymbol{r}_\pi[i] = \sum_a \pi(a|s_i) \sum_{s_j,r} r \cdot p(s_j, r|s_i, a)$, P is a $|\mathcal{S}| \times |\mathcal{S}|$ matrix in which $P_\pi[i,j] = \sum_a \pi(a|s_i) p(s_j, r|s_i, a)$.
With the system in hand, the rest is just to solve it as $\boldsymbol{v}_\pi = (I - \gamma P_\pi) \backslash \boldsymbol{r}_\pi$ or $\boldsymbol{v}_\pi = (I - \gamma P_\pi)^{-1} \boldsymbol{r}_\pi$.
Note that the $\mathcal{O}(|\mathcal{S}|^3)$ complexity is a nightmare for problems with large state spaces. Thus, it is desirable to change this method into an iterative method with lower computational complexity.
Thus, we arrive at the *iterative policy evaluation method*. It is simple and powerful, turning Bellman equation into an iterative formula achieves the convergence to the true values. One can also prove the convergence of iterative policy evaluation using Banach's fixed point theorem, by showing that the Bellman operator is a contraction.

Definition 7: Bellman Operator

Given an MDP with its dynamics p, a policy π and discount function γ, the *Bellman operator* $\mathcal{B}_\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ is defined by

$$(\mathcal{B}_\pi v)(s) \coloneqq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')] \tag{2.4}$$

Or equivalently in matrix form,

$$\mathcal{B}_\pi \boldsymbol{V} \coloneqq \boldsymbol{r}_\pi + \gamma P_\pi \boldsymbol{V} \tag{2.5}$$

where $\boldsymbol{r}_\pi$ is a $|\mathcal{S}| \times 1$ vector in which $\boldsymbol{r}_\pi[i] = \sum_a \pi(a|s_i) \sum_{s_j,r} r \cdot p(s_j, r|s_i, a)$, P is a $|\mathcal{S}| \times |\mathcal{S}|$ matrix in which $P_\pi[i,j] = \sum_a \pi(a|s_i) p(s_j, r|s_i, a)$.

Definition 8: Contraction

Let $\langle X, d \rangle$ be a complete metric space. Then a map $\mathcal{T} : X \to X$ is called a **contraction mapping** on X if there exists $q \in [0, 1)$ *s.t.*

$$\forall x, y \in X, d(\mathcal{T}(x), \mathcal{T}(y)) \leq q \cdot d(x, y)$$

We can prove that the Bellman operator, which is essentially turning the Bellman equation into an iterative formula, on the *estimated value function* or simply *value estimate* is a contraction. The unique fixed point must be the true value because that is when the Bellman equation holds.
The state-action value function q_π can be computed trivially by combining the computed v_π with the tabular reward function:

$$q(s, a) = r(s, a) + \gamma \cdot \sum_{s'} p_\pi(s, s') v(s'), \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{2.6}$$

Now that we have an algorithm to reliably solve the true values of a policy, we can think about how to generalize this algorithm to compute other interesting quantities, including for more complex cases involving goal-conditioned policies. I present the following examples:
In Chap. 5, we solved the cumulative rewards the agent would get by following a certain (goal-conditioned) policy π going from one state to another. This is done by augmenting $\mathcal{S}^+$ with the target state as an additional terminal state. Similarly, we also solved the cumulative discount from one state to another by making r all zeros except when reaching the target state, giving 1 instead;
In Chap. 6, we solved the distance between a pair of states, under a certain policy π, *i.e.*, how long it takes for policy π to travel from one state to another, are computed by replacing the rewards $\boldsymbol{r}_\pi$ with a vector full of -1s and setting γ to be 1. The convergence was assisted by the fact that the used environments were abundant with terminal states and the agent's policies will almost surely send the agent to a terminal state within finite timesteps, per Fact. 2.

2.3.2 DP for Optimal Policy

Relying additionally on the policy improvement theorem (Bellman et al., 1957), DP are empowered to find better policies, which ultimately leads to the optimal policies.
DP methods can naturally rely on *policy iteration* to find better policies, essentially alternating policy evaluation and policy improvement. However, such alternation may not always be trivially explicit. For example, **Value Iteration (VI)** is a representative DP method for finding the optimal value function by applying the Bellman optimality equations iteratively. VI conducts search in the constrained space of deterministic greedy policies and combines policy evaluation and policy improvement in one unifying step.
With the computed V-value function (for the optimal policy), we can recover the policy from the converted Q-value function via Eq. 2.6.
VI has other uses and can take in alternative forms. In Sec. 2.4.5 on Page. 23, we will introduce Q-learning, an approximate RL algorithm that draws heavy similarity to VI; In Chap. 5 (Page. 82), we use VI (over options) to solve an approximate plan given an estimated proxy problem, which essentially takes the same form as an MDP, to make plans, *s.t.* the agent can know which subgoal is to follow.
Now that we have finished introducing the DP algorithms relevant to this thesis (while skipping other less relevant use cases), we transition to the canonical RL methods.

2.4 (Approximate) RL

Approximate RL methods are used when there is no access to the MDP ground truth nor all combinations of state-action pairs. Thus, approximate RL represents the case most true to trial-and-error based sequential decision-making and is often abbreviated as just "RL". Learning from actual experience is striking because it requires no prior knowledge of the environment's dynamics, yet could still attain optimal behavior.
Approximate RL methods build on the principles of DP. Many RL algorithms, such as Q-learning (Sec. 2.4.5, Page. 23) and actor-critic methods (Sec. 2.2, Page. 16), use DP ideas but adapt them for function approximation. The similarity between approximate RL and DP is that it also relies on the two key steps of:

- **Policy Evaluation**: Just like DP methods, approximate RL aims to estimate value functions, but it does so in a way that does not assume access to environment MDP, and can often rely on function approximations when the states need to be inferred.
- **Policy Improvement**: Approximate RL and DP both use similar principles for policy improvement. In approximate RL, the learned value functions can guide policy updates, analogous to how DP updates policies based on value function estimates.

The difference is more pronounced in the aspects of:

- **Convergence and Stability**: While DP methods provide guarantees of convergence to the optimal solutions, approximate RL methods can struggle with convergence due to function approximation, particularly when using non-linear approximators like neural networks.
- **Exploration-Exploitation Tradeoff**: Because of the lack of access to all state-action pairs, approximate RL methods need to optimize for a fundamental tradeoff between exploration and exploitation, which correspond to trying diverse actions to find better previously unknown rewarding trajectories *v.s.* committing to known rewarding trajectories for better returns. The tradeoff also affects an RL agent in other ways. For example, from a "dataset"-label perspective, we could say that in RL problems, the data samples are dynamically collected by the agents' decisions. This also means the quality of the "dataset" is also determined by the quality of the past decisions, making learning quality dependent on behavior.

2.4.1 RL Agent Components

To implement the two core mechanisms, *i.e.*, policy evaluation and policy improvement, an (approximate) RL agent is often equipped with the follow components:

- A **value estimator** that approximates the (true) value function to conduct policy evaluation. Depending on the learning algorithm and learning capacity, policy evaluation by a value estimator may be misaligned with the objective performance it has in the environments, *i.e.*, in approximate RL, value estimates do not necessarily converge to the true values of the value function. Value estimators could also be used to estimate other interesting auxiliary "values", those beyond the expected on-policy future return, as in Chap. 5.
- A **policy** that can be improved based on the value estimates, the output of the value estimator. The improvements can be made by a search or optimization algorithms. The policy component of an agent could also be implicit. For example, we can always extract the greedy policy from a Q-value estimator if convenient, as shown later.
- An optional **model** of the environment. A model is something that mimics the behavior of certain aspects of the environment, or more generally, that allows inferences to be made about how the environment will behave. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for planning, by which we are not limited to deciding on a course of action by considering possible futures before they are actually experienced. A model could assist policy evaluation, as to be discussed as background planning, or could be an active component of the agent's policy, as to

be discussed as decision-time planning. A model could also take many forms, as some models mimic the environment MDP, predicting the next state and reward out of the current state and intended action, while others could even predict partial aspects of distant future states, *etc.*. Methods for solving RL problems that use models and planning are called **model-based methods**, as opposed to **model-free methods** that are explicitly trials-and-errors, viewed as almost the opposite of planning. A model-based method contains all components of a model-free method, and the latter can be seen as a foundation of the former. We are particularly interested in and primarily dealing with model-based methods in this thesis.

2.4.2 Train & Evaluate

An RL agent is deployed into its training environments with a certain budget of agent-environment interactions. During its interactions, the agent is expected to *autonomously* collect data, estimate the values of its policies and make improvements accordingly.

To understand the learned capabilities, an agent's performance is evaluated in expectation over environment instantiations, where in the real-world, these evaluation environments can often be different from the training environments.

One important objective of RL is to achieve high (generalization) performance on evaluation tasks after learning from a limited number of training tasks, where the evaluation and training distributions may differ; for instance, a policy for a robot may need to be trained in a simulated environment for safety reasons, but would need to be deployed on a physical device, a setting called sim2real. Discrepancy between task distributions is often recognized as a major reason why RL agents are yet to be applied pervasively in the real world (Igl et al., 2019).

In the chapters describing the contribution of this thesis, we employ on an experimental setting that emphasizes the evaluation of zero-shot generalization skills. Intuitively, these experiments evaluate if an agent could truly learn generalizable skills, instead of relying on memorization.

Our experiment settings often involve training on limited number of environments and testing on a whole distribution of unseen environments with the same nature as the training tasks (skills needed to finish the task remain consistent). To generalize well, the agents need to build learned skills which capture the consistent knowledge across tasks. I will introduce the details in the next chapters.

2.4.3 Temporal Difference Learning

Temporal Difference (TD) learning is a fundamental concept and methodology in credit assignment within approximate RL. Credit assignment refers to the challenge of determining which actions lead to a particular outcome (Minsky, 1961), and in the case of TD, it involves associating returns with states or state-action pairs. TD learning combines elements of Monte Carlo (MC) simulation and Dynamic Programming (DP) to enable efficient policy evaluation. Like MC methods, TD can learn directly from raw experience without access to a model of the environment's dynamics. Like DP, TD methods **bootstrap**, updating estimates based partly on other learned estimates, without waiting for the final outcome.

While MC methods must wait until the end of an episode to update the value of $V(S_t)$ (only after G_t is known), TD methods can update their estimates at each timestep. At timestep $t+1$, they immediately make

an update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. The simplest TD method performs the following update:

$$V(S_t) = V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{2.7}$$

immediately on transition to S_{t+1} and receiving R_{t+1}. The update rule 2.7 is called the **1-step TD update**, where we recognize $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ as the **(1-step) TD error** and $R_{t+1} + \gamma V(S_{t+1})$ as the **(TD-)update target**. Every 1-step TD update can be understood as: walk towards the update target $R_{t+1}+\gamma V(S_{t+1})$ from the current (estimated) value $V(S_t)$ with a step length of $\alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$ (decreasing the distance by ratio α). Note that the update target $R_{t+1} + \gamma V(S_{t+1})$ is also a random variable.

Fact 3: Convergence of 1-step TD

Under the episodic setting, given an MDP and a policy π, either discounted or not, 1-step TD achieves convergence to v_π asymptotically (in tabular setting).

The flexibility of TD learning, as well as its convergence guarantees in tabular and linear cases, set itself as a foundation for most RL methods, keeping generalized policy iteration valid even without direct access to MDPs.

TD has also received attention in neuroscience because of its connections to the reward-prediction-error hypothesis. It was discovered that the firing rate of dopamine neurons in the ventral tegmental area and substantia nigra appear to mimic the error function in the algorithm (Schultz et al., 1997).

Note that TD updates can also use targets constructed from value estimates over multiple steps, known as **multi-step TD**. However, we will omit a detailed discussion of these methods due to their limited relevance to this thesis.

2.4.4 Off-Policy Evaluation

In approximate RL, possibly because of the limited agent-environment interaction budgets or others, an agent is often required to estimate the values of one policy when acting upon another. Let us call the policy to learn about the *target policy*, and the policy used to generate behavior the *behavior policy*. In this case, we say that learning is from data "off" the *target policy*, and the overall process is termed off-policy learning, in contrast with on-policy learning, where the agents act accordingly to the values of the policy it estimates.

In Watkins and Dayan (1992), the author proved that learning a Q-value estimator with 1-step TD yields convergent value estimation results in tabular cases. And this even applies to off-policy learning if the update targets are constructed carefully using the target policy. This analysis, in theory, granted RL agents freedom to learn a Q-value estimator from any experience about the target policies. However, as later years has shown, when combined with function approximation, this approach still faces lots of challenges in practice, especially when the agent is asked to learn on experiences acquired by other agents, without access to their policies (Fujimoto and Gu, 2021).

Off-policy learning can also be conducted with V-value estimators using importance sampling. However, I will skip further discussions, because it is mostly irrelevant to the contributions of this thesis.

2.4.5 Q-Learning

We now turn to Q-learning, an approximate RL method of learning optimal policy, through TD learning of state-action values, commonly referred to as **Q-values**. Q-learning is off-policy compatible, meaning it can learn from experiences generated by a policy different from the one being optimized. This makes Q-learning particularly flexible and widely applicable. Q-learning is an approximate RL algorithm that draws heavy similarity to value iteration used in DP, when access to the environment MDP is provided (Sec. 2.3.2, Page. 19).

The Q-learning update rule is defined as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right) \tag{2.8}$$

where $Q(s_t, a_t)$ is the estimated value following taking action a_t in state s_t, r_{t+1} is the reward received during the transition, α is the learning rate, equivalent to a step size and $\max_{a'} Q(s_{t+1}, a')$ is the maximum estimated Q-value for the next state, reflecting the maximum value following taking the best action in the next state.

For a Q-learning update, the combined term $r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a')$ is recognized as the **update target** of Q-learning. In the variants of Q-learning, such as double Q-learning (Hasselt et al., 2016), the update target can be substituted with other terms. We will discuss this further in Sec.3.1.1 on Page. 31.

Q-learning converges to the optimal Q-values under certain conditions (*e.g.*, sufficient exploration of the state space, decaying learning rate) and ultimately leads to an optimal policy in the tabular case (Watkins and Dayan, 1992).

One of the most important modern deep RL methods, **Deep Q-Learning (**DQN**)**, combines Q-learning with **function approximation** using deep neural networks. We will introduce DQN in detail in Sec. 3.1.1 on Page. 30.

2.4.6 Function Approximation

In classic MDPs and DP methods, states are typically enumerated in a tabular form, where all states are indexed and open-for-access explicitly. In this tabular setting, the agent can only be in one state at a time, and state estimates do not influence each other. While tabular approaches are useful for theoretical analysis, they become impractical in real-world scenarios, especially when the state space is large or continuous and cannot be easily discretized.

In the more real-world setting - the **function approximation** case, an agent may be instead given observations that are representations of the states, or may have to use a function approximator to infer the states from the history of interactions. In either case, it has to employ a value estimator to map the state representations into estimated values.

In RL, function approximation primarily refers to using function approximators for value estimation, *i.e.*, policy evaluation. While, policies themselves, could have function approximators of their own.

When using observational inputs for value estimation, function approximators can help construct state representations - transformed or abstracted versions of the environment's true state, which is often abbreviated as simply the **state**. A state refers to the full configuration of the environment at a given time, containing all relevant information for decision-making, *i.e.*, the sufficient statistics; In contrast, a state representation

is a processed version of this state. When a state representation is formed by the agent itself, it typically involves feature extraction, dimensionality reduction, or other abstractions to make it easier for the agent to learn and generalize. With deep learning, these representations are often lower-dimensional embeddings that capture essential features while discarding irrelevant information. A state representation can take the shape of a vector, real-valued or even binary, or even a set of unordered objects, as shown in Chap. 4 (Sec. 4.2.3, Page. 60).

With function approximation, the two key problems of RL can be re-aligned towards representation (learning) and exploration, as suggested in Amortila et al. (2024).

In the tabular setting, methods like Temporal Difference (TD) learning can converge exactly to the value function because updates to each state are independent. However, with function approximation, updates to one state's value may affect the estimates of many other states, making exact convergence difficult. This introduces a challenge: improving the estimate of one state might degrade the estimates of others, especially when the number of states exceeds the learnable parameters. This poses a **generalization dilemma**: while interference between state estimates is problematic, it can also accelerate learning by improving estimates for similar states.

With optimization methods like gradient descent, surrogate losses (*e.g.*, L_1 or L_2 distances) are often used to guide learning, typically in conjunction. This means, a loss that encourage the value estimates to be closer to their update targets will be established, and an optimizer will be applied to minimize such loss to achieve the convergence to the update targets. In contrast, vanilla TD learning does not seek to optimize a loss but provably converges in the respective cases.

The learnable parts of the approximator are often abstracted and compiled into a collection of learnable **parameters**, which are also sometimes called **weights**.

2.4.7 Semi-Gradient Methods for Learning Function Approximators

Popularly, we use differentiable value estimators $V(s;\boldsymbol{w})$ parameterized by a weight vector $\boldsymbol{w}$ to enable stochastic gradient-descent methods for approaching the update targets.

$\boldsymbol{w}$ can be updated at each of a series of discrete timesteps as before, $t \in \{1, 2, \dots\}$, trying to minimize the losses based on the value errors. Stochastic gradient-descent (SGD) methods do this by adjusting the weight vector based on each example by a small amount in the direction that would most reduce the error on that example:

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \frac{1}{2}\alpha\nabla\left[v_\pi(S_t) - V_\pi(S_t, \boldsymbol{w}_t)\right]^2 = \boldsymbol{w}_t + \alpha\left[v_\pi(S_t) - V_\pi(S_t, \boldsymbol{w}_t)\right]\nabla V_\pi(S_t, \boldsymbol{w}_t) \tag{2.9}$$

where α is the learning rate, a positive step-size hyperparameter or **learning rate** for short.

Gradient descent methods are called "stochastic" when the update is done on only one or a few examples, selected stochastically. Over many steps, the overall effect is to minimize an average performance measure, such as the overall value error across all states.

Obviously, we cannot use (Eq. 2.9) to do update because the true value $v_\pi(S_t)$ is unknown. Thus, we must replace the update target $v_\pi(S_t)$ with an estimate U_t. If U_t is unbiased, *i.e.*, $\mathbb{E}[U_t|S_t = s] = v_\pi(S_t), \forall t$, then

$\boldsymbol{w}_t$ is guaranteed to converge to a local optimum under the usual SGD conditions with decreasing α. One simplest instance of this kind of method is to use the MC returns as the update targets, which leads to the gradient-MC method.

Having discussed the inefficiencies of using MC returns as the value estimation update targets, we naturally turn to the possibility of combining TD methods with function approximation. Unfortunately, despite TD's convergence in the tabular case, TD always employs biased targets, because TD update targets are constructed by bootstrapping existing value estimates. Combined with the requirements from gradient descent, this implies that TD methods will not produce a true gradient method. Formally, it can be shown that bootstrapping methods are not in fact instances of true gradient descent (Barnard, 1993), as they take into account the effect of changing the weight $\boldsymbol{w}_t$ on the estimate ($\nabla V_\pi(S_t, \boldsymbol{w}_t)$ in Eq. 2.9) but ignore its effect on the target ($[v_\pi(S_t) - V_\pi(S_t, \boldsymbol{w}_t)]$ in Eq. 2.9).

The methods that combine TD-learning and gradient-based optimization are recognized as *semi-gradient* methods, because they only take into consideration a part of the gradient. Although semi-gradient bootstrapping methods do not converge as robustly as gradient methods, they have shown generally good performance in application, especially the deep RL methods using neural networks as function approximators.

2.4.8 Off-policy Methods with Function Approximation

When learning off-policy with function approximation, new troubles emerge for semi-gradient methods. First, the update targets may need to be fixed with importance sampling ratios, depending on if a Q-value estimator is used; Second and most importantly, the state distribution will no longer match the target policy. Let us first look into why off-policy learning is more difficult with function approximation compared to the tabular case. With function approximation, value estimation becomes dependent on state representations. Thus, the updates for one state could affect multiple states with similar representations, whereas in the tabular case, the updates for one state have no influence on others. This means that, in the off-policy case, the tabular updates do not have to care about the state-frequencies when doing updates as long as the update targets are fixed using the importance sampling ratios. The blessing that the tabular case updates do not rely on any special distribution for stability has not been passed to the function approximation cases. In the function approximation case, the semi-gradient methods that we have introduced before rely on the state-frequencies for updates. This means we either have to "reweight" the updates, *i.e.*, to warp the update distribution back to the on-policy distribution using importance sampling methods, or we have to develop true gradient methods that do not rely on any special distribution for stability, which are not yet available for non-linear function approximators such as neural networks (Zhao et al., 2020). In fact, the problem of the coexistence of bootstrapping, off-policy learning and function approximation is so troublesome that it is considered as "the deadly triad", which is known for the resulting divergent value estimations (Sutton and Barto, 2018).

2.5 Temporal Abstraction

As discussed, the RL formulation of sequential decision-making relies on a one-step state transition model (decisions are made at the most atomic timesteps), where the action taken at timestep t impacts the state and

the immediate reward at $t+1$. This prompts RL methods to overwhelmingly learn to directly work on these finest-grain courses of action.

However, to be able to efficiently and effectively reason longer-term in the face of novelty, an agent must have the ability to utilize and behave, according to different appropriate timescales (Sutton and Barto, 2018). This is because an agent can learn more effectively if uninteresting details are abstracted away, and focus its computations on important decision timings. This is coupled with the fact that some predictions about the environment are counterintuitively more difficult in the finer timescales than the more coarse, where in the latter case, unnecessary details may be in a way effectively marginalized. This, together with the fact that the accumulation of errors by imperfect models during multistep planning, motivate the learning for temporal abstractions in the framework of RL, *e.g.* options.

Learning temporal abstractions is a longstanding problem and existing methods cover a variety of approaches, *e.g.* discovering partially defined policies skills (Thrun and Schwartz, 1994), learning more and more complex behaviors using temporal-transition hierarchies (Ring, 1997), a feudal approach where high level managers learn to allocate tasks to their sub-managers, which in turn learn how to satisfy them (Dayan and Hinton, 1992), a framework to consider the augmented MDP but also the underlying MDP in a seamless fashion (Sutton et al., 1999), *etc.*.

We lay the foundations of temporal abstractions including options and options models, discuss the discovery problem, then review both model-based (planning with option models) and model-free (option discovery) algorithms relevant to the contributions of this thesis.

2.5.1 Semi-Markov Decision Processes (SMDPs)

A Semi-Markov Decision Process (SMDP) provides a foundation for temporal abstractions, where, compared to MDPs, the amount of time between two decision points can vary (Puterman, 2014). In SMDPs, the transition functions $\mathbb{P}\{S^\tau = s' | S^0 = s, \pi_i\}$ are defined additionally over a *transition time* τ, for an agent to enter a next interesting state s' from s.

Building upon SMDPs, we can replace the action spaces with "macro" actions induced by different policies, abstracting the decision-making process, which leads to the options framework. These macro actions can also be nested, *i.e.*, consisted of a sequence of other macro actions, naturally resulting in a potential of hierarchies corresponding to different timescales. The aim of hierarchical RL is to find closed-loop policies at several levels of abstraction, also known as temporally-extended actions. Discovering useful and reusable temporally extended actions are core to temporal abstraction.

Instead of primitive actions, let us consider an agent with a set of policies to choose from at each decision-time, *i.e.*, the action space is replaced with a set of policies. Let the cumulative discounted reward under π_i be $R_s^{\pi_i}$. Considering the case, where the decision time for actions are only at discrete events, we have:

Fact 4: Bellman Optimality Equations for SMDP

$$V^*(s) = \max_{\pi_i} \left[R_s^{\pi_i} + \sum_{\tau=1}^{\infty} \gamma^{\tau-1} \cdot \sum_{s'} \mathbb{P}\{S^\tau = s' | S^0 = s, \pi_i\} V^*(s') \right] \tag{2.10}$$

$$Q^*(s, \pi_i) = R_s^{\pi_i} + \sum_{\tau=1}^{\infty} \gamma^{\tau-1} \cdot \sum_{s'} \mathbb{P}\{S^\tau = s' | S^0 = s, \pi_i\} \max_{\pi_i'} Q^*(s', \pi_i') \tag{2.11}$$

The equations above form sequences of actions defined over macro-actions. A sequence of actions forming a "macro" is one of the simplest kinds of abstraction. A macro can also be obtained as a sequence of other macros, which naturally results in a hierarchy in architecture. The equations indicate the potential of using macro-actions in sequential decision-making problems, giving rise to option-based frameworks.

2.5.2 Options: SMDPs with Policies as Macro Actions

Temporally extended actions are usually defined over a subset of the state space, with the primary aim to reduce the number of steps needed for the agent to solve a task. It serves as motivation to learn abstractions, which are partial solutions to a task that could be reused for other tasks.

Options are a way to achieve temporal abstraction in RL, *i.e.*, finding useful action sequences that span over multiple decision intervals (Sutton et al., 1999). The usefulness of these options can be evaluated by their robustness, re-usability, *etc.*. The appeal of options is that they are in some ways interchangeable with actions. Temporal abstraction allows agents to use sub-policies, and to model the environment over extended time scales, to achieve both better generalization and the divide and conquer of larger problems.

Each option comprises a way of behaving (a policy) and a way of stopping. Formally, for any set of options defined on any MDP, the decision process that selects only among those options, executing each to termination, is a SMDP (Puterman, 2014). An SMDP consists of 1) a set of states $\mathcal{S}$, 2) a set of options $\mathcal{O}$, 3) for each pair of state and option, an expected cumulative discounted reward, and (4) a well-defined joint distribution of the next state and transit time (Bacon et al., 2017). Naturally, options give rise to option-associated dynamic functions, value functions, Bellman optimality equations, *etc.*

Sutton et al. (1999) demonstrated the empirical potential to plan and learn at multiple time scales in the options framework, indicating options' effectiveness for speeding up learning, improving robustness and generalization abilities, *etc.*. However, the original experiments in Sutton et al. (1999) required the designer to use prior knowledge about the task to add pre-defined options, either providing the option-specific reward functions, or providing complete option policies.

This raises the critical question of where the options should come from, giving rise to the **option discovery** problem. Besides principled approaches such as option-critic (Bacon et al., 2017), common approaches to option discovery involve posing subsidiary tasks such as reaching a bottleneck state or maximizing the cumulative sum of a sensory signal other than reward. Given such subtasks, the agent can develop temporally abstract structure for its cognition by following a standard progression in which each subtask is solved to produce an option, the option's consequences are learned to produce a model, and the model is used in plan-

ning. Interestingly, option discovery problems can be bypassed to a certain degree, as what my collaborators and I have done for Chap. 5 with goal-conditioned planning.
Just as we can model the consequences of primitive actions and plan accordingly, so we can learn and plan with models of options' effects. However, this will not be as straight forward as operating on primitive actions. I will expand on this point in the next chapter.

2.5.3 Goal Conditioned RL & Options

Notably, goal-conditioned RL seeks to train agents to achieve a goal or a sequence of goals, and hence can be viewed as a formulation of instantiating option learning whose policies are shaped towards achieving certain outcomes (Sutton et al., 2023).
In some sequential decision-making problems, what matters is the achievement of certain goals instead of the maximization of returns. Researchers have thus tried to design reward functions that would align with the objectives of the problems. However, this remains an open-problem since the alignment can be non-trivial to establish, *i.e.*, maximizing the accumulation of the designed rewards often do not lead to the achievement of the important goals.
In Chap. 5, we will focus on goal-conditioned options, where the initiation set covers the whole state space $\mathcal{S}$. Each such option is a tuple $o = \langle \pi, \beta \rangle$, where $\pi : \mathcal{S} \to \text{Dist}(\mathcal{A})$ is the (intra-)option policy and $\beta : \mathcal{S} \to \{0, 1\}$ indicates when a goal state is reached.

Chapter 3

Literature Review: Model-Based Deep RL

This chapter presents the discussions of related works to the contributions of this thesis, as well as some more advanced preliminary background knowledge for understanding the methods used in later chapters.

Contents

5.1 Overview of This Thesis Milestone . . . 82

5.2 Methodology: Proxy Problems . . . 83

5.2.1 Definition . . . 83

5.2.2 Potential of Proxy Problems . . . 85

5.3 Methodology: Skipper Framework - Spatially & Temporally Abstract Planning . . . 88

5.3.1 Problem 1: Edge Estimation . . . 88

5.3.2 Problem 2: Vertex Generation . . . 92

5.3.3 Skipper Framework: Assemble Everything . . . 95

5.4 Discussions of Methodologies . . . 97

5.4.1 About Proxy Problems . . . 97

5.4.2 About Task Decomposition with Selected States / Sub-Goals . . . 97

5.4.3 About Temporal Abstraction . . . 98

5.4.4 About Spatial Abstraction . . . 98

5.4.5 About Planning Estimates . . . 99

5.5 Research Findings: Experiments . . . 99

5.5.1 Generalization Performance with 50 Training Environments . . . 100

5.5.2 Scalability Studies: Number of Training Tasks . . . 101

5.5.3 Validation of Claims . . . 104

5.5.4 Ablation Studies . . . 107

5.5.5 Sensitivity Studies . . . 109

5.5.6 Summary of Experiments . . . 110

5.6 Summary . . . 111

3.1 Deep RL: Neural Network based Methods

When function approximators in approximate RL methods are implemented with Neural Networks (neural nets, NNs)s, the resulting agents are often recognized as **Deep RL** agents, or **DRL** agents for short. This means, at least the value estimator will be parameterized as a neural network and will be optimized using a surrogate loss with certain neural network optimizers. The neural network optimizers are mostly based on gradient descent, *i.e.*, evolving the parameters roughly towards the direction of lower losses following the suggestion of the gradient directions. Gradient descent in neural nets is efficiently implemented with the Back-Propagation (BP) algorithm, which takes advantage of the nature of neural nets being composite functions and the chain rule of gradients (Rumelhart et al., 1986).

Neural nets are a popular choice for function approximator in RL, because of their abilities to learn complex mappings from observations or state representations to actions or values. While tabular RL algorithms struggle with high-dimensional observation / state / action spaces, neural networks allow for efficient generalization in large or continuous spaces by learning compact, useful representations, enabling RL agents to solve previously intractable problems. Notably, neural nets can approximate value functions (*e.g.*, Q-values or V-values) or policy functions. This ability paved the way for the renaissance of modern DRL, including algorithms like Deep Q-Networks (DQN, to be introduced in detail soon) and policy gradient methods[1].

Despite being subjected to the dangers of the deadly triad, we know from practice that DRL's value estimations, once meticulously tuned, can acquire convergence. Indeed, when we discuss DRL methods, we are most likely in a territory without guarantees.

To prepare the readers for a clear understanding of the methods used in the following chapters, we introduce one of the most impactful RL method - DQN (Mnih et al., 2015), that is fundamental to DRL methods dealing with discrete action spaces.

3.1.1 DQN

Based on Q-Learning (introduced in Sec. 2.4.5, Page. 23, originally proposed in Watkins and Dayan (1992)), Deep Q-Network (DQN) is a DRL method operating in discrete action spaces, based on off-policy Q-value estimation without an explicitly parameterized policy (Mnih et al., 2015). Before the emergence of DQN, DRL methods suffered greatly from training instabilities and sensitivity to hyperparameters. As a truly groundbreaking contribution, DQN achieved generally human-level performance on Atari games, and kickstarted the consequent progress in DRL. DQN set itself apart with the following features:

State Encoder & Value Estimator as Neural Networks: both the state representation encoder and the value estimator on top are implemented with neural networks and optimized with neural network parameter optimizers. DQN is fully parameterized, with learned state representations extracted from high-dimensional pixel-based observations.

Target Network for Training Stability: the agent maintains a time-delayed clone of the "policy network", *i.e.*, the bundle containing the state encoder and the value estimator. The clone is named the "target network" and is responsible for providing TD update targets for value estimator training (of the policy network), based

[1] We will skip the discussions of policy gradient methods in this thesis, since they are not too relevant to the contributions in the following chapters.

on the experimental observation that this would produce more stable update targets for value estimator learning. Periodically, the parameters of the target network will be synchronized with the latest parameters in the policy network.

Q-learning with Training Loss: In DQN, the value estimator is updated by the gradient descent-based optimizer to minimize a loss function. DQN uses surrogate loss functions, such as L_2 or Huber loss, to pull the estimated values of the current state towards an update target y constructed in the Q-learning fashion (introduced in Sec. 2.4.5, Page. 23). For example, over a sampled transition $\langle s, a, r, s', \omega' \rangle$, where ω' is a binary indicator of if the next state s' is terminal, the simple L_2 surrogate loss takes the following form:

$$\mathcal{L}_{\text{DQN}} \coloneqq (\hat{Q}_\theta(s, a) - y)^2$$

where y, in Mnih et al. (2015) (the original DQN paper), is an update target constructed with the help of the target network θ':

$$y_{\text{DQN}} \coloneqq \begin{cases} r & \text{if } \omega \text{ is true} \\ r + \gamma \max_{a'} \hat{Q}_{\theta'}(s', a') & \text{otherwise} \end{cases} \tag{3.1}$$

An alternative method of constructing update targets, named double DQN or DDQN for short, has shown more promising performance against the overestimation problem of Q-learning induced by the $\max$ operator (Hasselt et al., 2016). With DDQN, the update target is constructed jointly by the policy network θ and the target network θ':

$$y_{\text{DDQN}} \coloneqq \begin{cases} r & \text{if } \omega \text{ is true} \\ r + \gamma \hat{Q}_\theta(s', \operatorname{argmax}_{a'} \hat{Q}_{\theta'}(s', a')) & \text{otherwise} \end{cases} \tag{3.2}$$

Note that notations are abused here for simplicity: since both the policy and the target networks have their own paired state encoder, thus the s' input of $\hat{Q}_\theta$ and $\hat{Q}_{\theta'}$ are different, *i.e.*, they are produced by their respective state encoders.

Stores and Trains on Transitions with Experience Replay: DQN stores the interaction history as transitions in the experience replay, and samples the transitions in minibatches to conduct optimization based on batched stochastic gradient descent. An **experience replay** is a buffer for the interaction history between the agent and the environment. The data buffered in the experience replay can be later used for learning purposes, and can be organized in different ways, such as transitions (in DQN) or trajectories (in other methods). Agents that do not require the assistance of experience replay are often distinctively recognized as streaming methods (Elsayed et al., 2024).

ϵ-greedy Exploration: DQN employs ϵ-greedy exploration policy as the behavior policy. Let the target policy given the current value estimate be π, the ϵ-greedy is defined as:

$$\pi_{\epsilon\text{-greedy}} = \begin{cases} \text{uniform random policy} & \textit{w.p. } \epsilon \\ \pi(s) & \text{otherwise} \end{cases}$$

Note that because of DQN's compatibility with off-policy learning, its implementations often take advantage of a controlled annealing from $\epsilon = 1.0$ to a very small value to control the exploration-exploitation tradeoff within a limited agent-environment interaction budget. Detailed implementations can be flexible.

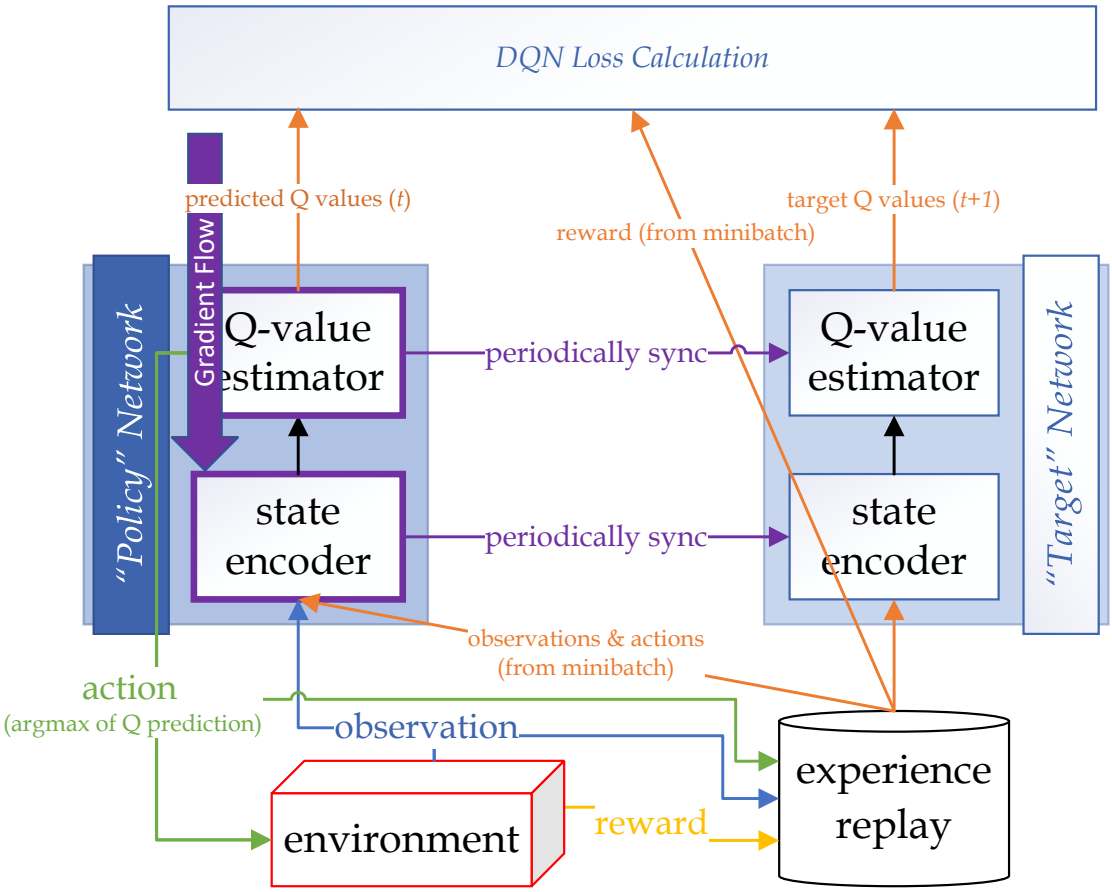


Figure 3.1: ***Dissection of*** DQN*: The policy network and the target network architectures are identical, both consists of a state encoder and a Q-value estimator. The target network is not trained, but updated periodically by synchronizing the parameters from the policy network. Only the policy network is trained through gradient descent-based optimization, and it can be extracted as an inference-only agent, i.e., all components except the policy network are for training purposes only. While in the literature, we often see the policy network as one whole unit, for more unified discussions, we decompose the policy network into the two components of the state encoder and the Q-value estimator.*

An overall illustration of DQN's training and behaviors is presented in Fig. 3.1.

3.1.2 Distributional RL

Distributional RL is a useful DRL technique that enables the estimation of the *distribution* of returns, instead of only the scalar expectation of the return. Distributional RL can be similarly analyzed through the lens of Bellman operators and contractions (as suggested in Sec. 2.3.1, Page. 18), which will show that the convergence guarantees of scalar RL updates are preserved (Bellemare et al., 2017).

In short, distributional RL changes the architecture of the value estimator's function approximator from having a scalar output to having a vectorized output, where the vectorized output is a representation of certain estimated distributions, as shown in Fig. 3.2. Naturally, the old surrogate losses used for scalar prediction must be also replaced, so they can help the predicted distributions converge to the update target distributions with the help of the gradient descent-based optimizers, depending on how the distributions are represented.

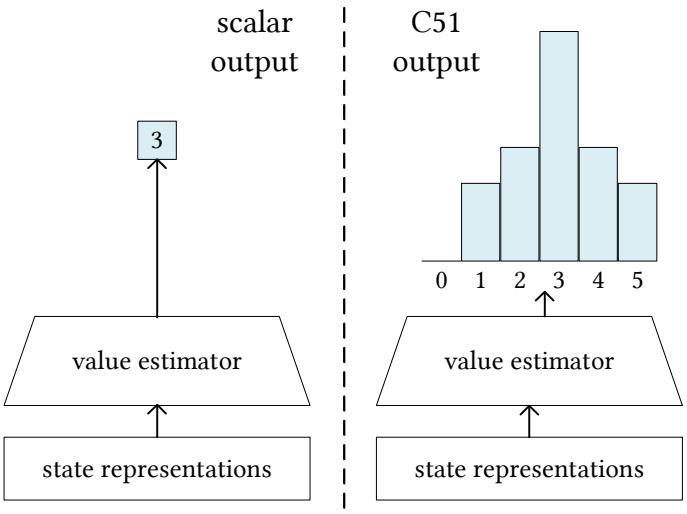


Figure 3.2: ***C51 Distributional Variant of*** DQN ***Value Estimators****: Q-value estimators output a prediction of values based on the input pair* $\langle s, a \rangle$*. For vanilla DQN, the output is a scalar; For C51 (Bellemare et al., 2017), the output is similar to histograms where the bins within a certain range of support are preset. For the right side (C51), the support is set to be* $\{0, 1, 2, 3, 4, 5\}$*, and the outputs are* `softmax`*-ed logits of the* 6*-dimensional outputs from the value estimator.*

There are several popular choices for the representation of the estimated distributions. For the contents of this thesis, we are particularly interested in the so-called C51-style distributional outputs (Bellemare et al., 2017).

C51 outputs a histogram-like representation of a discretized distribution over a preset support over certain value ranges. A histogram over the preset support will be the output, where the bins of the histograms do not necessarily have to be uniform. The expectation can be recovered by a weighted sum over the preset support. The surrogate losses for convergence take the forms of Kullback-Leibler divergence (KL-divergence) and the update targets are the histograms converted from the scalar update targets bootstrapping the weighted-sum expectation estimates (Eq. 3.1 & Eq. 3.2).

C51 greatly alleviates the problem introduced by the difference in the magnitude of outputs, and thus agents with auxiliary learners (that learn to estimate not only the values) are less subjected to conflicts of training losses due to imbalanced magnitudes. **Auxiliary learners** are additional estimators introduced during the training, typically used to improve performance in a specific way or help the model learn more robust features. Auxiliary learners and their training signals - auxiliary losses, may not be the primary objective of the model, but are designed to assist the primary learner (the primary loss function) by providing supplementary guidance. Sometimes in RL, only the value estimator is recognized as the non-auxiliary learner, among all learners in the RL system.

In this thesis, we by default used a variant of C51 distributional TD learning used in Schrittwieser et al. (2019) on all DQN-based baseline and agents. That is, the estimators output histograms instead of scalars. We regress the histogram towards the targets, where these targets are skewed histograms of scalar values, towards which KL-divergence is used to train. Note that this variant of C51 does not use the full distributional estimates of the subsequent states to construct update targets that span the full support, rather, only converting the

scalar update targets into two-hot histograms (Schrittwieser et al., 2019). The variant was preferred for its flexibility of not having to know the full distributions of the update targets, useful in planning situations. Also, in Chap. 5, we proposed a technique upon C51 distributional outputs, to interchangeably learn the distributions of cumulative discounts and cumulative distances simultaneously.

3.1.3 Goal-Conditioned RL: Source-Target Pairs & Hindsight Relabeling

Goal-conditioned RL is an extension of traditional RL where an agent's objective is to achieve specific goals rather than merely maximizing cumulative rewards. In MDPs, a goal is defined to be a set of states that match certain criteria, where such state set can be empty, a singleton or even infinite (Ghosh et al., 2019).
In goal-conditioned RL, the agent conditions its behaviors additionally on a goal (*e.g.*, a target state or a desired outcome) at each timestep. This is to say that goal-conditioned agents must learn a **goal-conditioned policy** that, given both the current state and the goal, takes actions to achieve the goal effectively. The goals could come from the agent itself, or an external instructor that is a part of the environment, *etc.*.
Goal-conditioned policies enable more flexible and adaptable learning, as the agent can generalize to different tasks by conditioning its behavior on various goals, rather than learning a separate policy for each task. From the viewpoint of options (introduced in Sec. 2.5.2, Page. 27), this also mean that one single goal-conditioned policy may be used to substitute a whole set of options. Also, evidently, a goal-conditioned policy can also be viewed as a special case of constrained option, whose initiation set is the whole state space $\mathcal{S}$ and the termination condition is the goal criterion.
Some goal-conditioned agents can be viewed from the perspectives of the feudal RL framework, a hierarchical RL framework in which a high-level *manager* provides low-level goals to its *workers* (Dayan and Hinton, 1992; Parr and Russell, 1997; Vezhnevets et al., 2017). Feudal RL framework is highly similar to the target-assisted planning framework, which my collaborators and I have abstracted for existing planning agents that utilizes generative models to produce targets to achieve (in Chap. 6, Page. 112), whose emphases are rather on the generator-estimator duo, inspired by the belief formation and belief evaluation systems that underpin the delusion mechanisms in the human brain.
Goal-conditioned agents often rely on "source-target pairs" for training, where "source" often represent a current state, *i.e.*, the state that the agent should make decisions from, and the "target" correspond to one of the states that match the goal (a state that belongs to the set of states that match the goal criterion). Source-target pairs are organized training data that could be used to let the agents learn about the relationship between one state and another. Note that pairs do not need to be explicitly stored: for example, two ordinarily sampled transitions can be trivially assembled during training to form a source-target pair, as follows

$$\langle s_1, a_1, r_1, s_1', w_1' \rangle \bigoplus \langle s_2, a_2, r_2, s_2', w_2' \rangle \rightarrow \langle s_1, a_1, r_1, s_1', w_1', s_2 \rangle$$

where s_1 acts as the source state and s_2 acts as the target state.
There are many ways to construct a source-target pair, but undoubtedly, **Hindsight Experience Replay (HER)** is among the most popular (Andrychowicz et al., 2017). HER's core mechanism is **hindsight relabeling**, where agent-environment interaction histories buffered in the experience replay while following their respective contemporary goals are relabeled with other goals (the ones that the agent was not following).

Intuitively, this makes the agent pretend that certain experience was gathered when following the relabeled goals.

Training target-assisted planning agents with only contemporary targets (the ones being followed) can lead to poor performance (Dai et al., 2021), since contemporary targets may be low-quality, hard to achieve, or lack in diversity (Moro et al., 2022; Davchev et al., 2022). The fact that hindsight relabeling creates much more diverse source-target pairs (compared to those with only the contemporary goals as targets) for the goal-conditioned estimators to generalize and gain deeper understanding of the relationship inside the source-target pairs, which in turn contributed to its massive success in training goal-conditioned policies.

From another perspective, HER is crucial for enhancing *sample efficiency* in goal-directed RL (Andrychowicz et al., 2017), as it enables learning from failed experiences (Dai et al., 2021). We will discuss HER more in Sec. 6.5.2 (Page. 124), about its connections with our contributions and its existing limitations.

Formally, HER augments a transition $\langle x_t, a_t, r_{t+1}, x_{t+1} \rangle$ with an additional observation $x^{\odot}$ (or its encoding), the relabeled target, creating a source-target pair, with one decision point for the current step and another for the relabeled target. **Relabeling strategies**, which correspond to how $x^{\odot}$ is selected, are critical for the performance of HER-trained agents (Shams and Fevens, 2022). Most popular choices are *trajectory-level*, meaning $x^{\odot}$ comes from the same trajectory as x_t. These include FUTURE, where $x^{\odot} = x_{t'}$ with $t' > t$, and EPISODE, with $0 \leq t' \leq T_{\perp}$.

The introduction of HER greatly enhanced the sample efficiency of learning about experienced targets. Meanwhile, the incompleteness of the accompanying relabeling strategies planted a hidden risk of delusions towards hallucinated targets, which will be discussed later in Chap. 6 (Page. 112), where we will also discuss relabeling strategies in detail.

3.2 Deep Learning Preliminaries

3.2.1 Generative Models: VAEs & Conditional VAEs

Aiming to maximize the likelihood of the observed data through a latent variable model, Variational Autoencoders (VAEs) are a type of generative model that learn to approximate the distribution of training data. Given an observed dataset $\{\boldsymbol{x}\}$, a VAE models the joint distribution $p(\boldsymbol{x}, \mathbf{z})$, where $\mathbf{z}$ represents its discovered latent variables (Kingma and Welling, 2014). We illustrate the mechanism of VAEs in Fig. 3.3.

Despite the effort to model the joint $p(\boldsymbol{x}, \mathbf{z})$, the ultimate objective of a VAE is to maximize the marginal likelihood $p(\boldsymbol{x})$, which is given by:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\mathbf{z})p(\mathbf{z})\, d\mathbf{z}$$

However, directly computing this integral is intractable. To address this, VAEs use variational inference to approximate the true posterior $p(\boldsymbol{z}|\boldsymbol{x})$ with a simpler, tractable surrogate $q(\boldsymbol{z}|\boldsymbol{x})$.

Such variational approach involves maximizing the Evidence Lower Bound (ELBO), which is a lower bound on the log-likelihood of the data, which a VAE seeks to learn. We can derive the ELBO from the log of the marginal likelihood $p(\boldsymbol{x})$:

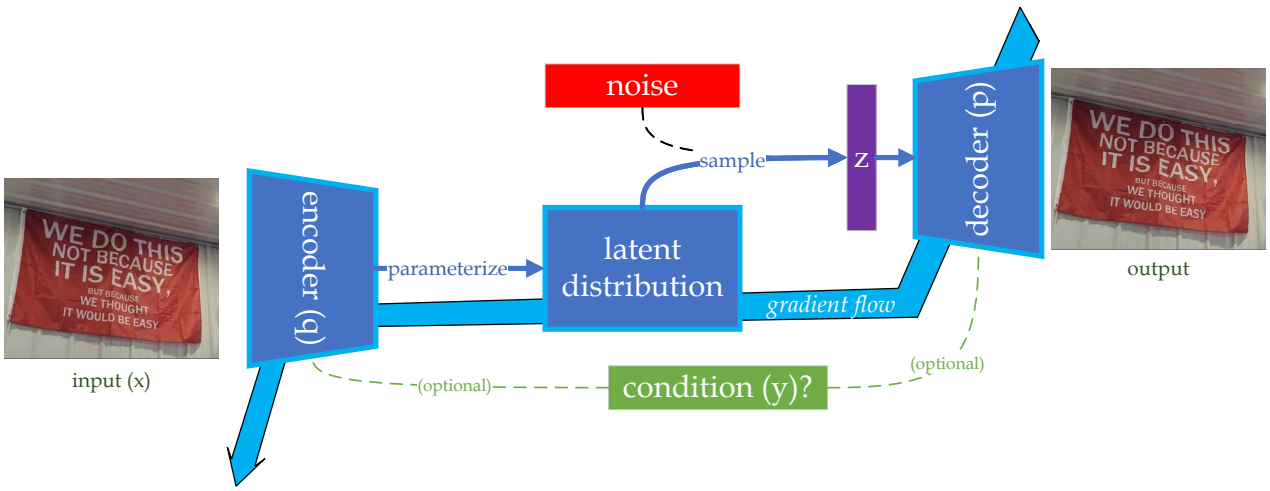


Figure 3.3: ***VAE Mechanisms****: the encoder q extracts parameters that define the distribution of the latent variable z, which is sampled and then used with the decoder z to parameterize the likelihood of the reconstructed output $\hat{x}$. The gradient flows back from the output, to the decoder and the encoder. The gradient flow through the sampled z is enabled by reparameterizing z with the help of an external noise (Kingma and Welling, 2014).*

$$\begin{aligned}\log p(\boldsymbol{x}) &= \log \int p(\boldsymbol{x}|\mathbf{z})p(\mathbf{z})\, d\boldsymbol{z} = \log \int \frac{q(\boldsymbol{z}|\boldsymbol{x})}{q(\boldsymbol{z}|\boldsymbol{x})} p(\boldsymbol{x}|\mathbf{z})p(\mathbf{z})\, d\boldsymbol{z} \\ &= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x},\mathbf{z})}{q(\boldsymbol{z}|\boldsymbol{x})}\right] + \mathrm{KL}(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z}|\boldsymbol{x}))\end{aligned} \tag{3.3}$$

The first term is the ELBO, and the second term is the Kullback-Leibler (KL) divergence between the variational distribution and the prior over the latent variables, which will be discarded due to intractability. The ELBO can be expanded into two terms:

$$\mathrm{ELBO}(\theta,\phi) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\mathbf{z})\right] - \mathrm{KL}(q(\boldsymbol{z}|\boldsymbol{x})\|p(\mathbf{z}))$$

In practice, the VAE learns the parameters θ and ϕ of the neural networks that parameterize $p(\boldsymbol{x}|\mathbf{z})$ and $q(\boldsymbol{z}|\boldsymbol{x})$, respectively, by maximizing the ELBO.

Maximizing the ELBO encourages the model to generate accurate reconstructions of the data while keeping the variational posterior close to the prior distribution. Depending on how the latent space is defined, the ELBO can be instantiated in different forms, some even with closed form solutions, such as in Kingma and Welling (2014), where an encoder extracts from input the parameters for defining a normal distribution. In Chap. 5, we used the encoder to instead parameterize a discrete bottleneck consists of a bundle of categorical distributions, from which a joint sample is treated as z, used as a partial description of a state (see Sec. 5.3.2). A hyperparameter β can be introduced to weight the two terms to control the learning, giving rise to β-VAE (Higgins et al., 2017):

$$\mathcal{L}(\theta,\phi,\beta) = -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\mathbf{z})\right] + \beta \cdot \mathrm{KL}(q(\boldsymbol{z}|\boldsymbol{x})\|p(\mathbf{z}))$$

Here, β is a scalar that controls the trade-off between the reconstruction term and the KL divergence term. When $\beta = 1$, this reduces to the standard VAE loss.

The **Conditional VAE (CVAE)** is an extension of standard VAE, where both the encoder and the decoder are conditioned on additional information, such as class labels or other external attributes. This allows the model to generate data conditioned on specific variables, making it suitable for tasks like conditional goal generation, where the generated goals depend on information collected up to the current state.

Formally, in a CVAE, the objective is to learn the conditional distribution $p(\boldsymbol{x}|\boldsymbol{y})$. This is done by learning the joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{y}$ is the condition, by marginalizing over latent z in the joint $p(\boldsymbol{x}, \boldsymbol{y}, \mathbf{z})$. The encoder and decoder are modified as follows:

Encoder: The distribution $q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$ is conditioned on both the data $\boldsymbol{x}$ and the condition $\boldsymbol{y}$.

Decoder: The likelihood $p(\boldsymbol{x}|\mathbf{z}, \boldsymbol{y})$ is conditioned on both the latent $\mathbf{z}$ and the condition $\boldsymbol{y}$.

The ELBO inequality for CVAE is:

$$\log p(\boldsymbol{x}|\boldsymbol{y}) \geq \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{y})}{q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})}\right]$$

With the same algebra, this results in the following objective function (for minimization):

$$\mathcal{L}_{\text{CVAE}}(\theta, \phi) = -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}\left[\log p(\boldsymbol{x}|\mathbf{z}, \boldsymbol{y})\right] + \text{KL}(q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})\|p(\mathbf{z}))$$

In Chap.5 and Chap.6, we use CVAE in conjunction with hindsight experience replay to train generative models that can imagine distance future states, conditioning on the current state.

3.2.2 Attention

Attention mechanisms, first popularized by Bahdanau et al. (2015) in the context of machine translation, have evolved to become one of the most powerful and widely used tools in machine learning. Allowing models to focus on different parts of the input sequence or data, the ability of attention has made it central to language models powered by the Transformer architecture (Vaswani et al., 2017). The ability to query sets of objects with varying degrees of attention has led to improvements in tasks such as question answering (Devlin et al., 2019), image captioning (Xu et al., 2015), and even RL (Chen et al., 2021). Through the introduction of mechanisms like multi-head attention and semi-hard attention, these models can capture more complex patterns and relationships within data, enabling a wide range of applications. One of the core contributions of this thesis, the consciousness-inspired - "spatial abstraction" mechanism, is implemented as a top-down semi-hard attention bottleneck, in Sec. 4.4 (Page. 65) and improved to work in synergy with temporal abstraction in Sec. 5.3.1 (Page. 89).

To understand formally how attention works, we now revisit a generic set query procedure (Routine 1), which is also illustrated in Fig. 3.4.

Routine 1: Querying a Set of objects $\{x_i\}$ with an Object x

1. The object $\boldsymbol{x}$ is transformed into a **query vector**. This is generally done via a linear transformation, typically denoted as:

$$\boldsymbol{q} = W_q \boldsymbol{x}$$

where W_q is a learnable weight matrix.

2. Each object in the set being queried $\{\boldsymbol{x}_i\}$ is independently transformed into two other vectors, forming two sets of the same cardinality, named the **key set** $\{\boldsymbol{k}_i\}$ and the **value set** $\{\boldsymbol{v}_i\}$, respectively. With linear transformations, these are computed as:

$$\boldsymbol{k}_i = W_k \boldsymbol{x}_i, \quad \boldsymbol{v}_i = W_v \boldsymbol{x}_i$$

where $\boldsymbol{k}_i$ and $\boldsymbol{v}_i$ are the transformed key and value vectors for $\boldsymbol{x}_i$ in set $\{\boldsymbol{x}_i\}$, and W_k, W_v are the learnable weight matrices for the key and value transformations.

3. Each key vector $\boldsymbol{k}_i$ in the key set is scored by similarity to the query vector $\boldsymbol{q}$, according to some similarity metric, typically the scaled dot-product similarity:

$$\text{score}(\boldsymbol{q}, \boldsymbol{k}_i) = \frac{\boldsymbol{q}^T \boldsymbol{k}_i}{\sqrt{d_k}}$$

where d_k is the dimensionality of the key vectors, and the similarity score is often scaled by $\sqrt{d_k}$ to prevent large values during training, which could lead to vanishing gradients.

4. The **attention weights** are then computed by applying a `softmax` to the scores:

$$\alpha_i = \frac{\exp(\text{score}(\boldsymbol{q}, \boldsymbol{k}_i))}{\sum_j \exp(\text{score}(\boldsymbol{q}, \boldsymbol{k}_j))}$$

where α_i represents the normalized attention weight for the i-th key, and the denominator sums over all key vectors to ensure the attention weights sum to 1, to form a probability distribution.

5. Finally, the value vectors are weighted by the normalized attention weight vector, and thus combined to yield the output vector:

$$\boldsymbol{y} = \sum_i \alpha_i \boldsymbol{v}_i$$

The output $\boldsymbol{y}$ represents a weighted combination of the values $\{\boldsymbol{v}_i\}$, where the weights correspond to the degree of "attention" the model pays to each key-value pair. Intuitively, this is the "answer" to the query of $\boldsymbol{x}$ given by the set $\{\boldsymbol{x}_i\}$.

Querying a set with *another set of vectors* is no different from independently applying the described procedure multiple times, for each object in the set. The number of outputs always matches the size of the query set, as each query element $\boldsymbol{q}_i$ generates an output vector.

Now let us look into some more advanced use cases of attention:

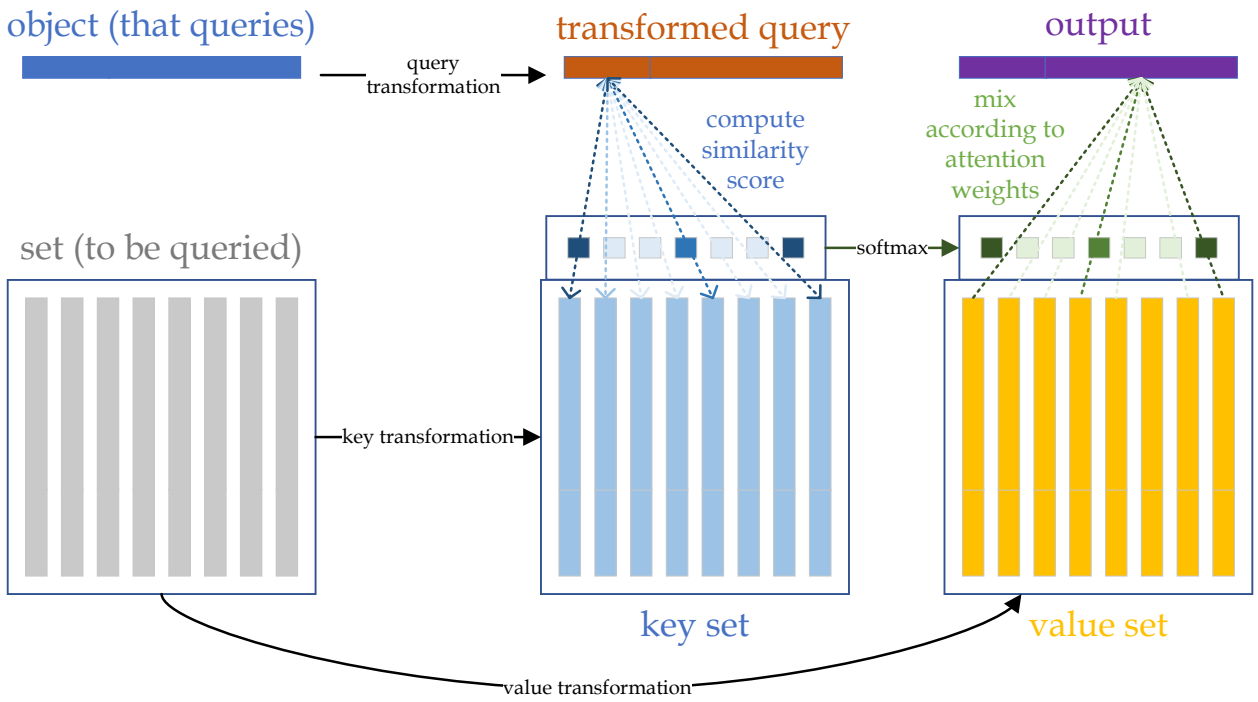


Figure 3.4: ***Querying A Set with an Object****: the object is first transformed into a query vector. Then, attention weights are computed based on the query's similarity to the transformed keys (from the set of objects being queried). Finally, the transformed values are fused to form the output according to the attention weights). Top-k attention (Sec. 3.2.2 on Page. 40) can be implemented by setting the non-top-k attention weights to 0, or equivalently the non-top-k scores to $-\infty$, due to the choice of* `softmax`*.*

Self-Attention

When a set queries itself, *i.e.*, when the one set is queried by a set identical to itself, such procedure is referred to as *self-attention*. In self-attention, the query, key, and value vectors are all derived from the same set of objects, and the model computes attention scores between elements within the same set. This allows each element in the set to attend to every other element, facilitating the extraction of long-range dependencies in sequences. The self-attention mechanism is key to models like the Transformer, where it enables the model to capture complex relationships within an input sequence (Vaswani et al., 2017).

Multi-Head Attention

Multi-head attention involves performing attention computations multiple times independently, each using a distinct set of transformations, such as different learned linear weights. The outputs from each transformation set (or "head") are then concatenated and projected back into the desired dimensionality. This approach enables the model to learn attention from multiple subspaces simultaneously, capturing various aspects of the relationships between objects.
In the linear case, multi-head attention can be formulated as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\boldsymbol{y}_1, \boldsymbol{y}_2, \dots)W_O$$

where Q, K and V are the concatenated matrix representations of the query set, the key set and the value set, respectively. Each head $\boldsymbol{y}_i$ is defined as:

$$\boldsymbol{y}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

where W_Q^i, W_K^i and W_V^i are the learnable linear transformation weights for the query, key, value of head i, respectively, with W_O being a final output transformation matrix. This allows the model to aggregate different attention patterns from different subspaces.
Multi-head attention was used in the set-based bottlenecked dynamic models introduced in Chap. 4 (Sec. 4.4, Page. 57).

Semi-Hard Attention

When computing attention weights, it can be beneficial to ignore the unrelated values and focus more strongly on the more matching elements. A modification of attention called **semi-hard attention** involves keeping only the top-k attention weights, where the output of a vector-set query is only a weighted sum over the top-k most matching values. Note that the output vector will still be a *soft* mixture of the top-k values, while the non-top-k values will be *hard*-discarded (Gupta et al., 2021). Here, the classical terminology about soft and hard attention weights is used: when the weights are binary values, a selection is essentially conducted, which is recognized as "hard"; While, if the attention weights are continuous values, the attention is recognized as "soft". Top-k attention weights are soft on the top-k indices of the attention weight vector, while being hard 0s on others, separating the influence of the non top-k ones.
Formally, semi-hard weights are obtained by selecting the top-k scores from the full attention weights, setting the remaining weights to zero, and then renormalizing the selected weights:

$$\alpha_i' = \begin{cases} \frac{\alpha_i}{\sum_{j \in \text{top-}k} \alpha_j} & \text{if } \alpha_i \text{ is one of the top-k scores,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

The formula above shows that a linear renormalization is conducted such that the selected top-k attention weights sum to 1. Then, as usual, the renormalized weights are applied to the value vectors. The renormalization can alternatively be implemented more efficiently by setting the non-top-k scores to be $-\infty$ before the `softmax` operation (Step 3 & 4 in Routine. 1 on Page. 38).
When querying a set $\{\boldsymbol{x}_1^K, \ldots, \boldsymbol{x}_n^K\}$ containing $n > k$ objects with another set $\{\boldsymbol{x}_1^Q, \ldots, \boldsymbol{x}_k^Q\}$ of exactly k objects, the score matrix will be $k \times n$ (see Step 3 in Routine. 1 on Page. 38). With ordinary semi-hard top-k attention, this query would result in an output set of k objects, each a transformation of its own top-k most matching object from $\{\boldsymbol{x}_i^K\}$. This means, the output set will be a set with k to k^2 objects, where each object is a transformation of k best-matching objects from $\{\boldsymbol{x}_i^K\}$, essentially creating a transformed subset of $\{\boldsymbol{x}_i^K\}$. This is how we used a semi-hard top-k attention as a mechanism to soft-select a subset of objects out of a larger set, with it created the backbone mechanism of spatial abstraction proposed in Chap. 4. In Chap. 5, spatial abstraction was made more generic to non-object-oriented state representations, *e.g.*, to work with feature maps of local patches of observations extracted by Convolutional Neural Networks (CNNs, LeCun et al. (1989)).

Bottom-Up *v.s.* Top-Down Attention

In both machine learning and human cognitive science, attention mechanisms can be classified into two general types based on the source of the attentional signal, *i.e.*, the signal that conditions the attention: **bottom-up attention** and **top-down attention**. While both types use learned attention weights to focus on different parts of the input data, the sources of conditioning signals in each approach differ significantly.
Bottom-Up attention is driven by the intrinsic properties of the input data. In this approach, the model attends to salient or noteworthy features in the input data based on its sensory characteristics, without any explicit guidance or prior knowledge. For example, in computer vision, regions of an image that contain high contrast, bright colors, or movement might naturally attract more attention (Mnih et al., 2014). Bottom-Up attention is data-driven and reactive, often used in tasks where the model must extract relevant information from raw data with minimal preconceptions.
Top-Down attention, instead, is guided by higher-level goals, intentions, or prior knowledge. In this approach, the attention mechanism is influenced by the intention of the model, allowing the model to focus on certain aspects of the data that are most relevant to achieving the desired outcome. For example, in a visual question answering (VQA) task, the model may direct its attention toward specific regions of an image that are relevant to answering the given question, based on context and prior knowledge, rather than simply attending to all potentially salient features (Antol et al., 2015).
We used top-down attention mechanisms to implement spatial abstractions: 1) in Chap. 4 to make the model pay attention to the aspects of the state representations most relevant to the intended actions during planning (Sec. 4.4, Page. 65), and 2) in Chap. 5 to make the estimators to focus on parts of the state representations most relevant to transitioning from one state to another target state (Sec. 5.3.1, Page. 89).

3.3 Model-based RL

Despite significant progress in applying RL to a variety of problems, its real-world application remains hindered by challenges such as poor sample efficiency, unsatisfactory generalization, and other issues (Lillicrap et al., 2016). In a standard RL system, the primary focus is on estimating a single quantity: the returns. While value estimation alone can suffice for optimal decision-making (Silver et al., 2021), relying solely on returns often proves inefficient or even intractable in many real-world scenarios.
Model-Based RL (**MBRL**) offers an alternative approach by incorporating predictive or generative models, which provide estimations of additional quantities beyond just returns. This inclusion of models opens up new avenues for improving efficiency and performance in RL.
The versatility of MBRL lies in its creative applications of models. For instance, models can be derived from domain expertise, learned in advance, or developed during the RL agent's training process. Furthermore, models in MBRL can be utilized at different stages of decision-making. **Decision-time** planning explicitly uses a model to help make decisions at decision-time (Alver and Precup, 2024). For example, MuZero performs a tree search at each timestep, querying a learned model to simulate the most desirable future actions (Schrittwieser et al., 2019). In contrast, **background** planning methods typically function as model-free during decision time, but leverage the model to enhance learning during training. Prominent examples of such methods include Dyna (Sutton, 1991) and Dreamer (Hafner et al., 2024), among others.

3.3.1 Sources of Models: Where They Come From

We begin by discussing the origins of models in MBRL agents, specifically distinguishing between **acquired models**, which do not require learning, and **learned models**, which are acquired during RL training.

Known Perfect Models

Computer Go has been successfully conquered by the MBRL agent AlphaGo and its variants (Silver et al., 2016). In this case, the model used for decision-time planning is the Go simulator itself, which provides perfectly accurate dynamics. With a perfect model, planning can be highly effective when combined with classical AI search methods, such as those with guaranteed performance (Kocsis and Szepesvári, 2006). However, task-specific models are not always available for all tasks. A more general and practical approach involves learning a model of the environment.

Models that are Learned

Many methods involve learning the models they utilize. Notable examples include the use of learned models in planning algorithms, such as the World Models framework (Ha and Schmidhuber, 2018; Zhou et al., 2025) and Model-Based Policy Optimization (Levine et al., 2016). These approaches have shown success in both simulated and real-world environments.
We now briefly review model learning strategies, distinguishing between approaches where the model and the RL agent are trained together as a unified system or separately.

- **Pre-Trained Models**: Many methods employ separate training stages for the model and the core RL components. For example, consider the unsupervised training methodology.

  Approaches like World Models (Ha and Schmidhuber, 2018; Zhou et al., 2025) benefit from a separated phase for model learning, known as the **exploration** or **pretraining** phase, where the RL agent is not involved. By using a fixed exploration policy, this strategy transforms the non-stationary nature of model learning in an RL setting into a more stable, stationary learning problem, thereby sidestepping certain challenges.

  Compared to end-to-end training methods, to be introduced shortly, a key disadvantage is that these approaches allocate a significant portion of the agent-environment interaction budget to a potentially long exploration phase (assuming the "unsupervised" training phase is not freely provided). Another issue is that the consequent models are trained only on the trajectories close to the initial states, as uniformly random policies usually do not get the agents far. Thus, these methods may struggle in environments with distinct local features that require the model to continually learn.

- **End-to-End Trained Models**: In supervised learning, "end-to-end" training refers to optimizing a model directly from inputs to outputs. While in RL, end-to-end indicates that both the model and the RL agent are trained together, where the model can aid in decision-making and value estimation. This means both the model and other system components learn from scratch.

  However, end-to-end training does not necessarily imply that the model and RL components are tightly coupled. For example, in Dyna, although the model and value estimator are learned concurrently, their learning processes are largely independent.

End-to-end training is attractive due to its procedural simplicity, especially in deep learning scenarios. However, it is not always suitable for every method. Since end-to-end models are learned from scratch alongside the RL system, the initial stages of RL learning can be difficult without an effective model. The model must rapidly learn to assist the RL system in a meaningful way.

Moreover, training a model end-to-end in RL control tasks is technically challenging, particularly due to non-stationarity. The data distribution evolves as the agent's policy changes. Additionally, when the model is coupled with RL components (*e.g.*, sharing the same state representation encoder), training signals often compete with each other. Different optimization objectives may not align, and an inadequately tuned bottleneck for all signals can lead to undesirable parameter landscapes. Although scalar trade-offs and alternative optimization objectives are commonly used to mitigate these issues, they are not always effective.

3.3.2 Timing of Planning: Decision-Time *v.s.* Background

Planning is a term referring to the use of more computation to improve predictions and behavior without additional agent-environment interactions, *i.e.*, essentially trading computation for sample efficiency (van Hasselt et al., 2019).
For clear initiation of the discussions and the coherence with terminologies proposed in Alver and Precup (2024), based on the timing of the model usage, we roughly categorize some existing planning methods into two categories: **decision-time** planning methods and **background** planning methods. While a decision-time planning agent *utilizes the model to directly enhance its behavior* when interacting with the environment, a background planning agent often seeks to *improve itself with a model when not interacting with the real environment.*

Decision-Time Planning

Decision-time planning is a computational embodiment of proactive decision-making. Intuitively, a decision-time planning agent evaluates potential future outcomes based on its understanding of the effects of actions and current value estimates, then selects the most promising action or sequence of actions to pursue. This knowledge is derived, at least in part, from the model.
A key advantage of decision-time planning is that, with an accurate model, behavior policies can quickly adapt toward optimality. This proactive approach also enhances adaptability in unexpected situations. Notable decision-time planning agents include MuZero (Schrittwieser et al., 2019) and PlaNet (Hafner et al., 2019).
Decision-time planning can also be framed as a search problem. We will now explore a representative methodology: Model Predictive Control (MPC).

Model Predictive Control (MPC) & Tree Search

Model Predictive Control (MPC) is a decision-time planning methodology that involves optimizing control inputs over a finite prediction horizon (Hansen et al., 2024). At each time step, MPC solves an optimization problem using a model to predict future states and determine the optimal control sequence. The optimization typically minimizes a cost function that balances objectives like tracking a desired trajectory, minimizing

energy consumption, or avoiding constraints. After solving the optimization, only the first action is applied, and the process can be repeated at the next timestep, updating the predictions with new observations. Beyond RL, MPC is also widely used in systems where constraints (*e.g.*, physical limits or safety requirements) must be explicitly respected. MPC serves as the foundation of many modern successful applications, *e.g.*, Monte-Carlo Tree Search (MCTS) which powers state-of-the-art game RL game-playing agents (Schrittwieser et al., 2019).

We can implement MPC with tree-search. The search tree structure is used to represent and evaluate possible future states of the system. Below are key aspects of how MPC utilizes the search tree:

1. **Nodes in the Search Tree**: In the context of MPC, each node in the search tree represents a possible state at a specific time step. MPC expands the search tree by appending new nodes corresponding to the predicted future states, which are generated by applying potential actions to the current state using the model;

2. **Budget-Constrained Search with Priority Queue**: The number of simulations (or steps of search) is typically constrained by a budget, meaning the agent performs a limited number of tree expansions. Once the budget is exhausted, the agent selects the immediate action based on the most promising path found within the search tree. For this, MPC tree search algorithm uses a priority search queue to manage which branches of the search tree are explored first. For example, some MPC's search priorities of the nodes are set according to a best-first search heuristic, which is designed to prioritize nodes that are more likely to lead to a favorable outcome. The priority heuristic in the MPC search process influences the order in which different branches of the search tree are explored, though it does not affect the finalization of the decision regarding the optimal action. The optimal action is always the one that leads to the most promising trajectory, as defined by the optimization objective. Only the first action of the optimal sequence is applied, and the search tree can be reconstructed at the next timestep if necessary, taking into account the new observations.

3. **Action Selection and Node Evaluation**: At each planning step, the MPC evaluates the nodes by calculating the expected future return (or other surrogate values) associated with each branch. The optimization process selects the action corresponding to the most promising node;

We now introduce a simple instance of the tree search algorithm used in Chap. 4, characterized by a priority search queue determined by certain priority heuristic.

For more intuitive understanding, we provide in Fig. 3.5 an example showing how the algorithm works with a best-first heuristic with $\gamma = 1$ and $|\mathcal{A}| = 3$ and maximum planning steps of 3.

In the case of Fig. 3.5, which depicts the tree-search algorithm used in decision-time planning in Chap. 4, the agent can choose to (re-)plan at every timestep using the learned model to maximize adaptability and to correct deviations in the outdated planning results.

Equivalence could be drawn from this planning approach to Monte-Carlo Tree Search (MCTS) (Silver et al., 2016, 2017a). While this method is more simplistic and assumes deterministic models to operate with.

Algorithm 1: Prioritized Tree-Search MPC

Input: s_0 (current state), $\mathcal{A}$ (action set), $\mathcal{M}$ (model), $\mathcal{Q}$ (value estimator), γ (discount)
Output: a^* (action to be taken)

$q = \text{queue}(); q_T = \text{queue}()$ //q_T for terminal nodes
$n_u = \text{NODE}(s_0, \text{root} = \text{True})$ //n_u denotes a node with branches unprocessed nor in q
while *True* **do**
  if $n_u.\omega$ **then**
    $q_T.\text{add}(\langle n_u, n_u.\sigma \rangle)$ //identified as a terminal state. n_u is added to q_T using bisection, together with the discounted sum of the simulated rewards along the way $n_u.\sigma$
  else
    for $a \in \mathcal{A}$ **do** $q.\text{add}(\langle n_u, a, n_u.\sigma + \gamma^{n_u.\text{depth}} \cdot Q(n_u.s, a) \rangle)$ //bisect *w.r.t.* priority ;
  if *isempty*(q) **then**
    break //tree depleted
  $n_c, a_c, v_e = q.\text{pop}()$ //get branch with highest priority; for in-distribution setting, priority is the estimated value of the leaf trajectory
  if *budget depleted* **then**
    break //termination criterion met
  $\hat{s}, \hat{r}, \hat{\omega} = \mathcal{M}(n_c.s, a_c)$ //simulate the chosen branch
  $n_u = \text{NODE}(\hat{s}, \text{parent} = n_c)$
  if $n_c.\textit{depth} > 0$ **then**
    $n_u.a_b = n_c.a_b$
  else
    $n_u.a_b = a_c$ //descendants trace root action
  $n_u.\omega = \hat{\omega}; n_u.\sigma = n_c.\sigma + \gamma^{n_c.\text{depth}} \cdot \hat{t}$
$n_c, a_c, v_e = q.\text{pop}(\text{'highest value'})$ //get branch with highest **value** within the expandables
$n^* = n_c$;
if $\neg$*isempty*(q_T) **then**
  $n_T = q_T.\text{pop}(\text{'highest value'})$ //get node with highest **value** within simulated terminal states
  if $n_T.\textit{value} \geq v_e \vee$ *isempty*(q) **then**
    $n^* = n_T$
if *isroot*(n^*) **then**
  $a^* = a_c$
else
  $a^* = n^*.a_b$

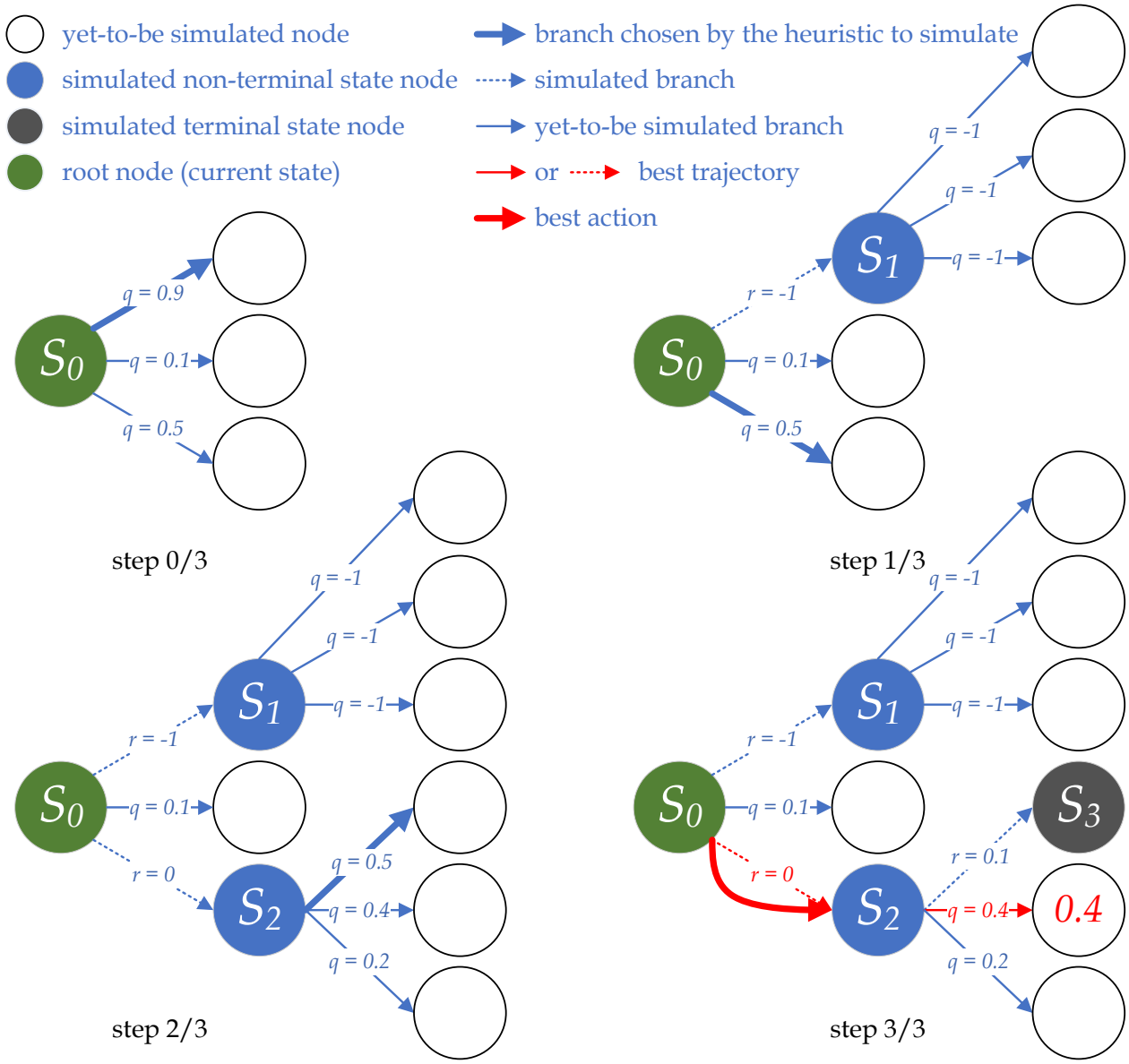


Figure 3.5: ***Example of the Best-First Heuristic-based Tree Search (for MPC)****: Step 0 / 3) Start of planning, with the root node and three branches. The branch* $\langle s_0, a_0 \rangle$ *is chosen due to the best-first heuristic. If we employ the random search heuristic, like what we do in OOD evaluation in Sec. 4.5 (Page. 57), a random branch would be chosen; Step 1 / 3) We expand the chosen branch, popped out of the priority queue. A new node is constructed, together with its out-reaching branches, which are added to the queue. Now the queue has* 5 *branches in it. The heuristic marks* $\langle s_0, a_2 \rangle$ *to be the next simulated branch; Step 2 / 3) Simulation of* $\langle s_0, a_2 \rangle$ *is finished and* $\langle s_2, a_0 \rangle$ *is marked; Step 3 / 3) Node S3 is imagined via* $\langle s_2, a_0 \rangle$ *but it is estimated to be a terminal state. Now, the tree search budget is depleted. We locate the root node branch* $\langle s_0, a_2 \rangle$ *which leads to the trajectory with the most promising return* 0.4.

Background Planning

Background Planning often involves more reactive model usages and does not directly optimize for the action to take at decision-time. We present one representative framework - Dyna, to showcase the potentials of this approach[2] (Sutton, 1991).

Dyna was first introduced in Sutton (1991) as an early RL planning framework designed to reinforce an agent's policy in a simulated environment, while it interacts with the real world[3].

Dyna trains its model based on the experiences from agent-environment interactions, enabling the model to generate simulated transitions. It then uses these simulated transitions to augment the training data for the value estimator. Importantly, there is no fundamental change to the learning process of the value estimator; the only difference is that the training data is enriched with simulations from the model. For more discussions regarding Dyna, please check Sec. 3.3.3 (Page. 48).

Background planning frameworks like Dyna offer sample efficiency benefits and are computationally less demanding during agent-environment interactions, as their additional computations occur offline. However, they cannot provide immediate behavior adaptations in the states the agent encounters. Furthermore, Dyna typically only learns to generate transitions from states visited in the past. As a result, Dyna is focused on the data distribution during training time, which can limit its ability to generalize OOD.

There are some notable works that followed the idea of Dyna, *e.g.*, Hafner et al. (2021); Łukasz Kaiser et al. (2020).

3.3.3 Usage of Models: What They Estimate & How They Help

As discussed, the versatility of MBRL methods makes them promising for improving RL performance across various dimensions. A model does not have to estimate the transition functions defined in an MDP; instead, it can capture other interesting and potentially partial aspects of the environment. Below, we explore several representative use cases of models:

Search for Best Actions with Dynamics Predictors

The most straightforward use of a model is to estimate the transition dynamics of the environment, *i.e.*, to predict how the environment changes given the actions that the agent may take.

In tabular settings (finite state and action spaces), learning transition dynamics is relatively simple. However, in non-tabular tasks, dynamics predictors may have to reconstruct future observations or given state representations, which can be noisy or contain irrelevant details. Despite progress, such as in Łukasz Kaiser et al. (2020), observation-level reconstruction often suffers from inefficiencies that complicate decision-making. Moreover, model architecture may have to be highly task-specific to handle these detailed reconstructions. For instance, object-oriented dynamics models work well for grid-like environments, while models suited for differential equations are better for continuous control tasks.

[2] In some sense, classical methodologies such as Dynamic Programming (DP) can also arguably fit in the category of background planning, as they seek to estimate the value function and optimal policy (Howard, 1960)

[3] Though it may seem counterintuitive to describe Dyna's behavior as planning, the authors argued that conducting RL in a simulated environment enhances the agent's behavior and can therefore be interpreted as a form of planning.

An alternative to detailed reconstruction is to build models in learned state representation spaces, where meaningful features are extracted and irrelevant details to decision-making are neglected. However, state representation learning introduces potential degeneracy: state transitions may be accurate yet meaningless without proper training signals. To address this, methods like Łukasz Kaiser et al. (2020) use reconstruction to seek establishing bijections between observations and states, aiming for lossless compression of observations. However, these methods, to a degree, still share the challenges of observation-level reconstruction (Silver et al., 2017b; Schrittwieser et al., 2019). To mitigate these issues, additional signals, such as value estimation, are needed to constrain state representations and ensure they are meaningful for prediction.

The Predictron (Silver et al., 2017b) is a model-based approach that learns state representations specialized for accurate value estimation and a model that operates within. By matching the Monte Carlo returns from the real environment, the model ensures consistency between generated and real environment returns. This approach enhances TD-based value estimation and has been extended to MuZero (Schrittwieser et al., 2019), which uses decision-time planning with MCTS. MuZero achieves state-of-the-art performance in board games and Atari by employing Predictron-based models for planning. However, training solely with task-specific reward signals can lead to overfitting and overreliance on memorization, limiting generalization across tasks. We will discuss this more in Chap. 4.

These dynamics predictors can be used effectively in tree-search algorithms during decision-time planning. A model that can estimate state transitions can also serve as environment simulators for background planning, which is to be introduced later separately.

Dynamics predictors can also be used to predict the outcomes of macro-action, giving rise to ideas such as **option models**. However, these models are often haunted by the uncertainty accumulated over the time horizon of the macro actions and can be highly unreliable, if sources of stochasticity are involved.

Propose Beneficial Goals with Temporarily-Abstract Target Generators

Models can also be used to directly generate goals that the agent may seek to achieve. This requires that the model are modelled in a temporally-abstract way, since the possible states / observations at the very next timestep typically do not serve as meaningful goals. For instance, LEAP used variational autoencoders (VAEs) to generate observations that may be used as waypoints to guide the overall navigation; While, in Chap. 5, Skipper used conditional VAE (CVAEs) to generate possible states given the environmental contexts, enabling the OOD generation of targets that help decompose the overall tasks into smaller and more manageable steps.

Learn for Free with Experience Generators & Experience Replays

Dyna and recent methods like Ha and Schmidhuber (2018); Łukasz Kaiser et al. (2020) use models to generate simulated transitions, building on generative modeling approaches. Recent work focuses on stochastic experience generation to improve MBRL in stochastic environments (Łukasz Kaiser et al., 2020; Du et al., 2020).

Experience Replay, popularized by DQN, stores transitions or trajectories experienced by the agent (Lin, 1992; Mnih et al., 2015). During background planning, the agent samples from the replay buffer to improve sample efficiency by reusing experiences, trading off large storage space requirements (Sutton, 1991).

Although DQN popularized experience replay, it is not always considered an MBRL method. In DQN, experience replay is used in a model-free, off-policy Q-learning fashion (See Sec. 2.4.5, Page. 23 for more details). van Hasselt et al. (2019) unifies DQN with Dyna, where experience replay serves as a model that "generates" on-policy transitions. This makes DQN a special case of Dyna, with a model trained by storing transitions. From this unification, an experience generator can be seen as a learnable version of experience replay (van Seijen and Sutton, 2015; van Hasselt et al., 2019). Parametric experience generators, trained to reduce storage requirements, can generate samples that may improve generalization if the model parameterization supports it. However, in deep learning, these generators often struggle to produce high-quality samples due to model inaccuracies. As van Hasselt et al. (2019) notes, inaccurate transitions can introduce uncommon or impossible states, damaging the value estimator by promoting undesirable generalization. Observation-level generators are especially prone to this issue (Silver et al., 2017b; Schrittwieser et al., 2019). In contrast, experience replay avoids these risks but is limited to samples from seen trajectories.

Enhance State Representations with Extra Predictions

It has been shown that incorporating additional predictive models that can back-propagate into the state encoder can improve the quality of state representations. These extra predictions may be used during decision-time planning, or they may never be called upon. In the latter case, the predictive models exist solely to shape the state representation through training, effectively acting as an implicit regularization signal.
In the former case, the agent proposed in Chap. 4 enhances its object-oriented state representation encoder with predictive losses, including state dynamics, reward and termination predictions, as well as value estimation, with all predicted quantities being explicitly utilized during decision-time planning. In contrast, for the latter case, UNREAL uses reward and termination signal predictions to prevent the state representation from overfitting to value estimation, making it more aware of the environment's dynamics (Jaderberg et al., 2017).

Optimize Policy Directly with Differentiable Models

Differentiable models can play an important role in enabling the direct optimization of policies by providing a smooth, learnable approximation of the environment's dynamics. These models allow for the end-to-end optimization of both the policy and the model in a unified framework. By using differentiable dynamics models, gradient-based optimization techniques can be used to directly update the policy based on planning predictions. Differentiable models enable the use of techniques such as MPC within a differentiable framework, allowing for direct gradient updates to both the model and the policy, improving overall performance (Heess et al., 2015).

3.3.4 Inspiring Developments in Model-Based RL

We turn to more recent literature and survey the interesting progress for MBRL research related to this thesis, focusing on perspective methods seeking to address the innate difficulties of existing approaches from different perspectives.

Partial Planning

If the state space or feature space is large, then the expected next state or distribution over it can be difficult for models to estimate, as has been repeatedly shown (Alver et al., 2024).
In MBRL, **partial models** refer to models that predict partial aspects of the environment rather than capturing the full dynamics, which may be more complex or less relevant to the agent's decision-making process. For instance, partial models may predict only the immediate rewards associated with state-action pairs, or generate high-level abstractions such as goals or waypoints, as seen in frameworks like LEAP and Skipper. By simplifying the environment representation, partial models can help reduce the complexity of learning and planning, focusing the agent's attention on the most important aspects of the environment for its current objective. This approach can improve computational efficiency and allow for better generalization (Bengio, 2019).
Partial models resemble the idea of spatial abstraction we mentioned in Chap. 4 and Chap. 5.
Similarly, partial models can also reflect abstraction in the temporal dimension (Alver et al., 2024). For example, the model in Talvitie and Singh (2008) makes certain predictions based on certain few points in history (past observations) when triggered at a certain decision point (timestep). This idea predates the sequence modeling methods based on modern deep learning, which take advantage of the recent advances of deep learning to facilitate attention to history back-in-time.

Planning with Temporal Abstraction

When planning a long trajectory, searching at minuscule timescales is computationally demanding and in nature difficult, due to the exponentially growing search outcomes and imperfectness of the learned model.
An *option model* seeks to capture the dynamics given by a set of options inside an MDP. As pointed out in Sutton et al. (1999), planning with an option-model is preferred by its sample efficiency and the less exposure to the error accumulation problem of atomic timestep planning using an imperfect model. Though option models in real practice face the challenges brought by the discovery of options, the changing of option, *etc.*, I have proposed a solution as the Skipper framework in Chap. 5.

Directions of Planning

Our previous discussions on model behaviors focused on the forward simulation from certain states, whether from present, to the relative future. Somewhat counter-intuitively, van Hasselt et al. (2019) investigates benefits of using a *backward model*, *i.e.*, a Dyna transition generator which simulates a transition from the current state and the last action to the last state. When the backward model is still learning much, *i.e.*, model is still inaccurate, temporal credit assignment, *e.g.*, temporal-difference updates, will be conducted on fictional starting state with the target constructed from a true state, therefore lowering the chance of catastrophic delusional updates (van Hasselt et al., 2019). When the model is accurate enough, this would perform in the same way as we would expect from a good forward Dyna (van Hasselt et al., 2019). The concept of delusions in RL is heavily discussed in Chap. 6.

3.4 Other Related Background Knowledge

3.4.1 Consciousness in the First Sense & Conscious Planning

In this thesis, my collaborators and I frequently draw inspiration from human conscious planning when designing agent behaviors targeting the challenges of generalization. Note that this thesis does not claim to understand the exact mechanisms behind consciousness in the human brain. Instead, the methods presented in this thesis aim to endow agents with capabilities akin to conscious planning. In other words, we seek to design agents that behave similarly to conscious planning agents, leveraging consciousness-driven behaviors such as Out-Of-Distribution (OOD) generalization (Bengio, 2019).

In Dehaene et al. (2020), the authors outline conscious behaviors along two orthogonal axes. The first axis, corresponding to consciousness in the first sense, often abbreviated as C1, describes how information processing is selectively focused. When humans reason for decision-making, their attention is directed to the most relevant aspects of space and time, facilitating efficient generalization. This form of consciousness is often modeled by the global workspace hypothesis (Baars et al., 1997). The second axis, representing consciousness in the second sense, pertains to an agent's awareness of its own internal state (van Gulick, 2004). While this axis is intriguing and often receives significant attention, it is essential not to overlook the more fundamental axis of conscious behaviors (the first sense).

The top-down attention-based spatial abstraction mechanism proposed in Chap. 4 and used in Chap. 5 draws heavy inspirations from consciousness in the first sense.

More discussions regarding consciousness and its impact on decision-making can be found in Sec. 4.1 on Page. 58.

3.4.2 Generalization-Focused Environments & Experimental Settings

In existing works, the failure modes of RL agents are often overlooked, possibly due to a lack of access to environmental ground truth in benchmark environments. To address this, my collaborators and I developed several sets of environments aimed at evaluating the generalization capabilities of RL agents, specifically in the face of distributional shifts, compositional challenges, and OOD systematic generalization scenarios (Quiñonero-Candela et al., 2022).

Generalization-Focused Environments

The two most representative sets of these environments are `RandDistShift` and `SwordShieldMonster`, or simply `RDS` and `SSM`, which are the primary focus of later introductions. These environments were designed with the goal of testing how agents adapt to the challenges of generalization. Unlike traditional environments, however, `RDS` and `SSM` are environment factories that can generate specific instances of tasks based on our design specifications.

Both `RDS` and `SSM` are built upon the `DistShift`[4] environment from the MiniGrid repository (Chevalier-Boisvert et al., 2018, 2023), which itself is based on the work by Leike et al. (2017).

[4] https://minigrid.farama.org/environments/minigrid/DistShiftEnv/

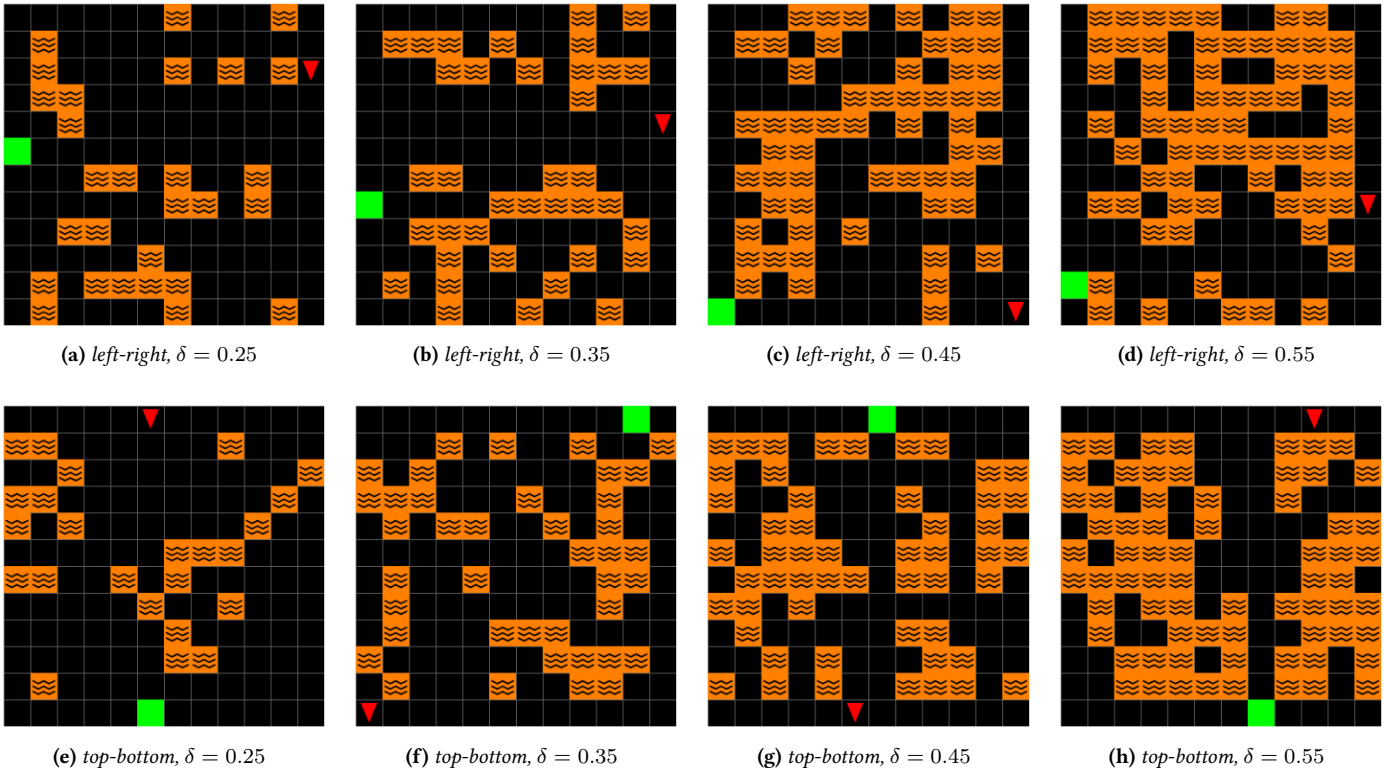
A significant advantage of using gridworlds as the foundation for these environments is that all instances are solvable by dynamic programming (DP), due to the simplicity of enumerating state-action pairs and constructing a transition table.
It is important to note that the challenges presented by these tasks are more complex than they may initially appear. For instance, the random placement of episode-terminating lava traps introduces compositionality challenges, while the abundance of terminal states complicates path planning. Certain locations in some environment instances become inaccessible to the agent, leading to problematic targets, as discussed in Chap. 6.
We deliberately made both `RDS` and `SSM` fully observable by modifying the original agent-centric views (which render the observation) to a top-down bird's-eye view of the entire gridworld. These fully observable tasks prioritize reasoning over causal mechanisms rather than learning representations from complicated observations.

RDS

`RDS` is a navigation task where the agent must navigate to the goal without stepping into the episode-terminating lava grids. A terminal sparse reward of $+1$ is given when the agent successfully gets to the goal grid.
In each `RDS` instance, the agent and the goal spawn at opposite edges of the field. Between these edges, lava is randomly placed according to a difficulty parameter, δ, which ensures that a viable path to the goal exists. The agent must navigate through this lava field to reach the goal and receive the terminal reward. When we constrain that the agent only spawns at the left or the right edge of the grid world, the corresponding `RDS` instances are out of the distribution of those instances generated by constraining the spawning at the top or bottom edges. We illustrate these in Fig. 3.6.
The evaluative task instances propose a systematic generalization challenge to the agents (Frank et al., 2009). Systematic generalization refers to the ability of a cognitive system (a human brain or a computational agent) to apply learned knowledge or rules to new, previously unseen situations in a consistent and structured manner.
One key feature of `RDS` instances is that all non-terminal states form a connected component. This means that, starting from any non-terminal state, the agent can always reach another non-terminal state within a limited number of steps.
Each `RDS` instance can be paired with three different action spaces, each defining a distinct transition kernel:

- **"Turn-or-Forward" dynamics**: Depending on the agent's facing direction, it can either turn clockwise, turn counterclockwise, or move forward to the grid in front.
- **"Absolute-Direction" dynamics**: The agent chooses to move to one of the four adjacent grids based on absolute directions, regardless of its facing direction.
- **"Turn-and-Forward" dynamics**: The agent chooses to turn clockwise, counterclockwise, or $180°$ and move forward, or simply step forward without turning.

While `RDS` instances are generally deterministic, randomness is occasionally introduced to test the agent's ability to handle stochasticity.
`SSM` was built on top of the features of `RDS`.



(a) *left-right*, $\delta = 0.25$ **(b)** *left-right*, $\delta = 0.35$ **(c)** *left-right*, $\delta = 0.45$ **(d)** *left-right*, $\delta = 0.55$

(e) *top-bottom*, $\delta = 0.25$ **(f)** *top-bottom*, $\delta = 0.35$ **(g)** *top-bottom*, $\delta = 0.45$ **(h)** *top-bottom*, $\delta = 0.55$

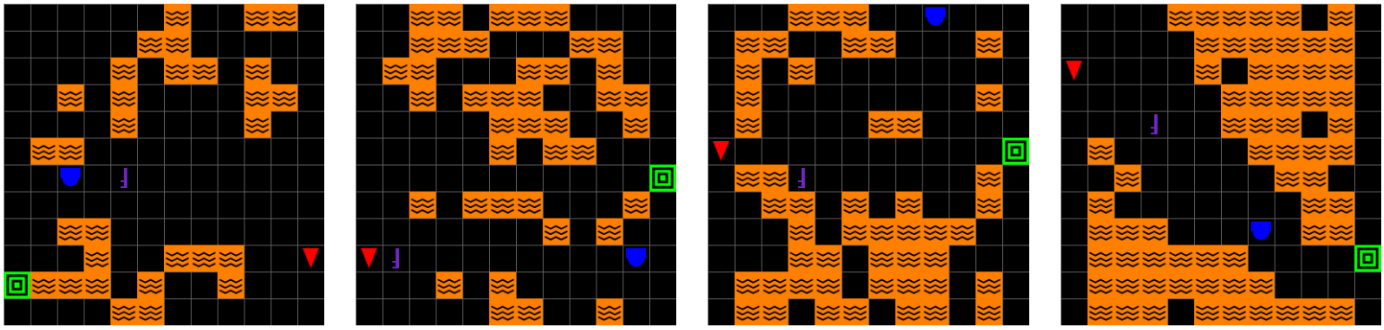
Figure 3.6: ***Left-Right & Top-Down RDS Instances****: We render observations of* `RDS` *instances of different orientation (left-right or top-down) and different difficulties* $\delta \in \{0.25, 0.35, 0.45, 0.55\}$*. The agent location and facing direction are rendered with a red triangle. The lava grids are rendered with orange boxes with waves. The goal locations are rendered as green squares. In Chap. 4, the agents are trained on left-right instances and evaluated on only top-down instances, which are outside the training distributions. Such experimental setting is how we evaluated the zero-shot OOD generalization abilities of the agent in Chap. 4.*

SSM

In `SSM`, the agent moves one step at a time in four absolute directions to navigate the field, implementing only the "Absolute Direction" dynamics. The agent must collect both a sword and a shield, which are randomly placed, before reaching the "monster" guarding the treasure. If the agent reaches the monster prematurely, the episode ends without reward. The paths to the sword, shield, and monster are guaranteed to exist in all environmental instances. The sword and shield are collected by moving to their respective grid cells, and once acquired, they cannot be dropped.

This design introduces temporary unreachability in the state structure, as non-terminal states are not fully traversable from one to another. Semantically, this segments the states of `SSM` into four situations, *i.e.*, equivalence classes, determined by two binary indicators: the agent's possession of the sword and shield. For example, $\langle 0, 1 \rangle$ denotes that the sword has not been acquired, but the shield has been acquired.

Visualizations of `SSM` instances are in Fig. 3.7. The state structure and the situations in `SSM` are visualized in Fig. 3.8. `SSM` was designed with the expectation to handle future work directions regarding abstract planning, as explained in Sec. 7.3 on Page. 143.



(a) $\delta = 0.25$ **(b)** $\delta = 0.35$ **(c)** $\delta = 0.45$ **(d)** $\delta = 0.55$

Figure 3.7: *SSM Instances*: *We render observations of* `SSM` *instances of difficulties* $\delta \in \{0.25, 0.35, 0.45, 0.55\}$. *The renderings of the agent location and the lava grids are the same as in* `RDS`. *While, the locations of the newly introduced sword and shield are rendered as a purple rapier shape and a blue heater shield shape, respectively. The monster location is marked with green-black loops.* `SSM` *was introduced in Chap. 6 (Page. 112) to expose target-assisted planning agents' failure modes regarding problematic targets. In Sec. 6.6 (Page. 125), the agents are trained only on instances with* $\delta = 0.4$ *and then evaluated on a range of OOD difficulties* $\delta \in \{0.25, 0.35, 0.45, 0.55\}$. *Such experimental setting is how we evaluated the zero-shot OOD generalization abilities of the agent in both Chap. 5 & Chap. 6. In Chap. 5, only* `RDS` *is used. While in Chap. 6, both* `RDS` *and* `SSM` *are used.*

Generalization-Focused Experimental Settings

We designed our customized MiniGrid-based environments with the intention to use them in a generalization-focused experimental setting. In this setting, agents are trained in a gridworld with one or a limited number of layouts and then tested on unseen layouts. This setup, referred to as `DistShift`, is designed to evaluate how well agents generalize to different tasks (Mendonca et al., 2020). This setting not only reflects how RL may be applied in real-world scenarios, but also tests the agents' abilities to generalize via *understanding* instead of *memorization*. I provide a brief summary of the experimental settings used in the main thesis chapters in Tab. 3.1.

It is similar to the "train-on-few then test-on-unseen" approach used in the non-curriculum experiments in ProcGen (Cobbe et al., 2020), which focuses on the challenges of learning representations from image-based observations. However, unlike ProcGen, `RDS` and `SSM` utilize simple observations to minimize the challenges of representation learning, instead focusing on the inherent difficulty of the tasks themselves. This deliberate design choice isolates the complexities of decision-making and generalization, which are the central focus of this thesis, from the challenges posed by representation learning.

Note that, despite the focus on generalization, our experimental setting is considered a **transient** learning setting with its limitations, instead of a *continual* learning setting that would be necessary to achieve more general intelligence.

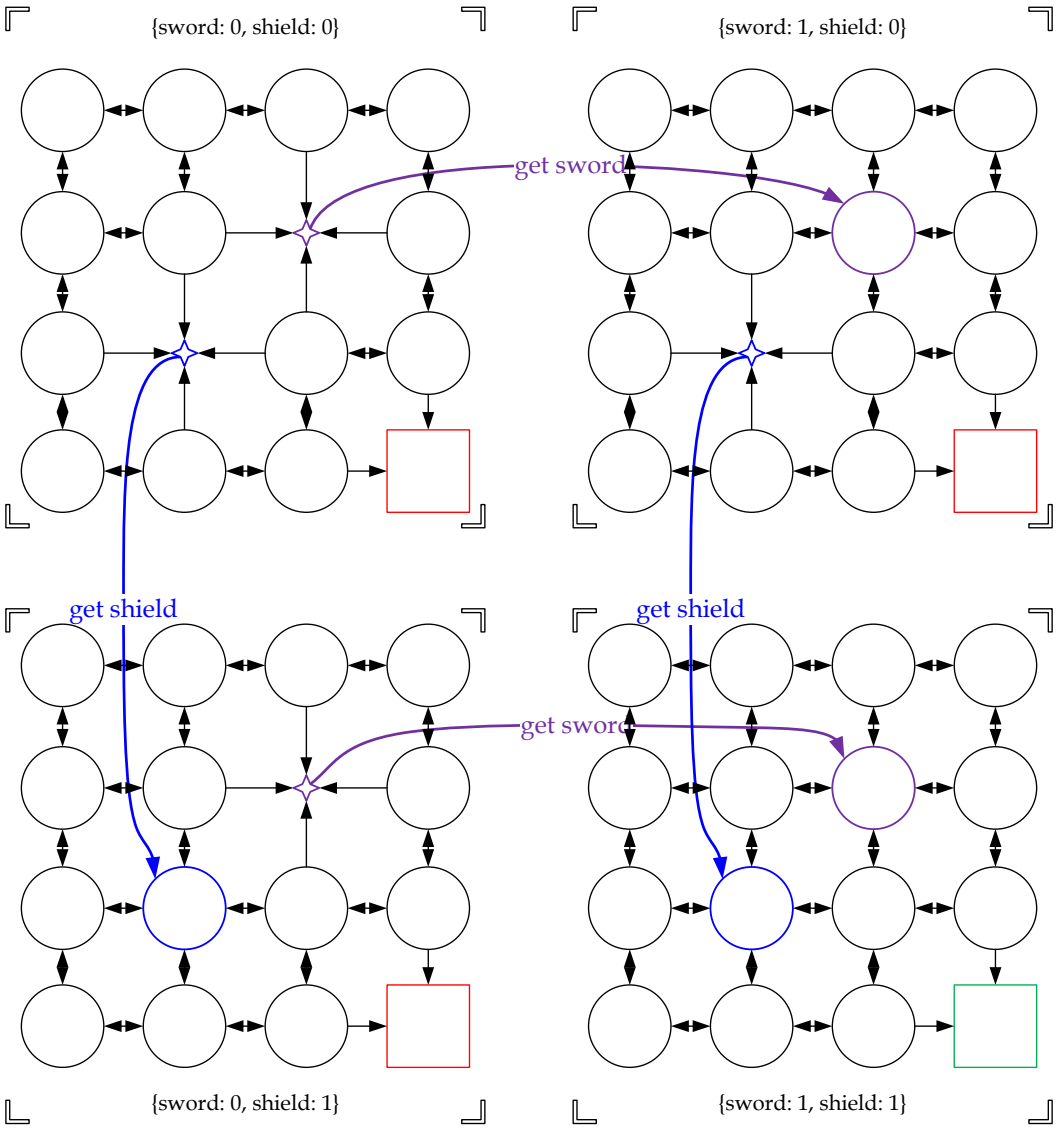


Figure 3.8: ***Visualization of SSM State Structure****: we analyze an instance of* 4×4 *SSM task instance with difficulty* $\delta = 0$*, i.e., no lava traps generated, approximately* 4 *times as large as a corresponding RDS instance. It can be seen that intra-situation transitions are only facilitated with pivotal transitions that involve the acquisition of the sword or the shield. The only possible situation transitions are* $\langle 0, 0 \rangle$ *to* $\langle 0, 1 \rangle$*,* $\langle 0, 0 \rangle$ *to* $\langle 1, 0 \rangle$*,* $\langle 1, 0 \rangle$ *to* $\langle 1, 1 \rangle$*, and* $\langle 0, 1 \rangle$ *to* $\langle 1, 1 \rangle$*. To effectively solve an unseen SSM instance, an agent needs to understand the environment, make plans based on its location and the world layout, deciding if it is the sword or the shield it should be getting first, before success.*

Setting	RDS	SSM	Training	Evaluation
Chap. 4	Used (sizes from 6×6 to 10×10, 3 action spaces & variants)	Not used	Train with ∞ environment instances with $\delta = 0.35$ (2.5×10^6 interactions)	Evaluate with *transposed* instances with δ from $\{0.25, 0.35, 0.45, 0.55\}$
Chap. 5	Used (size 12×12)	Not used	Train with $\{1, 5, 25, 50, 100, \infty\}$ environment instances with $\delta = 0.4$, random initialization (1.5×10^6 interactions)	Evaluate with δ from $\{0.25, 0.35, 0.45, 0.55\}$
Chap. 6	Used (size 12×12)	*Used* (sizes 8×8 & 12×12)	Train with 50 environment instances with $\delta = 0.4$, random initialization (1.5×10^6 interactions)	Evaluate with δ from $\{0.25, 0.35, 0.45, 0.55\}$

Table 3.1: ***OOD-Focused Experiment Settings for Different Chapters****: The experimental settings have evolved throughout the course of my PhD, I will introduce them in more detail in later chapters where they are used. In Chap. 4, we explored the most diverse set of experiments, ranging from transposed instances to varying world sizes. In Chap. 5, we adjusted the training difficulty to $\delta = 0.4$ to be OOD from all evaluation, and removed the transposed instances to better capture the generalization gap and align with the notion of distribution shifts (Soulos et al., 2024). We accelerated training by granting the agents a uniformly random initialization (in terms of their initial locations in the gridworlds). Furthermore, we limited the action space to the "absolute-direction" dynamics. We also mostly limited the number of training environments to reflect limited training scenarios in the real world. We tested Skipper's performance scalability in terms of the number of training environments. In Chap. 6, we converged at an OOD-focused setting, where the generalization gap can be intuitively visualized, while introducing the new* `SSM` *environments.*

Chapter 4

Conscious Planning: Spatially-Abstract Decision-Time Reasoning

Contents

6.1 Overview of This Thesis Milestone . . . 112

6.2 Methodology: Target Evaluation & Relabeling . . . 114

6.2.1 Target-Assisted Planning & Targets . . . 114

6.2.2 Source-Target Pairs & Hindsight Relabeling . . . 116

6.3 Methodology: Understanding Hallucinated State Targets . . . 116

6.3.1 G.0: ∞-Feasible . . . 116

6.3.2 G.1 - Permanently Infeasible (Hallucinated) . . . 117

6.3.3 G.2 - Temporarily Infeasible (Hallucinated) . . . 117

6.3.4 Examples . . . 117

6.3.5 Non-Singleton Targets . . . 118

6.4 Methodology: Evaluating Targets Correctly and Robustly . . . 118

6.4.1 Desiderata for Evaluator . . . 119

6.4.2 Learning Rule & Architecture for Feasibility . . . 119

6.4.3 Training Data for Feasibility . . . 120

6.4.4 Computational Overhead . . . 122

6.5 Discussions of Methodologies . . . 123

6.5.1 About TAP Agents . . . 123

6.5.2 About Hindsight Relabeling . . . 124

6.5.3 About Delusions & Delusional Behaviors . . . 124

6.5.4 About Goal Mis-Generalization . . . 125

6.6 Research Findings: Experiments . . . 125

6.6.1 Rejecting Infeasible Goals in Decision-Time Planning (Exp. 1/8 - 4/8) . . . 126

6.6.2 Rejecting Destabilizing Updates in Background Planning (Exp. 5/8 - Exp. 6/8) 133
6.6.3 Feasibility Convergence to Non-Singleton Targets (Exp. 7/8 & 8/8) 134
6.7 Summary 135

4.1 Overview of This Thesis Milestone

We present an end-to-end, model-based deep RL agent that dynamically attends to relevant parts of the environment during its decision-time planning. The agent uses a top-down attention based bottleneck mechanism over a set-based (object-oriented) state representation to limit the number of entities considered during planning. In experiments, we investigate the bottleneck-equipped agent with several sets of customized environments featuring different challenges. We consistently observe that the design allows the planning agents to better generalize their learned task-solving abilities in compatible unseen environments by attending to the relevant objects, leading to better out-of-distribution generalization performance.

Whether when humans plan our paths home from the offices or from a hotel to an airport, we typically focus on a small subset of relevant variables, *e.g.*, the change in position or the presence of traffic. An interesting hypothesis of how this path planning skill generalizes across scenarios is that it is based on computation associated with the conscious processing of information (Baars, 1993). Conscious attention focuses on a few necessary environment elements, with the help of an internal abstract representation of the world (van Gulick, 2004). This behavioral pattern, also known as consciousness in the 1st sense (C1) (Dehaene et al., 2020), was theorized to be the foundation of humans' exceptional adaptability and learning efficiency (Baars, 2002). A central characterization of conscious processing is that it involves a computational **bottleneck**, which forces the human brain to handle dependencies among only limited few environmental characteristics at a time. Though focusing on a subset of the available information may seem limited, such limitation counterintuitively facilitates generalization abilities to situations where the ignored variables are different and irrelevant (Bengio, 2019; Goyal and Bengio, 2022).

In this project, we (the collaborators and I) introduce these ideas into RL, as most of the big successes of RL have been obtained by deep, model-free agents (Mnih et al., 2015; Silver et al., 2016, 2017a). While model-based RL has generated significant research due to the potentials of using an extra model (Moerland et al., 2022), its empirical performance has typically lagged behind, with some recent notable exceptions (Schrittwieser et al., 2019; Łukasz Kaiser et al., 2020; Hafner et al., 2021).

We take inspiration from human conscious planning behaviors to build set-based (object-oriented) neural network architectures which learn a useful state representation and in which attention can be focused on a small set of variables at decision-time, where the aspect of "partial planning" (see Sec. 3.3.4 on Page. 50) is enabled by modern deep RL techniques (Talvitie and Singh, 2008; Khetarpal et al., 2021). Specifically, we propose an end-to-end latent-space model-based RL agent which does not require reconstructing the observations, as in most existing works, and uses Model Predictive Control (MPC) for decision-time planning (Richards, 2005; Rao, 2009), which we introduced in Sec. 3.3.2 on Page. 43. From an observation, the agent encodes a set of objects as a state, with an attention bottleneck mechanism to **reason** over selected subsets of the state.

Our experiments show that the introduced inductive biases improve a systematic OOD generalization, where consistent environmental dynamics are preserved across seemingly different tasks.

4.2 Discussions of Methodologies

Much of the background and context for this project have already been covered in Chap. 2 and Chap. 3. We now focus on aspects specific to this project that will help explain the rationale behind our design choices.

4.2.1 About Reconstruction, Observation-Level & State-Level Planning

Many model-based RL methods utilize models that operate in the observation space or over state representations obtained with reconstruction-based losses (Wang et al., 2018b; Łukasz Kaiser et al., 2020). Appropriate as these models might be for tasks with few sensory inputs, *e.g.*, continuous control of robots with joint states, these approaches encounter difficulties with high-dimensional inputs such as images, because they must focus on predictable yet potentially useless aspects of the environmental observations (Moerland et al., 2022). Besides the need to reconstruct irrelevant parts of the environment, it is unclear if representations built by a reconstruction loss (*e.g.*, L_2 in the observation space) are effective for an model-based RL agent to plan or produce desired behaviors at all (Silver et al., 2017b; Hafner et al., 2021; Hamrick et al., 2021), *e.g.*, values, rewards, *etc.*. Taking a similar approach to those in Silver et al. (2017b); Schrittwieser et al. (2019), in this chapter, we build a latent space representation jointly shaped by all the training signals without using reconstruction (to serve value estimation and planning). We briefly discussed this topic previously in Sec. 3.3.3 on Page. 47.

4.2.2 About Staged Training with World Models

World Models are a popular methodology for model-based RL agents (Ha and Schmidhuber, 2018; Łukasz Kaiser et al., 2020; Moerland et al., 2022). These models require two explicit stages of training: 1) an representation of the environment is trained using exploration, sometimes in an unsupervised fashion; 2) the representation is fixed and used for planning and model-based RL. World models are favored for the lack of distributional shift and their adjacency to supervised deep learning methods. However, despite these advantages, the world model methodology relies on the implicit assumption that the environments are highly homogeneous, *i.e.*, the trained model that could handle the experiences from the exploration stage can handle the more rewarding trajectories as well. This is obviously not true in many environments, and the applicability of such method deteriorates rapidly with the increment of exploration difficulties. Such implicit assumption also demonstrates the limited OOD generalization abilities of such approach and indicate that world models may not be robust enough to generalize in novel situations. This can be viewed from a transient learning *v.s.* continual learning perspective, discussed in Sec. 3.4.2 on Page. 54. In certain existing literature, world models are also understood to be the methodology of building predictive models based on the actions taken by the agents. We do not use such aspects when we discuss the world models.
Furthermore, despite that the learned representations of a world model are expected to be effective in predicting the evolution of environmental states, they should not be expected to be effective for value estimation

at all. This is because of the properties of neural networks: without a value estimation training signal, the learned representation would typically not be effective for value estimation; End-to-end model-based RL agents, *e.g.*, Silver et al. (2017b); Schrittwieser et al. (2019), are able to learn the representation of the world simultaneously with the value estimator, hence have the potential to adapt better to non-stationarity in the control tasks.

4.2.3 About Vectorized *v.s.* Set-based State Representations

Most existing deep RL methods employ vectorized state representations, where the agents' observation is encoded into a vector of fixed dimensionality of features (Mnih et al., 2015; Hessel et al., 2018). while, set-based encoders, *a.k.a.* object-oriented architectures, are designed to extract a set of unordered vectors (often named **objects** in such context), from which the desired signals can be predicted via permutation-invariant computations, which is extensively investigated in Zaheer et al. (2017); Lee et al. (2019). We illustrate the difference between the two approaches in Fig. 4.1. In recent years, the potential of set-based representations are discovered for RL, in terms of effectively building environmental state representations, in terms of compositional generalization (Wang et al., 2018a; Vinyals et al., 2019; Davidson and Lake, 2020; Mu et al., 2020; Löwe et al., 2020). In this chapter, we exploit the compositionality of these set-based state representations to enable the discovery of sparse interactions patterns among the encoded objects, as well as to facilitate the core bottleneck mechanism, analogous to the dynamic selection empowered by consciousness in the 1st sense (C1). The combination of the set-based representation and the bottleneck provides a set of inductive biases consistent with dynamically selecting only the relevant aspects of the environmental state through the proposed attention mechanism. The sparsity of the dependencies captured by the learned dynamics model is enforced by the small size of the working memory bottleneck: each transition can only relate a few objects together, no more than the size of the bottleneck (Bengio, 2019).

4.3 Methodology: Model-based RL with Set Representations

We present a baseline end-to-end agent, which uses a set-based representation and carries out latent space decision-time planning, but **without** a consciousness-inspired small bottleneck (Alver and Precup, 2024). This agent serves as a baseline to investigate the OOD generalization capabilities brought by the bottleneck, which is to be introduced in Sec. 4.4 on Page. 65.

As discussed previously, a model-free agent (or a model-free part of a model-based agent) relies on the mapping from observations to values, which is in turn a combination of an **state representation encoder** (encoder for short) and a **value estimator**. Aiming at a set-based representation, we designed the encoder to map an observation vector to a set of objects. Also, the value estimator is designed to be a permutation-invariant set-to-vector neural network, mapping the learned state representation to a value estimate. With our design, to maximize the gain from end-to-end learning, the same state representation is shared for all the agents' predictions, including the dynamics model's prediction of future states, rewards, *etc.*, to be discussed soon.

4.3.1 State Representation Encoder

We propose a neural network architecture targeting image-based observations. For this, we "slice" the CNN-output feature maps each position to characterize the feature of an object, similar to the approach in Carion et al. (2020), as shown in Fig. 4.1.

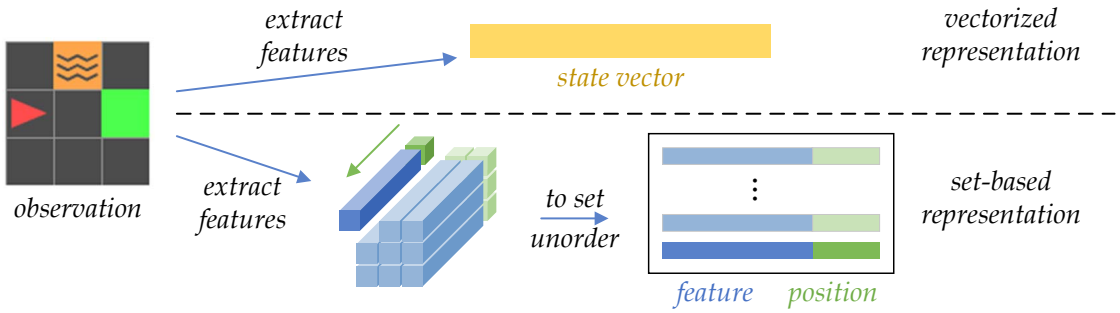


Figure 4.1: ***Set-based State Representation Encoder****: Compared to vectorized encoders, the feature map extracted by some feature extractor, e.g., a CNN, is "sliced" into individual feature vectors and concatenated with positional information. Note that such concatenation is a design choice that serves the dynamics model training purposes. Post concatenation, all resulting vectors are treated as objects, indicating all entities an agent could perceive given the current observation.*

When a 3-dimensional CNN feature tensor is sliced into an unordered set of 1-dimensional feature vectors, the positional information of each feature will be lost. To recover such lost information, we concatenate each obtained feature vector with a positional embedding. This method is different from the common practice of mixing positional information by addition, as often used in transformer architectures (Vaswani et al., 2017). This particular design aims to address a challenge in training a dynamics model with set-based representations, which is to be discussed in Sec. 4.3.3 on Page. 61.

4.3.2 (State-Action) Q-Value Estimator

This Q-value estimator is implemented in the form $Q : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$, where $\mathcal{S}$ is the *learned* state representation space by the previously introduced set-based encoder and $\mathcal{A}$ is a discrete action set. The architecture we propose for this component uses an improved architecture upon DeepSets (Zaheer et al., 2017) and Set Transformer (Lee et al., 2019), as shown in Fig. 4.2. This architecture performs reasoning based on the relationship among the encoded objects of the input set, resembling token-based methods in natural language processing (Porada et al., 2021).

4.3.3 Transition Model

We design a set-to-set transition model that maps from s_t, a_t to $\hat{s}_{t+1}$, $\hat{r}_t$ and $\hat{\omega}_{t+1}$. The model is a sample model that predicts the outcome of a transition.

For clarity, we separate the computational flow of the transition model into: 1) the **dynamics model**, responsible for simulating how the state would evolve with an intended action a_t and 2) the **reward-termination estimator** which maps s_t, a_t to $\hat{r}_t$ and $\hat{\omega}_{t+1}$.

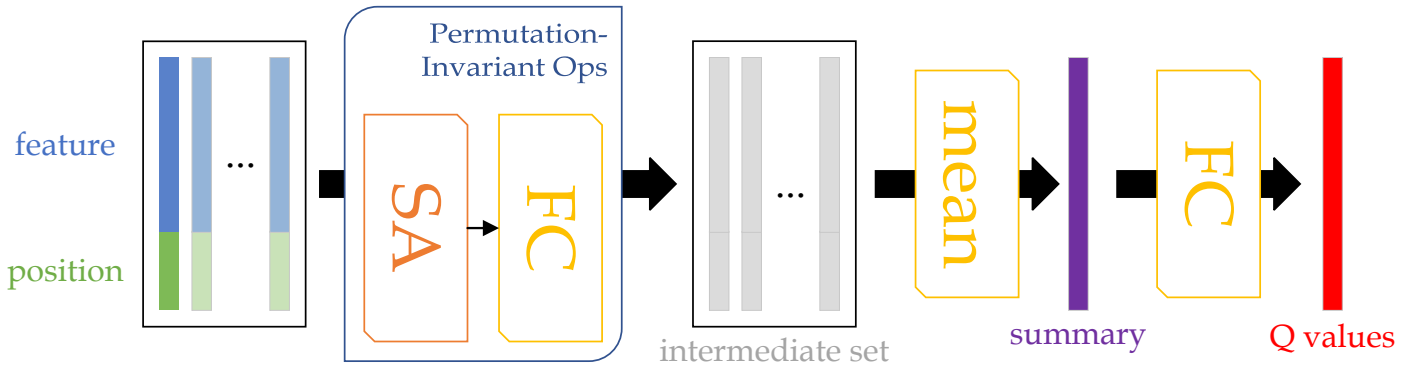


Figure 4.2: ***Value Estimator*** Q **&** ***Generic Set-to-Vec Architecture****: We improved upon the DeepSets and Set Transformer architectures (Zaheer et al., 2017; Lee et al., 2019) by swapping the Fully Connected (FC) Multi-Layer Perceptron (MLP) before pooling with transformer layers (Self-Attention (SA) + object-wise Fully Connected (FC)) (Vaswani et al., 2017). After passing through the transformer layers, the intermediate set (gray) will entangle features and positions, i.e., the feature and the positional information of each object no longer have their own separate dimensions.*

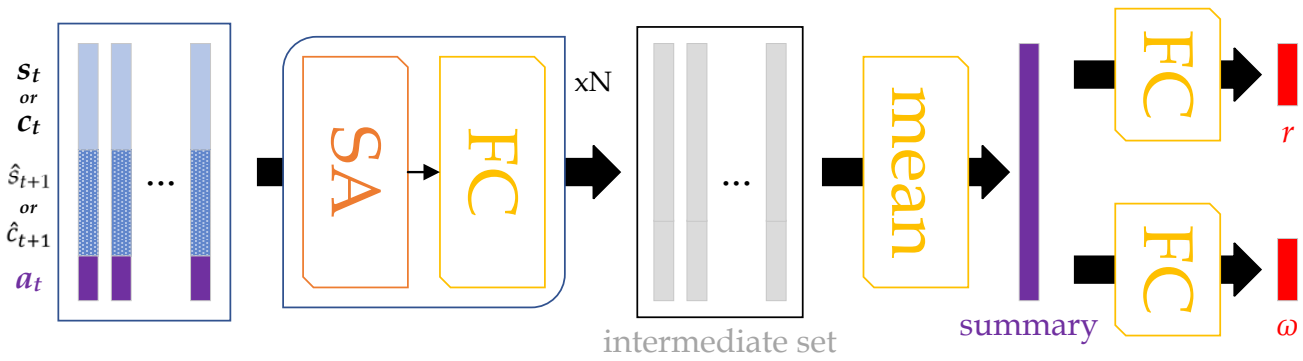


Figure 4.3: ***Reward-Termination Estimator based on Set Transitions****: The computational flow requires* 3 *inputs. The* 1^{st} *input is either the full set* s_t *for agents without the bottleneck, or bottleneck set* c_t*, for the bottlenecked agents (from the selection). The* 2^{nd} *input is the respective output of the dynamics model for the next timestep, i.e., the predicted full set* $\hat{s}_{t+1}$ *or* $\hat{c}_{t+1}$*. The* 3^{rd} *input is the embedding of the action that governs the simulated transition, i.e., the intention of the agent in the search step. The first two inputs are aligned and concatenated with the tiled action embeddings to form an augmented set, as shown to the left of the figure. The augmented set is then passed through the transformer layers to form an intermediate set, which is then mean-pooled and projected to the two predictions. With deterministic environments, it will be sufficient to predict the reward and termination with only* s_t *and* a_t*.*

While designing a reward-termination estimator should be straightforward (a 2-headed augmented architecture similar to the value estimator, as shown in Fig. 4.3), the dynamics model requires predicting changes on sets of *unordered* objects (set-to-set). However, since the position and the feature of each object can change at the same time, the predicted set of objects may not align with their counterparts from the input set trivially. These additional degrees of freedom raise the computational difficulties of set-to-set predictions, often rendering the learning process ineffective. To address such challenge, in existing deep learning literature, a common approach is to use alignment methods, *e.g.*, aligned loss functions such as Hausdorff distance or Chamfer matching. Nevertheless, these approaches are computationally inefficient and subject to local optima (Barrow et al., 1977; Borgefors, 1988; Kosiorek et al., 2020). With these drawbacks in mind, we deliberately separated the dimensions of feature and position in our objects: this design not only make the permutation-invariant computations position-aware, but also allow surprisingly simple end-to-end training losses over the dynamics, similar to those used for agents based on vectorized state representations. This is made possible by forcing the dimensions of positional information, *i.e.*, the "position tails", to stay unchanged during the dynamics prediction. Such anchor addresses the difficulty of alignment by design: the degree of freedom of changing positional tails is no longer allowed. Objects "labeled" with the same positional information in the output of the dynamics model $\hat{s}_{t+1}$ and the training sample input s_{t+1} must be aligned, forming pairs of objects with changes solely in the feature dimensions. This is further explained in Fig. 4.4.

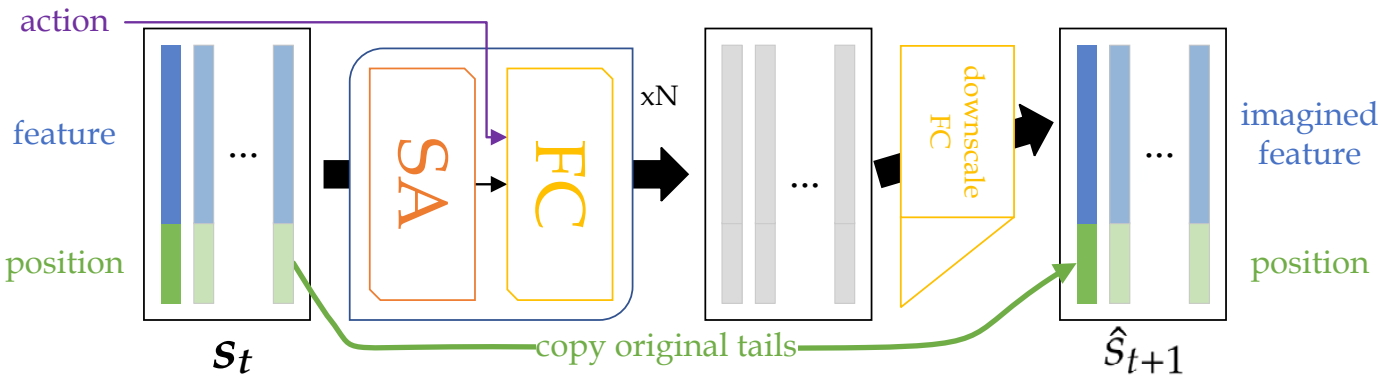


Figure 4.4: ***Set-Based Dynamics Model****: This dynamics model architecture takes state set s_t and action embedding a_t as input and outputs the imagined next state $\hat{s}_{t+1}$. For the FC sub-layers of the transformer layers, we inject an action embedding s.t. the transformer computations become action-conditioned. This process is illustrated in detail in Fig. 4.5. After getting the intermediate set, we use linear projections to downscale each object to the correct dimensionality, while forcing the positions untouched and directly copied from the input s_t. Note that though the objects in the sets (input-intermediate-output) are inter-aligned, within each set they are still unordered, i.e., permutation-invariant.*

The action-conditioned transformer layer serves as the backbone of set-to-set learning in this chapter. A vanilla transformer layer consists of two consecutive sub-layers, the multi-head SA and the fully connected, each containing a residual pass. To make a vanilla transformer layer action-conditioned, we first embed the discrete actions into a vector and then concatenate it to every intermediate object output by the SA sub-layer. A detailed illustration of the action-conditioned transformer layer is provided in Fig. 4.5.

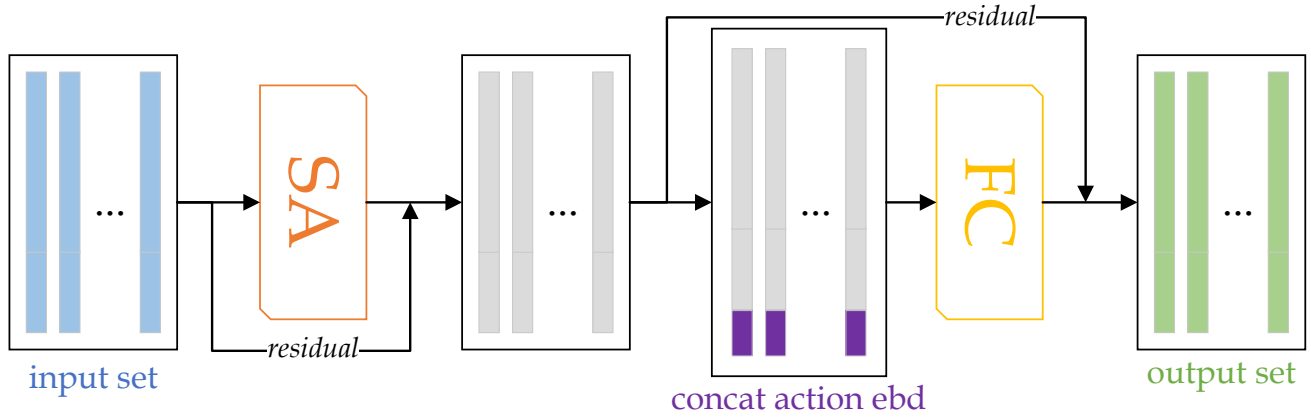


Figure 4.5: ***Action-Conditioned Transformer Layer****: compared to the vanilla transformer layers, we concatenate additionally the action embedding to every intermediate object embeddings in the FC sub-layers of transformer layers. The FC sub-layer implements* $X' = X + f(cat[X, a])$*, where* X *is the input set and* $cat([X, a]])$ *is the concatenation of action embedding* $\boldsymbol{a}$ *to every object embedding in* X*, with* X' *being the output set.* f *needs to correctly downscale the dimensionality of its input to match that of* X*. Layer normalization layers are omitted for simplicity.*

4.3.4 Training

The baseline model-based agent is trained the following losses (over sampled transitions):

- Value Estimation $\mathcal{L}_{\text{TD}}$: push the current value estimates towards the update targets. These targets can be acquired with techniques such as Double DQN (DDQN) (Hasselt et al., 2016). In experiments, C51-style distributional architectures as used for all scalar predictions including values and rewards, making $\mathcal{L}_{\text{TD}}$ a KL-divergence (Bellemare et al., 2017).

- Dynamics Prediction $\mathcal{L}_{\text{dyn}}$: A L_2 penalty established between the aligned $\hat{s}_{t+1}$ and s_{t+1}, where $sg(\hat{s}_{t+1})$ is the imagined next state given o_t, a_t and s_{t+1} is the true next state encoded from o_{t+1}. The stop-gradient function sg indicates that the gradient updates will treat $\hat{s}_{t+1})$ as a constant tensor and do not compute the associated partial derivatives. This loss is made possible by our separate-dimension design of feature-position object representations.

- Reward Prediction $\mathcal{L}_r$: given the C51 architecture, the KL-divergence between the imagined reward $\hat{r}_{t+1}$ predicted by the model and the true reward r_{t+1} of the observed transition.

- Termination Prediction $\mathcal{L}_\omega$: this is an optional binary cross-entropy loss from the imagined termination $\hat{\omega}_{t+1}$ to the ground truth ω_{t+1}, obtained from environment feedback.

The overall loss for end-to-end training of this set-based model-based baseline agent is thus:

$$\mathcal{L} = \mathcal{L}_{\text{TD}} + \mathcal{L}_{\text{dyn}} + \mathcal{L}_r + \mathcal{L}_\omega$$

As discussed earlier, jointly shaping the state representations avoids the collapse to trivial solutions and makes the representations useful for predicting all signals of interest.

In our implementation, no re-weighting is used for each loss term. This simplicity is possible because most individual losses are entropy-like and thus similar in magnitudes.

4.4 Methodology: Consciousness-Inspired Bottleneck

Planning should focus on the relevant parts of the environment that matter the most for the intended plan. In this section, we introduce an inductive bias which facilitates C1-capable planning, which can be applied to the baseline agent introduced above. Such inductive bias is built on the observation that, with each action taken, only a limited few aspects of the environment should change, as in the real world.

From the input of a full state set s_t, we aim to capture all the relevant aspects of the state in a *small* bottleneck set, which is expected to *contain all the important transition-related information*. With such bottleneck set c_t, we can perform predictions about the change in the environmental state, the immediate reward, *etc.*. To be more precise, as illustrated in Fig. 4.6, the model performs 1) selection of the bottleneck set from the full state-set (to acquire c_t), 2) dynamics simulation on the bottleneck set (from c_t to $\hat{c}_{t+1}$) and 3) integration of predicted bottleneck set to form the predicted next state (to acquire $\hat{s}_{t+1}$).

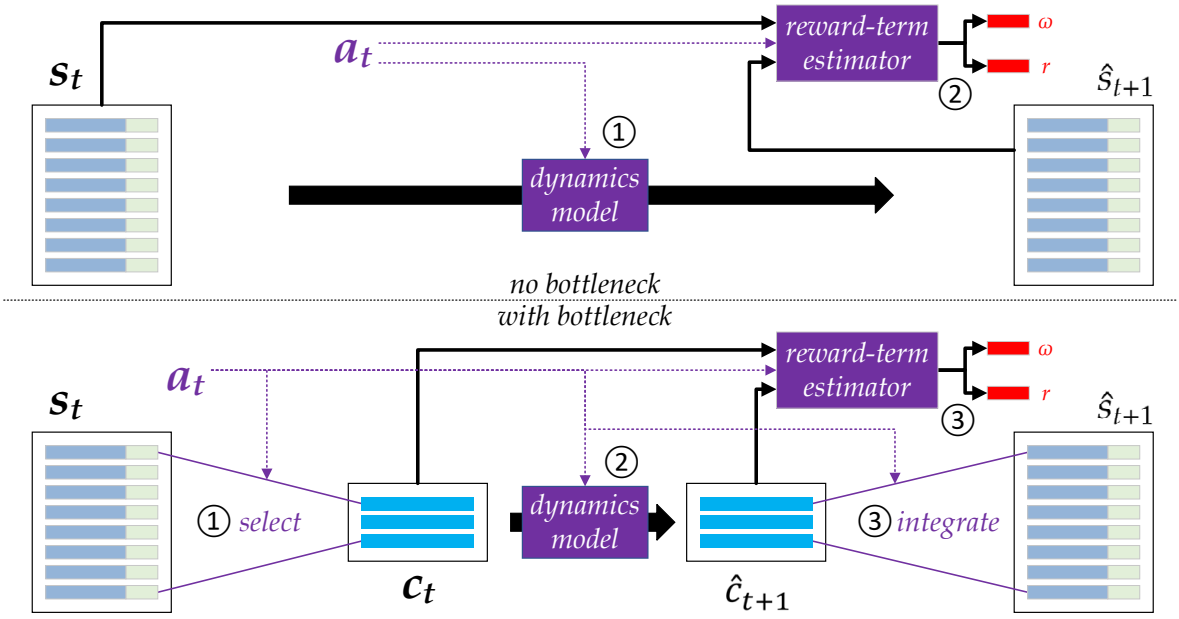


Figure 4.6: ***Stages of Bottleneck-Enabled Set-Based Dynamics Model****: Similar to the baseline model illustrated in Fig. 4.4, this component also takes the state set s_t and the action embedding a_t as input, and outputs the predicted next state $\hat{s}_{t+1}$, together with the transitional reward r and the termination signal ω. However, it produces some very different intermediate predictions / representations. 1) a bottleneck set c_t is soft-selected from the whole state set s_t via semi-hard top-down attention, conditioned on the intended action. Details of this operation are illustrated in Fig. 4.7; 2) dynamics are applied to the bottleneck set c_t to form $\hat{c}_{t+1}$. This part is implemented with the same architecture presented in Fig. 4.5; 3) similarly, the reward and termination signals are predicted out of c_t, $\hat{c}_{t+1}$ and a_t, as depicted in Fig. 4.3. At the same time, the changes introduced in $\hat{c}_{t+1}$ are Integrated with s_t to obtain $\hat{s}_{t+1}$, the imagined next state, with an attention-like operation, as shown in Fig. 4.8. The two computational flows in stage 3 are naturally parallelizable.*

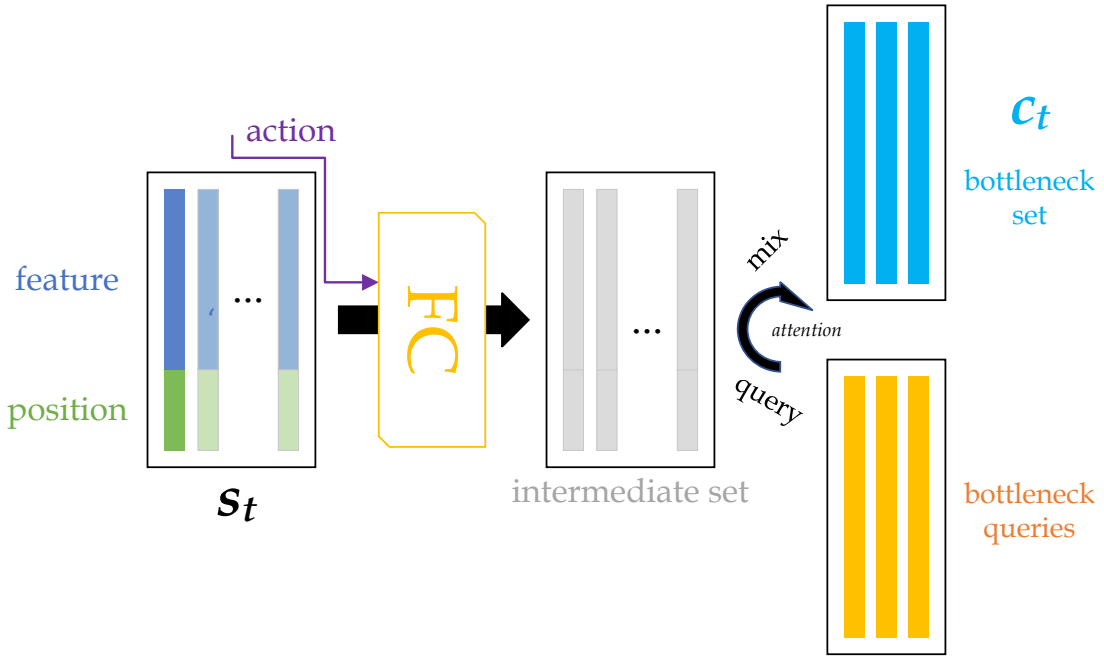


Figure 4.7: ***Design of the Bottleneck Selector****: the bottleneck set c_t is obtained by querying the (transformed) state set s_t, denoted as the intermediate set, with a learned query set of size k, denoted as the bottleneck queries, using a custom semi-hard attention (Gupta et al., 2021). The selection is conditioned on the chosen action. Note that, for this component, self-attention should NOT be used to transform s_t to create the value set (in terms of key-query-value) for the output, otherwise a mixture of all objects would be created, defeating the purpose of the bottleneck.*

4.4.1 Conditional State Selection

We soft-select a bottleneck set c_t of n objects from the potentially large input state set s_t of $m \gg n$ objects. Then we only model the transition for the selected objects in c_t. To implement this, we creatively implement an action-conditioned key-query-value attention mechanism, where the source of keys and values come from s_t and a_t, and the queries come from some learned dedicated set of vectors and of the action considered, as shown in Fig. 4.7.

To dodge the difficulties of a hard attention bottleneck, we use a semi-hard top-k attention mechanism to facilitate the selection of the bottleneck set. This semi-hard attention technique limits the influence of the irrelevant objects on the bottleneck set c_t while allowing for a gradient to propagate on the assignment of relative weight to different objects. With purely soft attention, weights for irrelevant objects are never 0 and learning to disentangle objects may be more difficult. Please refer to Sec. 3.2.2 on Page. 40 for more details regarding how we used semi-hard top-k attention to make sure that the output set c_t is a transformed subset of the state set s_t (the set being queried).

4.4.2 Dynamics / Reward-Termination Prediction on Bottleneck Sets

We use the same architecture as described in Sec. 4.3 (in Fig. 4.3), but taking the bottleneck objects as input rather than the full state set.

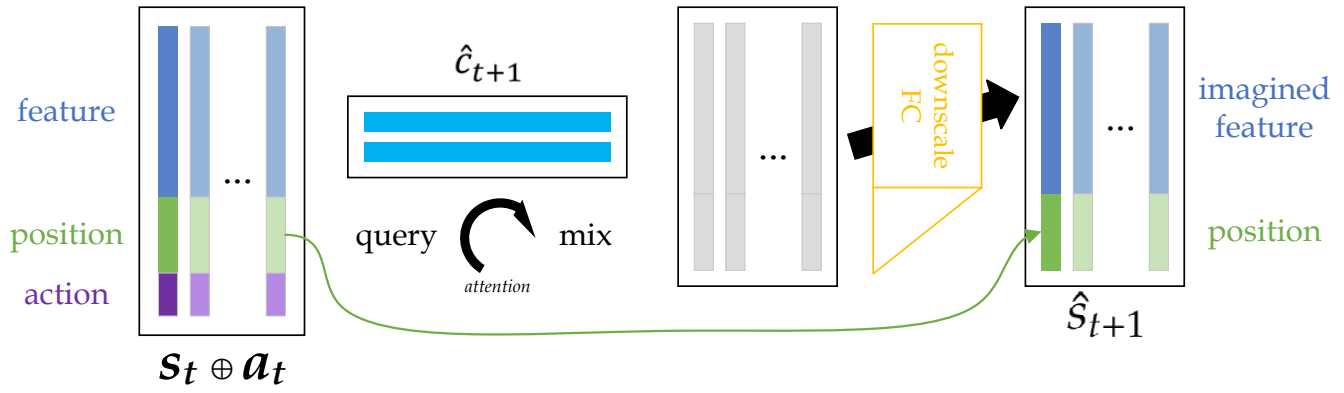


Figure 4.8: ***Design of the Bottleneck Integrator****: This component takes the 3 inputs, the state set s_t, the bottleneck set $\hat{c}_{t+1}$ and the action embedding a_t. The output $\hat{s}_{t+1}$ is generated by using the action-augmented s_t to query the imagined bottleneck set $\hat{c}_{t+1}$ with an attention-like operation without the* `softmax` *on attention weights. Note that there is the similar operation of downscaling objects to features and copying the positional tails. For more details of the query operation, please refer to Sec. 3.2.2 on Page. 37. Note that because there is no SA used to transform the full state set s_t, meaningful changes during $s_t \to s_{t+1}$ must be modeled by the query of the bottleneck set $\hat{c}_{t+1}$. Thus, the integrator cannot be used to bypass the bottleneck.*

4.4.3 Change Integration

The evolution on the bottleneck set should cover the change of the full state set. Thus, we implement an integration operation to 'soft paste-back' the changes of the bottleneck state onto the state set s_t, yielding the imagined next state set $\hat{s}_{t+1}$. Intuitively, this integration is somewhat similar to an inverse operation of selection process. This 'soft paste-back' is implemented with attention-like operations, more specifically querying $\hat{c}_{t+1}$ with s_t, conditioned on the action a_t. We emphasize attention-"like" here because, since certain objects in s_t should not interact with anything from the bottleneck set at all, we allow the attention weights to be 0 by forgoing the `softmax` operation when computing the attention weights. Details of the architecture regarding change integration are in Fig. 4.8.

4.4.4 Discussion

The proposed bottleneck is a natural complement to the baseline agent introduced previously. In particular, planning and training are carried out the same way as discussed in Sec. 4.3.

One may wonder if it is appropriate to have the value estimator share the bottleneck with the transition model. To estimate a value, the estimator needs to consider the future object interactions, which likely include the objects that do not matter for the current transition; Additionally, it is not necessary to implement an explicit bottleneck mechanism on the value estimator, since the mean-pooling operation functions as the selection if some objects are learned to be ignored.

We call the baseline agent equipped with the bottlenecked inductive biases in this section the Conscious Planning (CP) agent. We expect the CP agent to demonstrate the following advantages:

- More Effective Generalization: only relevant objects participate in each planning step, thus each planning step is handling a smaller scale, more manageable problem. Thus, generalization should be improved even in OOD scenarios, because the transitions do not depend on the parts of the state ignored by the bottleneck.
- Lower Computational Cost: directly employing transformers to simulate the full state dynamics results in a complexity of $\mathcal{O}(|s_t|^2 d)$, where d is the dimensionality of the objects, due to the use of self-attention, while the bottlenecked variants lowers the related parts to $\mathcal{O}(|s_t||c_t|d)$.

The partial predictions enabled by the bottleneck can be viewed from the perspectives of partial / local models. While, previous works have more emphasis on the selective attention towards points in history (temporal focus) (Talvitie and Singh, 2008), and our bottleneck-equipped model has the attention towards aspects within state and state transitions.

4.4.5 Birdseye View of Overall Design

We assemble the components and present the organization of the proposed CP agent in Fig. 4.9.

4.5 Research Findings: Experiments

We wish to empirically validate the advantages of generalization brought by the introduced consciousness-inspired bottleneck. This is to be done with rigorously controlled experiments and ablation studies.

4.5.1 Environment / Task Description

We use `RDS` as the backbone environment in the experiments of this chapter. Crucial for the experimental insights and isolating the unwanted technical difficulties such as learning from complicated visual observations, `RDS` tasks provide clear object definitions, with clear and intuitive dynamics based on object interactions. For more details regarding the environment, please check Sec. 3.4.2 on Page. 52.
For demonstration purposes, the main set of experiments we will be using in this chapter are conducted on 8×8-sized `RDS` instances paired with "turn-or-forward" dynamics. In the later parts, we will provide additional test settings with different dynamics and different world sizes (Sec. 4.5.7, Page. 77).
For better generalization, the agent needs to understand how to avoid lava in general (and not at specific locations, since their placement changes) and to reach the target locations as quickly as possible. For the agent to be able to *understand* the environment dynamics instead of *memorizing* specific task layouts, we generate a new environment for each episode (training or evaluation), facilitating essentially a multi-task setting. In each training episode, the agent starts at a random position on the leftmost or rightmost edge, and the goal is placed randomly somewhere on the opposite edge. Consistent with all `RDS` instances, the difficulty parameter δ controls partially how seemingly different the OOD evaluation tasks are to the in-distribution training tasks, though we know the underlying dynamics of all these tasks are consistent. For training episodes, the difficulty is fixed to $\delta = 0.35$[1].

[1]Based on `RDS`, this was the 1st OOD-focused experimental setting used in this thesis, The setting was later refined in Chap. 5 and Chap. 6.

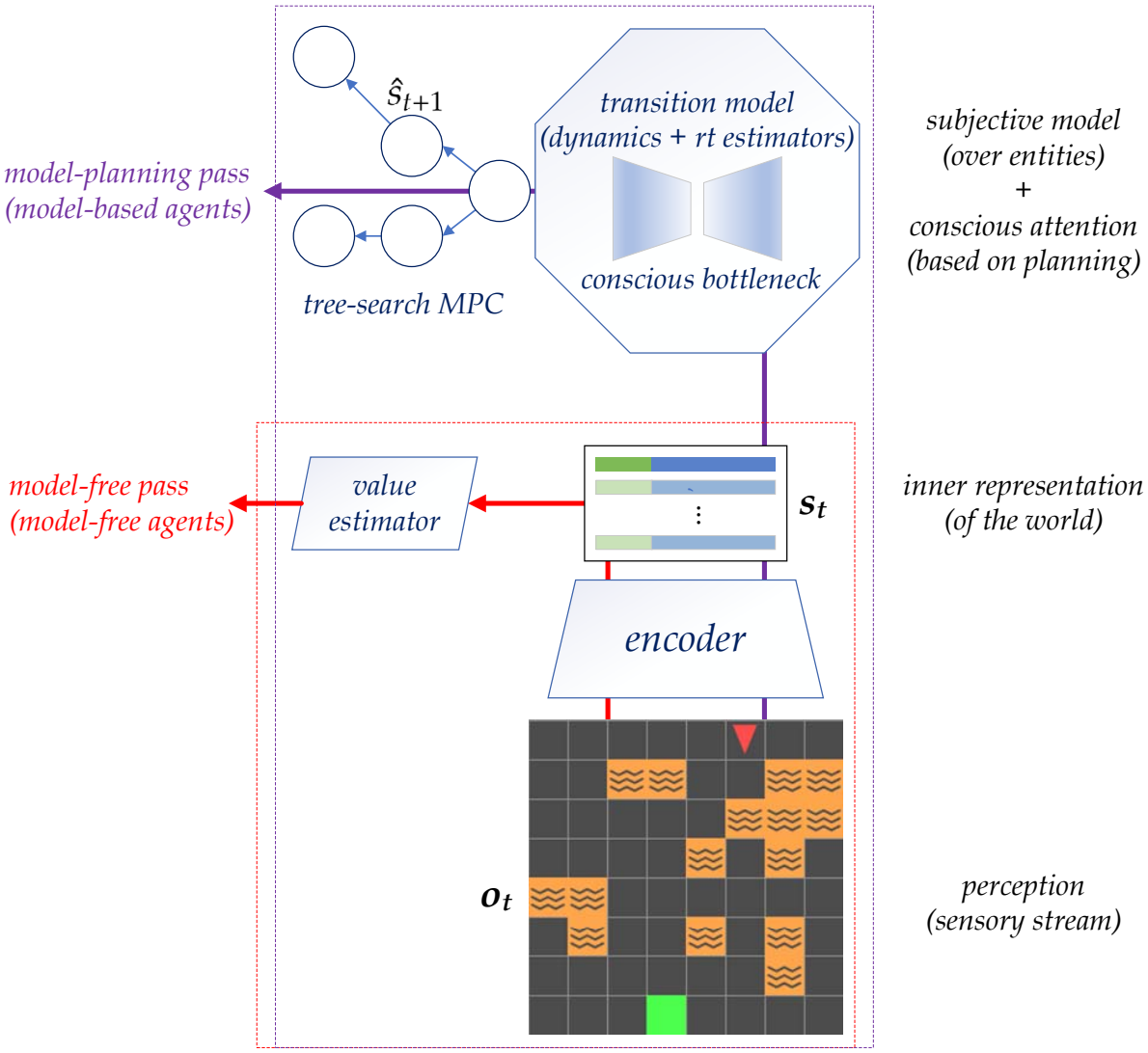
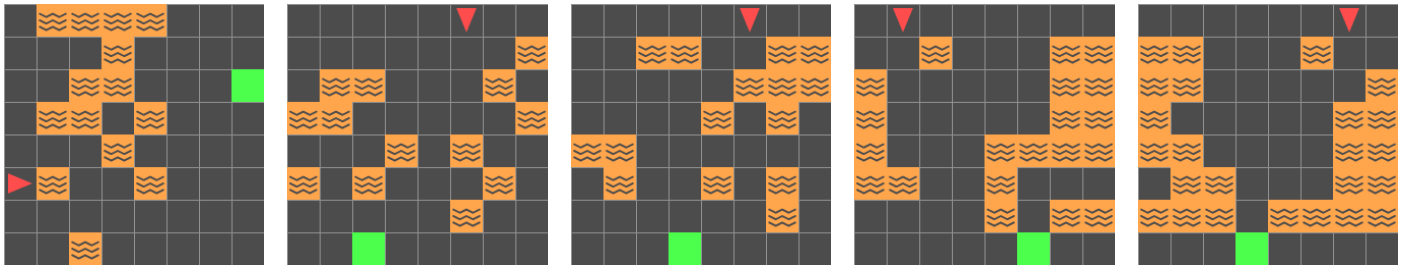


Figure 4.9: ***Overall Organization of Proposed Components for CP Agent****: Both the model-free baseline and the model-based baseline without the bottleneck can be extracted from the figure as well.*

To be specific, the OOD generalization we refer to in this chapter is *the agents' ability to generalize its learned task skills across seemingly different tasks with common underlying dynamics, i.e.*, systematic generalization (Frank et al., 2009). For OOD evaluation, the agent is expected to adapt to new tasks with consistent underlying dynamics in a 0-shot fashion, *i.e.*, with the agent's parameters fixed (Sylvain et al., 2020). In other words, the agent will be challenged with distribution shifts (Mendonca et al., 2020; Quiñonero-Candela et al., 2022). The OOD evaluation tasks are designed to challenge the trained agents. These tasks include changes both in the support (orientation) and in the distribution (difficulty): the agents are deployed in *transposed* layouts that they have never seen before with varying levels of difficulties ($\{0.25, \mathbf{0.35}, 0.45, 0.55\}$) that they were not trained on. The differences of in-distribution (training) and OOD (evaluation) environments are illustrated in Fig. 4.10. To be more specific, in transposed tasks, an agent starts at the top or bottom edge and the goal grid is on the farthest edge (bottom or top), whereas a training environment has the agent and goal on the left or right edges.



(a) *In-Dist*, $\delta = 0.35$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 4.10: ***Multitask RL Setting, with In-Distribution and OOD Tasks on RDS**: **a)** example of training environments; **b) - e)** examples of OOD evaluation environments (transposed with a gradient of OOD difficulties δ). For each episode (training or OOD), we randomly generate a new environmental instance, which we call a "task", from a sampling distribution controlled by certain δ. Note that the training environments and the OOD testing environments have almost no intersections.*

4.5.2 Compared Methods

We based all compared agents on a common model-free baseline: a set-based variant of DDQN (introduced in Sec. 3.1.1 on Page. 31, introduced in Hasselt et al. (2016)) with prioritized replay and distributional outputs. We make all compared methods share architectures as much as possible to ensure fair comparisons. Details of the compared methods, their adaptation, and hyperparameters are provided in the Appendix (Sec. 8.1, Page. 159).

4.5.3 Agent Variants

We compare the proposed approach, labeled CP (for Conscious Planning) in the figure legends, against the following methods (variants):

- **UP** (for "Unconscious Planning"): the agent proposed in Sec. 4.3, lacking the bottleneck.
- **model-free**: this model-free set-based agent is the common backbone for all the compared set-based model-based agents. It consists of only the encoder and the value estimator, sharing their architectures with CP and UP, as shown in the red box in Fig. 4.9.
- Dyna: a set-based model-based RL agent which includes a model-free agent and an observation-level transition model, *i.e.*, a transition generator, which shares the same architecture as the CP transition model (with the same hyperparameters as the best performing CP agent), yet applied on the observation-level without an encoder. If performing perfectly, the agent could essentially double the batch size of the model-free baseline by augmenting training batches with an equal number of generated transitions.
- Dyna*: A Dyna variant using the ground truth environment model for transition generation. This is expected to demonstrate Dyna's best-possible performance.
- **WM-CP**: A world model CP variant that is different for following a 2-stage training procedure (Ha and Schmidhuber, 2018). First, the model (together with the state representation encoder) is trained with 10^6 random transitions. After this, the encoder and the model are fixed and RL (value estimation) begins.

- NOSET: A UP-counterpart with vectorized representations and no bottleneck mechanism.

4.5.4 Performance Evaluation (RDS with Turn-Or-Forward Dynamics)

In-Distribution

In Fig. 4.11, we present the in-distribution training performance for the compared agents, *i.e.*, the agents are evaluated with tasks sampled from the same distribution of tasks as that of training.
For **UP**, **CP** and the corresponding **modelfree** baselines, the performance curves indicate convergence to optimal success rates, with confidence intervals overlapping, showing no statistically significant difference in performance. These demonstrate that the 3 agents are effective in learning to solve the in-distribution tasks.
During **WM**'s "warm-up" period, the model learns a representation that captures the underlying dynamics. After the warm-up, the encoder and the model parameters are fixed, and only the value estimator learns to predict the state-action values based on the learned representation. From Fig. 4.11, we can observe that the increase in performance is not only delayed due to the warm-up phase (during which rewards are not taken into account), but also visibly harmed, presumably because the value estimator has no ability to shape the representation to better suit its needs.
For in-distribution evaluation, Dyna performs badly while Dyna* perform relatively well. We suspect that this is due to the delusional transitions generated at the early stages of training, from which the value estimator never recovers[2]. However, despite that Dyna* is by design free of this trouble and exhibits satisfactory training performance, we can see that it does not achieve satisfactory OOD performance (later in Fig. 4.12), due to the limited OOD generalization abilities of background planning methodology (Alver and Precup, 2024).
NOSET performs very badly even in-distribution, per Fig. 4.11. In later experiments, we show that NOSET seems only able to perform well in a more classical, monotask RL setting, suggesting its reliance on memorization. There, we provide more results regarding the model accuracy.

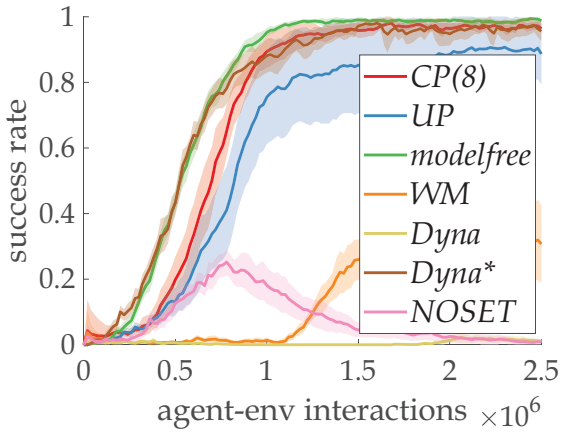
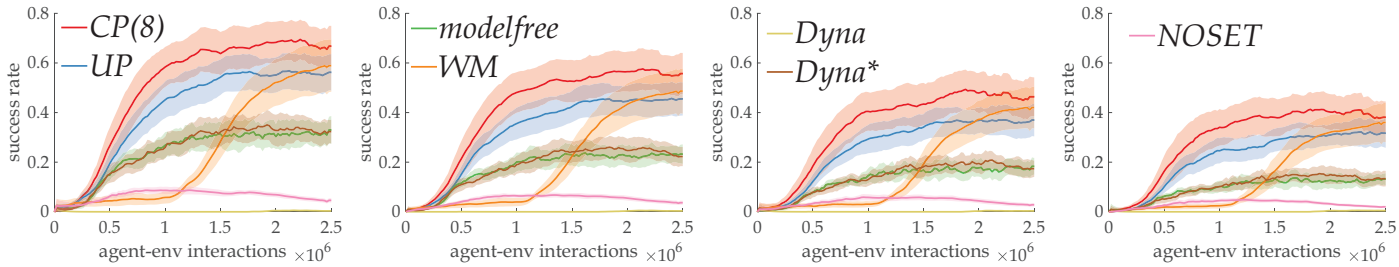


Figure 4.11: ***In-Distribution Task Performance****: the x-axis shows the training progress (2.5×10^6 agent-environment interactions). The y-axis values are generated by agent snapshots at times corresponding to the x-axis values. The bands denote 95%-CI. CP, UP, model-free and Dyna* agents all learn to solve the in-distribution tasks quickly. All error bars are 95% confidence intervals obtained from 20 independent seed runs.*

[2]This was validated and investigated by concurrent work Jafferjee et al. (2020) and importantly in later work Lo et al. (2024), which is not only heavily connected to Chap. 5, but also draws great similarity to approaches used in Chap. 6.

OOD Evaluation Performance

The zero-shot evaluations focus on testing agents' performance facing a gradient of OOD difficulties. The results are presented in Fig. 4.12. **CP(8)**, **CP** with $k = 8$ for top-k semihard attention, shows a clear performance advantage over UP, validating the OOD generalization capability. The Dyna* baseline, essentially the performance upper bound of Dyna-based planning methods, shows no significant performance gain in OOD tests compared to model-free methods. **WM** may have the potential to reach similar performance as CP, yet it needs to warm up the encoder with a significant portion of the agent-environment interaction budget. We investigate further into the potentials of the WM baseline in Sec. 4.5.8 on Page. 79.



(a) *OOD*, $\delta = 0.25$ **(b)** *OOD*, $\delta = 0.35$ **(c)** *OOD*, $\delta = 0.45$ **(d)** *OOD*, $\delta = 0.55$

Figure 4.12: ***OOD performance of Compared Agents under a Gradient of Difficulties****: The figures show a consistent pattern: the MPC-based end-to-end agent equipped with a bottleneck (**CP**) performs the best among all compared methods. All error bars are 95%-CI obtained from* 20 *independent seed runs. All sub-figures share the same* y*-range / scale.*

We want to also verify how much decision-time planning in novel situations could address the generalization gap. For this, we compare the planning agents' performance by enabling / disabling planning (relying on the model-free pass only) in OOD evaluation scenarios. The results are shown in Fig. 4.13.

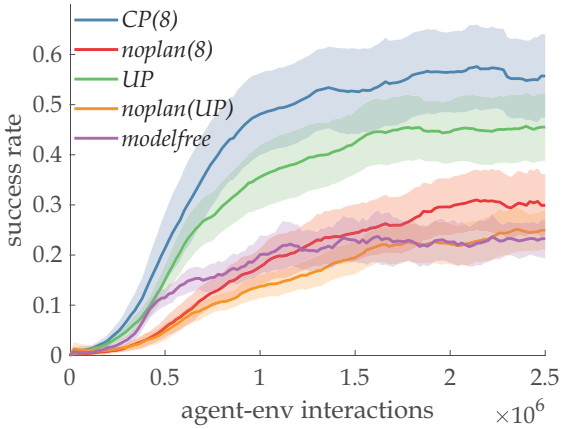
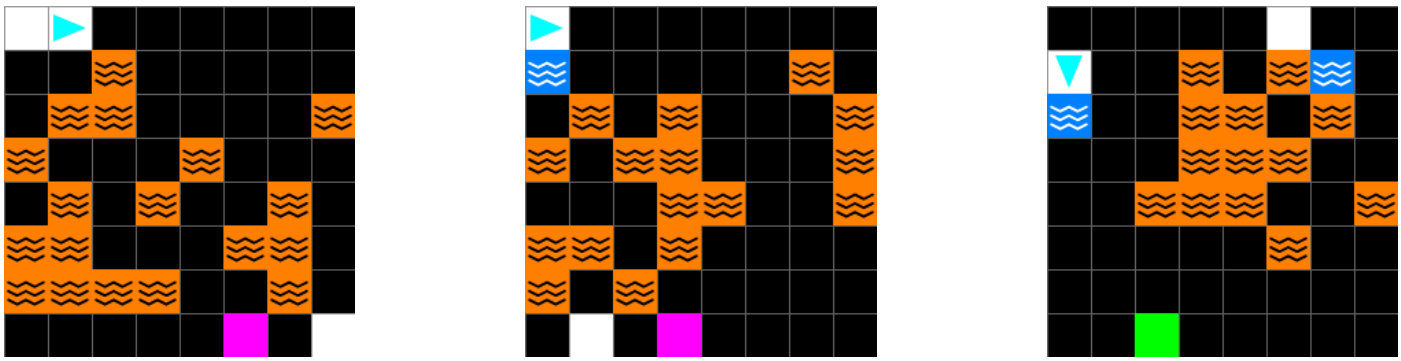


Figure 4.13: ***Reasoning Addresses Generalization Gap, Bottleneck benefits OOD capability****: the* x*-axis shows the training progress (*2.5×10^6 *agent-environment interactions). The* y*-axis values are generated by agent snapshots at times corresponding to the* x*-axis values. noplan(8) and noplan(UP) correspond to the CP(8) and UP variants with planning disabled during OOD tests. Comparing noplan against modelfree, we see that planning during training is beneficial for both value estimation and representation learning. All error bars are 95% confidence intervals obtained from* 20 *independent seed runs.*

Verification of Selection

We present some examples to visually verify the object selection during the planning steps in Fig. 4.14. Note that though these visualizations provide an intuitive understanding of the agents' behavior, they do not serve statistical purposes. In **a)** and **b)**, the agent only turns in its current grid without changing any other grids, thus it exhibits quite random attention towards the other grids. While in c), we can see that the agent takes consideration into the grid (the blue lava grid, color-inverted from orange) that it is facing before taking a step-forward action, since it predicts that such grid will be changed given the intended actions.
Additionally, we collected the coverage ratio of all the relevant objects by the selection phase in all the in-distribution and OOD evaluation cases. The collected data on bottleneck sizes $k = 4$, $k = 8$ and $k = 16$ indicate that the coverage is almost perfect very early on during training. We do not provide these curves because the convergence to 100% is so fast that the curves would all coincide with the line $y = 1$, with some minor fluctuations of the confidence intervals.



(a) *Turn Left* **(b)** *Turn Right* **(c)** *Step Forward*

Figure 4.14: ***Visualization of Bottleneck Selection given Observation & Specific Actions****: These figures are extracted from a fully trained CP agent under OOD evaluation. The bottleneck is set to very small for clearer visualization purposes. We invert the color of the selected grid by the selector's output semihard attention weights.*

4.5.5 Ablation Studies

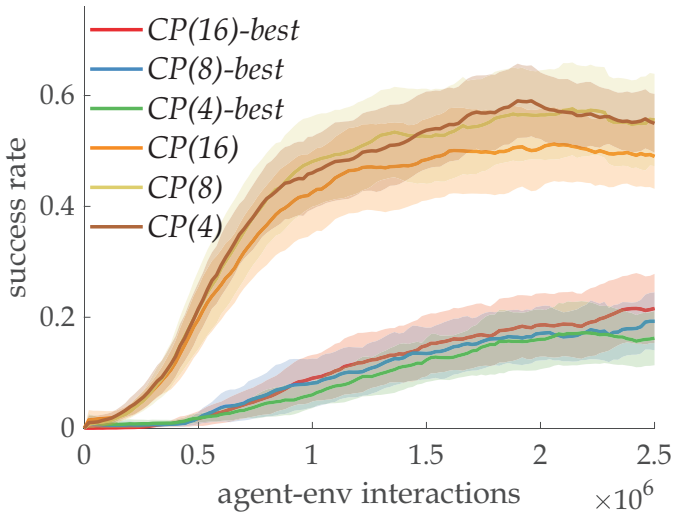
The ablation studies focus on validating the individual design choices by comparing the proposed agents to variants with certain designs altered.

Bottleneck & Planning

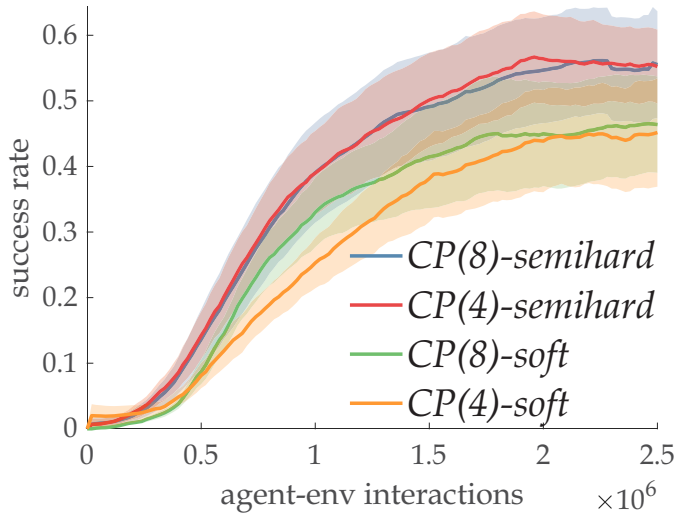
With ablation studies, we validate the effectiveness of our design regarding the bottleneck and planning. We present the key results in Fig. 4.15.

Model Accuracy

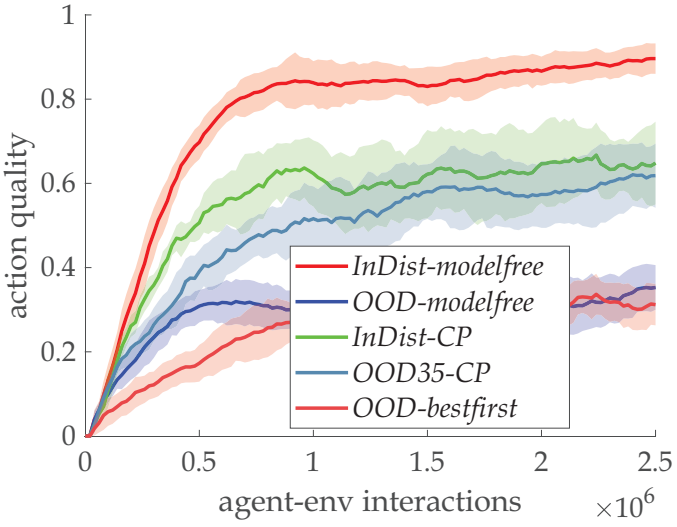
This set of experiments intends to understand how well the bottleneck sets capture the underlying dynamics of the environments. For each transition, with the help of DP, we partition the grid points into two classes: one containing all relevant objects that have an impact on reward or termination or changed during the



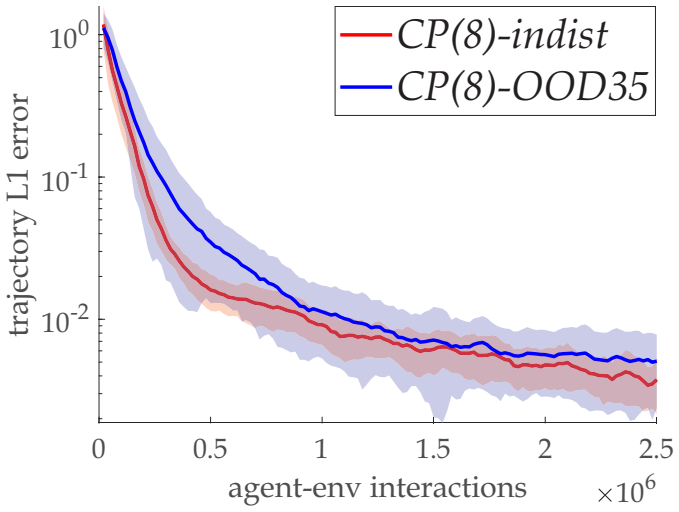
(a) ***Value-guided tree search does not generalize well in OOD evaluation****: random heuristic significantly outperforms best-first heuristic*



(b) ***Attention Type****: when used in bottleneck selection, semi-hard attention outperforms soft attention*



(c) ***Action Optimality****: for in-distribution evaluation, the methods both perform well. Interestingly, the model-free agent performs superior possibly due to its simple value-based greedy policy. However, in OOD evaluation, only the CP agent with the random heuristic shows neither significant deterioration nor signs of overfitting*



(d) ***Tree Search Dynamics Accuracy****: we average over the cumulative L_1 error in the simulated state of the chosen trajectories during tree search. The curves show no signs of overfitting, as the cumulative trajectorial dynamics errors for OOD evaluation are decreasing continuously.*

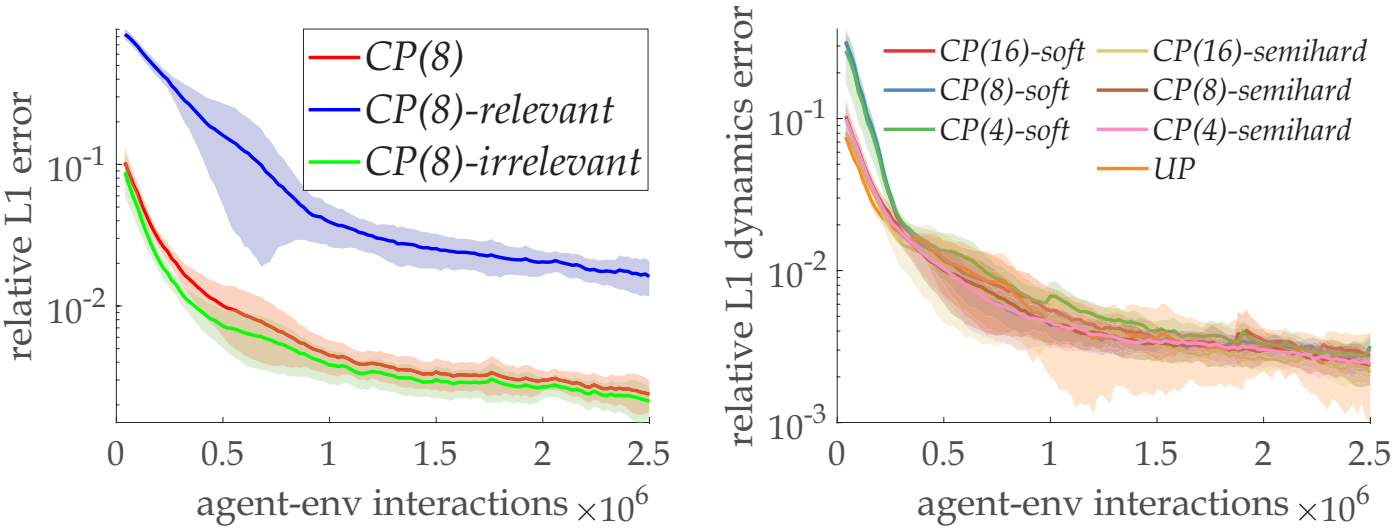
Figure 4.15: ***Ablation Results regarding Spatial Abstraction via the Proposed Bottleneck Mechanism****: With $\delta = 0.35$, each error bar (95%-CI) is obtained from 20 independent seed runs.*

transition; While, the other contains the remaining grid points. As a result, the dynamics errors are split into two terms which correspond to the accuracy of the model simulating the relevant and irrelevant objects, respectively.

Acknowledging the differences in the norm of the learned latent representations for each run, due to the stochasticity in the parameter initialization and optimization processes, we use the *normalized* element-wise mean of L_1 (absolute value) difference between $\hat{s}_{t+1}$ and s_{t+1}. The residual is normalized by the element-wise mean L_1 norm of s_{t+1}. As a metric of model accuracy, we name such metric *relative* L_1. This metric shows the degree of deviation in dynamics learning: the lower it is, the more consistent are the learned and observed dynamics.

Fig. 4.16 **a)** presents the progression of *relative* L_1 error of the CP agent during the in-distribution learning process. With the help of the bottleneck, the error for the irrelevant parts converge very quickly while the model focuses on learning the relevant changes in the dynamics. We provide the model accuracy curves of the **WM** and Dyna baselines in the later parts of the experiments.

For reward and termination estimations, our results show no significant difference in estimation accuracies, with varying bottleneck sizes. However, they do have significant impact on the quality of the learned dynamics. In Fig. 4.16 **b)**, we present the convergence of the relative dynamics accuracy of different CP and UP variants. CP agents generally learn as fast as UPs, which indicates low overhead for learning the selection and integration. We will present a more detailed sensitivity analyses for the bottleneck size later in Sec. 4.5.6.



(a) *Split of Relative L_1 error*

(b) *Comparison of Relative L_1 error*

Figure 4.16: *In-Distribution Model Performance: Each band shows the mean curve (bold) and the 95% confidence interval (shaded) obtained from 20 independent seed runs. **a)**: Partitioning of the relative L_1 dynamics prediction errors into that of the relevant objects and the irrelevant ones: The difference in the errors shows that the bottleneck learns to ignore the irrelevance while prioritizing on the relevant parts of the state; **b)**: Comparison of the overall relative L_1 errors (not partitioned). For CP variants, the numbers in the parentheses correspond to the bottleneck sizes and the suffixes the types of attention for the bottleneck selection. Semi-hard attention learns more quickly than soft attention at early stages, but they both converge to similar accuracy levels. This is likely because semi-hard attention is forced to pick few objects and thus to ignore irrelevant objects even at early stages of training.*

Action Regularization

For the stability of the CP agent, we applied an additional regulatory loss that predicts the action a_t with c_t and $\hat{c}_{t+1}$ as input, resembling an inverse model (Conant and Ross Ashby, 1970). The loss is implemented with categorical cross-entropy, similar to how we handled the termination prediction loss. As shown in Fig. 4.17, This additional training signal is shown in experiments to produce better OOD results, especially when the bottleneck is small.

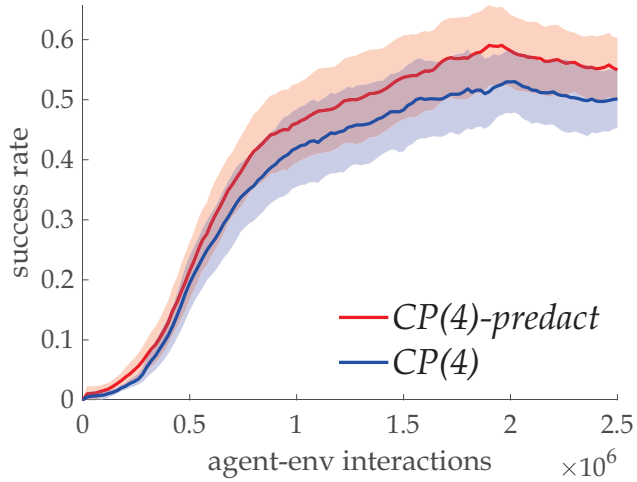


Figure 4.17: *Impact of Action Regularization Loss: we present the results on an ablation study on the impact of the action regularization loss on the OOD evaluation success rates of CP(4) agents, with difficulty $\delta = 0.35$ on RDS. The "predact" configuration is by default enabled in all experiments. Each point of the band correspond to the mean and confidence interval are obtained from 20 independent seed runs.*

4.5.6 Sensitivity Studies

We investigate how sensitive the compared agents are to certain hyperparameters and to the sizes of the tasks.

Bottleneck Size

Without a question, the most important hyperparameter of the CP agents is k, which controls the size of the bottleneck set. We present the results of a sensitivity study regarding the hyperparameter k in Fig. 4.18.

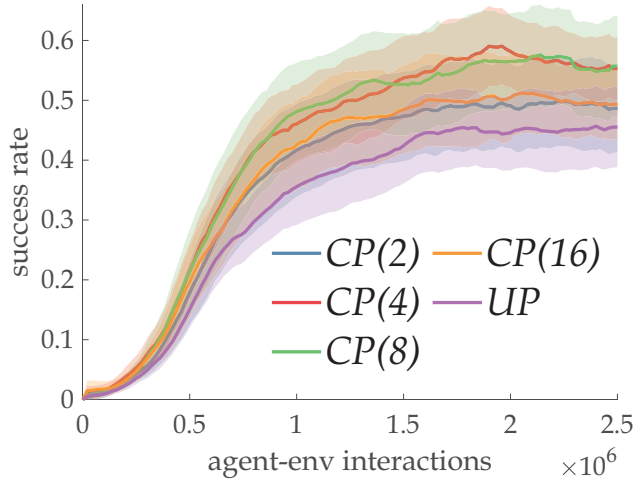


Figure 4.18: ***Sensitivity to Bottleneck Sizes (controlled by k):*** *bottleneck sizes $k = 4$ and $k = 8$ perform similarly the best within $k \in \{2, 4, 8, 16\}$. Note that the performance of CP agents (agents with bottlenecks) is consistently better than those without (UP), showing the bottlenecks' effectiveness for generalization*

Planning Steps

We investigate the number of planning steps allowed in each MPC session at each decision point. Intuitively, if the planning steps are too few, then the planning would have little gain over model-free methods. While, if the planning steps are too many, we suffer from cumulative planning errors and potentially prohibitive wall time.

Thus, we have a strong intuition that an appropriate value for the number of planning steps could potentially achieve a good tradeoff. To search for such good value, we tried different numbers of planning steps for CP(8) variant. Note that the planning steps during training and OOD evaluation are set to be the same, to ensure that the planning during evaluation would be carried out to the same extent during training. The results visualized in Fig. 4.19 suggested that 5 planning steps achieves the best performance in OOD with difficulty 0.35. Base on this, for all other experiments reported in this chapter, 5 is set to be the default number of planning steps.

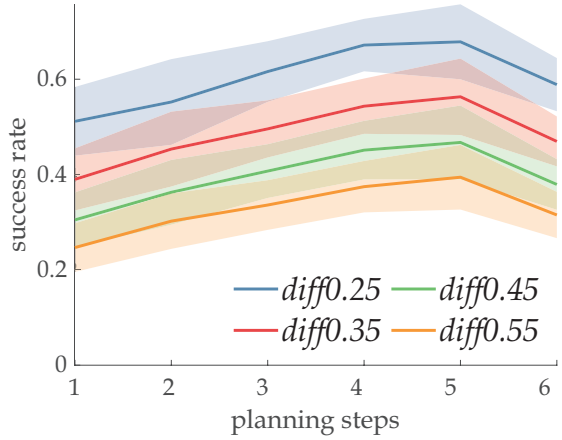


Figure 4.19: ***Success Rates given Different Numbers of Planning Steps****: Success rates of CP(8) agent under OOD difficulty $\delta = 0.35$ are presented. each data point is obtained by averaging from the last 20% of 20 independent seed runs.*

World Sizes

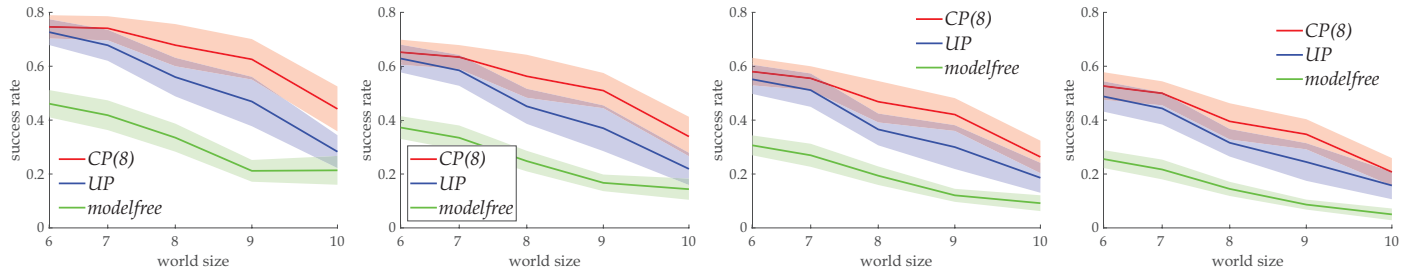
To inspect the scalability of the proposed method (in terms of the number of objects in the full state sets), we compare the methods CP(8), UP and model-free in a range of gridworld sizes (from 6×6 all the way to 10×10). The results are presented in Fig. 4.20.

4.5.7 More Experiments, Discussions & Failed Attempts

We go beyond the main experiments ("turn-or-forward" `RDS` instances) to gain some additional insights.

Alternative Dynamics

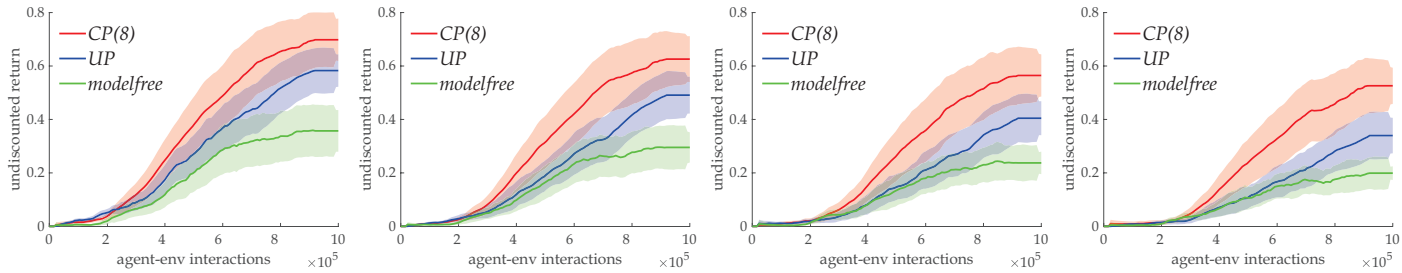
We want to know if the conclusions of previous experiments would still hold, if the agents are given tasks with different dynamics. For this purpose, we modify the original tasks in the previous experiments by a new set of "Turn-And-Forward" dynamics: the action space is re-designed to include 4 composite actions, each of which first turns to some directions (forward, left, right or back based on the agent's current facing direction in the environment) and then move forward if possible (remain in the boundaries of the grid world).



(a) *OOD*, $\delta = 0.25$ **(b)** *OOD*, $\delta = 0.35$ **(c)** *OOD*, $\delta = 0.45$ **(d)** *OOD*, $\delta = 0.55$

Figure 4.20: ***Scalability of OOD Performance under a Spectrum of Difficulties and World Sizes****: The x axes are ticked with #grids in each gridworld size, representing the number of entities for in the state set, thus non-linear w.r.t. the world sizes. We can observe that, generally, the smaller the world sizes, the better and the closer the performance of all 3 methods are. The fact that the CP(8) performance deteriorates slower than UP suggests that the bottleneck may contribute to more scalable performance in tasks with larger amount of entities. All error bars are obtained from 20 independent seed runs.*

This new set of environment dynamics ("turn-AND-forward") can be seen as a composition of the original dynamics ("turn-OR-forward") and hence produces shorter planning trajectories. In Fig. 4.21, we observe that all three methods are performing better compared to the original tasks and the previous conclusions about experiments are again validated.



(a) *OOD*, $\delta = 0.25$ **(b)** *OOD*, $\delta = 0.35$ **(c)** *OOD*, $\delta = 0.45$ **(d)** *OOD*, $\delta = 0.55$

Figure 4.21: ***OOD Performance of Compared Agents in "Turn-and-Forward" Tasks****: All error bars are obtained from 20 independent seed runs.*

Shaping Representations with Many Signals

We have gathered more empirical evidence regarding the non-conflicting training of the state-representation based on the signals. This means, it is possible to train a state representation that can be used to predict several interesting quantities to the RL agent at the same time. In terms of the accuracy of model learning, according to our results of the **WM** baseline, removing the value estimation training signal would result in a poorer representation, but not in a lower accuracy when predicting other relevant signals; Similarly, removing termination signal would not impact the convergence of reward prediction accuracy or that of the dynamics prediction, despite that the RL performance would be impacted. Also, removing the reward signal or the next state dynamics prediction signal leads to collapse of the MPC based behavior policy. Yet, the convergence of the other remaining training signals is not significantly affected. With these observations,

we would suggest that we have, at least in this task setting, learned a set-based representation capable of predicting all interesting quantities.

Failed Effort: Straight-Through Hard Subset Selection with Gumbel

We initially tried to use Gumbel subset selection (Xie and Ermon, 2021) to implement a hard selection based bottleneck, but we later realized that such approach will not work. We expected the model to pick the right objects by generating a binary mask, and then use the masked objects as the bottleneck set. On the surface, this two-staged design would align more with the consciousness theories, and would yield clearer interpretability. However, we came to realize that such method overlooked an implicit chicken-and-egg problem. This problem can be intuitively described as follows: to learn how to pick, the model should first understand the dynamics. Yet if the model does not pick the right objects frequently enough, the dynamics would never be understood. Mathematically, this problem is more recently described as the "degeneracy" of discrete bottlenecks. Our proposed semi-hard / soft approaches address such problem by essentially making the two staged selection and simulation as a whole for the gradient-based optimization.

4.5.8 Details of Baselines

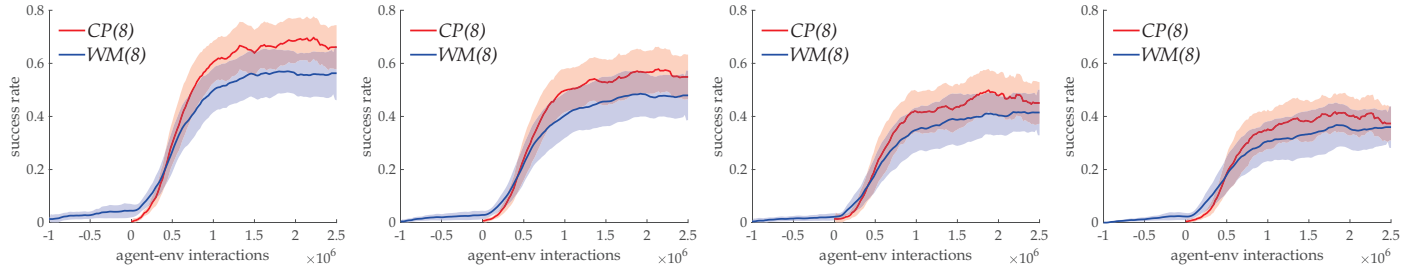
WM

Our **WM** baselines, *i.e.* agents with World Model (WM) trained in stages, share the same architectures (and hyperparameters) as their CP or UP counterparts (Zhou et al., 2025). The only difference is that **WM** adopts a 2-staged training strategy: In the first 10^6 agent-environment interactions, only the model is trained (together with the state representation encoder), and thus, the state representations are only shaped by the model's training losses (without the participation of RL signals). In the first stage, the agent relies on a uniformly random policy. After 10^6 interactions, the agent freezes its state representation encoder as well as the model to carry out value estimator learning separately. Compared to CP or UP, the exploration scheme is delayed but unchanged.

We are also curious about how the WM baseline would evolve after the 2.5×10^6-step cutoff. For this, we provide an additional set of experiments featuring a "free" unsupervised learning phase of 10^6 agent-environment interactions, essentially prolonging the runs of **WM** baselines by extra 10^6 steps. As presented in Fig. 4.22, results suggest that **WM** could not achieve similar performance as that of CP, likely because the state representation is not jointly shaped for value estimation and thus cannot perform well enough for such purpose. The results show promise of the methodology of representation learning with joint signals. However, this is not to deny the usability and applicability of the world model methodology in general.

Dyna

As usual, for fair comparison, Dyna baselines share the model-free part of the architecture as CP or UP. The models used by the Dyna baselines are powered by the proposed set-to-set architecture, working on the observation-level. For each sampled batch of transitions, we feed the sampled current observations and actions to the Dyna model to generate imagined next observations and train additionally on this second batch.



(a) *OOD*, $\delta = 0.25$ **(b)** *OOD*, $\delta = 0.35$ **(c)** *OOD*, $\delta = 0.45$ **(d)** *OOD*, $\delta = 0.55$

Figure 4.22: ***Extended Run to Demonstrate the OOD Performance Differences of CP and WM agents****: for fair comparison, WM(8) is the WM variant which uses the same architecture as CP(8). Results of WM(8) are shifted along the x-axes for a free unsupervised world model learning phase of 10^6 steps. All error bars are obtained from 20 independent seed runs.*

NOSET

The NOSET baseline utilizes traditional vectorized representations. We use the CNN feature extractor before the encoder, but instead of transforming the feature map into a set, we flatten it and then linearly project it to some specific dimensionality (256), which will be used as the vectorized state representation, similar to most existing DRL practices. Since all set-based operations would now be improper for the vectorized representation, they are substituted with 3-layered, 512-wide MLPs. The dimension of the vectorized state representation, the widths, and the depths of the FC layers are optimized through coarse grid tuning. We find that architectures exceeding the chosen size are hardly superior in terms of performance.

To maximize the performance of this baseline, we designed a 2-layered dynamics model, which employ a residual connection, with the expectation that the model might learn incremental changes in the dynamics. We first verified the competence of this baseline on a classical mono-task setting, as shown in Fig. 4.23. However, to a degree as expected, in our experiments with randomly generated environments for each episode (the "multi-task setting"), the NOSET baseline performs miserably.

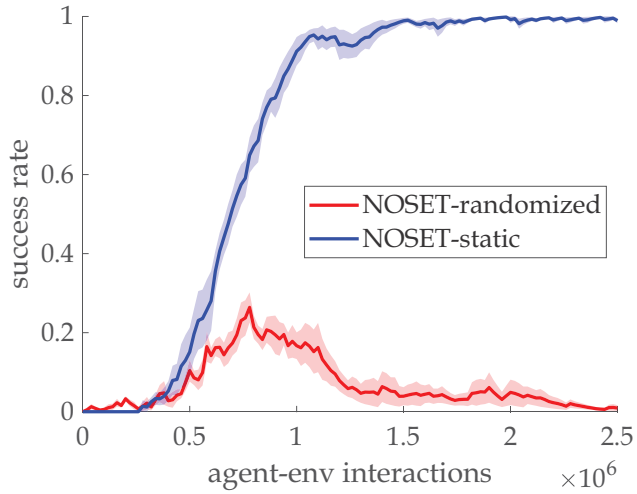


Figure 4.23: NOSET ***Baseline Performance on Multitask and Monotask Settings****: Each band is consisted of the mean curve and the confidence interval shades obtained from 20 independent seed runs.*

4.5.9 Summary of Experiments

The experimental results allow us to draw the following conclusions, within the scope of our experimental setting:

- Set-based representations enable at least in-distribution generalization across different environment instances in our OOD-oriented multi-task setting, where the agents are forced to discover dynamics that are preserved across environments;
- Model-free methods seem to rely more on memorization instead of understanding, as they face more difficulties in OOD evaluation scenarios, which emphasizes systematic generalization;
- Decision-time MPC exhibits better performance than Dyna in the tested OOD generalization settings;
- Online joint training of the representation with all the relevant signals could bring benefits to RL, as suggested in Jaderberg et al. (2017).
- In accordance with our intuition, transition models with bottlenecks tend to learn dynamics better in our tests;
- From further experiments, we observe that bottleneck-equipped agents may also be less affected by larger size environments with more encoded objects, possibly due to their prioritized learning of interesting entities.

4.6 Summary

In this chapter, we introduced a consciousness-inspired bottleneck mechanism into model-based RL, facilitated by set-based representations, end-to-end learning and tree search MPC. In the multi-task systematic generalization-focused settings, the bottleneck allows selecting the relevant objects for planning and hence enables significant OOD performance.
The proposed bottleneck mechanism will be extended for spatial abstraction, a foundation of Chap. 5.

Chapter 5

Skipper Framework: Spatio-Temporal Abstractions for Planning

Contents

7.1 Summary of Contributions 137

7.1.1 Conscious Planning: Spatially-Abstract Decision-Time Reasoning 137

7.1.2 Skipper Framework: Spatio-Temporal Abstractions for Planning 139

7.1.3 Rejecting Hallucinations: Addressing Delusional Planning Behaviors 140

7.2 Limitations 141

7.2.1 Limitations of Work on Conscious Planning (Chap. 4) 141

7.2.2 Limitations of Work on Skipper (Chap. 5) 142

7.2.3 Limitations of Work on Rejecting Hallucinations (Chap. 6) 143

7.3 Future Work 143

5.1 Overview of This Thesis Milestone

Inspired by the humans' abstract planning behaviors, my collaborators and I propose Skipper, an RL / planning framework using both spatial and temporal abstractions organically to generalize better in novel situations. Skipper automatically decomposes the given task into smaller, more manageable sub-tasks, and thus enables sparse decision-making and focused computation on the relevant parts of the environment. Skipper's task decomposition relies on the extraction of a "proxy problem", represented as a directed graph, in which vertices and edges are learned end-to-end from hindsight. Our theoretical analyses provide performance guarantees under appropriate assumptions and establish where our approach is expected to be helpful. Our carefully controlled experiments validate Skipper's significant advantage in zero-shot generalization, compared to some existing state-of-the-art hierarchical planning methods.

Human abstract planning can attend to relevant aspects in both time and space. This kind of planning could competently break down long-horizon tasks into smaller scale and more manageable steps, each of which

can be narrowed down even more. From the previously introduced perspective of consciousness in the first sense (C1) (Dehaene et al., 2020), this type of planning focuses attention on mostly the important decision points (Sutton et al., 1999) and relevant environmental aspects (Tang et al., 2020), thus *operating abstractly both in time and in space.*

In contrast, existing RL agents either operate solely based on intuition ("system-1", model-free methods) or are limited to reasoning over mostly relatively shortsighted plans (rudimentary "system-2", model-based methods, including the CP agent in the previous chapter) (Kahneman, 2017). These intrinsic design limitations hinder the agents' real-world application, which demands robustness against generalization challenges (Mendonca et al., 2020).

Building on intuitions gathered from our previous work on conscious planning (Chap. 4), we develop a planning framework, named Skipper, that plans while skipping over unnecessary details in both space and time, seeking to decompose the complex task at hand into smaller sub-tasks, by constructing "proxy problems". A proxy problem can be viewed as a simplified version of the given task, and is represented as a graph where 1) the vertices consist of states imagined by a generative model, corresponding to sparse decision points; and 2) the edges, which define temporally-extended transitions, are constructed by focusing on a small amount of relevant information from the states, using an attention mechanism heavily investigated in the previous chapter. Once a proxy problem is constructed and the agent solves it to form a plan, each edge will define a new sub-problem, on which the agent will focus solving next. While granting agents the flexibility to construct necessary abstractions for the problem at hand, the divide-and-conquer strategy based on proxy problems allows constructing "partial" solutions that generalize better to novel situations. Our theoretical analyses tie the quality of the solution to the proxy problem to the solution of the overall problem, making the Skipper framework distinguish itself as a principled approach.

For experiments, we use an implementation of the Skipper framework to demonstrate its advantages in terms of out-of-training-distribution generalization. These experiments are conducted on a more refined multi-task, systematic generalization focused setting, where the agents are only trained on limited few task instances but are expected to be evaluated in a range of OOD scenarios in a zero-shot fashion. The detailed and controlled experiments that the proposed framework, in most cases performs significantly better in terms of zero-shot generalization, compared to the baselines and to some state-of-the-art hierarchical planning methods (Nasiriany et al., 2019; Hafner et al., 2022).

5.2 Methodology: Proxy Problems

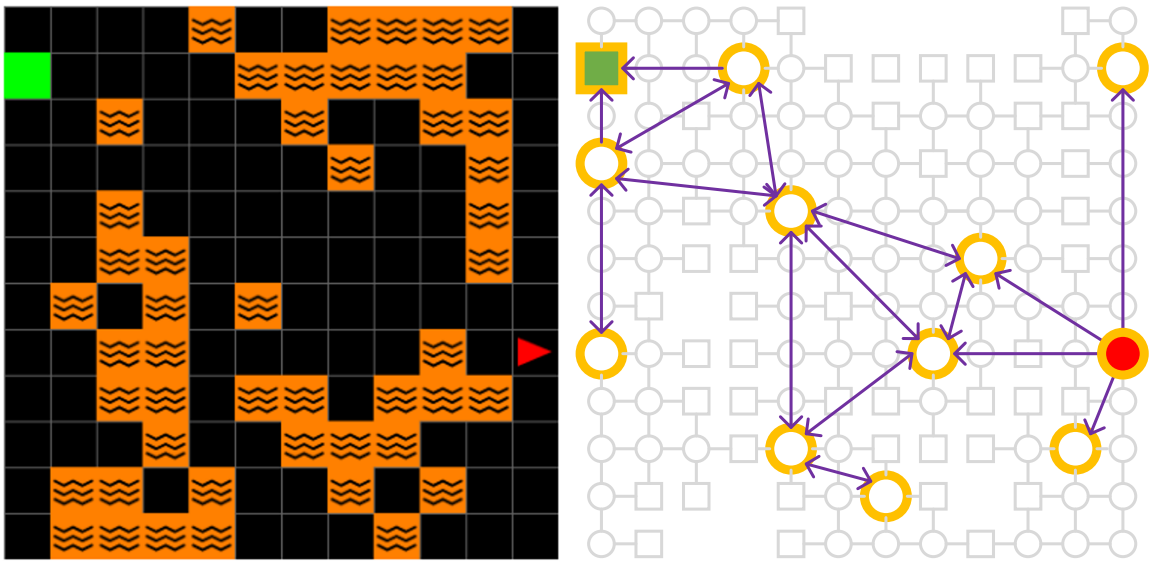
5.2.1 Definition

Central to the temporal and spatial abstraction abilities of the proposed Skipper framework, proxy problems are utilized to handle task decomposition. Representing simplified versions of the more complicated given tasks, proxy problems are proxies for solving the more complicated given tasks. Each solution of the proxy problem is equivalent to a plan that can be carried out in the original problem. While, each step of the plan is a sub-problem in the original problem.

Proxy problems are sparse graphs constructed at decision-time, whose vertices are states and whose (directed) edges estimate transition statistics between the vertices, as shown in Fig. 5.1. For clarity, we call the

states selected to be (or proposed by the generator to be) vertices of the proxy problems **checkpoints**, to differentiate them from other uninvolved states. Note that the current state should always be included as one of the vertices. At decision-time, during evaluation, the checkpoints are proposed by a generative model, a **generator**, and represent some states that the agent might experience in the current episode, often denoted as $S^{\odot}$ in this chapter. Each edge is annotated with estimates of the cumulative discount and reward associated with the transition between the connected checkpoints; these estimates are learned over the **relevant** aspects of the environment ("spatially-abstract") and **depend** on the agent's capability ("capability-aware"). As the low-level policy responsible for solving the sub-problems (implementing the transitions from one checkpoint to the next) improves, the edges strengthen.

Planning in a proxy problem is by design temporally abstract, since the sub-problems are defined as the transitions between pairs of checkpoints, which are sparse decision points. Handling each checkpoint transition is also by design spatially abstract, as an option corresponding to such a task would base its decisions only on some aspects of the environment relevant to the sub-problem at hand (Konidaris and Barto, 2009; Bengio, 2019), leading to improved generalization as well as computational efficiency (Zhao et al., 2021). In the language of probabilistic modelling, the proxy problems alleviate the difficulties of long horizon planning by *marginalizing the change of more irrelevant aspects* of the space and time within trajectories, and *constraining the options s.t.*, the changes that matter to the objective are simpler to predict (Bengio, 2019).



(a) *RDS Instance* **(b)** *Proxy Problem*

Figure 5.1: ***A Proxy Problem Proposed by*** Skipper ***on an RDS Task****: the MDP of the original problem is in colored gray and the terminal states (the lava grids and the goal) are marked with squares. The overall objective of the task is to navigate from the red position, to the goal (green). As shown, distant goals can be reached by leveraging a proxy problem, which can lead to a plan that involves hopping through a series of checkpoints (orange).*

Proxy problems can be formally described in the language of SMDPs (Sec. 2.5.1, Page. 26), where each directed edge is implemented as a checkpoint-conditioned option. Thus, edges in a proxy problem can be defined by

the discount and reward matrices Γ^π and V^π, where γ_{ij}^π and v_{ij}^π are defined as:

$$\gamma_{ij}^\pi \coloneqq \mathbb{E}_\pi \left[\gamma^{T_\perp} | S_0 = s_i, S_{T_\perp} = s_j\right] \tag{5.1}$$

$$v_{ij}^\pi \coloneqq \mathbb{E}_\pi \left[\sum_{t=0}^{T_\perp} \gamma^t R_t | S_0 = s_i, S_{T_\perp} = s_j\right] \tag{5.2}$$

5.2.2 Potential of Proxy Problems

By planning with Γ^π and V^π, *e.g.*, using SMDP value iteration (Sutton et al., 1999), we can form a plan that hops through the checkpoints in the proxy problem, while simultaneously traveling among states in the original problem.

But why should we use this methodology? What could be the advantage? The following results establish that if the proxy problems can be estimated well (Condition 5.3), the obtained plan via the proxy problem will be a good quality solution for the original problem (Bound 5.4):

Theorem 1: Performance under Proxy Problems

Let μ be the SMDP policy (high-level planner that solves the proxy problems and produce a plan) and π be the low-level policy (that implements the sub-problems of transitioning from one checkpoint to another). Let $\hat{V}^\pi$ and $\hat{\Gamma}^\pi$ denote learned estimates defining the edges of the proxy problem. If their estimation accuracy satisfies:

$$\begin{aligned} |v_{ij}^\pi - \hat{v}_{ij}^\pi| < \epsilon_v v_{\text{range}} \ll (1-\gamma) v_{\text{range}} \qquad & \textbf{and} \\ |\gamma_{ij}^\pi - \hat{\gamma}_{ij}^\pi| < \epsilon_\gamma \ll (1-\gamma)^2 \qquad & \forall i, j. \end{aligned} \tag{5.3}$$

Then, the estimated value of the composite $\hat{v}_{\mu\circ\pi}(s)$ is accurate up to error terms linear in ϵ_v and ϵ_γ:

$$\hat{v}_{\mu\circ\pi}(s) \coloneqq \sum_{k=0}^{\infty} \hat{v}_\pi(s_k^\odot | s_{k+1}^\odot) \prod_{\ell=0}^{k-1} \hat{\gamma}_\pi(s_\ell^\odot | s_{\ell+1}^\odot) = v_{\mu\circ\pi}(s) \pm \frac{\epsilon_v v_{\text{range}}}{1-\gamma} \pm \frac{\epsilon_\gamma v_{\text{range}}}{(1-\gamma)^2} + o(\epsilon_v + \epsilon_\gamma) \tag{5.4}$$

where $\hat{v}_\pi(s_i|s_j) \equiv \hat{v}_{ij}^\pi$ and $\hat{\gamma}_\pi(s_i|s_j) \equiv \hat{\gamma}_{ij}^\pi$, and $v_{\text{range}} \coloneqq v_{\max} - v_{\min}$ denotes the range of values, *i.e.*, the maximum possible value minus the minimum possible value.

Proof. Before anything, we need to acknowledge that we are in the presence of a hierarchical structure involving two policies. The high level policy μ determines (potentially) stochastically a sequence of checkpoints to form a plan. The low-level policy π is assumed to be evolving throughout training, and fixed for the moment. The composite policy $\mu \circ \pi$ is non-Markovian: it depends both on the current state and the current target checkpoint. Thus, there is no notion of a classical "state value", except when the agent arrives at a checkpoint, *i.e.*, when a high level action (checkpoint selection) needs to be chosen.

To simplify the proof that proceeds further, we adopt the view that discounts can be implemented equivalently with sudden termination probabilities. In this equivalent view, v_π denotes the undiscounted expected sum of reward before reaching the next checkpoint, and more interestingly γ_π denotes the binomial random variable of non-termination during a sub-problem, *i.e.*, the transition to the selected checkpoint.

With the above in mind, we can write the gain (in undiscounted cumulated rewards) V from the initial checkpoint $S_0^{\odot}$ as:

$$V = V(S_0^{\odot}|S_1^{\odot}) + \Gamma(S_0^{\odot}|S_1^{\odot})V_1 = \sum_{k=0}^{\infty} V(S_k^{\odot}|S_{k+1}^{\odot}) \prod_{i=0}^{k-1} \Gamma(S_i^{\odot}|S_{i+1}^{\odot}), \tag{5.5}$$

where $S_{k+1} \sim \mu(\cdot|S_k)$, $V(S_k^{\odot}|S_{k+1}^{\odot})$ is the random variable of gain during the transition from $S_k^{\odot}$ to $S_{k+1}^{\odot}$, and the random variable $\Gamma(S_k^{\odot}|S_{k+1}^{\odot})$ can take a value of either 0 or 1, depending on whether the trajectory terminated prematurely or reached $S_{k+1}^{\odot}$.
It should be clear that the action space of μ is the a list of checkpoints $\{s_0^{\odot} = s_0, s_1^{\odot}, s_2^{\odot}, \cdots\}$ in the same proxy problem. Thus, for the expected (ground truth) value of $\mu \circ \pi$, taking the expectation of V_0, we have:

$$v_{\mu\circ\pi}(s_0) \coloneqq \mathbb{E}_{\mu\circ\pi}[V|S_0 = s_0] = \sum_{k=0}^{\infty} v_{\pi}(s_k^{\odot}|s_{k+1}^{\odot}) \prod_{i=0}^{k-1} \gamma_{\pi}(s_i^{\odot}|s_{i+1}^{\odot}) \tag{5.6}$$

Having obtained the ground truth value, in the following, we are going to consider the estimates with small error terms:

$$|v_{\pi}(\cdot) - \hat{v}_{\pi}(\cdot)| < \epsilon_v v_{\max} \ll (1-\gamma)v_{\max} \qquad \text{and} \qquad |\gamma_{\pi}(\cdot) - \hat{\gamma}_{\pi}(\cdot)| < \epsilon_{\gamma} \ll (1-\gamma)^2 \tag{5.7}$$

To simplify the algebra, we assume without loss of generality that all rewards are non-negative, *s.t.*, the values are guaranteed to be non-negative as well. Note that the assumption for the reward function to be non-negative is only a cheap trick to ensure we bound in the right direction of ϵ_{γ} errors in the discounting, but that the theorem would still stand if it were not the case: simply replace $v_{\max}$ with $v_{\max} - v_{\min}$). With the assumption, we can write the upper bound as:

$$\hat{v}_{\mu\circ\pi}(s) \coloneqq \sum_{k=0}^{\infty} \hat{v}_{\pi}(s_k^{\odot}|s_{k+1}^{\odot}) \prod_{i=0}^{k-1} \hat{\gamma}_{\pi}(s_i^{\odot}|s_{i+1}^{\odot}) \tag{5.8}$$

$$\leq \sum_{k=0}^{\infty} \left(v_{\pi}(s_k^{\odot}|s_{k+1}^{\odot}) + \epsilon_v v_{\max}\right) \prod_{i=0}^{k-1} \left(\gamma_{\pi}(s_i^{\odot}|s_{i+1}^{\odot}) + \epsilon_{\gamma}\right) \tag{5.9}$$

$$\leq v_{\mu\circ\pi}(s) + \sum_{k=0}^{\infty} \epsilon_v v_{\max} \prod_{i=0}^{k-1} \left(\gamma_{\pi}(s_i^{\odot}|s_{i+1}^{\odot}) + \epsilon_{\gamma}\right) + \sum_{k=0}^{\infty} \left(v_{\pi}(s_k^{\odot}|s_{k+1}^{\odot}) + \epsilon_v v_{\max}\right) k\epsilon_{\gamma}\gamma^k + o(\epsilon_v + \epsilon_{\gamma}) \tag{5.10}$$

$$\leq v_{\mu\circ\pi}(s) + \epsilon_v v_{\max} \sum_{k=0}^{\infty} \gamma^k + \epsilon_{\gamma} v_{\max} \sum_{k=0}^{\infty} k\gamma^k + o(\epsilon_v + \epsilon_{\gamma}) \tag{5.11}$$

$$\leq v_{\mu\circ\pi}(s) + \frac{\epsilon_v v_{\max}}{1-\gamma} + \frac{\epsilon_{\gamma} v_{\max}}{(1-\gamma)^2} + o(\epsilon_v + \epsilon_{\gamma}) \tag{5.12}$$

Similarly, we can derive a lower bound:

$$\hat{v}_{\mu\circ\pi}(s) \coloneqq \sum_{k=0}^{\infty} \hat{v}_\pi(s_k^\odot | s_{k+1}^\odot) \prod_{i=0}^{k-1} \hat{\gamma}_\pi(s_i^\odot | s_{i+1}^\odot) \tag{5.13}$$

$$\geq \sum_{k=0}^{\infty} \left(v_\pi(s_k^\odot | s_{k+1}^\odot) - \epsilon_v v_{\max}\right) \prod_{i=0}^{k-1} \left(\gamma_\pi(s_i^\odot | s_{i+1}^\odot) - \epsilon_\gamma\right) \tag{5.14}$$

$$\geq v_{\mu\circ\pi}(s) - \sum_{k=0}^{\infty} \epsilon_v v_{\max} \prod_{i=0}^{k-1} \left(\gamma_\pi(s_i^\odot | s_{i+1}^\odot) - \epsilon_\gamma\right) - \sum_{k=0}^{\infty} \left(v_\pi(s_k^\odot | s_{k+1}^\odot) - \epsilon_v v_{\max}\right) k\epsilon_\gamma \gamma^k + o(\epsilon_v + \epsilon_\gamma) \tag{5.15}$$

$$\geq v_{\mu\circ\pi}(s) - \epsilon_v v_{\max} \sum_{k=0}^{\infty} \gamma^k - \epsilon_\gamma v_{\max} \sum_{k=0}^{\infty} k\gamma^k + o(\epsilon_v + \epsilon_\gamma) \tag{5.16}$$

$$\geq v_{\mu\circ\pi}(s) - \frac{\epsilon_v v_{\max}}{1-\gamma} - \frac{\epsilon_\gamma v_{\max}}{(1-\gamma)^2} + o(\epsilon_v + \epsilon_\gamma) \tag{5.17}$$

□

Let us carefully interpret and discuss the implication of the proved theorem:

No Assumption on Optimality

The methodology of using proxy problems and the theorem above assume that it would be difficult to learn effectively a π for all sub-problems: when the target is too far, and that we would rather use a proxy to construct a path with shorter-distance transitions. Therefore, we never want to make any optimality assumption on π, otherwise our approach is pointless: If the low-level policy π is perfect, then the best high-level policy μ should always directly choose the task goal as target. A triangular inequality can be shown that with a perfect π and a perfect estimate of v_π and γ_π, the performance will always be optimal by selecting $s_1^\odot = s_g$. The basis of divide-and-conquer strategies, including proxy problems in the Skipper framework, is that learning an all-capable policy is difficult in practice. The lack of assumption here is very appropriate in our view.

Assumption on Checkpoint Feasibility

The analyses assumed that the target checkpoints are all feasible for convenience. While in reality (in the approximate RL setting without access to all states), since checkpoints need to be generated, the checkpoints could end up being hallucinated targets which cannot be achieved. To be able to handle this more realistic setting, the estimators responsible for estimating γ^π need to produce the correct estimate 0. Such a problem is not unique to Skipper and I will discuss this problem in great detail in Chap. 6.

Assumption of Estimation Accuracy & Improving Low-Level Policy

Thm. 1 provides guarantees on the solution to the overall problem, whose quality depends on both the quality of the estimates (distances/discounts, rewards) and the quality of the policy. Note that the theorem guarantees accuracy to the solution of the overall problem conditioned on a snapshot of the current policy, which should evolve towards optimal during training. We need to interpret this carefully: 1) bad policies with good estimation will lead to an accurate yet potentially bad overall solution; 2) No matter the quality of the policy,

with a bad estimation, it may result in a poor estimate of solutions; 3) It is expected that a near-optimal policy with good estimations will lead to a near-optimal solution.
These amplify the necessity to be careful about if the learning rules for the estimates and the policy could converge to the correct values (Sec. 5.3.1, Page. 91).

Applicable Scenarios due to Assumptions

Thm. 1 indicates that once the agent achieves high accuracy estimation of the model for the proxy problem and a near-optimal lower-level policy π, it converges toward optimal performance.
Despite the theorem's generality, in the experiments, we limit ourselves to the scope where the accuracy assumption can be met non-trivially, *i.e.*, while avoiding degenerate proxy problems whose checkpoint transitions involve no rewards, that is, in tasks with sparse terminal rewards (*e.g.*, `RDS` and `SSM`), where the goals are included as permanent vertices in the proxy problems. This is a case where the accuracy assumption can be met non-trivially, *i.e.*, while avoiding degenerate proxy problems whose edges involve no rewards. Following Thm. 1, we design the Skipper framework to implement the construction and planning over proxy problems.
In Sec. 7.2.2 in Chap. 7 (Page. 142), we will present some detailed discussion regarding the limitation brought by the assumptions used in the analyses.

5.3 Methodology: Skipper Framework - Spatially & Temporally Abstract Planning

We now look into the design of Skipper - a framework that implements the task decomposition provided by **proxy** problems. The design of the framework has the following highlights:

- **Decision-time planning** is used for its ability to generalize better in novel situations;
- **Spatio-temporal abstraction**: temporal abstraction breaks down the given task into smaller ones, while spatial abstraction over the state features improves local generalization;
- **Higher quality proxies**: we introduce pruning techniques to improve the quality of sub-problems;
- **Learning end-to-end from hindsight, off-policy**: to maximize sample efficiency and the ease of training, we propose to use auxiliary (off-)policy methods for edge estimation, and learn a context-conditioned checkpoint generation, both from hindsight experience replay;

Designing Skipper requires solving two big problems, which are explained as follows, corresponding to the edges and the vertices of the proxy problems, respectively.

5.3.1 Problem 1: Edge Estimation

First, we discuss how Skipper provides the edges needed to assemble the proxy problems. In this sub-section, we assume that the vertices of the proxy problems are already given. We start the discussions on edges

because it is easier to understand and can provide intuitions about the more difficult problem of how Skipper can construct the vertices of the proxy problems.

Spatial Abstraction

Each edge in the proxy problem defines a sub-problem. Some sub-problems are good and desirable, and some are not. Sub-problems are characterized by their locality: transitioning from one checkpoint to another often only involves limited few aspects of the environmental state. It is thus crucial to use inductive biases that can take advantage of the locality instead of using generic neural network architectures.

To parallel with and differentiate from temporal abstraction, we call the behavior pattern of an agent to selectively focus on parts of the environment relevant for the sub-problem - "spatial abstraction". Spatial abstraction is thus core to the previous chapter.

Inspired by conscious information processing in brains (Dehaene et al., 2020), the intuitions gathered in the previous chapter, and existing approach in Sylvain et al. (2020), we introduce a local perceptive field selector, σ, consisting of an attention bottleneck that (soft-)selects the top-k local segments of the full state (*e.g.*, a feature map by a typical convolutional encoder); all segments of the state compete for the k attention slots, *i.e.*, irrelevant aspects of state are discouraged or discarded, forming a *partial* state representation (Mott et al., 2019; Tang et al., 2020; Zhao et al., 2021; Alver and Precup, 2023). More specifically, this soft-selection mechanism is implemented using a top-down attention mechanism that queries the set of local receptive fields extracted by the CNN. The $h \times w \times c$ feature map is "sliced" and transformed into a set containing $h \times w$ local unordered feature objects, each of dimension c, with each of these vectors being a local receptive field on top of a local patch of the observation (while in Chap. 4, each vector maps to exactly one cell in the gridworld). The top-down mechanism conditions itself on the sub-problem, *i.e.*, the intention to transition from the current state to the target state. The output of the soft-selection is a vector that aggregates the information from the top-k most relevant local patches of the CNN features. For more details regarding top-k and top-down attention mechanisms, please see Sec. 3.2.2 on Page. 37.

This method of selecting top-k receptive fields can be viewed as a more relaxed approach, generalizing the top-down attention controlled bottleneck with set-based representations in Chap. 4 to be compatible with general visual inputs processed by Convolutional Neural Networks (CNNs) (LeCun et al., 1989) without the need for set-based state representation encoding.

We provide a visualization of how spatial abstraction works in Fig. 5.2. Through σ, the auxiliary estimators, to be discussed soon, force the bottleneck mechanism to promote aspects relevant to the local estimation of connections between the checkpoints. The rewards and discounts are then estimated from the partial aspects of the environmental state relevant to the sub-problem at hand, conditioned on the agent's policy.

Checkpoint-Achieving Policy

The checkpoint-achieving policy implements π. We ask Skipper to obtain the low-level policy π by maximizing an intrinsic reward, which incentivize the achievement of a target checkpoint $S^{\odot}$. In this approach, the choice of intrinsic reward is flexible; for example, one could use a reward of $+1$ when S_{t+1} is within a small radius of $S^{\odot}$ according to some distance metric, or use reward-respecting intrinsic rewards that enable

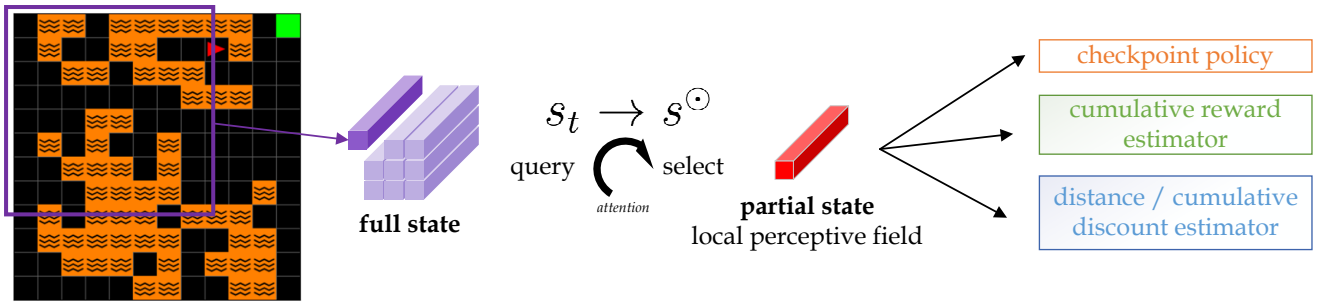


Figure 5.2: ***Illustration of the Spatial Abstraction Mechanism****: the top-down semi-hard attention is conditioned on an encoding of the checkpoint transition, established over two state-like representations.*

more sophisticated behaviors, as in (Sutton et al., 2023). In the following, for simplicity, we will denote the checkpoint-achievement condition with equality: $S_{t+1} = S^{\odot}$.

Relationship Estimators

Proxy problems require that the relationship estimates between checkpoints be capability-aware, *i.e.*, conditioned on the low-level policy π. Thus, the learners must be off-policy compatible.
With the consideration in mind, we designed learning rules that estimate the relationship between checkpoints with auxiliary reward signals that need not be tuned for each application to different tasks (Zhao et al., 2020). Notably, these estimates are learned with C51-style distributional RL (Bellemare et al., 2017), where the output of each estimator takes the form of a histogram over scalar support (Sec. 3.1.2, Page. 32).

Cumulative Reward following the design of proxy problems, the first interesting edge estimate to learn is the cumulative-discounted task reward v^{π}_{ij} along the possible trajectories leading to the target checkpoint s_j under π from a starting state s_i. We learn this by policy evaluation on an auxiliary reward that is the same as the original task reward everywhere except when reaching the target. Given a hindsight sample $\langle x_t, a_t, r_{t+1}, x_{t+1}, x^{\odot} \rangle$ and the corresponding encoded sample $\langle s_t, a_t, r_{t+1}, s_{t+1}, s^{\odot} \rangle$, we train V_{π} with KL-divergence as follows:

$$\begin{aligned} &\hat{v}_{\pi}(s_t \rightarrow s^{\odot}, a_t) \equiv \hat{v}_{\pi}(\sigma(s_t|s_t \rightarrow s^{\odot}), a_t|\sigma(s^{\odot}|s_t \rightarrow s^{\odot})) \\ &\leftarrow \begin{cases} R(s_t, a_t, s_{t+1}) + \gamma \hat{v}_{\pi}(\sigma(s_{t+1}|s_{t+1} \rightarrow s^{\odot}), a_{t+1}|\sigma(s^{\odot}|s_{t+1} \rightarrow s^{\odot})) & \text{if } s_{t+1} \neq s^{\odot} \\ R(s_t, a_t, s_{t+1}) & \text{if } s_{t+1} = s^{\odot} \end{cases} \end{aligned} \tag{5.18}$$

where $\sigma(s|s_t \rightarrow s^{\odot})$ is the spatially-abstracted from the full state s given intention to go from s to $s^{\odot}$, and $a_{t+1} \sim \pi(\cdot|\sigma(s_{t+1}|s_{t+1} \rightarrow s^{\odot}), \sigma(s^{\odot}|s_{t+1} \rightarrow s^{\odot}))$. Similar methods on estimating the reachability between states also exist in literature, however, without the considerations for spatial-abstraction.

Cumulative Distances / Discounts Besides v_{π}, the second important estimate for a checkpoint transition is the discounts γ_{π}, which is needed for proxy problems.

Notably, learning the cumulative discount leading to $s_\odot$ under π is numerically difficult, even with C51, because the target values are heavily skewed towards 1 if $\gamma \approx 1$. Note that this is not to say that reward estimation in Eq. 5.18 is numerically simple, since reward functions can be diverse.

We propose a way to bypass the numerical difficulties by take a detour to instead effectively estimating cumulative (truncated) distances (or trajectory length) under π. Similar to how we learn the cumulative rewards $\hat{v}_\pi$, such distances can be learned also with policy evaluation, where the auxiliary reward is $+1$ on every transition, except at the targets:

$$\begin{aligned}
&\hat{D}_\pi(s_t \to s^\odot, a_t) \equiv \hat{D}_\pi(\sigma(s_t|s_t \to s^\odot), a_t|\sigma(s^\odot|s_t \to s^\odot)) \\
&\leftarrow \begin{cases} 1 + \hat{D}_\pi(\sigma(s_{t+1}|s_{t+1} \to s^\odot), a_{t+1}|\sigma(s^\odot|s_{t+1} \to s^\odot)) & \text{if } s_{t+1} \neq s^\odot \\ 1 & \text{if } s_{t+1} = s^\odot \\ \infty & \text{if } s_{t+1} \text{ is terminal and } s_{t+1} \neq s^\odot \end{cases}
\end{aligned} \tag{5.19}$$

where $a_{t+1} \sim \pi(\cdot|\sigma(s_{t+1}|s_{t+1} \to s^\odot), \sigma(s^\odot|s_{t+1} \to s^\odot))$.

Recovering Discounts from Distances With the learned distribution of cumulative distances by a C51 architecture (Dabney et al., 2018), the cumulative discount can then be extracted by swapping the support of the output histogram with the corresponding discounts values. This process is illustrated in Fig. 5.3. This bypasses the problem of $\mathbb{E}[\gamma^D] \neq \gamma^{\mathbb{E}[D]}$, because the probability of having a trajectory length of 5 under policy π from state s_t to $s_\odot$ is the same as a trajectory having discount γ^5.

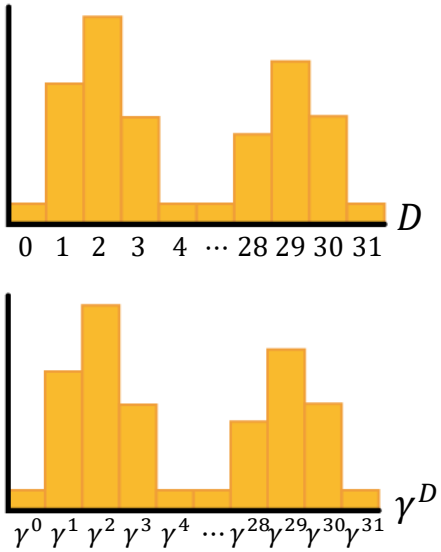


Figure 5.3: ***Simultaneously Estimating Distributions of Discounts & Distances****: by swapping the support with the corresponding discount values, the distribution of the cumulative discount can be inferred from the output histogram of cumulative distances. Vice versa.*

The learned distances are not only used to recover the discounts needed for proxy problems, but are also used to prune unwanted checkpoints to simplify the proxy problem via k-medoids pruning (Sec. 5.3.2, Page. 94), as well as pruning far-fetched edges (Sec. 5.3.2, Page. 95), both to be introduced soon.

Justifying Update Rules for Edge Estimation

As previously analyzed, the quality of the estimations of the cumulative rewards and the cumulative discounts during checkpoint transitions determine if effective solutions of the proxy problems can be found.

Thus, we want to understand if the update rules proposed previously would lead to high-quality estimates that the proxy problems need.
The cumulative reward and cumulative discount are estimated by applying policy evaluation given π, on the two sets of auxiliary reward signals, respectively.
The following equality for the cumulative discounted reward random variable can be shown:

$$V_\pi(s_t, a_t|s^\odot) = R(s_t, a_t, S_{t+1}) + \gamma V_\pi(S_{t+1}, A_{t+1}|s^\odot) = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} R(S_\tau, A_\tau, S_{\tau+1}), \tag{5.20}$$

where $S_{t+1} \sim p(\cdot|s_t, a_t)$, $A_{t+1} \sim \pi(\cdot|S_{t+1}, s^\odot)$. Let $V_\pi(s|s^\odot) \coloneqq V_\pi(s, A|s^\odot)$ with $A \sim \pi(\cdot|s, s^\odot)$. Noticing that $V_\pi(S_{t+1}, A_{t+1}|s^\odot) = 0$ if $S_{t+1} = s^\odot$, the equivalence is established towards Rule. 5.18 (Page. 90).
Similarly, we connect the cumulative discount random variable with Rule. 5.19 (Page. 91). Let $\Gamma_\pi(s_t|s^\odot) \coloneqq \Gamma_\pi(s_t, A_t|s^\odot)$ with $A_{t+1} \sim \pi(\cdot|S_{t+1}, s^\odot)$:

$$\Gamma_\pi(S_t, A_t|s^\odot) = \gamma \cdot \Gamma_\pi(S_{t+1}, A_{t+1}|s^\odot) = \gamma^{T_\perp - t} \mathbb{I}\{S_{T_\perp} = s^\odot\} \tag{5.21}$$

where $T_\perp$ denotes the timestep when the trajectory terminates, and with $\Gamma_\pi(S_{t+1}, A_{t+1}|s^\odot) = 1$ if $S_{t+1} = s^\odot$ and $\Gamma_\pi(S_{t+1}, A_{t+1}|s^\odot) = 0$ if $S_{t+1} \neq s^\odot$ is terminal.

5.3.2 Problem 2: Vertex Generation

Proxy problems can only be constructed with both the vertices and the edges in hand. Since we do not assume the access to an oracle, to be able to use proxy problems, the vertices (checkpoint states) must be generated using a learned model.
Different from most model-based approaches that utilize a predictive model to simulate the changes of the environment, we instead propose a temporally abstract checkpoint generator which aims to directly imagine the distant future states *without needing to know how exactly the agent might reach them nor worrying about if they are reachable.* The feasibility of checkpoint transitions will be handled by the connection estimates instead. In the language of some existing literature, the generator here can be called a "goal generator". We also discuss the implication of such approaches in target-assisted frameworks in Sec. 6.2.1 on Page. 114.

Checkpoint Generator: Split Context & Partial Descriptions

We want to design a checkpoint generator which generalizes well across diverse tasks, *i.e.*, propose useful checkpoints even in novel scenarios. This means the generator should be able to capture the underlying causal mechanisms across tasks (a challenge for existing model-based methods) (Zhang et al., 2020). For this, we propose that the checkpoint generator learns to split the state representation into two parts: 1) an episodic / task **context** and 2) a **partial description**.
We design the contexts and partial descriptions to coordinate as follows: in different contexts, the same partial description could correspond to different states. However, within a given context, the same partial description should always map to the same state. In other words, the partial description should capture the

defining distinct feature of a certain state, while the combination of the context with the partial description recovers the information of a full state.

In a navigation problem, for example, as in Fig. 5.4, a context could be a representation of the layout of the gridworld (of a specific task instantiation), and the partial description could represent the 2D-coordinates of the agent's location.

Note that the representations of partial descriptions do not necessarily capture the relevant transitional information between two states. This is why the design of partial description / context inside the generator should not be confused with the partial states created by spatial abstraction in Sec. 5.3.1 on Page. 89.

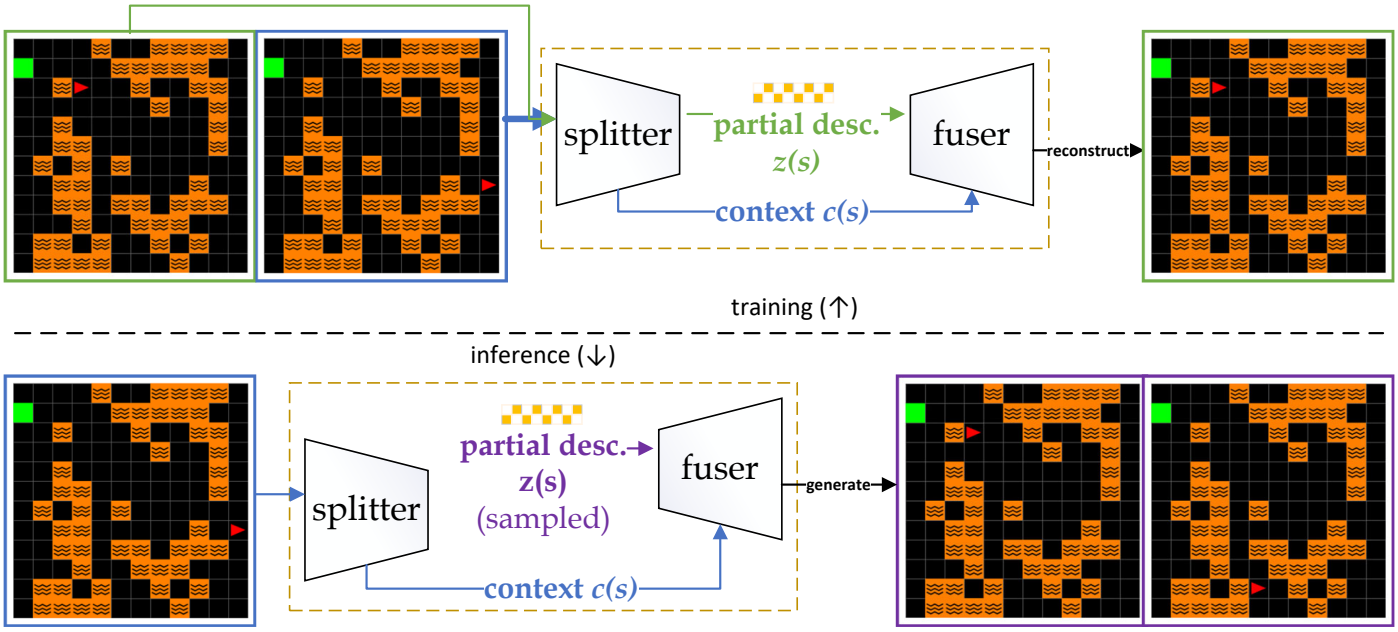


Figure 5.4: ***Checkpoint Generator Training & Inference****: during training, both x_t (current observation) and $x^{\odot}$ (relabeled target observation) participate in the calculation of the training losses. The partial description is extracted from $x^{\odot}$, while the context is extracted from x_t. Both are extracted by the same two-headed neural network, marked as the "splitter". The training losses seek to reconstruct $x^{\odot}$, by making the model learn that with a context from any current state, e.g., that of x_t, $x^{\odot}$ should be reconstructed as long as its corresponding partial description is used; during inference, the context from the current observation is used together with sampled partial descriptions to generate checkpoints.*

As shown in Fig. 5.4, the proposed information split is achieved by an inductive bias of two parameterized trainable functions: the *splitter* $\mathcal{E}_{CZ}$, mapping the input s into a representation of a context $c(s)$ and a partial description $z(s)$, as well as the *fuser* $\bigoplus$ that recovers s from the extracted $\langle c, z \rangle$. To achieve consistent context extraction across states in the same episode, at training time, we force the context to be extracted from other states in the same episode, instead of the input (upper pass in Fig. 5.4).

Crucial for grounding the two representations into reality and stabilizing the hierarchical planning framework that is Skipper, we rely on a reconstruction loss, despite having previously criticized this methodology in Sec. 3.3.3 and having avoided it in the previous work. As previously discussed, in Skipper, we rely on a generative model instead of a predictive model, which relies more on techniques such as reconstruction. A reconstruction loss can be established on either the observation-level or a learned state level, depending on

the need of the application. With full-observability, when the observation space is simpler than the state space to learn, we would recommend the reconstruction there, given the environment is fully observable.

The two representations are learned by the generator, powered by a conditional Variational Auto-Encoder (CVAE) architecture (Sec. 3.2.1, Page. 35) from hindsight-labeled targets (Sec. 3.1.3, Page. 34). The generator learns a distribution $p(s^{\odot}|C(s_t)) = \sum_z p(s^{\odot}|C(s_t), z)p(z|C(s_t))$, where $C(s_t)$ is the extracted context from s_t and zs are the partial descriptions. We train the generator by minimizing the evidence lower bound on $\langle s_t, s^{\odot} \rangle$ pairs chosen with HER. The processes of training and inference with the proposed checkpoint generator are illustrated in Fig. 5.4.

With hindsight relabeling, we may need a more careful control over the distribution of the future checkpoints, which we seek to learn with the generator. This distribution, unlike the outcome of 1-step transition, could be more diverse and non-stationary given a changing behavior policy, depending on the relabeling strategies (Andrychowicz et al., 2017). In our implementation, the targets are labeled with the EPISODE relabeling strategy. Improper hindsight relabeling, such as using EPISODE exclusively as here, can cause delusional behaviors in target-assisted agents such as Skipper. We will discuss this issue in detail in the next chapter.

Similar to the discrete and categorical architectures used in Hafner et al. (2024); Hansen et al. (2024), we constrain the partial description in the forms of bundles of binary variables, which requires the techniques of straight-through gradient estimators (Bengio et al., 2013). By design, these partial description latents can be easily sampled or composed during inference, *i.e.*, decision-time planning. Compared to models such as that in Director (Hafner et al., 2022), which generates intermediate sub-goals on the on-policy trajectory, our proposed generator can generate and handle a more diverse distribution of states, beneficial for generalization in novel scenarios.

The simple representations of the partial descriptions also granted us the simplicity to represent the sub-problems with a simple concatenation of the partial descriptions of two checkpoints. This is used in the top-down attentions for the spatial abstraction mechanism.

Pruning with k-medoids

In this chapter, the proposed conditional generation model is limited in the sense that they are return-unaware. Without learning how to be return-aware, the proxy problem can instead be improved by making it more sparse, and making the proxy problem vertices more evenly spread in state space. To achieve this, we propose a pruning process based on the classical k-medoids clustering algorithm (Kaufman and Rousseeuw, 1990), which only requires pairwise distance between states as input. During the construction of a proxy problem, a larger number of checkpoints are first generated. Then, these checkpoints are clustered using k-medoids, after which all checkpoints but the cluster centers are discarded. The cluster centers are always real generated checkpoints instead of imaginary weighted sums of state representations that will be produced by clustering algorithms such as k-means (Lloyd, 1982).

Notably, for sparse reward tasks, the generator cannot guarantee the presence of the rewarding checkpoints, such as the task goal, to be in the proposed proxy problem. We could remedy this by explicitly learning the generation of the rewarding states with another conditional generator. These rewarding states should be kept as vertices (immune from pruning).

We present the pseudocode of the modified k-medoids algorithm for pruning overcrowded checkpoints in Alg. 2. The changes upon the original k-medoids algorithm are marked in purple, which correspond to a forced preservation of the special input data points as cluster centers: when k-medoids is called after the unpruned graph is constructed, $\mathcal{S}_\vee$ is set to be the set containing the goal state only. This is intended to make the remaining cluster centers (selected checkpoints) span more uniformly in the state space, while preserving the goal. This is particularly proper if the proxy problems are going to be conserved throughout the episode and reused.

There are some additional technical details we need to pay attention to, particularly because k-medoids was designed to handle undirected distances. Let the estimated distance matrix be D, where each element d_{ij} represents the estimated trajectory length it takes for π to fulfill the transition from checkpoint i to checkpoint j. Since k-medoids cannot handle infinite distances (*e.g.*, from a terminal state to another state), to still maximize for the span, the distance matrix D is truncated, and then we take the elementwise minimum between the truncated D and D^T to preserve the one-way distances. The matrix containing the elementwise minimums would be the input of the pruning algorithm.

Algorithm 2: Checkpoint Pruning with k-medoids

Data: $X = \{x_1, x_2, \ldots, x_n\}$ (state indices), D (estimated distance matrix), $\mathcal{S}_\vee$ (states that must be kept), k (#checkpoints to keep)
Result: $\mathcal{S}_\odot \equiv \{M_1, M_2, \ldots, M_k\}$ (checkpoints kept)

Initialize $\mathcal{S}_\odot \equiv \{M_1, M_2, \ldots, M_k\}$ randomly from X
make sure $\mathcal{S}_\vee \subset \mathcal{S}_\odot$
repeat
  Assign each data point x_i to the nearest medoid M_j, forming clusters $C_1, C_2, \ldots, C_k$;
  foreach *medoid* M_j **do**
    Calculate the cost J_j of M_j as the sum of distances between M_j and the data points in C_j;
  Find the medoid M_j with the lowest cost J_j;
  if M_j *changes* **then**
    make sure $\mathcal{S}_\vee \subset \mathcal{S}_\odot$
    Replace M_j with the data point in C_j that minimizes the total cost;
until *Convergence (no cost improvement)*;

In addition to vertex pruning, we also prune the edges according to a distance threshold, *i.e.*, all edges with estimated distance over the threshold are deleted from the complete graph of the pruned vertices. This is enabled by how we interchangeably learned the cumulative discounts and distances. Such approach biases the plans, *i.e.*, the solutions of the proxy problems, towards shorter-length, smaller-scale sub-problems, as distant checkpoints are difficult for π to achieve.

5.3.3 Skipper Framework: Assemble Everything

An overall view of the proposed Skipper framework with all components assembled is illustrated in Fig. 5.5. The pseudocode of Skipper is provided in Alg. 3, together with the hyperparameters used in later experiments.

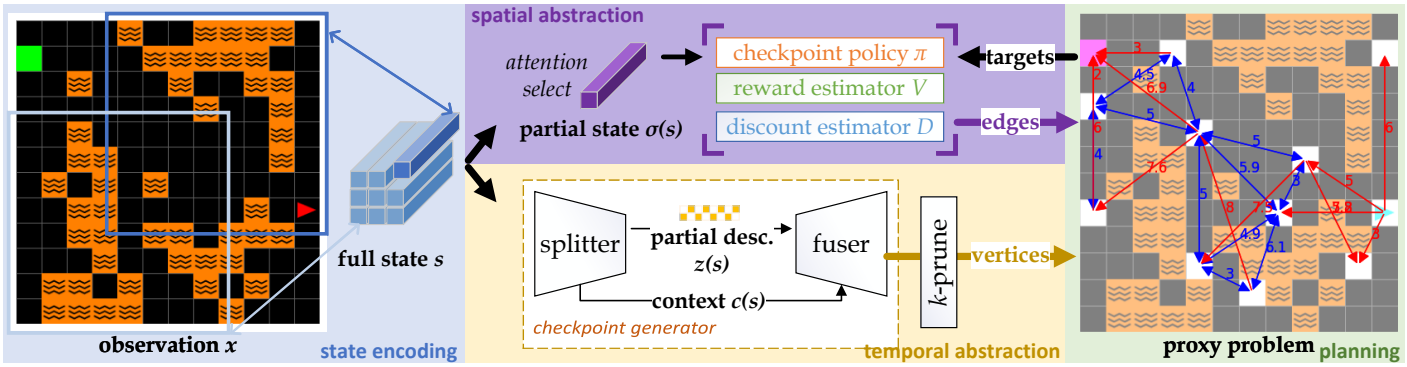


Figure 5.5: Skipper ***Framework****: 1) Partial states, used for low-level policy and edge estimation, consist of a few local perceptive fields, soft-selected via top-k attention (Gupta et al., 2021). Skipper's edge estimations and low-level behaviors π are based on the partial states. 2) The checkpoint generator learns by splitting the full states into contexts and partial descriptions, and fusing them to reconstruct the input. It imagines checkpoints by sampling partial descriptions and combining them with the episodic contexts; 3) We prune the vertices and edges of the denser graphs to extract sparser proxy problems. Value iteration can be used to solve the proxy problems and the solutions are equivalent to plans in the original problem. Each step in the plan becomes a sub-problem. In the proxy problem example, blue edges are estimated to be bidirectional and red edges have the other direction pruned.*

Algorithm 3: Skipper with Random Checkpoints (implementation choice in purple)

for *each episode* **do**
  // — start of the **subroutine** to construct the proxy problem
  generate more than necessary (32) checkpoints by sampling from the partial descriptions given the extracted context from the initial state;
  ($k = 12$)-medoids pruning upon estimated distances among all checkpoints; // prune vertices
  use estimators to annotate the edges between the nodes (including a terminal state estimator to correct the estimates);
  prune edges that are too far-fetched according to distance estimations (threshold set to be 8, same as replan interval); // prune edges
  // — end of the **subroutine** to construct the proxy problem
  for *each agent-environment interaction step until termination of episode* **do**
    if *decided to explore (DQN-style annealing ϵ-greedy)* **then**
      take a random action;
    else
      if *abstract problem just constructed* **or** *a checkpoint / timeout reached ($\geq$ 8 steps since last planned)* **then**
        [**OPTIONAL**] call the **subroutine** above for Skipper-**regen**;
        run value iteration (for 5 iterations) on the proxy problem, select the target checkpoint;
      follow the action suggested by the checkpoint-achieving policy;
    if *time to train (every 4 actions)* **then**
      sample hindsight-relabelled transitions and train checkpoint-achieving policy, checkpoint generator & estimators (including a terminal state estimator);
    save interaction into the trajectory experience replay;
  convert trajectory into HER samples (relabel 4 random states as additional goals);

5.4 Discussions of Methodologies

5.4.1 About Proxy Problems

The representation of a proxy problem draws similarity to an early work on landmark-based approximate value iteration (Mann et al., 2015).

Concurrently, (Zadem et al., 2024) introduced a three-level hierarchical planning method aimed at decomposing an overall task into abstract goal regions, with sub-steps defined over goal transitions; Lo et al. (2024) proposed to decompose the task by goal-space MDPs, essentially proxy problems, and conduct background planning (Sec. 3.3.2, Page. 47) to improve performance.

Similar to Skipper, whole sets of relationship estimation learning, *e.g.*, for reachability, between states and goals are proposed and analyzed in both works. Different from Skipper, both works were not motivated from OOD generalization and focused on a classical single-task RL setting.

For Goal Space Planning (GSP) from Lo et al. (2024) specifically, extensive studies were done on multiple classical single-training task environments, spanning from tabular methods to deep RL. Lo et al. (2024) was motivated by the fact that background planning methods often behave worse than the model-free baselines due to the generation of invalid states (a behavior named **hallucination**, to be discussed in Sec. 6.3 on Page. 116). GSP sought to address the performance issue by constraining background planning to a given set of (abstract) subgoals. The method applied can be viewed from the lens of target-assisted frameworks and is highly similar to our mitigation strategies in Chap. 6 on Page. 112.

For methods that abstract the original MDP with representative states, such as proxy problems, planning can be efficient, while the resulting policy could be suboptimal. Lo et al. (2024) argued that the sub-optimality issue forces a trade-off between increasing the size of the abstract MDP (to increase the policy's expressivity) and increasing the computational cost of decision-time planning. We acknowledge such limitation to the Skipper framework (and so for other decision-time methods such as Zadem et al. (2024)) and agree it may be a good motivation for using background planning in a classical single-task focused RL setting. On a high-level, proxy problems indeed trade optimality for robustness. We presented the framework's sensitivity to the number of checkpoints in proxy problems in Sec. 5.5.5 on Page. 110. While acknowledging the shortcomings of the methodology of Skipper, we have to point out that background planning methods, including GSP, assumes that a know-all do-all policy can be effectively learned, which is rarely the case in practice. In the analyses in Sec. 5.2.2 (Page. 85), we explicitly discussed why we assumed no such condition.

While both Zadem et al. (2024) and Lo et al. (2024) require oracles to work, Skipper operates at mainly the state-level and is able to generate the checkpoints, *i.e.*, the subgoals, autonomously.

5.4.2 About Task Decomposition with Selected States / Sub-Goals

Some early works such as McGovern and Barto (2001) suggested to use bottleneck states to abstract given tasks into manageable steps. In Kim et al. (2021), promising states to explore are generated and selected with shortest-path algorithms. Similar ideas were attempted for guided exploration (Erraqabi et al., 2022; Kulkarni et al., 2016).

Similar to Hafner et al. (2022), Czechowski et al. (2021) generate fixed-steps ahead subgoals, while Bagaria et al. (2021) augments the search graph by states reached fixed-steps ahead. Nasiriany et al. (2019); Xie et al. (2021); Shi et al. (2022) employ CEM to plan a chain of subgoals towards the task goal (Rubinstein, 1997).
Similar to our approach, Nair et al. (2018); Florensa et al. (2018) use models to imagine subgoals while Eysenbach et al. (2019) search directly on the experience replay. Skipper utilizes proxy problems to abstract the given tasks via spatio-temporal abstractions (Bagaria et al., 2021). Checkpoints can be seen as sub-goals that generalize the notion of "landmarks" or "waypoints" in Sutton et al. (1999); Dietterich (2000); Savinov et al. (2018). Zhang et al. (2021) used latent landmark graphs as high-level guidance, where the landmarks are sparsified with weighted sums in the latent space to compose subgoals. In comparison, our checkpoint pruning selects a subset of generated states, which is less prone to issues created by weighted sums.

5.4.3 About Temporal Abstraction

Somewhat similar to the idea behind attention, choosing a checkpoint target is a selection towards certain decision points in the dimension of time, *i.e.*, a form of temporal abstraction. Constraining options, Skipper learns the options targeting certain "outcomes", which dodges the difficulties of option collapse (Bacon et al., 2017) and option outcome modeling by design. It should be acknowledged that the constraints shift the difficulties to generator learning (Silver and Ciosek, 2012; Tang and Salakhutdinov, 2019). We expect this shift to bring benefits in cases where states are easy to learn and generate, and / or in stochastic environments where the outcomes of unconstrained options are difficult to learn. Constraining options was also investigated in unsupervised settings (Sharma et al., 2020).

5.4.4 About Spatial Abstraction

Spatial abstraction is a special, dynamic kind of "state abstraction" (Sacerdoti, 1974; Knoblock, 1994). When state abstraction is being done in a non-dynamic way, it roughly equates to a synonym for state space partitioning due to the focus (Li et al., 2006). Spatial abstraction, characterized to capture the behavior of conscious planning in the spatial dimension, focuses on the **intra-state** partial selection of the environmental state for decision-making. It corresponds naturally to the intuition that state representations should contain useful aspects of the environment, while not all aspects are useful for a particular intent (sub-problem). Efforts toward spatial abstraction are traceable to early hand-coded proof-of-concepts proposed in Dietterich (2000). In Zadaianchuk et al. (2021); Fu et al. (2021); Shah et al. (2022), 3 more recent model-based approaches, spatial abstractions are attempted to remove visual distractors.
Until only recently, attention mechanisms had primarily been used to construct state representations in model-free agents for sample efficiency purposes, overlooking their potentials for generalization (Mott et al., 2019; Manchin et al., 2019; Tang et al., 2020). Emphasizing on generalization, our previous work (Zhao et al., 2021) used spatially-abstract partial states in decision-time planning, as introduced in Chap. 4. We proposed an attention bottleneck to dynamically select a subset of environmental entities during the atomic-step forward simulation, without explicit goals provided as in Zadaianchuk et al. (2021). Skipper's checkpoint transition is a step-up from our old approach, where we now show that spatial abstraction, an overlooked missing

flavor, is as crucial for longer-term planning, arguably as important as temporal abstraction (Konidaris and Barto, 2009).
Concurrently, GSP sought to constrain background planning to a given set of (abstract) subgoals and learning only local, subgoal-conditioned models (Lo et al., 2024). To a degree, the claimed compatibility to learn only local and sub-goal conditioned models is only materialized with our implemented spatial abstraction mechanisms, in this chapter.

5.4.5 About Planning Estimates

Hierarchical planning must be established over estimates that can guide the low-level behaviors. Zhang et al. (2021) proposed a distance estimate with an explicit regression. With Temporal-Difference Models (TDMs) (Pong et al., 2020), LEAP (Nasiriany et al., 2019) embraces a sparse intrinsic reward based on distances to the goal states. Contrasting with our distance estimates, there is no empirical evidence of TDMs' compatibility with stochasticity and terminal states. Eysenbach et al. (2019) employs a similar distance learning technique to extract the shortest path distance between states present in the experience replay; while our proposed estimators learn the distance conditioned on evolving policies. Similar aspects were also investigated in Nachum et al. (2018).

5.5 Research Findings: Experiments

The primary goal of our experiments is to understand how the proposed Skipper framework can bring advantages of (zero-shot) generalization. To fully understand the results, we must have precise control of the difficulty of the training and evaluation tasks. Also, to validate if the empirical performance of our agents matches the formal analyses (Thm. 1), we need to know how close to the (optimal) ground truth our edge estimations and checkpoint policies are. These objectives lead to the need for environments whose ground truth information (optimal policies, true distances between checkpoints, *etc.*) can be computed. For these reasons, we chose to employ `RDS` as our main experiment environments.
Despite that the experiments in this chapter employ essentially the same kind of environment as the previous chapter, the experimental settings are quite different.
Across all experiments, we sample training tasks from an environment distribution of difficulty $\delta = 0.4$: each cell in the field has probability 0.4 to be filled with lava while guaranteeing a path from the initial position to the goal. The evaluation tasks are sampled from a gradient of OOD difficulties - 0.25, 0.35, 0.45 and 0.55, where the training difficulty acts as mean[1].
We have taken measures to accelerate the experiments and isolate challenges from the aspects of exploration. During evaluation, the agent is always spawned at the opposite side from the goals. During training, the agent's position is uniformly initialized to speed up training. We provide results for non-uniform training

[1]This is an updated setting from that used in Chap. 4 (Sec. 4.5, Page. 68). The training difficulty of $\delta = 0.4$ now is OOD from those of evaluation and is right in the middle of the OOD evaluation difficulty range. The transposed OOD evaluation instances are no longer used because without it 1) the OOD setting becomes closer to the distribution shfit setting commonly used; and 2) the performance discrepancy between the evaluation curves on the training tasks and the evaluation tasks can directly reflect the generalization gap. See Sec. 3.4.2 on Page. 54 for more discussions.

initialization in the later experiments. This change means that the agents can be trained for only 1.5×10^6 interactions to achieve convergence.
Meanwhile, to increase generalization difficulty for long(er) term planning, we conduct experiments done on large, 12×12 maze sizes, compared to the maximum 10×10 used in Chap. 4. The compared agents include:

Skipper-**once**: A Skipper variant that generates one proxy problem at the start of the episode, and the replanning (choosing a checkpoint target based on the existing proxy problem) only triggers a quick re-selection of the immediate checkpoint target;

Skipper-**regen**: A variant that re-generates a proxy problem when replanning is triggered;

modelfree: A model-free baseline agent sharing the same base architecture with the Skipper variants - a prioritized distributional DDQN (Dabney et al., 2018; Hasselt et al., 2016);

Director: An adapted Director agent (Hafner et al., 2022). Since Director discards trajectories that are not long enough for training purposes, we give it extra agent-environment interaction budget to make sure that the same amount of training data is gathered as for the other agents;

LEAP: An adapted LEAP agent for discrete action spaces. We waived the interaction costs for its generator pretraining stage, only showing the second stage of RL pretraining.

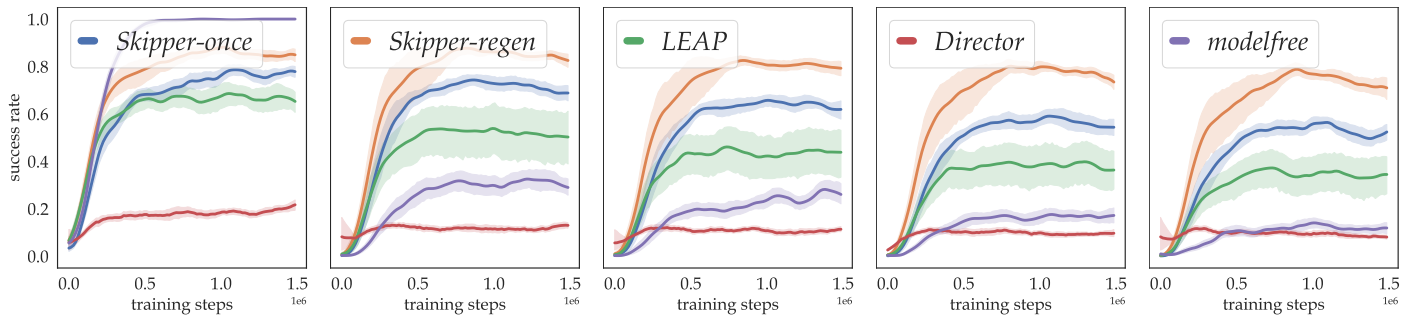
Please refer to Sec. 8.2 on Page. 161 for more details regarding implementation of these agents.

5.5.1 Generalization Performance with 50 Training Environments

The main set of experiments is focused on how well the compared agents could generate after training on 50 tasks, a representative configuration of different numbers of training tasks including $\{1, 5, 25, 50, 100, \infty\}$[2] Fig. 5.6 shows how the agents' performance (measured by the episodic task success rates) evolves during training. These results are obtained with 50 fixed training tasks (different 50 task instances for each independent seed run, with each task instance generated at $\delta = 0.4$). In Fig. 5.6 **a)**, we observe how well an agent performs on its training tasks (note the differences in initialization between an evaluation episode and a training episode). If an agent performs well here but badly in **b)**, **c)**, **d)** and **e)**, like the **modelfree** baseline, then we suspect that it overfitted on training tasks, likely indicating a reliance on memorization (Cobbe et al., 2020). This was also discussed in the previous chapter.
Due to the non-overlapping confidence intervals, we observe a statistically significant advantage in the generalization performance of the Skipper agents throughout training, compared to the other methods. It is worth noting that the **regen** variant exhibits significant performance advantages over all other methods, including the **once** variant. This is likely because, the frequent reconstruction of the graph makes the agent less prone to being trapped in a low-quality proxy problem, and thus provides extra adaptability in novel scenarios.
During training, it should be acknowledged that Skipper variants behave less optimally than expected, despite the strong generalization on evaluation tasks. As our later ablation results show, concurring with our

[2] ∞ training tasks mean that an agent is trained on a different task for each episode. In reality, this may lead to prohibitive costs in creating the training environment.

previous theoretical analyses, such a phenomenon is an outcome of inaccuracies both in the proxy problem and the checkpoint policy. One major symptom of an inaccurate proxy problem is that the agent would chase delusional targets. We address this behavior in the next chapter.



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.6: ***Evaluation Performance Evolution of Compared Agents during Training****: the x-axes correspond to training progress, while the aligned y-axes represent the success rate of episodes (optimal is 1.0). Each agent is trained with 50 tasks. Each data point is the average success rate over 20 evaluation episodes, and each error bar (95% confidence interval) is processed from 20 independent seed runs. Training tasks performance is shown in a) while OOD evaluation performance is shown in b), c), d), e).*

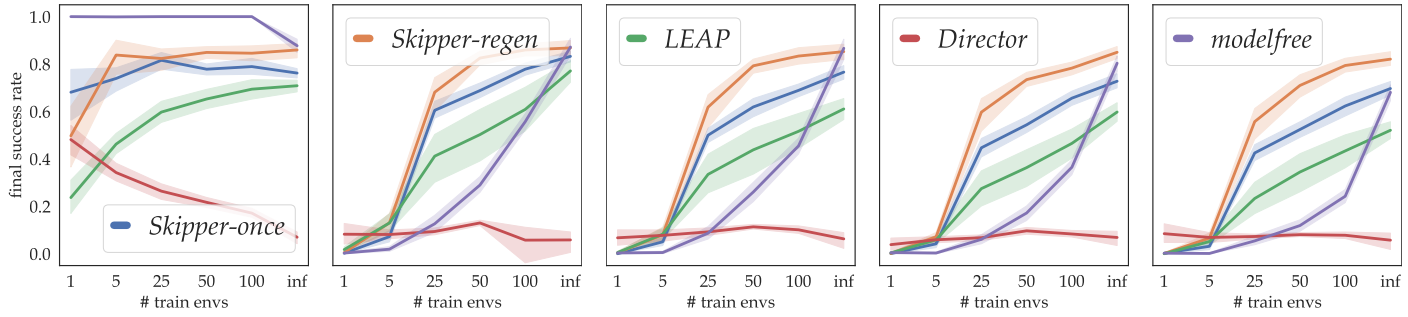
Performing better than the **modelfree** baseline, LEAP obtains reasonable generalization performance, that is, if we are willing to waive the extra budget it needed for pretraining. In the next chapter, we show that LEAP benefits largely from removing hallucinated state targets, which validates that optimizing for a path in the latent space may be prone to errors caused by delusional subgoals. Lastly, we see that the Director agents suffer in these experiments despite their good performance in the single environment experimental settings reported by Hafner et al. (2022). One may be surprised by how badly a state-of-the-art hierarchical planning methods such as Director performed. For this, we will present later additional experiments to show that Director is certainly strong in the more classical mono-task experimental setting, but not strong in a multi-task, generalization-focused setting (the one we adopted in this chapter): Director performs well in single environment configurations as expected, but its performance deteriorates fast with more training tasks. Such results indicate poor scalability in terms of generalization, and thus a limitation to its real-world applications.

5.5.2 Scalability Studies: Number of Training Tasks

Inspired by Cobbe et al. (2020), we want to investigate the scalability of the agents' generalization abilities *w.r.t.* the number of training tasks. A method is favorable if it could achieve the highest generalization performance with the least amounts of training tasks.

To this end, in Fig. 5.7, we present the results of the agents' final evaluation performance after training over different numbers of training tasks.

We can observe that, with few or more training tasks, Skippers and the baseline show consistent improvements in generalization performance. While both LEAP and Director behave similarly as in the previous 50-task experiments. Notably, the **modelfree** baseline can reach similar performance as Skipper, but only when trained on a different task in each episode, which is generally infeasible in the real world beyond simulator-based environments.

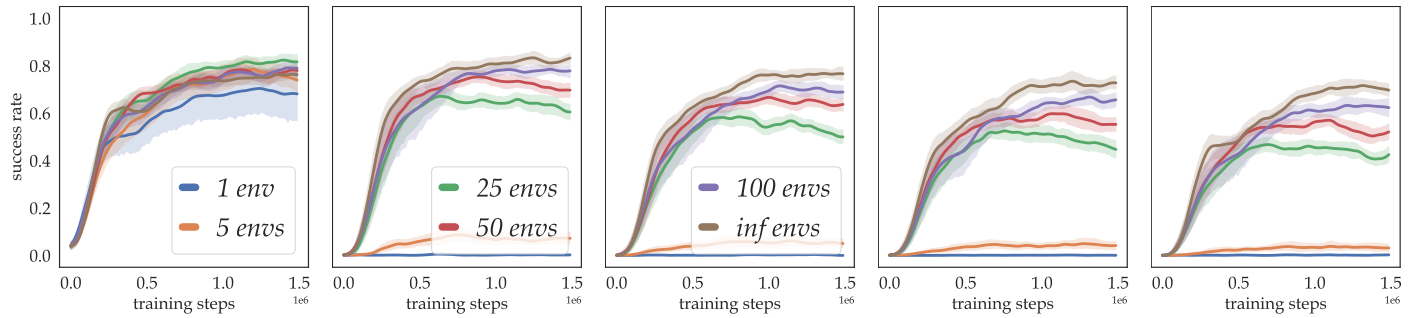


(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.7: ***Evaluation Performance of Agents with Different Numbers of Training Tasks****: each data point and corresponding error bar (95% confidence interval) are based on the final performance from* 20 *independent seed runs. Training task performance is shown in a) while OOD performance is shown in b), c), d), e). Notably, the Skipper agents as well as the adapted LEAP behave poorly during training when being trained on only one task. This is likely because the split of context and partial information in the checkpoint generator cannot be achieved. Training on one task invalidates the purpose of the proposed generalization-focused checkpoint generator.*

Skipper-**once Scalability**

We present the performance evolution (throughout training) of Skipper-**once** with different numbers of training tasks, in Fig. 5.8.



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

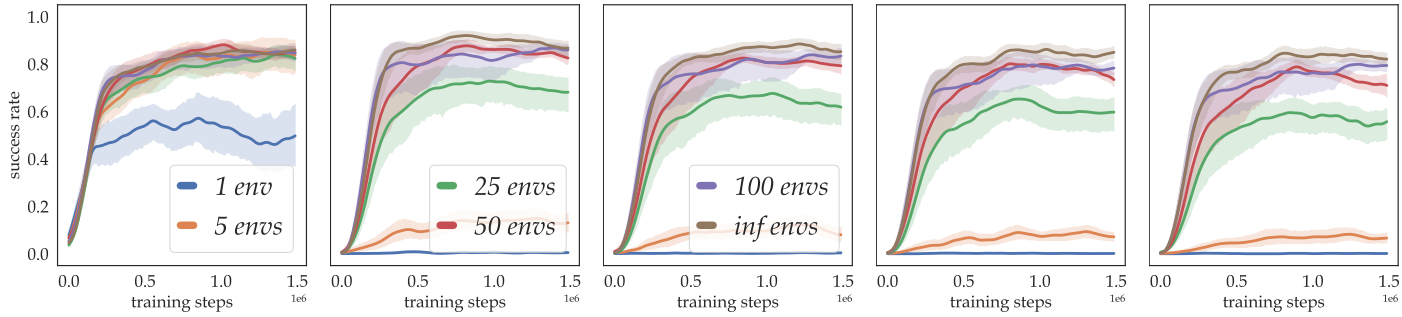
Figure 5.8: ***Evaluation Performance Evolution of*** Skipper-***once with Different Numbers of Training Tasks****: each error bar (95% CI) was obtained from* 20 *seed runs.*

Skipper-**regen Scalability**

We present the performance evolution (throughout training) of Skipper-**regen** with different numbers of training tasks, in Fig. 5.9.

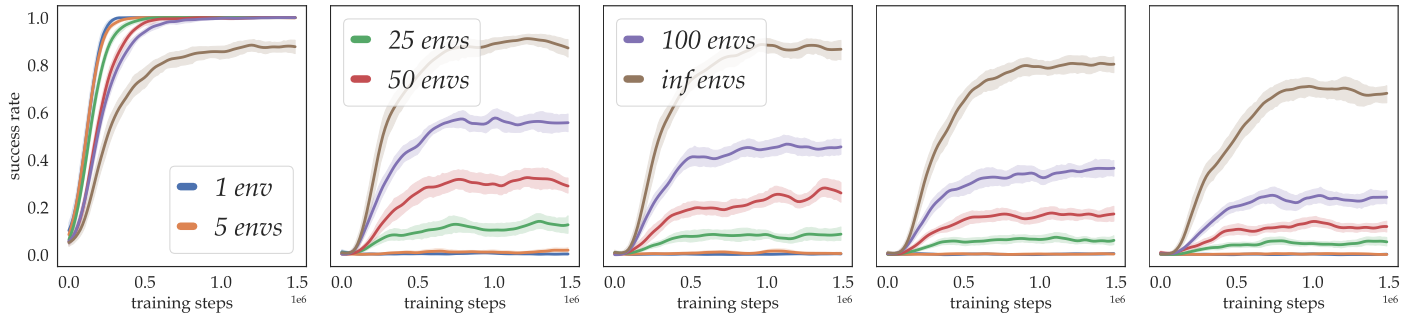
modelfree Scalability

We present the performance evolution (throughout training) of the **modelfree** with different numbers of training tasks, in Fig. 5.10.



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.9: ***Performance of*** Skipper***-regen with Different Numbers of Training Tasks****: each error bar (95% CI) was obtained from* 20 *seed runs.*

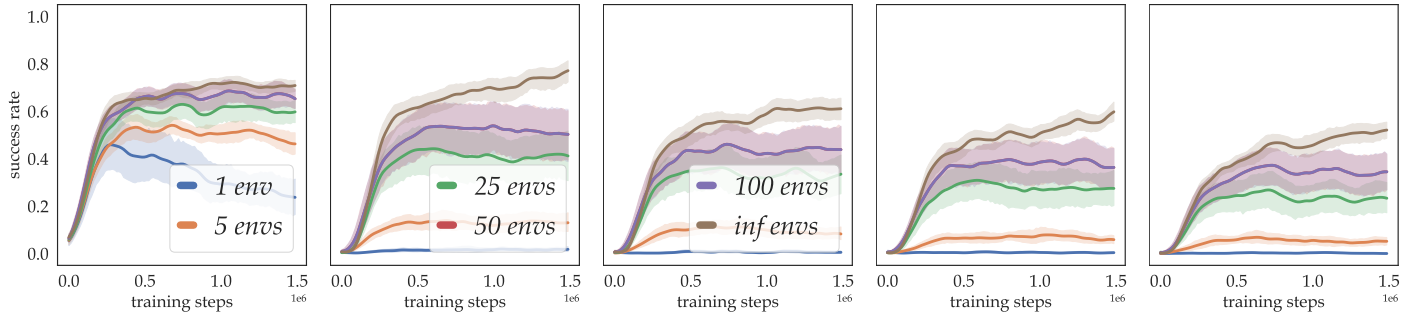


(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.10: ***Evaluation Performance Evolution of modelfree with Different Numbers of Training Tasks****: each error bar (95% CI) was obtained from* 20 *seed runs.*

LEAP Scalability

We present the performance evolution (throughout training) of the adapted LEAP baseline with different numbers of training tasks, in Fig. 5.11.

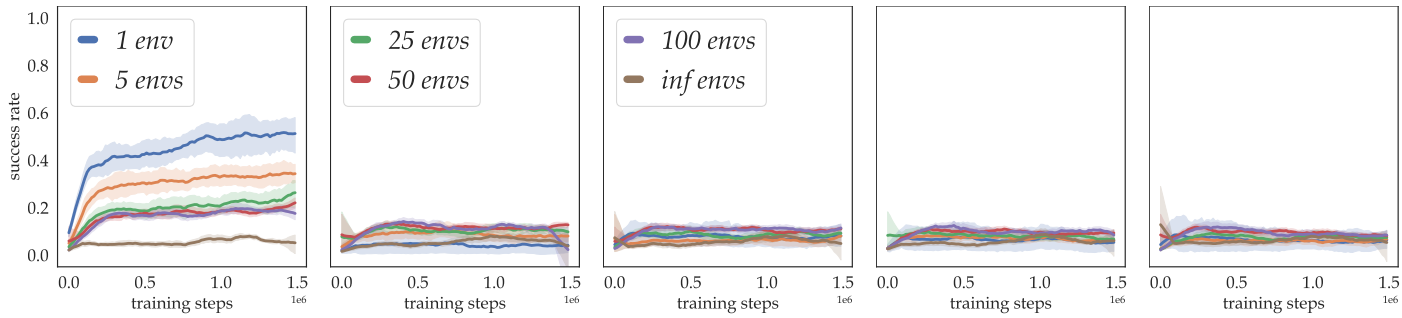


(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.11: ***Evaluation Performance Evolution of*** LEAP ***with Different Numbers of Training Tasks****: each error bar (95% CI) was obtained from* 20 *seed runs.*

Director **Scalability**

We present the performance evolution (throughout training) of the adapted Director baseline on different numbers of training tasks, in Fig. 5.12.

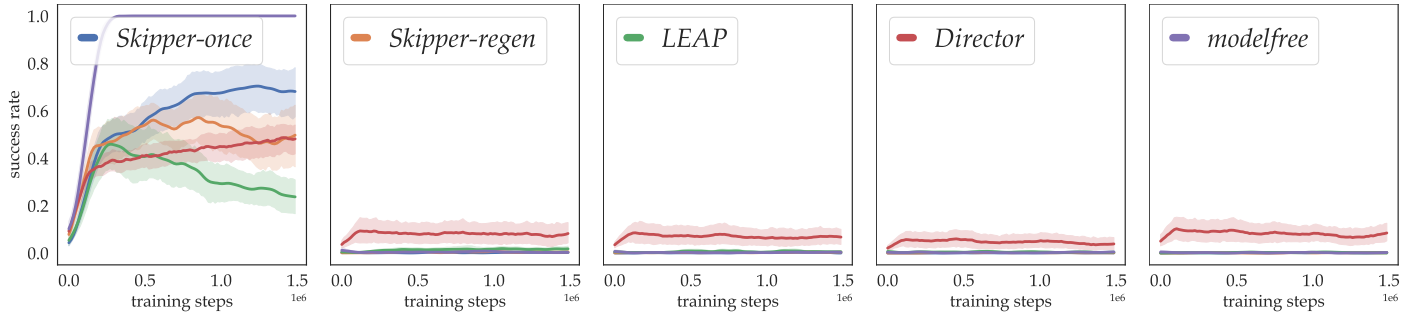


(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.12: ***Evaluation Performance Evolution of*** Director ***with Different Numbers of Training Tasks****: each error bar (95% CI) was obtained from* 20 *seed runs.*

Detailed OOD Performance with Different Numbers of Training Tasks

The performance of all agents on all training configurations, *i.e.*, different numbers of training tasks, are presented in Fig. 5.13, Fig. 5.14, Fig. 5.15, Fig. 5.16, Fig. 5.17 & Fig. 5.18.



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

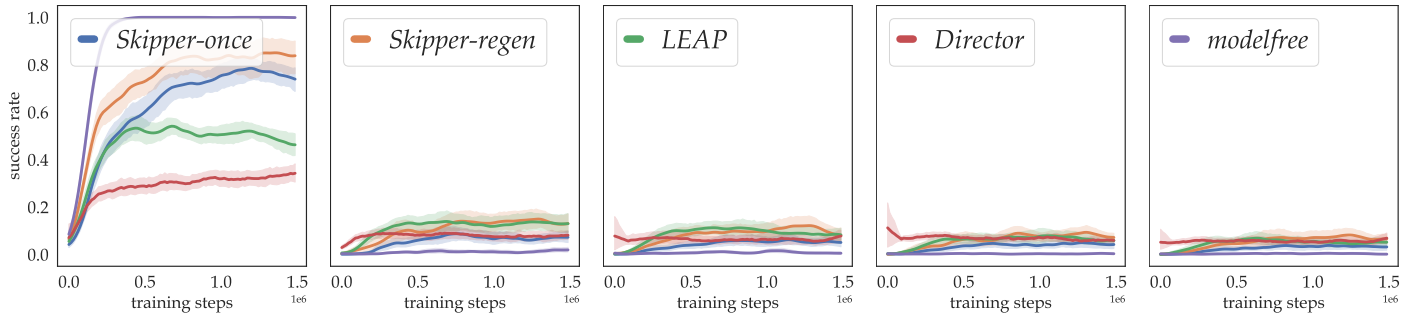
Figure 5.13: ***Evaluation Performance of Agents Trained on*** 1 ***Task****: each error bar (95% CI) was obtained from* 20 *seed runs.*

5.5.3 Validation of Claims

This sub-section focuses on validating the two important claims of advantages brought by the Skipper framework: 1) this framework is compatible with stochasticity and 2) empirically validate if the agent's performance is as described as the theorem.

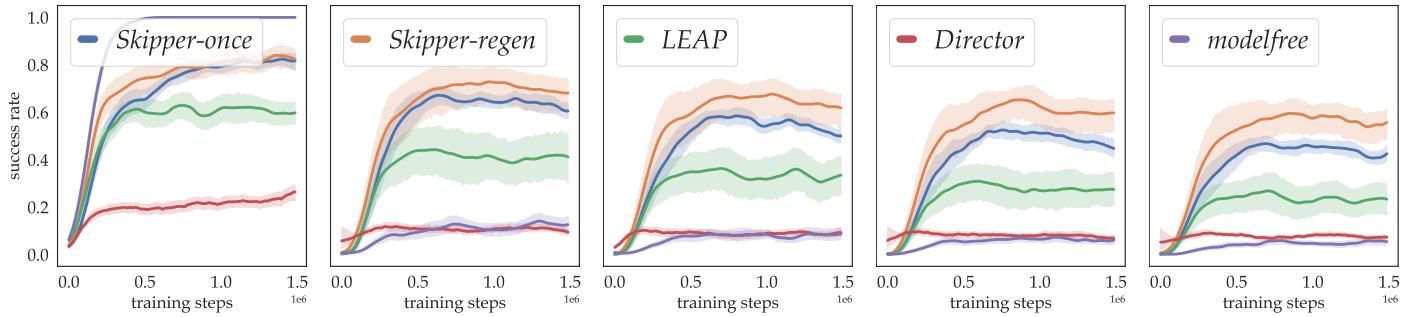
Validation of Effectiveness on Stochastic Environments

We present the performance evolution (throughout training) of the agents in stochastic variants of `RDS`. We modify `RDS` dynamics by implementing ϵ-greedy actions, *i.e.*, with probability $\epsilon = 0.1$, each action is



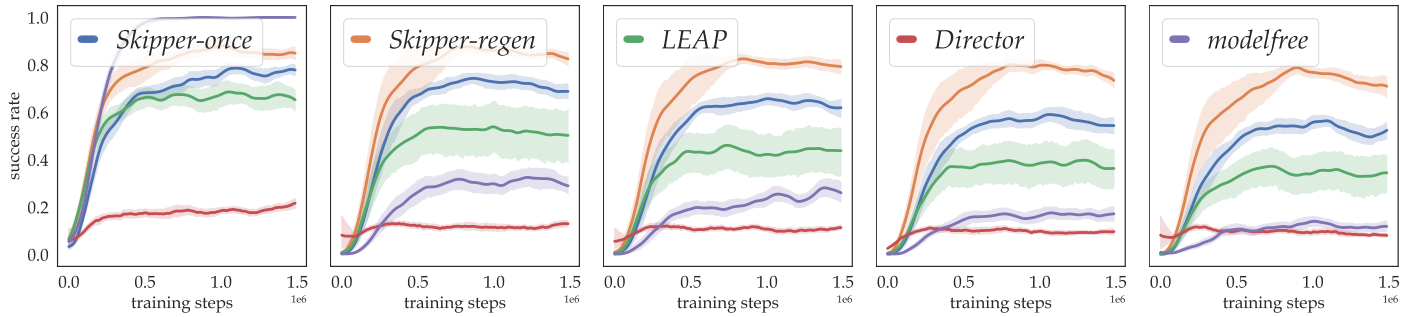
(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.14: ***Evaluation Performance of Agents Trained on*** 5 ***Tasks****: each error bar (95% CI) was obtained from* 20 *seed runs.*



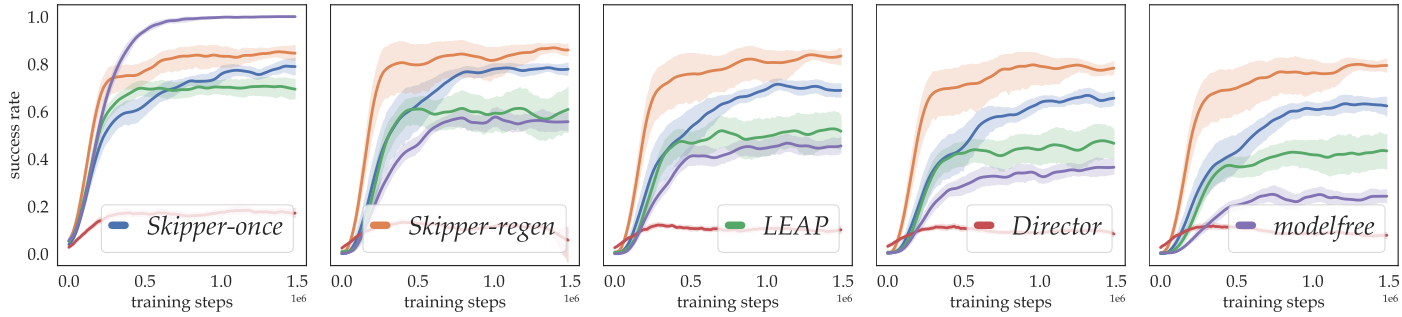
(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.15: ***Evaluation Performance of Agents Trained on*** 25 ***Tasks****: each error bar (95% CI) was obtained from* 20 *seed runs.*



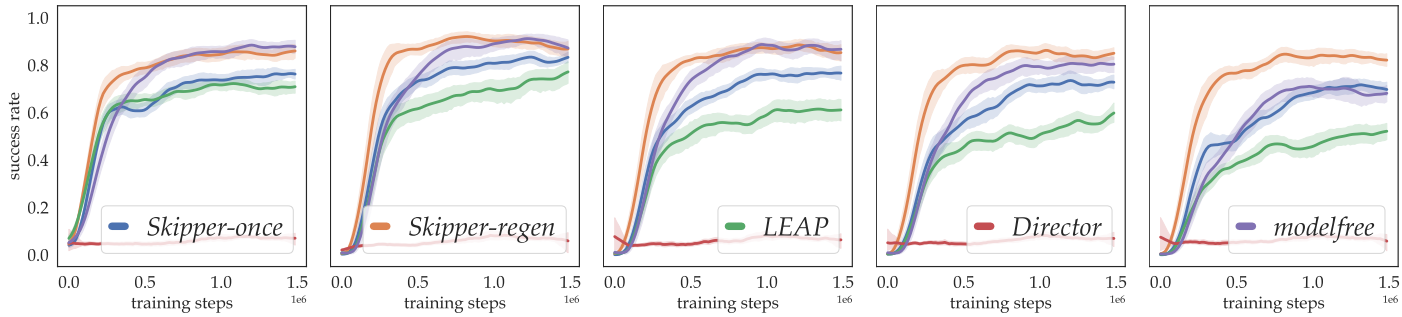
(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.16: ***Evaluation Performance of Agents Trained on*** 50 ***Tasks****: each error bar (95% CI) was obtained from* 20 *seed runs.*



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

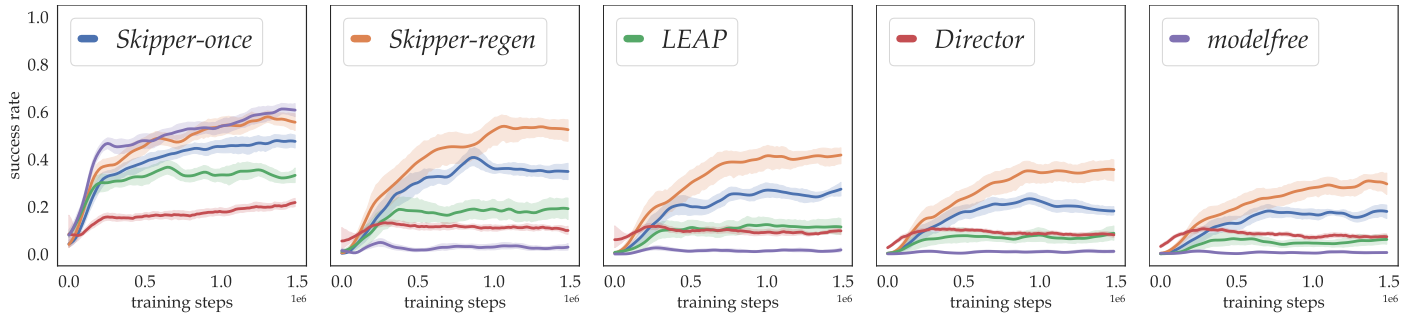
Figure 5.17: ***Evaluation Performance of Agents Trained on*** 100 ***Tasks****: each error bar (95% CI) was obtained from* 20 *seed runs.*



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.18: ***Evaluation Performance of Agents Trained on*** ∞ ***Tasks (a new task each training episode)****: each error bar (95% CI) was obtained from* 20 *seed runs.*

changed into a random action. We present the 50-training tasks performance evolution in Fig. 5.19. The results validate the compatibility of our agents with stochasticity in environmental dynamics. Notably, the performance of the baseline deteriorated to worse than even Director with the injected stochasticity.



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

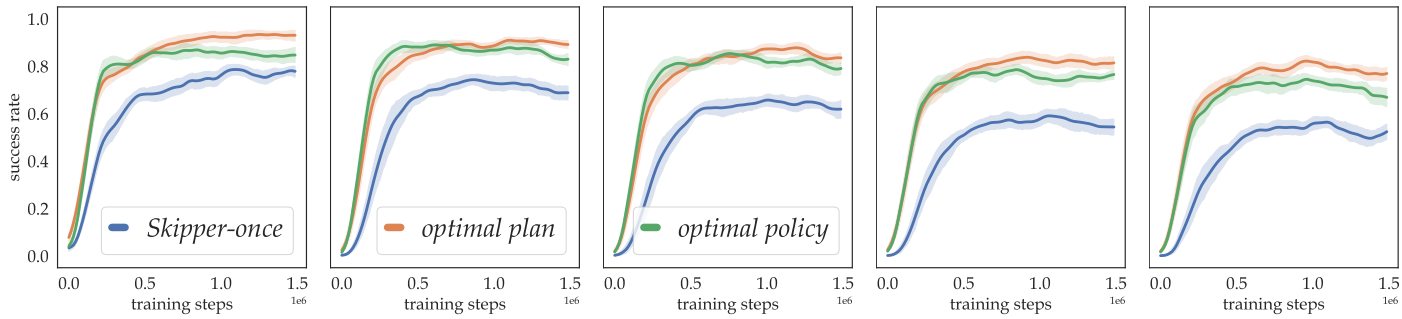
Figure 5.19: ***Evaluation Performance of Agents in Stochastic Environments****:* ϵ*-greedy style randomness is added to each primitive action with* $\epsilon = 0.1$*. Each agent is trained with* 50 *tasks and each curve is processed from* 20 *seed runs.*

Accuracy of Proxy Problems & Checkpoint Policies

We present in Fig. 5.20 the results on the accuracy of proxy problems as well as the checkpoint policies of the Skipper-**once** agents, trained with 50 tasks. We created two variants of Skipper-**once** based on the DP-solved ground truths.

Concurring with our previous theoretical analyses, these results indicate that the performance of Skipper was indeed bottlenecked by the accuracy of the proxy problem estimation on the high-level and the optimality of the checkpoint policy on the lower level.

Notably, the results for the generalization performance across training tasks, as in **a)** of 5.20, indicate that the lower-than-expected performance is an outcome composed of errors in the two levels, the sub-optimality of the low-level policy as well as the inaccuracies of the proxy problem estimates.



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

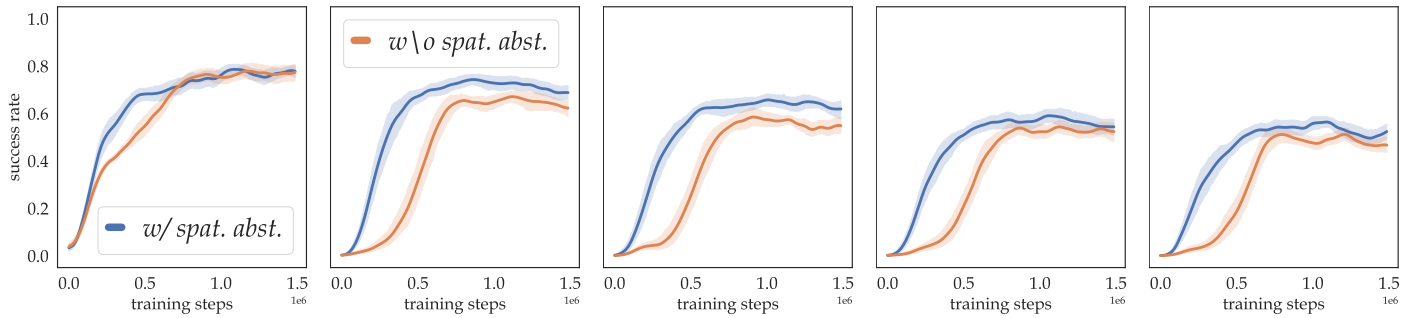
Figure 5.20: ***Evaluation Performance of*** Skipper-***once v.s. Oracle Agents****: both the optimal policy and optimal plan variants are assisted by ground truths solved by DP. The default deterministic setting induces the fact that combining optimal policy and optimal plan results in* 1.0 *success rate. The figures suggest that the learned agent is limited by errors both in the proxy problem estimation and the checkpoint policy* π*. Each agent is trained with* 50 *tasks and each curve is processed from* 20 *seed runs.*

5.5.4 Ablation Studies

We use the following ablation studies to verify the effectiveness of each proposed component of the Skipper framework as well as certain experimental settings. These experiments are mainly done with the more lightweight Skipper-**once** variant on 50 training tasks.

Spatial Abstraction

We present in Fig. 5.21 the ablation results on the spatial abstraction component with Skipper-**once** variant, trained with 50 tasks. The alternative component, without the spatial abstraction, is an MLP on a flattened full state. The results confirm significant advantage in terms of generalization performance as well as sample efficiency in training, introduced by spatial abstraction.

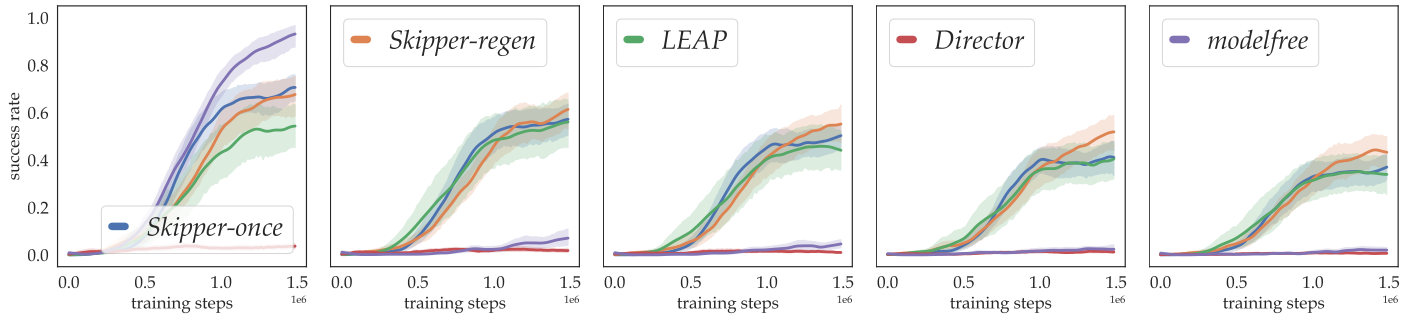


(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.21: ***Ablation for Spatial Abstraction on*** Skipper-***once agent****: each compared method is trained with* 50 *environments and each curve is processed from* 20 *seed runs.*

Training Initialization: uniform *v.s.* same as evaluation

This set of experiments is used to justify the changes in initialization, that we previously introduced to accelerate the experiments. We compare the agents' performance with and without uniform initial state distribution. The non-uniform starting state distributions introduce additional difficulties in terms of exploration. As Presented in Fig. 5.22, these results are obtained from training on 50 tasks. We conclude that given similar computational budget, using non-uniform initialization only slows down the learning curves without introducing significant changes to our conclusions, and thus we use the new initialization setting throughout this chapter by default.

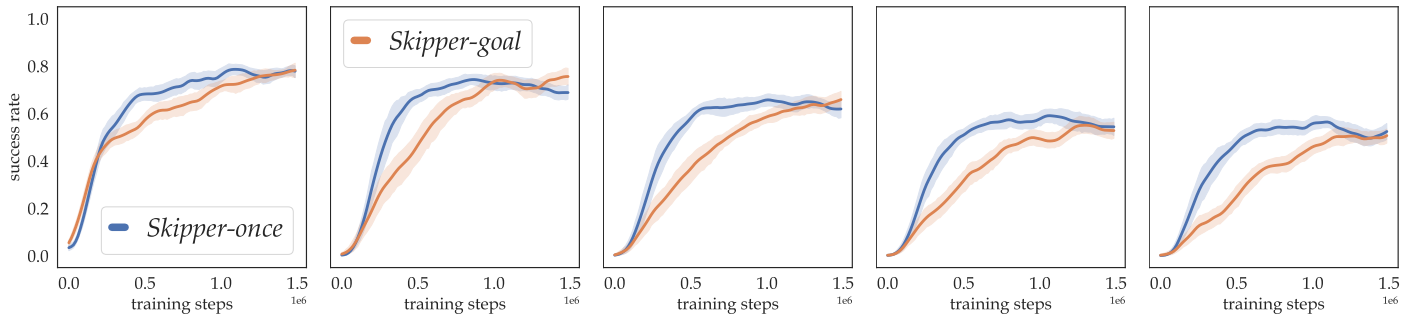


(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.22: ***Ablation Results on*** 50 ***Training Tasks without Uniform Initial State Distributions****: each curve is processed from* 20 *seed runs.*

Planning over Proxy Problems

We provide additional results for the readers to intuitively understand and validate the effectiveness of planning over proxy problems, which is the core to this chapter, This is done by comparing the results of Skipper-**once** with a baseline Skipper-**goal** that blindly selects the task goal as its target all the time. We present the results based on 50 training tasks in Fig. 5.23. Concurring with our vision on temporal abstraction, we can see that by utilizing proxy problems and solving more manageable sub-problems leads to faster convergence. The Skipper-**goal** variant catches up later when the policy slowly improves to be capable of solving longer distance navigation.

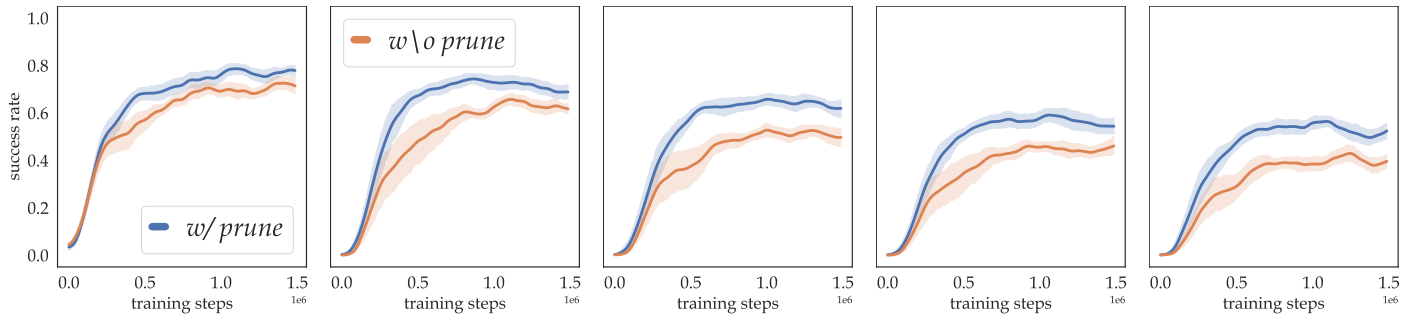


(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.23: ***Ablation Result for the Effectiveness of Proxy Problems****: Each agent is trained with* 50 *tasks and each curve is processed from* 20 *seed runs.*

Vertex Pruning

We want to validate if the proposed k-medoids based vertex pruning technique is useful and could produce better generalization abilities by improving the quality of the proxy problems.
In all previous experiments, each proxy problem is reduced from 32 vertices to 12 with k-medoids. We compare the performance of the default $32 \rightarrow 12$ configuration against a baseline that generates 12-vertex proxy problems without pruning and present the results in Fig. 5.24.
From these results, we can observe that generating more checkpoints than needed then pruning produce higher quality proxy problems than directly generating the exact number of checkpoints without pruning. And thus, we can deduce that the proposed pruning process not only increases the generalization but also the stability of performance.



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.24: ***Ablation Results on*** 50 ***Training Tasks for*** k***-medoids Vertex Pruning****: each curve is processed from* 20 *seed runs.*

5.5.5 Sensitivity Studies

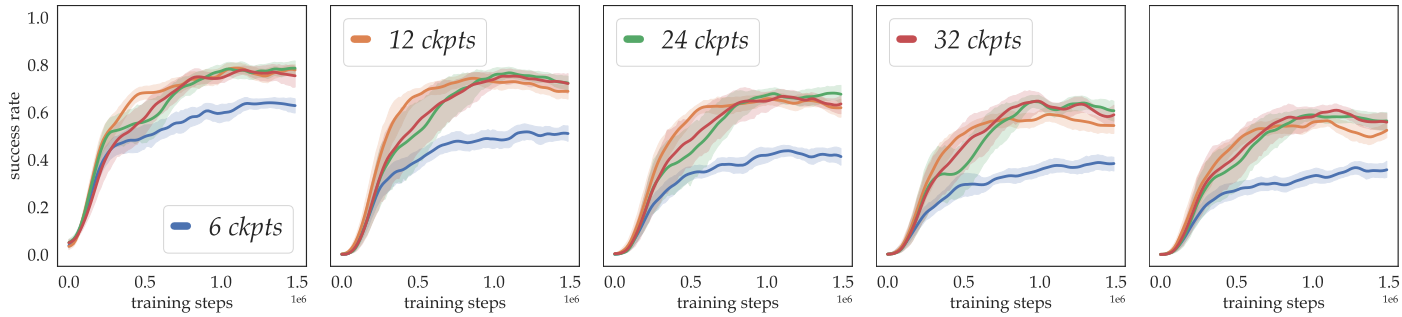
In this sub-secction, we conduct experimental studies to understand how sensitive the proposed Skipper framework is to certain important hyperparameters.

Number of Checkpoints in Proxy Problem

We now conduct a sensitivity study on the number of checkpoints (number of vertices) in each proxy problem. We present the results of Skipper-**once** on 50 training tasks with different numbers of post-pruning checkpoints (all reduced from 32 by pruning), in Fig. 5.25. From the results, we can see that as long as the number of checkpoints is above 6, Skipper exhibits good performance. We therefore chose 12, the one with a rather small computation cost, as the default hyperparameter.

As a default, for our experiments, we adopted the episodic success rate as the performance metric, because using discounted return would cause high variance in the curves without intuitive upper bounds depicting the upper limit of the agents' performances. This choice of metric induces a tradeoff, since this metric does not differentiate the agents taking longer timesteps to succeed in tasks. This acknowledgement is also relevant for this sensitivity study.

When there are too many overcrowded checkpoints each potentially contributing to planning errors, more sub-optimal decisions emerge. Thus, we expect with even more checkpoints, the performance will actually decrease.



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 5.25: ***Sensitivity of*** Skipper-***once to Number of Checkpoints in Proxy Problem****: Each agent is trained with* 50 *tasks. All curves are processed from* 20 *independent seed runs. The error bars are* 95% *confidence intervals.*

5.5.6 Summary of Experiments

Within the scope of the previous experiments, we conclude that Skipper provides significant benefits for generalization; And it can achieve even better generalization when exposed to more training tasks;

From the content presented above, we can deduce additionally that:

- Spatial abstraction based on the local perception field is crucial for the generalization abilities;
- Skipper performs well by reliably decomposing the given tasks, and achieving the sub-tasks robustly. Its performance is bottleneck-ed by the accuracy of the estimated proxy problems as well as the checkpoint policies. This matches well with our theory;
- LEAP in its original form fails to generalize well within its original form and can generalize better when combined with the ideas proposed in this project; We believe that Director may generalize better only in domains where long and informative trajectory collection is possible;

- We verified empirically that, Skipper is compatible with stochasticity, as expected.

5.6 Summary

In this chapter, we proposed, analyzed and validated Skipper, which builds combined spatio-temporal abstractions and generalizes its learned skills better than other comparable methods.

Chapter 6

Rejecting Hallucinations: Addressing Delusional Planning Behaviors

Contents

8.1 Auxiliaries—CP 159

8.1.1 Additional Experimental Insights 159

8.1.2 Experiment Configurations 160

8.2 Auxiliaries—Skipper 161

8.2.1 Skipper 161

8.2.2 LEAP 164

8.2.3 Director 165

8.3 Auxiliaries—Delusions 166

8.3.1 Implementation of PERTASK 166

8.3.2 Implementation Details for Experiments 167

8.3.3 Applying the Evaluator on Dreamerv2 168

6.1 Overview of This Thesis Milestone

In planning processes of computational decision-making agents, generative or predictive models are often used as "generators" to propose "targets" representing sets of expected or desirable states. Unfortunately, learned models inevitably hallucinate infeasible targets that can cause delusional behaviors and safety concerns. We first investigate the kinds of infeasible targets that generators can hallucinate. Then, we devise a strategy to identify and reject infeasible targets by learning a target feasibility evaluator. To ensure that the evaluator is robust and non-delusional, we adopted a design choice combining off-policy compatible learning rule, distributional architecture, and data augmentation based on hindsight relabeling. Attaching to a planning agent, the designed evaluator learns by observing the agent's interactions with the environment and the targets produced by its generator,

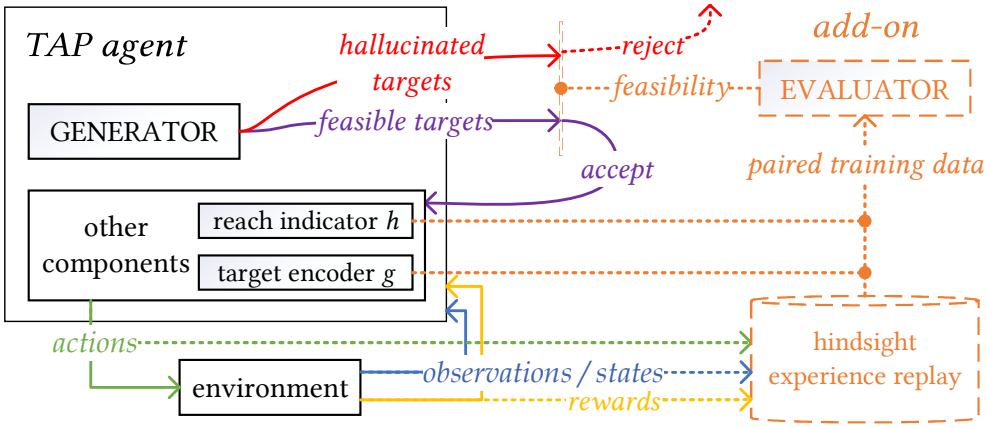


Figure 6.1: ***Target Evaluator attached to a Target-Assisted Planning (TAP) Agent****: An abstracted framework that encompasses, but is not limited to, methods listed in Tab. 6.1. The generator proposes candidate target embeddings* $\boldsymbol{g}^{\odot}$*. Our proposed evaluator can be attached to learn to reject infeasible targets (related parts marked in dashed lines).*

without the need to change the agent or its generator. Our controlled experiments show significant reductions in delusional behaviors and performance improvements for various kinds of existing agents.

The advent of computational modeling has spurred advancements in computational decision making agents, most notable of which is, model-based RL. This work is focused on those models that play the role of *generators*, which imagine or specify future outcomes for agents. Some generators imagine next states or observations, while others specify subgoals (sets of states) to accomplish. No matter how they are represented, we refer to these generator outputs as *targets* and methods that make such use of generators *Target-Assisted Planning (TAP)*.

TAP is a new perspective for unifying many planning / reasoning agents with wildly different behaviors. For instance, some rollout-based model-based RL methods are TAP, such as those following the classic Dyna or Monte-Carlo tree-search frameworks, utilize fixed-horizon transition models to simulate experiences (Sutton, 1991; Schrittwieser et al., 2019; Łukasz Kaiser et al., 2020); While, TAP also encompasses methods that directly generate arbitrarily distant targets acting as candidate sub-goals to divide-and-conquer the tasks into smaller, more manageable steps (Zadem et al., 2024; Zhao et al., 2024; Lo et al., 2024).

It is the defining characteristic of TAP - the usage of generators, that brings us to the topic of this work: an often unstated assumption is that generated targets are always feasible. However, the desired generalization abilities of generative models are inevitably accompanied by *hallucinations* (Xu et al., 2025; Zhang et al., 2024b; Xing et al., 2024; Jesson et al., 2024; Aithal et al., 2024) - the "dark side" that produces infeasible targets that can never be experienced by anyhow. Hallucinations impact TAP agents differently based on their planning behaviors. In *decision-time* TAP methods (Alver and Precup, 2024), where models are used to make an immediate decision on what to do next, hallucinated targets can lead to delusional plans that compromise performance and safety (Langosco et al., 2022; Bengio et al., 2024). For *background* TAP agents that train on simulated experiences constructed with generated targets, delusional values estimated from hallucinated targets can be catastrophically destabilizing (Jafferjee et al., 2020; Lo et al., 2024).

Hallucination and creativity are the two-sides of the same coin that is the generalization ability of the learned models. In psychiatry, it is widely accepted that human brains reject infeasible intentions hallucinated by the

belief formation system (that can often hallucinate, similar to the generators in TAP agents) using a belief evaluation system, which acts like a firewall (Kiran and Chaudhury, 2009). Inspired by such discussions, we propose to assist existing TAP agents with a target evaluator, which can be used to reject infeasible targets and in turn delusional behaviors. To maximize compatibility, the evaluator is designed as a minimally-intrusive add-on to attach to existing TAP agents Zhao et al. (2020): it learns by observing the TAP agent's interactions with the environment and the targets produced, without the need to change the agent's behaviors or the architecture.. Our main contributions are as follows:

1. We systematically categorized and characterized infeasible targets *w.r.t.* time horizons
2. We discussed the desiderata of learning a minimally-intrusive non-delusional target evaluator
3. We proposed a combination of a) off-policy compatible update rule that enables the evaluator to learn by observing, b) an evaluator architecture compatible with different time horizons that enables a unified solution for most TAP agents and c) two assistive hindsight relabeling strategies performing data augmentation to provide training data beyond those collected via interactions, which ensure that the evaluator itself does not produce delusional evaluations
4. We implemented the solution, as illustrated in Fig. 6.1, for several types of existing TAP agents, as discussed in Tab. 6.1, and showed that agents can better manage generated targets, reduce delusional behaviors and significantly improve performance

6.2 Methodology: Target Evaluation & Relabeling

6.2.1 Target-Assisted Planning & Targets

In this work, for generality, we consider a target to be an embedding representing a set of states. Each target $\boldsymbol{g}^{\odot} \mapsto \{s^{\odot}\}$ is paired with an indicator function h outputting $h(s', \boldsymbol{g}^{\odot}) = 1$ if $s' \in \{s^{\odot}\}$ and 0 otherwise. For the interest of time horizons, we also introduce τ - the maximum number of time steps an agent is allowed to reach a state in $\boldsymbol{g}^{\odot}$.

Let $D_{\pi}(s, \boldsymbol{g})$ represents *1st* timestep t *s.t.* $h(s_t, \boldsymbol{g}^{\odot}) = 1$, if π (conditional or not) is followed from state s. We define τ-**feasibility** of $\boldsymbol{g}^{\odot}$ from state s under π as $p(D_{\pi}(s, \boldsymbol{g}^{\odot}) \leq \tau) \coloneqq \sum_{t=1}^{\tau} p(D_{\pi}(s, \boldsymbol{g}^{\odot}) = t)$. $\boldsymbol{g}^{\odot}$ is called τ-**feasible** if $p(D_{\pi}(s, \boldsymbol{g}^{\odot}) \leq \tau) > 0$, and τ-**infeasible** otherwise. Notably, τ-feasibility is an intrinsic property of a state-policy-target tuple, not a heuristic measure.

Aligning with the RL objective of maximizing returns, a target is good if it leads to rewarding outcomes, *i.e.*:

$$\begin{aligned} &\mathcal{U}_{\pi,\mu}(s, \boldsymbol{g}^{\odot}, \tau) \coloneqq \\ &r_{\pi}(s, \boldsymbol{g}^{\odot}, \tau) + \gamma_{\pi}(s, \boldsymbol{g}^{\odot}, \tau) \cdot V_{\mu}(s_{\min(D_{\pi}(s, \boldsymbol{g}^{\odot}), \tau)}) \end{aligned} \tag{6.1}$$

where $\min(D_{\pi}(s, \boldsymbol{g}^{\odot}), \tau)$ denotes the timestep when the commitment to $\boldsymbol{g}^{\odot}$ is terminated (by h or τ), $s_{\min(D_{\pi}(s, \boldsymbol{g}^{\odot}), \tau)}$ is the state the agent ended up in, $r_{\pi}(s, \boldsymbol{g}^{\odot}, \tau) \coloneqq \sum_{t=1}^{\min(D_{\pi}(s, \boldsymbol{g}^{\odot}), \tau)} \gamma^{t-1} r_t$ is the cumulative discounted reward

Table 6.1: ***Discussed Methods, Properties & How to use the Feasibility Evaluator***

Method	TAP Category	Delusional Planning Behaviors	How Our Solution Helps	Implementation Details & Challenges
Dyna (Sutton, 1991)	Fixed-Horizon Background Planning	The imagined transitions could contain hallucinated (next) observations / states, whose delusional value estimates could destabilize the bootstrapping-based TD learning	Cancel updates involving rejected next states (not evaluated to be reachable within 1 timestep).	**Implemented** (for 1-step Dyna): If the output histogram of the evaluator has significant density on the bin corresponding to $t = 1$, then accept the generation, or else, reject
Dreamer (Hafner et al., 2024)	Fixed-Horizon Background Planning	The imagined trajectories could contain infeasible, hallucinated states	Do not let the rejected imagined states participate in the construction of update targets for the actor-critic system (See Sec.8.3.3).	**Implemented** (insufficient compute for results, Sec. 8.3.3): Use the deterministic state $\boldsymbol{s}$ as the state representation to feed to the evaluator (also for imagined future target states). Establish h with Mahalanobis distance on the state representations and use the discount, reward and value predictions to force behavioral realism. Truncate λ-returns until infeasible imagined target states.
Director (Hafner et al., 2022)	Fixed-Horizon Decision-Time Planning (mainly)	The internally sampled goals may be unreachable	Reject unreachable goals and re-sample reachable ones	Similar to our implementation for Dreamer.
MuZero (Schrittwieser et al., 2019)	Fixed-Horizon Decision-Time Planning	The predicted states in the tree search could be unreachable hallucinations	Reject hallucinated state generations, regenerate node in tree search if necessary	Similar to our implementation for 1-step Dyna
SimPLe (Łukasz Kaiser et al., 2020)	Fixed-Horizon Background Planning	The predicted next observation could be an unreachable hallucination	Reject learning against the delusional estimates (potential)	Similar to our implementation for 1-step Dyna
Skipper (Zhao et al., 2024)	Arbitrary-Horizon Decision-Time Planning	Hallucinated subgoals could lead to decision-time planning committing to them, leading to unsafe behaviors	Use an evaluator to learn that the expected cumulative discount is 0 when aiming to reach the hallucinated subgoals. This disconnects the hallucinated subgoals from the current state in the planning	**Implemented**: diversify the source-target pairs with GENERATE and PERTASK mixtures. $\mathcal{G}$ is discrete and h is a trivial comparison.
GSP (Lo et al., 2024)	Arbitrary-Horizon Background Planning	Hallucinated subgoals could lead to value estimation destabilization, like in Dyna.	Use output histogram of the add-on evaluator to correct the delusions by GSP's own estimators. Use the "support swap" technique.	Similar to our implementation for Skipper
LEAP (Nasiriany et al., 2019)	Arbitrary-Horizon Decision-Time Planning	Hallucinated subgoals could help fake a sequence of subgoals that is too good to be true and committed to during planning	Use an evaluator to learn that the expected cumulative distance is infinite when aiming to reach the hallucinated subgoals. This makes sure that subgoal sequences containing hallucinated subgoals will not be favored	**Implemented**: pay attention to the representation space of the sub-goals.
PlaNet (Hafner et al., 2019)	Arbitrary-Horizon Decision-Time Planning	Hallucinated subgoals could help fake a sequence of subgoals that is too good to be true and committed to during planning	Reject the delusional subgoals and therefore reject the delusional subgoal sequences	Same as our implementation for LEAP (both uses CEM for planning (Rubinstein, 1997))

Similar colors are used to denote similar implementations for the solution proposed in this work.

along the way, $\gamma_\pi(s, \boldsymbol{g}^\odot, \tau) \coloneqq \gamma^{\min(D_\pi(s,\boldsymbol{g}^\odot),\tau)}$ is the cumulative discount, and $V_\mu(\cdots)$ is the future value for following μ afterwards.

Eq. 6.1 shows that if $\boldsymbol{g}^\odot$ is τ-infeasible, *i.e.*, $s_{min(D_\pi,\tau)} \notin \boldsymbol{g}^\odot$, then TAP methods blindly using $\boldsymbol{g}^\odot$ to determine V_μ will produce delusional evaluations - the cause of delusional planning behaviors. For example, feasibility-unaware methods, *e.g.*, Sutton (1991); Schrittwieser et al. (2019); Hafner et al. (2024), assume that targets are always reachable as long as they can be generated. Meanwhile, planned trajectories involving infeasible targets are delusional; There are also some feasibility-aware methods, *e.g.* Nasiriany et al. (2019); Zhao et al. (2024); Lo et al. (2024), in which agents estimate certain metrics to decide if a target is feasible. However, as we will discuss later, they often produce incorrect estimates, thus may still favor infeasible targets. In later sections, we propose an evaluator that simultaneously estimates the τ-feasibility and D_π of the proposed targets, where these estimations are used to decide if the evaluation of a target should be trusted or if the target should be rejected.

6.2.2 Source-Target Pairs & Hindsight Relabeling

To learn the feasibility of a target from a given state, "source-target pairs" are needed, which are tuples involving a source state and a target embedding. The quality of these paired training data is critical for the training outcome (Dai et al., 2021; Moro et al., 2022; Davchev et al., 2022). Hindsight Experience Replay (HER) can be seen as a way to enhance the diversity of the pairs, by re-using targets that happened to have been achieved on existing trajectories, and pretending that they were the chosen targets during the interactions (Andrychowicz et al., 2017). **Relabeling strategies**, corresponding to how $s^\odot$ is obtained, are critical for HER's performance (Shams and Fevens, 2022; Eysenbach et al., 2021). Most existing relabeling strategies are *trajectory-level*, meaning that $s^\odot$ comes from the same trajectory as s_t. These include FUTURE, where $s^\odot \leftarrow s_{t'}$ with $t' > t$, and EPISODE, with $0 \leq t' \leq T_\perp$. HER greatly enhanced the sample efficiency of learning about experienced targets. Meanwhile, its limitations to learning about only experienced targets planted a hidden risk of delusions towards hallucinated targets for TAP agents, to be discussed later. For more discussions regarding Hindsight Experience Replay (HER) and hindsight relabeling strategies, please see Sec. 3.1.3 on Page. 34.

6.3 Methodology: Understanding Hallucinated State Targets

Categorizing targets proposed by the generator helps inform us about how to correctly learn the feasibility of targets, *s.t.* hallucinated targets can be properly rejected.

Let us warm-up with singleton targets, *i.e.*, $\boldsymbol{g}^\odot$ has a single element $\hat{s}^\odot$, and propose a characterization of singleton targets into 3 *disjoint* categories, which we call G.0, G.1 and G.2. Then, we will extend to the non-singleton targets composed of these 3 "ingredients".

6.3.1 G.0: ∞-Feasible

Given source state s, a generated singleton target $\boldsymbol{g}^\odot$ is called G.0 if it maps to one state which is ∞-feasible from s, with some policy π. Note that G.0 includes τ-infeasible states for given finite values of τ.

6.3.2 G.1 - Permanently Infeasible (Hallucinated)

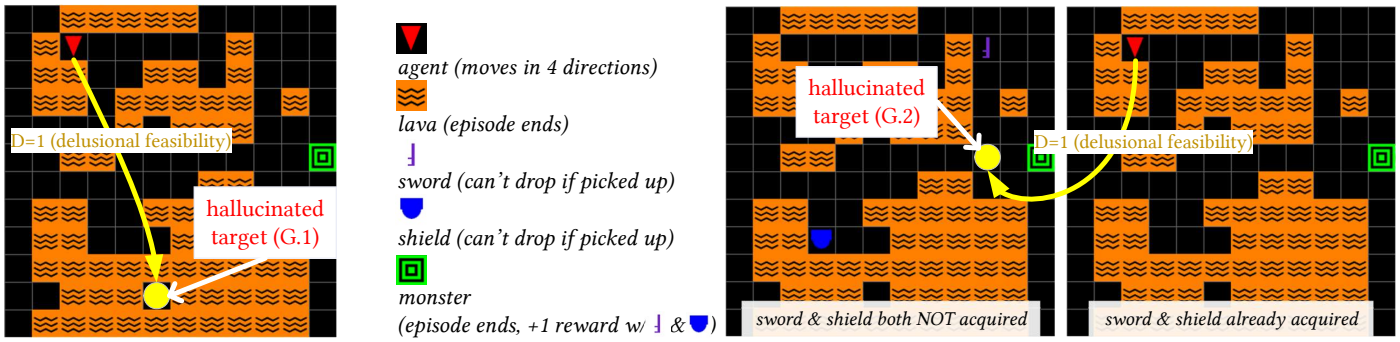
G.1 includes generated "states" that do not belong to the MDP at all, *i.e.*, a target "state" $\hat{s}^{\odot}$ is G.1 if $\forall s, \pi,\ p(D_\pi(s, \hat{s}^{\odot}) < \infty) = 0$.

6.3.3 G.2 - Temporarily Infeasible (Hallucinated)

This type includes those MDP states that are *currently infeasible from state* s. Unlike G.1, G.2 states could be G.0 if they were evaluated from a different source state. G.2s can often be overlooked, not only because hallucinations are mostly studied in contexts that do consider the source state s, but also because they only exist in some special MDPs.

6.3.4 Examples

To provide intuition about these concepts, we use the MiniGrid platform to create a set of fully-observable environments, minimizing extraneous factors to focus on the targets (Chevalier-Boisvert et al., 2023). We call this environment `SwordShieldMonster` (`SSM` for short); In `SSM`, agents navigate by moving one step at a time in one of four directions across fields of randomly placed, episode-terminating lava-traps, while searching for both a sword and a shield to defeat a monster with a terminal reward. The lava-traps' density is controlled by a difficulty parameter δ, but there is always a feasible path to success. Approaching the monster without both the randomly placed sword and shield ends the episode. Once acquired, either of the two items cannot be dropped, leading to a state space where not all states are accessible from the others. Thus, `SSM` states are partitioned into 4 semantic classes, defined by 2 indicators for sword and shield possession. For example, $\langle 0, 1\rangle$ denotes "sword not acquired, shield acquired".
G.1 generations in this environment may be semantically valid, *e.g.*, an `SSM` "state" with the agent surrounded by lava, as in Fig. 6.2 (top row), or totally absurd, *e.g.*, an `SSM` observation without an agent.



G.1-induced delusional behavior — *G.2-induced delusional behavior (s in $\langle 1, 1\rangle$, $s^{\odot}$ in $\langle 0, 0\rangle$)*

Figure 6.2: ***Delusional Plans in SSM****: In both cases, a TAP agent, lacking understanding about the hallucinated targets (yellow dots), mis-evaluates their feasibility, leading to delusional plans which suggest that the task goal can be achieved via the infeasible targets.*

G.2 states can be once G.0 but are now blocked due to a past transition, *e.g.*, after acquiring the sword in `SSM`, the agent transitions from class $\langle 0, 0\rangle$ to $\langle 1, 0\rangle$, sealing off access to $\langle 0, 0\rangle$ or $\langle 0, 1\rangle$; G.2 can also appear due to the initial state distribution d: some states can only be accessed from specific initial states, *e.g.*, an agent

spawned in $\langle 1, 0\rangle$ cannot reach $\langle 0, 0\rangle$ or $\langle 0, 1\rangle$. An example of delusional behavior caused by a G.2 target is provided in Fig. 6.2 (bottom row).

Despite rising concerns regarding the safety of TAP agents (Bengio et al., 2024), their delusional behaviors remain under-investigated, largely due to the **lack of access** to ground truths needed to identify hallucinations and their resulting delusional behaviors. Thus, it is critical to analyze with clear examples and conduct rigorous controlled experiments where the ground truth of targets could be solved with Dynamic Programming (DP) (Howard, 1960), which is why we created `SSM` and used it later for experiments.

6.3.5 Non-Singleton Targets

For the general case, where a generated target embedding $\boldsymbol{g}^{\odot}$ potentially corresponds to a set of "states" $\{\hat{s}^{\odot}\}$, the elements of the associated set may span all categories, *i.e.*, the target can correspond to a mixture of G.0, G.1 & G.2 states. Tab. 6.2 summarizes the possible target compositions and their properties[1].

Table 6.2: ***Categorization of Targets based on Composition, Characteristics, Risks & Delusion Mitigation Strategies***

Target Composition	State Correspondence	∞-Feasibility $p(D_\pi(s, \boldsymbol{g}^{\odot}) < \infty)$	Feasibility Errors & Resulting Delusional Planning Behaviors	Data Augmentation (Relabeling) Strategies against Feasibility Delusions
Only or Single G.0	non-hallucinated feasible states from s	> 0	**E.0**: May think G.0 states are infeasible, thus turn to riskier alternatives, *e.g.*, G.1 or G.2	EPISODE for G.0 (and G.2) states in the same episode + PERTASK for G.0 (and G.2) beyond the episode
Only or Single G.1	hallucinated "states" not belonging to the MDP	should $= 0$	**E.1**: May think G.1 states are favorable, thus commit to them. Impacted by ill-defined $V_\mu(\cdots)$	GENERATE for G.1 (and G.0 & G.2) states, to be proposed by the generator
Only or Single G.2	hallucinated MDP states infeasible from s	should $= 0$	**E.2**: May think G.2 states are favorable, thus commit to them	PERTASK for G.2 (and G.0) beyond episode + EPISODE for G.2 (and G.0) states in the same episode
Some G.0	at least one non-hallucinated state from s	$= p(D_\pi(s, \boldsymbol{g}^{\odot}_{-}) < \infty) > 0$ (Thm. 6.4.1)	**E.0**	EPISODE + PERTASK
Only G.1 & G.2	set of ONLY hallucinated states	should $= 0$	**E.1** & **E.2**	GENERATE or GENERATE + PERTASK

6.4 Methodology: Evaluating Targets Correctly and Robustly

Knowing that hallucinations cannot be eradicated in general, we intend to lower their risks by adopting the brain-inspired solution - to reject infeasible targets post-generation. If done effectively, the negative impact of hallucinated targets becomes limited to the resource cost of generating and rejecting targets, to be discussed in detail. This approach is in contrast with directly trying to address hallucinations in the generators case-by-case, which we deem to have an unbreakable glass ceiling and not versatile enough to be generalized to

[1]There may be no explicit mapping from a target embedding to a set of "states" and thus any target can always map to arbitrarily many G.1 "states". Thm. 6.4.1 explains that this problem is benign.

generic TAP methods. In contrast, agents without evaluators accept proposed targets unconditionally and thus are at risk of delusional planning behaviors.

For a feasibility evaluator to be effectively differentiate the proposed targets, it should correctly estimate the feasibility of targets which maps to all kinds of states (G.0, G.1 & G.2). However, learning to estimate feasibility is not as trivial as it seems, because improper training could naturally lead to delusional feasibility estimations, which cannot be simply addressed by scaling up training. If the evaluator has delusions of feasibility, then its incorporation becomes futile, as hallucinated targets could still be favored.

For estimation errors, we similarly warm up with those of the singleton targets. For clarity, we use matching identifiers E.0, E.1, and E.2 to denote the estimation errors of feasibility towards G.0, G.1, and G.2 "states", respectively. These discussions are presented in Tab. 6.2.

When targets correspond to general sets of states, we have:

Result 6.4.1. *Let $\boldsymbol{g}^{\odot}$ be a target embedding. Its feasibility from state s satisfies:*

$$\forall \pi, p(D_\pi(s, \boldsymbol{g}^{\odot}) \leq \tau) = p(D_\pi(s, \boldsymbol{g}^{\odot}_{-}) \leq \tau)$$

where $\boldsymbol{g}^{\odot}_{-}$ is a target that correspond to the set of states of $\boldsymbol{g}^{\odot}$ with all infeasible states (G.1 & G.2) removed.

This result indicates that a target is infeasible if and only if it consists entirely of infeasible states, allowing us to focus on learning processes that identify such cases.

6.4.1 Desiderata for Evaluator

We used the following important considerations to guide our design for an appropriate feasibility evaluator.

- **[automatic]** the evaluator must learn to automatically differentiate the feasibility of all kinds of targets without pre-labeling: *we need to exploit h*
- **[minimally intrusive]** the evaluator should be generally applicable to existing TAP agents, without changing the agents too much to disturb the generally-sensitive RL components: *we need to ensure its behavior as an add-on and it can be conditioned on the policy π of the agent, to learn alongside the agent by merely observing*
- **[unified]** the evaluator should have a unified behavior compatible with different τs: *we can design it in a way to learn the τ-feasibilities for many τ values simultaneously*

6.4.2 Learning Rule & Architecture for Feasibility

Following the considerations, we propose to use the following learning rule to *indirectly* learn the targets' feasibility by *directly* learning the distribution of $D_\pi(s, \boldsymbol{g}^{\odot})$.

$$D_\pi(s, \boldsymbol{g}^\odot) \leftarrow 1 + D_\pi(s', \boldsymbol{g}^\odot)\text{ , with} \tag{6.2}$$
$$\begin{cases} D_\pi(s, \boldsymbol{g}^\odot) \equiv D_\pi(s, a, \boldsymbol{g}^\odot), a \sim \pi(\cdot|s, \boldsymbol{g}^\odot) \\ D_\pi(s', \boldsymbol{g}^\odot) \coloneqq \infty \text{ if } s' \text{is terminal and } h(s', \boldsymbol{g}^\odot) = 0 \\ D_\pi(s', \boldsymbol{g}^\odot) \coloneqq 0 \text{ if } h(s', \boldsymbol{g}^\odot) = 1 \end{cases}$$

This results in an off-policy compatible policy evaluation process over a parallel MDP almost-identical to the task MDP, but adapted for $\boldsymbol{g}^\odot$, where all transitions yield "reward" $+1$ and states satisfying $\boldsymbol{g}^\odot$ are changed to terminal with state value 0. Every time an infeasible target embedding is sampled for training, the update rule will gradually push the estimate towards ∞, for all sampled source state s.

Our design only learns D_π in a way that can lead to τ-feasibilities $p(D_\pi(s, \boldsymbol{g}^\odot) \leq \tau)$. For this purpose and the consideration for a unified design, we propose to use Eq. 6.2 in conjunction with a C51-style distributional architecture (Bellemare et al., 2017), which outputs a distribution represented by a histogram over pre-defined supports. When we set the support of the estimated $D_\pi(s, \boldsymbol{g}^\odot)$ to be $[1, 2, \cdots, T]$ with T sufficiently large, the learned histogram bins via Eq. 6.2 will correspond to the probabilities of $p(D_\pi(s, \boldsymbol{g}^\odot) = t)$ for all $t \in \{1, \ldots, T-1\}$. This technique of using C51 distributional learning enables the extraction of τ-feasibility $p(D_\pi(s, \boldsymbol{g}^\odot) \leq \tau)$ from a learned T-feasibility with $p(D_\pi(s, \boldsymbol{g}^\odot) = t)$ over $t \in \{1, \ldots, T\}$, thus learning all τ-feasibility with $\tau < T$ *simultaneously*. Take the example of the 1-step Dyna agent we implemented for experiments (Sec. 6.6.2): if the estimated histogram has little probability density for $p(D_\pi(s, \boldsymbol{g}^\odot) = 1)$, then the target (simulated next state) is likely hallucinated and should be rejected, avoiding a potential delusional value update.

Note that the C51 architecture also allows us to extract the distribution of $\gamma_\pi(s, \boldsymbol{g}^\odot, \tau)$, which, as defined in Sec. 6.2 (Page. 114), the cumulative discount with a chosen target. This is done via transplanting the output histogram of $D_\pi(s, \boldsymbol{g}^\odot)$ over $[1, 2, \ldots, \tau, \tau{+}1, \tau{+}2, \ldots]$ onto the changed support of $[\gamma^1, \gamma^2, \cdots, \gamma^\tau, \gamma^\tau, \gamma^\tau, \ldots]$

6.4.3 Training Data for Feasibility

With the proper learning rule and architecture, we now need to ensure that the evaluator have proper training data and does not become delusional. In Sec. 6.2, we mentioned the incompleteness of the relabeling strategies, which will be discussed in detail now: **1)** Certain relabeling strategies naturally create exposure issues, even for G.0 targets. For instance, FUTURE only relabels with future observations, thus only exposes a learner to future feasible targets, confusing the evaluator when a "past" target is proposed during planning; **2)** Trajectory-level relabeling is, by design, limited. Short trajectories, common in many training procedures, cover limited portions of the state space and prevent evaluators from learning about distant targets, risking delusions when such distant targets are proposed. Short trajectories can be the product of experimental design (initial state distributions, maximum episode lengths (Erraqabi et al., 2022), or environment characteristics, *e.g.*, density of terminal states).

Avoiding feasibility delusions requires learning from all kinds of targets, including those that can never be experienced. This is to counter the exposure bias - the discrepancy between (most existing) TAP agents'

behaviors (involving all targets that can be generated) and training (learning from only experienced targets), identified in Talvitie (2014).
We introduce 2 data augmentation (relabeling) strategies to expand training source-target pair distributions.

GENERATE: Expose Candidates Targets (to be generated)

The first strategy, named GENERATE, is to *expose the targets that will be proposed during planning to the evaluator*, so that it can figure out if these targets are infeasible.
We can implement this as a Just-In-Time (JIT) relabeling strategy that relabels a sampled (un-relabeled) transition for training with a generated target (provided by the generator). We can expect GENERATE to be effective, as evaluators will get exposed to hallucinated targets that the generator could offer. Note that GENERATE requires the use of the generator, thus it incurs additional computational burden, depending on the complexity of target generation. The JIT-compatibility lowers the need for storage and provides timely coverage of the generators' changing outputs, especially helpful for non-pretrained generators. The idea behind GENERATE can be traced back to Talvitie (2014).

PERTASK: Expose Experienced Targets Beyond the Episode

The second strategy, named PERTASK, is to *expose the evaluator to **all** targets $g(s^{\odot})$ experienced before*, so that it could realize if some previously achieved targets not present in the current episode are infeasible from the current state.
We implement PERTASK by relabeling transitions with (the target embedding of) observations from the same training task, sampled across the entire replay. PERTASK can be seen as an generalization of the "random" strategy in Andrychowicz et al. (2017) to multi-task training settings. Importantly, PERTASK exposes the evaluator to E.2 delusions and to long-distance E.0 caused by trajectory-level relabeling on short trajectories. An example is shown in Fig. 6.3.

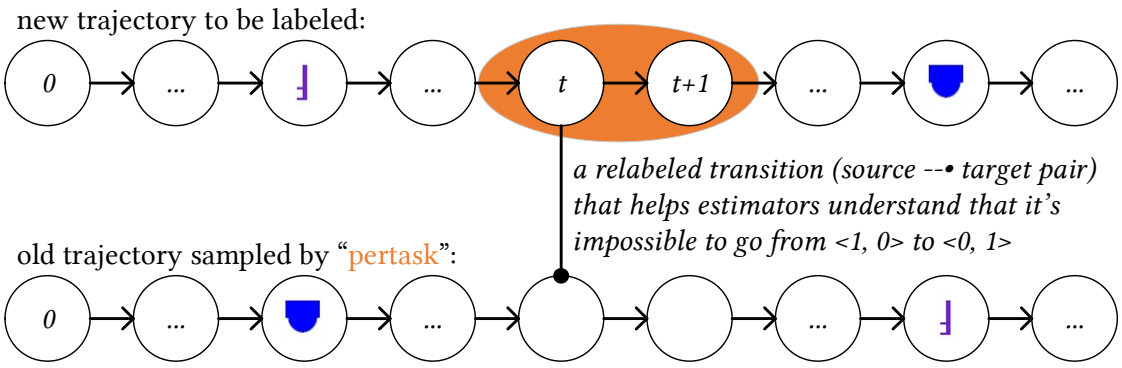


Figure 6.3: ***An Example of How PERTASK Reduces E.2 Errors by Sampling Across Episodes****: The new trajectory acquired the sword first and the shield later, while the old trajectory acquired the shield first and then the sword. When relabeling a transition in the new trajectory (in $\langle 1, 0 \rangle$), a target observation in the old trajectory (in $\langle 0, 1 \rangle$) can be paired to make an agent realize the infeasibility of the relabeled target, thus reducing E.2 delusions.*

Applicability

PERTASK cannot address E.1 delusions. Meanwhile, GENERATE can be used against **some** G.2 targets that the generator hallucinates. PERTASK is a specialized and computationally efficient strategy to reduce feasibility delusions towards *all* experienced G.2 target states and importantly also the long-distance E.0 errors that GENERATE cannot handle. PERTASK is expected to be more effective than GENERATE in generalization-focused scenarios, where the distribution of G.0 & G.2 targets proposed by the generator during evaluation can go beyond those trained under GENERATE.

Importantly, relabeling strategies such as FUTURE, EPISODE and PERTASK rely on the existence of g that maps a state into a target embedding, which is commonly found in TAP agents (Andrychowicz et al., 2017). However, if only the target set indicator function h is available, we may need to accumulate $\langle s, \boldsymbol{g} \rangle$ tuples for which $h(s, \boldsymbol{g}) = 1$, and the use them to train a g. Or, in the cases where feasibility is only used for rejection, such as when dealing with simulated experiences and tree search, we could also rely on only GENERATE, which does not require g; Sometimes, it is h that needs to be constructed. We provide detailed discussions for applying our solution on DreamerV2 in Sec. 8.3.3, with a focus on how to construct a proper h.

Mixtures

Both GENERATE & PERTASK bias the training data distribution, making the evaluator spread out its learning efforts to the source-target pairs possibly distant from each other. Despite increasing training data diversity, distant pairs are less likely to contribute to better evaluation compared to the closer in-episode ones offered by EPISODE, as close-proximity G.0 targets matter the most.

Creating a mixture of sources of training data can increase the diversity of source-target combinations. For HER specifically, each atomic strategy, enumerated in Tab. 6.3 and illustrated in Fig. 6.4, exhibits different estimation accuracies for different types of source-target pairs, including short-distance and long-distance ones involving all of G.0, G.1 and G.2.

When the training budget is fixed, *i.e.*, training frequency, batch sizes, *etc.*, stay unchanged, the mixing proportions of strategies pose a tradeoff to the learning of different kinds of source-target pairs. In the experiments, we mainly used (E+P+G), which is a mixture of $2/3$ EPISODE and $1/3$ PERTASK, with $1/4$ chance using GENERATE JIT, resulting in a mixture of 50% EPISODE, 25% PERTASK and 25% GENERATE. (E+P+G) exploits the fact that assisting EPISODE with GENERATE and PERTASK often results in better performance in evaluator training, striking a balance between the investment of training budgets for the feasible and infeasible targets (Nasiriany et al., 2019; Yang et al., 2021).

6.4.4 Computational Overhead

The portion of overhead for the evaluation of targets is straightforward, as each target will be fed into the neural networks (paired with a source state) for a forward pass at inference time. This portion of the overhead depends on the complexities of evaluator's architecture and of the state / target representations. We can expect fast evaluations with lightweight evaluators.

It is the strategy post evaluation that determines the overall overhead, which depends on the planning behavior of the TAP agent that the evaluator is attached to. For background TAP agents that generate batches

of targets, the improper ones can be rejected and the whole batch can be all rejected without problem (no Dyna update this time). For decision-time TAP agents, targets act as subgoals and when they are rejected, the agent can either re-generate or commit to more random explorations.

Strategies	Advantages	Disadvantages	Gist
EPISODE	Efficient for evaluator to learn close-proximity relationships	When used exclusively to train evaluator, 1) cannot handle E.2 and 2) prone to E.0 - cannot learn well from short trajectories; Can cause G.2 target states when used to train generators	Creates training data with source-target pairs sampled from the same episodes
FUTURE	Can be used to learn a conditional generator with temporal abstractions	In addition to the shortcomings of EPISODE (those for evaluators only), this additionally causes E.0 when used as the exclusive strategy for evaluator training	Creates training data with temporally ordered source-target pairs from the same episodes
GENERATE	Addresses E.1 with data diversity (also E.2 when generator produces G.2)	Relies on the generator with additional computational costs; Potentially low efficiency in reducing E.0.	Augments training data to include candidate targets proposed at decision time
PERTASK	Addresses evaluator delusions (E.2 & E.0 for long-distance pairs)	low efficiency in learning close-proximity source-target relationships	Augments training data to include targets that were experienced

Table 6.3: ***Detailed Comparison of Atomic Hindsight Relabeling Strategies****: EPISODE and FUTURE are widely used as they increase sample efficiency towards G.0 states significantly; GENERATE and PERTASK, proposed in this work, should be applied against delusions in relevant scenarios.*

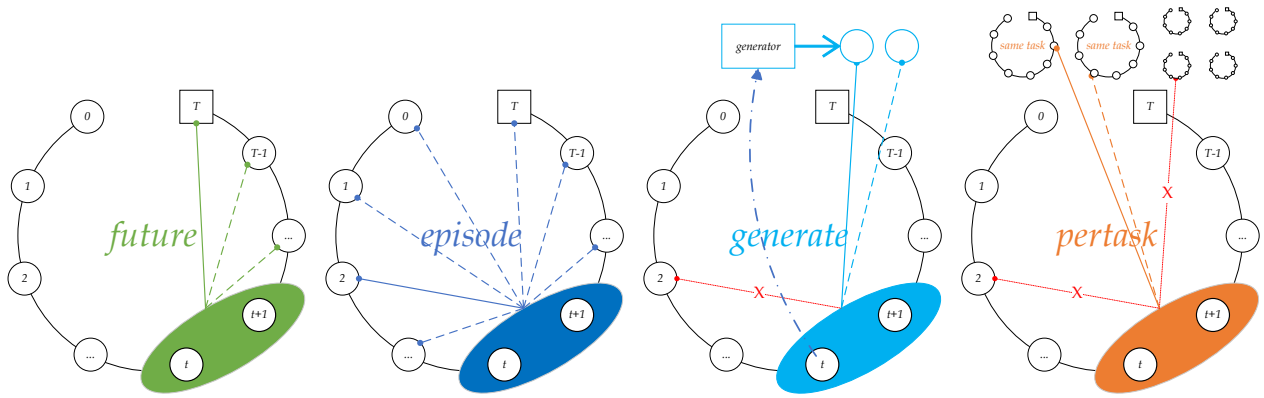


Figure 6.4: ***Representative Atomic Hindsight Relabeling Strategies & Newly Proposed Ones****: The first two strategies, FUTURE and EPISODE, are widely used as they create relabeled transitions that help evaluators efficiently handle G.0 target states during planning. The last two, GENERATE and PERTASK, are effective at addressing delusions, making them useful in specific scenarios.*

6.5 Discussions of Methodologies

6.5.1 About TAP Agents

Most rollout-based TAP methods are oblivious to model hallucinations and utilize all generated targets without question. These include fixed-step background methods such as Sutton (1991); Łukasz Kaiser et al. (2020); Yun et al. (2024); Lee et al. (2024) and decision-time methods based on tree-search, such as Schrittwieser et al.

(2019); Hafner et al. (2019); Zhao et al. (2021); Zhang et al. (2024a); TAP methods compatible with arbitrarily distant targets ($\tau = \infty$) often struggle to produce non-delusional feasibility-like estimates for hallucinated targets. Thus, they cannot properly reject infeasible targets despite having their own "evaluators". These include background methods such as Lo et al. (2024) and decision-time methods for path planning (Nasiriany et al., 2019; Yu et al., 2024; Duan et al., 2024), OOD generalization (Zhao et al., 2024), and task decomposition (Zadem et al., 2024).

6.5.2 About Hindsight Relabeling

Hindsight relabeling is highlighted for its improved sample efficiency towards G.0 targets, around which most follow-up works revolved as well (Andrychowicz et al., 2017; Dai et al., 2021). However, sample efficiency is not the only concern in TAP agents, as delusions toward generated targets can cause delusional behaviors leading to other failure modes. Shams and Fevens (2022) studied the sample efficiency of atomic strategies, without looking into their failure modes. Deshpande et al. (2018) detailed experimental techniques in sparse reward settings using FUTURE. In (Yang et al., 2021), a mixture strategy similar to GENERATE improved estimation of feasible targets, though its impact on hallucinated targets was not investigated. Note that the performance of existing HER-trained agents is often limited by their reliance on FUTURE or EPISODE, whose delusions this paper intends to address.

Learning feasibility of targets has long been on the list of many research agendas for its wide applicability (Tian et al., 2020; Bae et al., 2024; Myers et al., 2024). Our way of using hindsight relabeling to create source-target pairs (instead of using the next states / observations as targets) circumvents issues of learning Monte-Carlo distance functions, which were heavily investigated in Akella et al. (2023). The mixture strategies also circumvent the failure modes of hindsight relabeling that were presented in Eysenbach et al. (2021).

6.5.3 About Delusions & Delusional Behaviors

Delusional value estimates of hallucinated states are hypothesized to plague background planning (Jafferjee et al., 2020). Lo et al. (2024) introduced a temporally-abstract background TAP method to limit temporal-difference updates to only a few trustworthy targets. Langosco et al. (2022) classified goal mis-generalization, a behavior describing when an agent competently pursues a problematic target, a form of delusional planning behaviors. Talvitie (2017) tried to trains the model to correct itself when error is produced. Zhao et al. (2024) gave first examples of delusional behaviors caused by hallucinated targets in decision-time TAP agents. Recently, when I watched David Silver's keynote presentation during the first Reinforcement Learning Conference (RLC), I came to know that previously his AlphaGo team struggled with delusions in the Shoji game. He claimed that his solution was "trial and error" and I suspect that it would be approximately a simulation-based variant of GENERATE. Previously, there were method-specific approaches proposed against delusional planning behaviors, such as by constraining to certain probabilistic models (Deisenroth and Rasmussen, 2011; Chua et al., 2018), or training a target evaluator separately on a collected dataset. The proposed solution in this work, however, is an add-on auxiliary learner that does not seek to introduce additional constraints or changes to the generative models.

6.5.4 About Goal Mis-Generalization

Langosco et al. (2022) proposed a dichotomy of two dimensions to analyze the failure modes for goal-conditioned RL agents: capability mis-generalization *v.s.* goal mis-generalization, corresponding to *the competence of implementing targets* and *the validity of targets*.
While such a perspective is a sensible abstraction from a behavioral standpoint, our work was not established on this perspective. Nevertheless, we are happy to discuss the connections between our work and Langosco et al. (2022):
Directly reducing hallucination (which is not the strategy of this work) directly corresponds to reducing goal mis-generalization. However, in TAP frameworks, goal mis-generalization is also reduced by the estimator that can reject "mis-generalized targets".
Reducing feasibility delusions about targets (with our proposed strategies) reduces both capability mis-generalization and goal mis-generalization. To see both sides, we must point out that 1) evaluating a proposed target requires capability generalization, since it includes the understanding of if the agent can reach certain target and how the agent would reach it (tied to the agent's capabilities); and 2) fewer feasibility delusions about targets in TAP frameworks means problematic targets are less likely favored.

6.6 Research Findings: Experiments

To investigate the effectiveness and generality of our proposed solution against delusional behaviors caused by hallucinated targets, we use a simple and unified implementation of our evaluator (3-layers of `ReLU` activated `MLP` with output bin $T = 16$) for 8 sets of experiments, encompassing decision-time *v.s.* background planning, TAP methods compatible with arbitrary τs and fixed τs, singleton and non-singleton targets, on controlled environments with respective emphases on G.1 and G.2 difficulties. The implementations of our solutions for these experiments can be extended to various existing TAP methods, as suggested in Tab. 6.1.

Exp.$^{1}/_{8}$**:** Skipper (a decision-time TAP method with singleton targets, compatible with arbitrary τ) on `SSM`

Exp.$^{2}/_{8}$**:** LEAP (a decision-time TAP method with singleton targets, compatible with arbitrary τ) on `SSM`

Exp.$^{3}/_{8}$**:** Skipper on `RDS`

Exp.$^{4}/_{8}$**:** LEAP on `RDS`

Exp.$^{5}/_{8}$**:** Dyna (a background TAP method with singleton targets, $\tau = 1$) on `SSM`

Exp.$^{6}/_{8}$**:** Dyna on `RDS`

Exp.$^{7}/_{8}$**:** Feasibility estimation of *non-singleton* targets with arbitrary τ on `SSM`

Exp.$^{8}/_{8}$**:** Feasibility of *non-singleton* targets with arbitrary τ on `RDS`

Data Augmentation Strategies

: In the following sub-sections, we will focus on a particular implementation of evaluator which utilizes (E+P+G) for data augmentation. Since we have the full degrees of freedom in deciding the mixture ratios of the involved relabeling strategies, *i.e.*, EPISODE, GENERATE & PERTASK, we will provide more results that

encompass more relabeling strategies. These results could not only provide the readers with more understanding of the empirical characteristics of the relabeling strategies but also can serve as an ablation test for the two alternative relabeling strategies, *i.e.*, GENERATE & PERTASK. The variant relabeling strategies are as follows:

- **(E+G)** - a mixture against E.1. EPISODE with 50% chance using GENERATE JIT, resulting in a half-half mixture of EPISODE & GENERATE
- **(E+P)** - against E.2. Half EPISODE & half PERTASK

We can see that (E+P+G) is the middle ground between (E+G) and (E+P), with a comprehensive coverage for both G.0, G.1 and G.2 cases. This is why we have chosen (E+P+G) as the default for our evaluator, since we do not wish to assume access to the state space structures of the environments. Note that for the readers' convenience, we have used consistent colors for each variant throughout.

6.6.1 Rejecting Infeasible Goals in Decision-Time Planning (Exp. $1/8$ - $4/8$)

For decision-time TAP agents, we are interested in understanding how rejecting hallucinated targets can influence their abilities to generalize their learned skills after learning from a limited number of training tasks. This also means, the evaluator is expected to learn to generalize its identification of infeasible targets in novel situations (with consistent dynamics), by identifying the patterns of the infeasible targets.
For such experimental purpose, we use distributional shifts provided in `SSM` to simulate real-world OOD systematic generalization scenarios in evaluation tasks (Frank et al., 2009). For each seed run on `SSM`, we sample and preserve 50 training tasks of size 12×12 and difficulty $\delta = 0.4$. For each episode, one of the 50 tasks is sampled for training. Agents are trained for 1.5×10^6 interactions in total. To speed up training, we make the initial state distributions span all the non-terminal states in each training task, making trajectory-level relabeling even more problematic.

Methods

To demonstrate the generality of our proposed solution against hallucinated targets for decision-time TAP, we apply it onto two methods which utilize targets differently:

Skipper (Zhao et al., 2024): generates candidate target states that, together with the current state, constitute the vertices of a directed graph for task decomposition, while the edges are pairwise estimations of cumulative rewards and discounts, under its evolving policy. A target is chosen after applying *value iteration*, *i.e.*, the values of targets are the $\mathcal{U}$ values of the planned paths.[2]

LEAP (Nasiriany et al., 2019): LEAP uses the cross-entropy method to evolve the shortest sequences of sub-goals leading to the task goal (Rubinstein, 1997). The immediate sub-goal of the elitist sequence is then

[2] As shown in Tab. 6.1, our adaptation for Skipper can be extended to methods utilizing arbitrarily distant targets, including background TAP methods such as GSP (Lo et al., 2024)

used to condition a lower-level policy. Compared to Skipper, LEAP is more prone to delusional behaviors, since one hallucinated sub-goal can render a whole sub-goal sequence delusional.[3]
For Exp. $1/8$ - $4/8$, targets are observation-like generations, where G.1 & G.2 can be clearly identified. See Chap. 8 on Page. 167 for more implementation details.

Evaluative Metrics

Feasibility Errors At each evaluation timing, we use the average errors of $\hat{\mathbb{E}}[D_\pi]$ against the ground truths as a proxy to understand the convergence of the evaluators' estimated feasibility of targets.

Delusional Behavior Frequencies We monitor the frequency of a hallucinated target (made of G.1 and G.2) being chosen by the agents (as the next sub-goal for Skipper, as a part of the sub-goal chain for LEAP), *i.e.*, delusional planning behaviors.

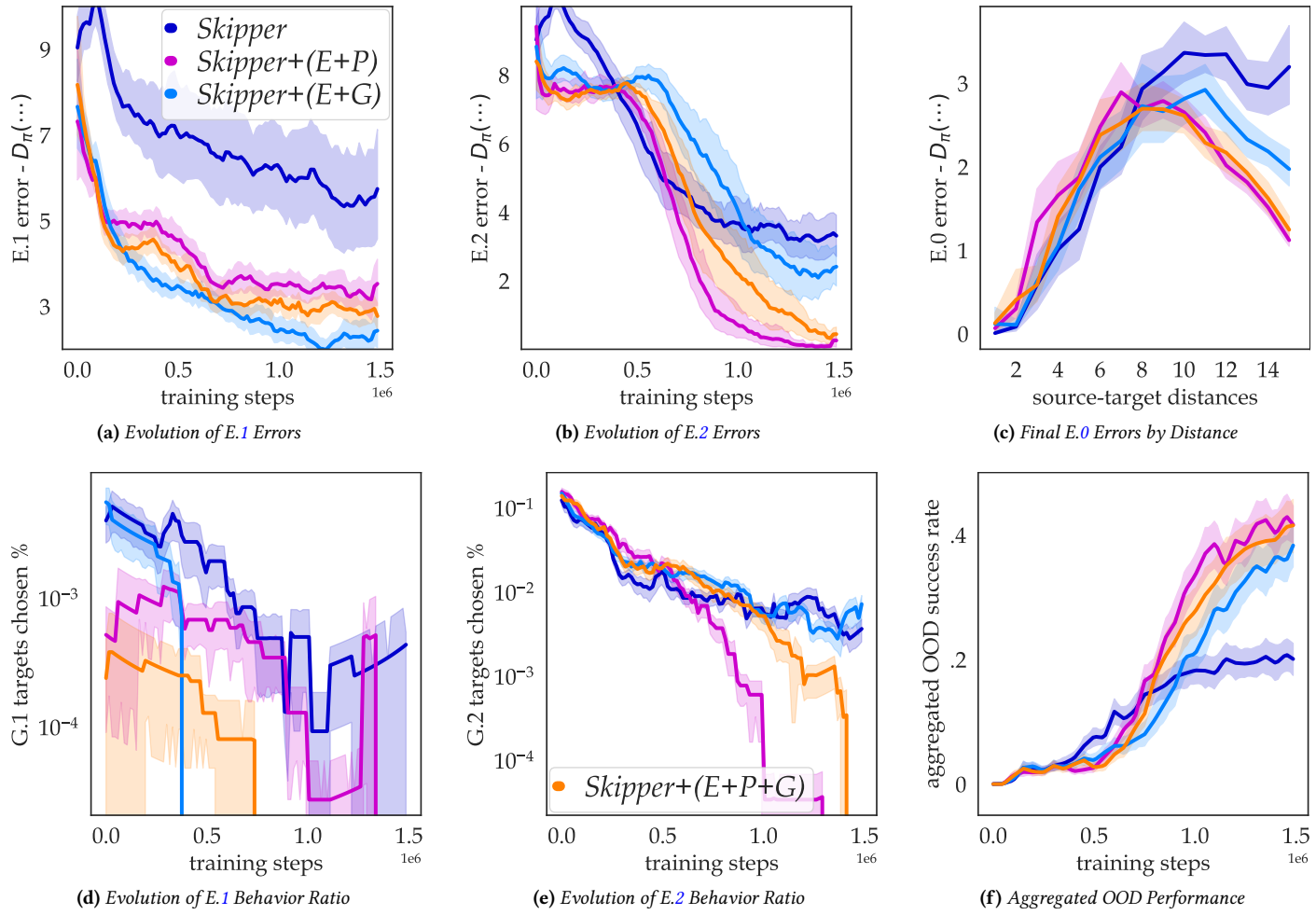
OOD Generalization Performance We analyze the changes in agents' OOD generalization performance. The evaluation tasks (targeting systematic generalization) are sampled from a gradient of OOD difficulties - $0.25, 0.35, 0.45$ and 0.55. Because of the lack of space, we present the "aggregated" OOD performance, such as in Fig. 6.5 **d)**, by sampling 20 task instances from each of the 4 OOD difficulties, and combine the performance across all 80 episodes, which have a mean difficulty matching the training tasks. To maximize the evaluation difficulty, the initial state is fixed in each evaluation task instance: the agents are not only spawned to be at the furthest edge to the monster, but also in semantic class $\langle 0, 0 \rangle$, *i.e.*, with neither the sword nor the shield in hand.

Skipper on SSM (Exp. $1/8$)

We compare the original form of Skipper with Skipper+, a variant that is assisted by the proposed evaluator. Details of the variants are shown in the captions of Fig. 6.5.

Hallucination We first investigate generator's rates of hallucinations. As shown in Fig. 6.6, the generator produces targets that correspond to G.1 and G.2 with the rate of around 3% and 5%, respectively.
We use hindsight-relabeled transitions to train the generators in the two methods, to demonstrate how different ways of training the generator could affect the rates of hallucinations. G.2 can appear more frequently if the generator is trained to imagine more diverse kinds of targets than needed. For example, a conditional target generator which learns from EPISODE will be more likely to produce G.2 target states (compared to FUTURE). This was why we mostly used FUTURE to train the generators in the related experiments.
For the HER-trained generators, Fig. 6.6 **a)**, shows that different training targets for the generator could lead to different degrees of hallucinations, in terms of G.1 and G.2, but not 0. Importantly, Fig. 6.6 **b)** indicates that, FUTURE generates G.2 target states significantly less frequently than EPISODE and PERTASK, as the other two wasted training budget on G.2 target states, especially PERTASK that brings in more problematic training

[3] As shown in Tab. 6.1, our implementation for LEAP can be extended to planning methods proposing sub-goal sequences, such as PlaNet (Hafner et al., 2019)

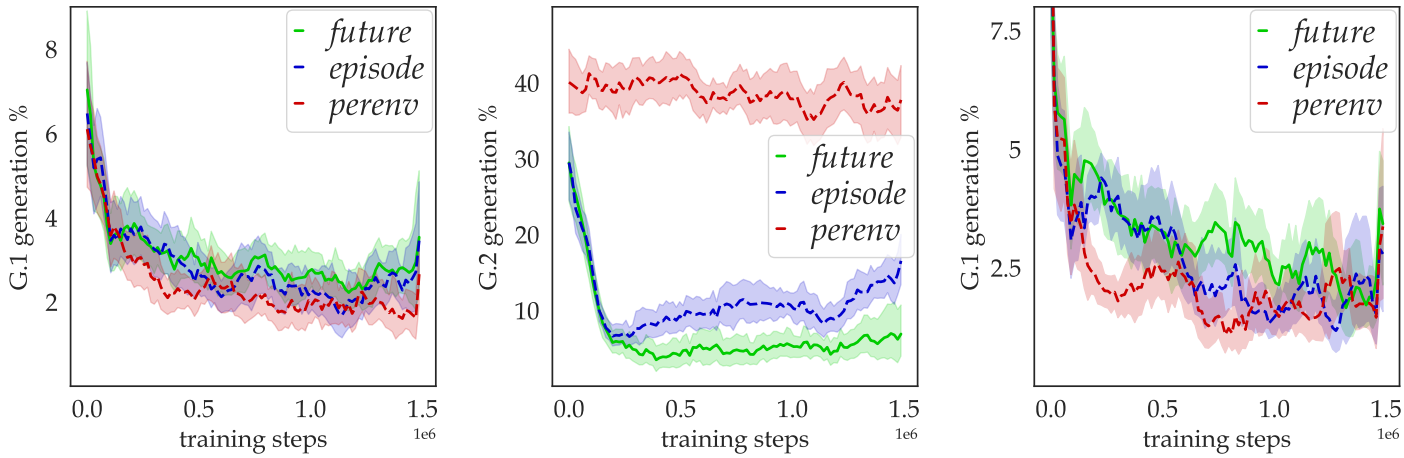


(a) *Evolution of E.1 Errors* **(b)** *Evolution of E.2 Errors* **(c)** *Final E.0 Errors by Distance*

(d) *Evolution of E.1 Behavior Ratio* **(e)** *Evolution of E.2 Behavior Ratio* **(f)** *Aggregated OOD Performance*

Figure 6.5: ***Details of*** Skipper***'s Performance on SSM****: All error bars (95%-CI) are established over* 20 *seed runs. We compare the original form of Skipper, which learns its own feasibility estimates of target states in its own way, against three Skipper+ variants, which have our proposed evaluator injected to assist the evaluation of the feasibility of targets, powered by the (E+G), (E+P) & (E+P+G) relabeling strategies, respectively.* ***a)****: Final E.0 errors separated across a range of ground truth distances. Both estimated and true distances are conditioned on the evolving policies;* ***b)****: E.1 errors measured as* L_1 *error in estimated (clipped) distance throughout training;* ***c)****: G.2-counterparts of* ***b****;* ***d)*** *&* ***e)****: the changes of frequencies in delusional behaviors throughout training, for G.1 and G.2 composed targets, respectively. The curves denote the frequencies of G.1 and G.2 targets becoming the imminent subgoals that Skipper seeks to achieve next;* ***f)****: Each data point represents OOD evaluation performance aggregated over* 4×20 *newly generated tasks, with mean difficulty matching training. The decomposed results for each OOD difficulty are presented in Fig. 6.7.*

samples from long distances. *In all other experiments, we only compare variants with FUTURE for the generator training.*
The generator is consistently used for both Skipper and LEAP in these 4 sets of experiments.



(a) *G.1 Candidate Ratio in SSM* **(b)** *G.2 Candidate Ratio in SSM* **(c)** *G.1 Candidate Ratio in RDS*

Figure 6.6: ***Hallucination Frequencies****: All error bars (95%-CI) are established over 20 seed runs.* ***a)*** *Evolving ratio of G.1 "states" among all candidates at each target selection, throughout training; Subfigure* ***b)*** *is the E.2-counterpart of* ***a)*** *on SSM; Subfigure* ***c)*** *is the RDS-counterpart of* ***a).***

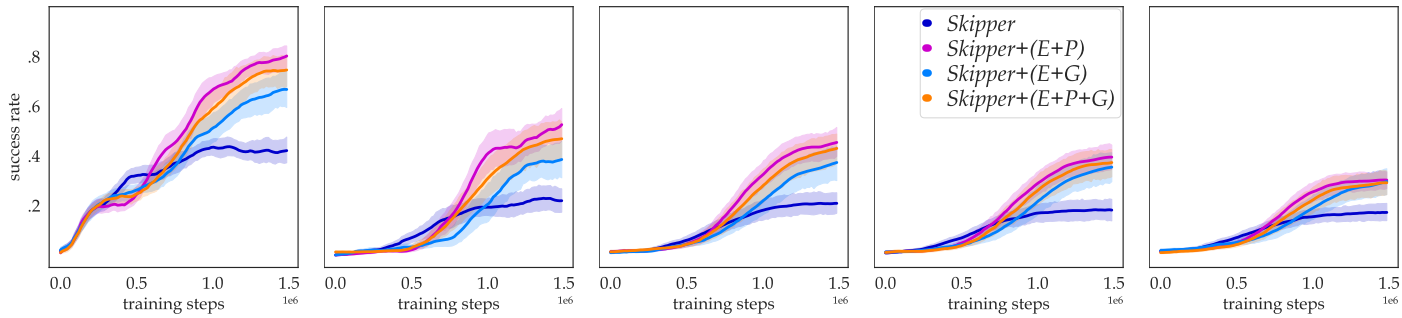
Feasibility Errors Skipper relies on a built-in cumulative discount estimator whose estimations can be converted to feasibility estimates that our evaluator seeks to learn. Thus, we can examine the errors of the feasibility estimates corrected by the injected evaluator to understand how the proposed evaluator could reduce feasibility delusions of arbitrary-horizon TAP methods. From Fig. 6.5 **b**) and **c)**, we can see that feasibility estimates corrected by our evaluator have significantly less errors compared to the original, towards both G.1 and G.2 targets. As a perk for Skipper+'s utilization of PERTASK for E.2 delusions (included in (E+P+G)), its positive effect on far-away G.0 targets are also shown in Fig. 6.5 **a)**. It can be seen that the evaluator is generally helpful for Skipper to understand the feasibility of all G.0, G.1 and G.2 targets.

Frequency of Delusional Plans The purpose of identifying infeasible targets is to reduce delusional plans that involve them. We provide detailed results on this in Fig. 6.5, where we observed significant reduction in delusional plans involving both G.1 and G.2 targets.

Generalization Comparing Skipper and Skipper+, we can deduce from Fig. 6.5 that generally, lower E.2 errors (**c**) lead to less frequent delusional behaviors (**d** & **e**)), which in turn improves the OOD performance in **f)**. This indicates that rejecting infeasible targets can help decision-time TAP agents in systematic OOD generalization.

Breakdown of Task Performance In Fig. 6.7, we present the evolution of Skipper variants' performance on the training tasks as well as the OOD evaluation tasks throughout the training process. Note that Fig. 6.5 **h)** is an aggregation of all 4 sources of OOD performance in Fig. 6.7 **b-e)**.

From the performance advantages of the hybrid variants (in both training and evaluation tasks), we can see that learning to address delusions during training brings better understanding for novel situations posed in OOD tasks.



(a) *training,* $\delta = 0.4$ **(b)** *OOD evaluation,* $\delta = 0.25$ **(c)** *OOD evaluation,* $\delta = 0.35$ **(d)** *OOD evaluation,* $\delta = 0.45$ **(e)** *OOD evaluation,* $\delta = 0.55$

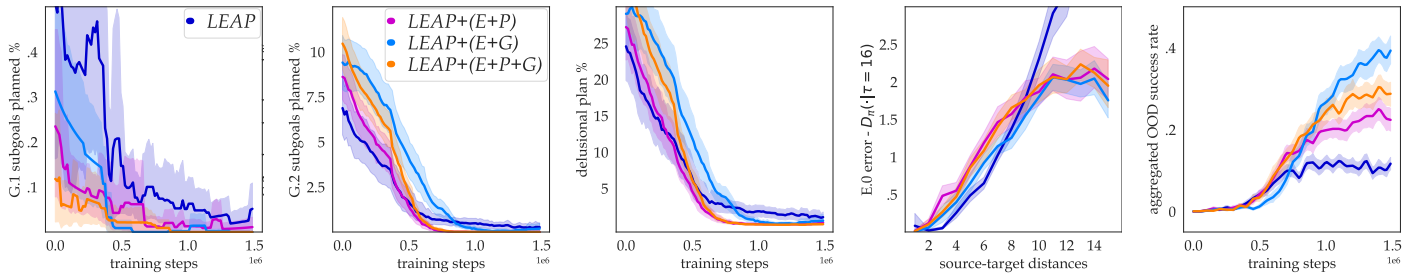
Figure 6.7: ***Evolution of OOD Performance of*** Skipper ***Variants on SSM****: All error bars (95%-CI) are established over* 20 *seed runs.*

LEAP on SSM (Exp. 2/8)

This set of experiments seeks to demonstrate that the proposed feasibility evaluator is applicable to other decision-time TAP agents, utilizing their generators in different ways. For this purpose, we study LEAP performance on SSM, with or without the help of the target rejection provided by the feasibility estimators. LEAP is different from Skipper, as its decision-time planning process constructs a singular sequence of subgoals leading to the task goal. Due to a lack of backup subgoals, even if one among them is problematic, the whole resulting plan would be delusional, making LEAP much more prone to failures compared to Skipper, where candidate targets can still be reused if deviation from the original plan occurred.

SSM has a relatively large state space that requires more intermediate subgoals for LEAP's plans. However, an increment of the number of subgoals also dramatically increases the frequencies of delusional plans, damaging the agents' performance. Because of this, our experimental results of LEAP on SSM with size 12×12 became difficult to analyze because of the rampant failures. We chose instead to present the results on SSM with size 8×8 here.

For LEAP, we use some different metrics to analyze the effectiveness of the proposed strategies in addressing delusions. This is because, if LEAP's evaluator successfully addressed delusions and learned not to favor the problematic targets (G.1 and G.2), then they will not be selected in the evolved elitist sequence of subgoals. This makes it inconvenient for us to use the distance error in the delusional source-target pairs during decision-time as a metric to analyze the reduction of delusional estimates, because of their growing scarcity. As we can see from Fig. 6.8, similar arguments about the effectiveness of the proposed hybrid strategies can be made, to those with Skipper. The hybrids with more investment in addressing E.1, *i.e.*, (E+G) and (E+P+G), exhibit the lowest E.1 errors (**a**)). Similarly, (E+P) and (E+P+G) achieve the lowest E.2 errors (**b**)). In **e**), we see that the 3 hybrid variants achieve better OOD performance than the baseline E. Specifically, (E+G) achieved the best performance. This is likely because that it induced the highest sample efficiency in terms of learning the estimations towards G.0 subgoals, as shown in **d**). Assistive strategies such as GENERATE and PERTASK do

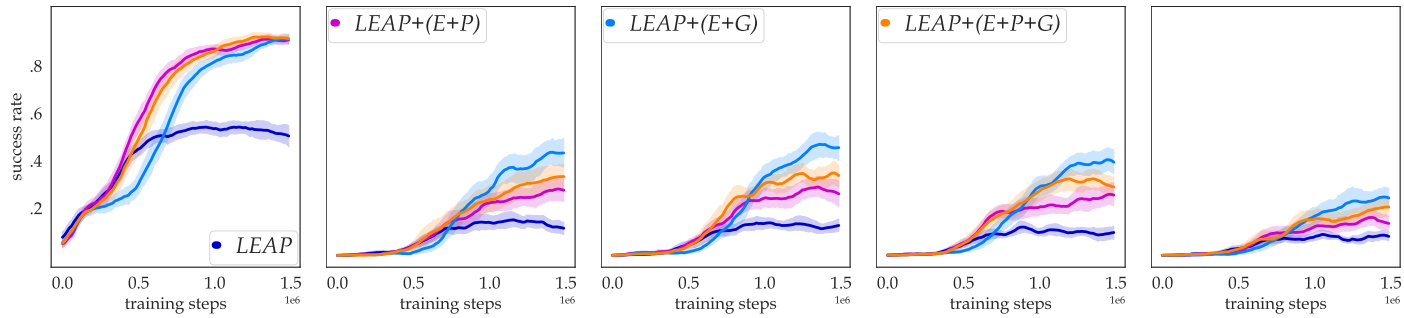


(a) *G.1 Ratio Planned* **(b)** *G.2 Ratio Planned* **(c)** *Delusional Plan Ratio* **(d)** *E.0 Errors* **(e)** *Aggregated OOD Perf.*

Figure 6.8: LEAP ***on SSM****: All error bars (95%-CI) are established over* 20 *seed runs.* ***a)*** *Ratio of G.1 subgoals among the planned sequences;* ***b)*** *Ratio of G.2 subgoals in the planned sequences;* ***c)*** *Ratio of evolved sequences containing at least one G.1 or G.2 target;* ***d)*** *The final estimation accuracies towards G.0 targets after training completed, across a spectrum of ground truth distances. In this figure, both distances (estimation and ground truth) are conditioned on the final version of the evolving policies;* ***e)*** *Each data point represents OOD evaluation performance aggregated over* 4×20 *newly generated tasks, with mean difficulty matching the training tasks.*

not only induce problematic targets, but also G.0 ones that can shift the training distribution towards higher sample efficiencies in the traditional sense.

Breakdown of Task Performance In Fig. 6.9, we present the evolution of LEAP variants' performance on the training tasks as well as the OOD evaluation tasks throughout the training process. Note that Fig. 6.8 **e)** is an aggregation of all 4 sources of OOD performance in Fig. 6.9 **b-e)**.



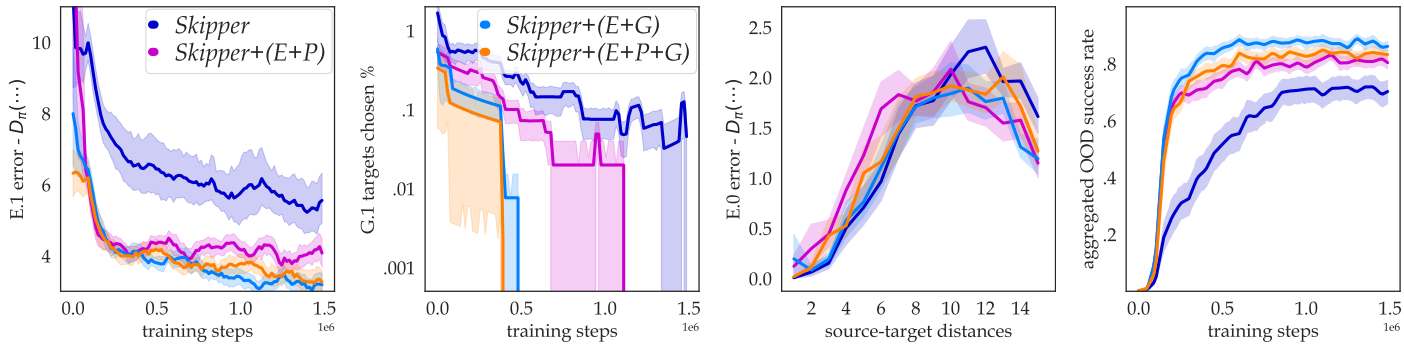
(a) *training,* $\delta = 0.4$ **(b)** *OOD evaluation,* $\delta = 0.25$ **(c)** *OOD evaluation,* $\delta = 0.35$ **(d)** *OOD evaluation,* $\delta = 0.45$ **(e)** *OOD evaluation,* $\delta = 0.55$

Figure 6.9: ***Evolution of OOD Performance of*** LEAP ***Variants on SSM****: All error bars (95%-CI) are established over* 20 *seed runs.*

Skipper **on RDS (Exp.** $^3/_8$**)**

This set of experiments focus on the feasibility evaluator's abilities in the face of G.1 challenges. We present Skipper's evaluative curves in Fig. 6.10.

From Fig. 6.10 **d)**, we can see that, probably because of the lack of dominant G.2 + E.2 cases, the OOD performance of even the most basic EPISODE variant is high, despite the hybrid variants perform even better. (E+G), *i.e.* the hybrid with the most investment in GENERATE (aiming at E.1), performs the best both in terms of E.1 delusion suppression **(a)**), and OOD generalization **(d)**), as expected. In RDS, the short-distance E.0

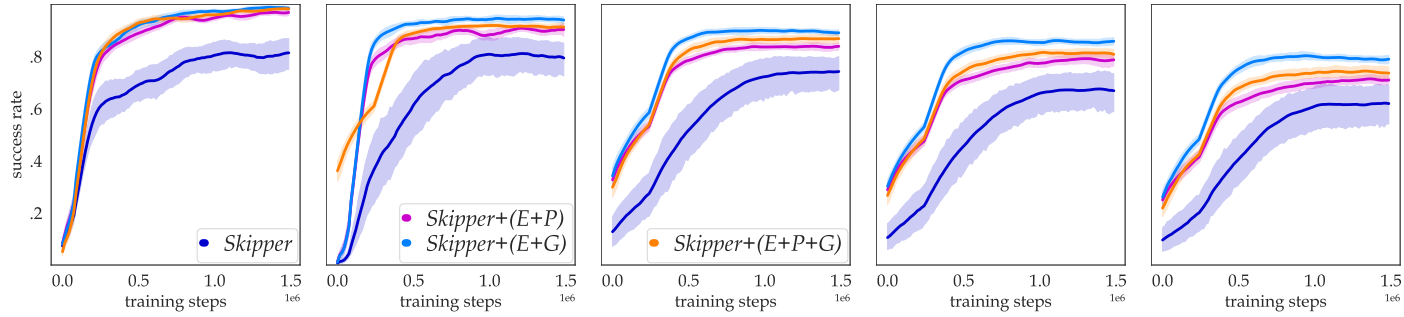


(a) *E.1 Estimation Errors* **(b)** *G.1 + E.1 Behavior Ratio* **(c)** *E.0 Errors* **(d)** *Aggregated OOD Perf.*

Figure 6.10: Skipper***'s Performance on RDS****: All error bars (95%-CI) are established over* 20 *seed runs.* ***a)*** *E.1 delusions in terms of* L_1 *error in estimated distance is visualized, throughout the training process.* ***b)*** *The curves represent the frequencies of choosing G.1 "states" whenever a selection of targets is initiated;* ***c)*** *The final estimation accuracies towards G.0 target states after training completed, across a spectrum of ground truth distances. In this figure, both distances (estimation and ground truth) are conditioned on the final version of the evolving policies; The state structure of* RDS *does not allow G.2 target states and the corresponding E.2 delusions;* ***d)*** *Each data point represents OOD evaluation performance aggregated over* 4×20 *newly generated tasks, with mean difficulty matching the training tasks.*

estimation accuracy as well as the OOD performance of P are not as bad as in SSM. This is possibly because RDS has much smaller state spaces, where EPISODE and PERTASK produce more similar results (than in large state spaces of SSM).

Breakdown of Task Performance In Fig. 6.11, we present the evolution of Skipper variants' performance on the training tasks as well as the OOD evaluation tasks throughout the training process. Note that Fig. 6.10 **d)** is an aggregation of all 4 sources of OOD performance in Fig. 6.11 **b-e)**.

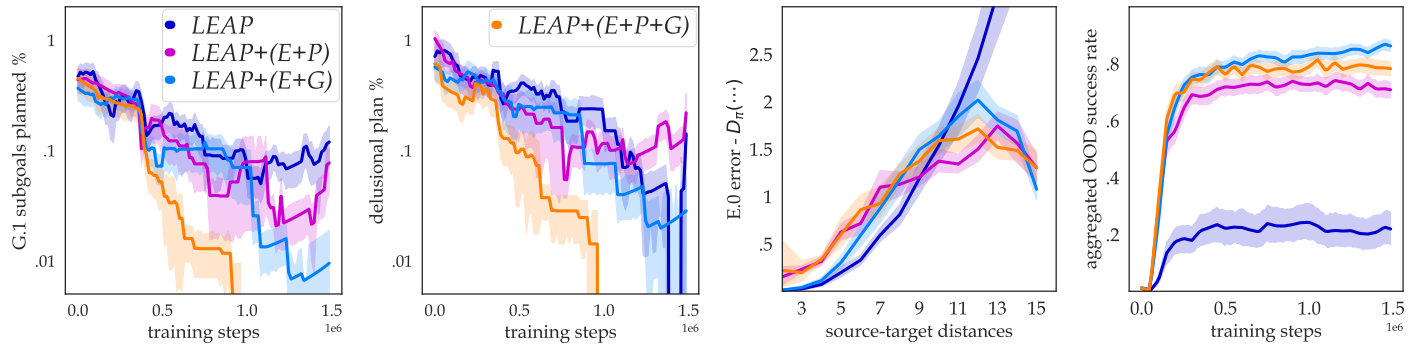


(a) *training,* $\delta = 0.4$ **(b)** *OOD evaluation,* $\delta = 0.25$ **(c)** *OOD evaluation,* $\delta = 0.35$ **(d)** *OOD evaluation,* $\delta = 0.45$ **(e)** *OOD evaluation,* $\delta = 0.55$

Figure 6.11: ***Evolution of OOD Performance of*** Skipper ***Variants on RDS****: All error bars (95%-CI) are established over* 20 *seed runs.*

LEAP on RDS (Exp. 4/8)

This set of experiments focus on LEAP's performance on RDS. Similarly, we present the evaluative metrics in Fig. 6.12.

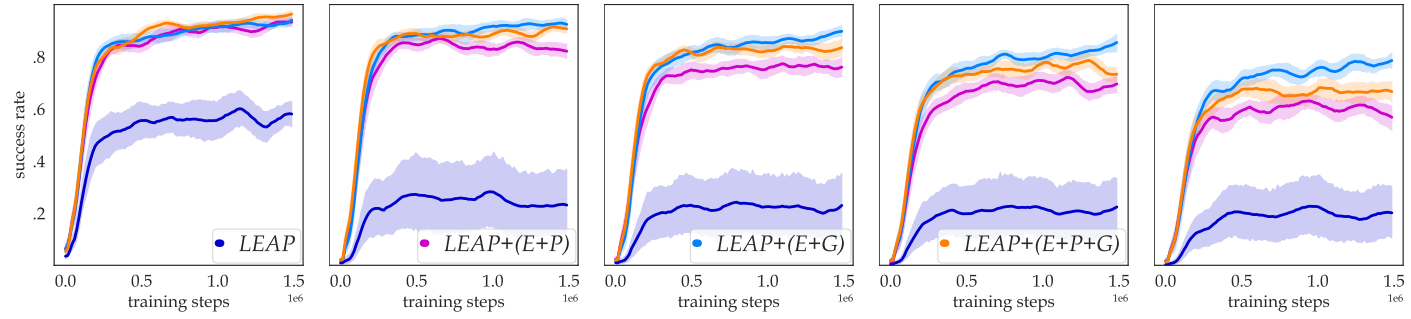


(a) *G.1 Ratio in Planned Sequence* **(b)** *Delusional Plan Ratio* **(c)** *E.0 Estim. Errors* **(d)** *Aggregated OOD Performance*

Figure 6.12: LEAP***'s Performance on RDS****: All error bars (95%-CI) are established over* 20 *seed runs.* ***a)*** *Ratio of G.1 subgoals among the planned sequences;* ***b)*** *Ratio of planned sequences containing at least one G.1 target;* ***c)*** *The final estimation accuracies towards G.0 target states after training completed, across a range of ground truth distances. In this figure, both distances (estimation and ground truth) are conditioned on the final version of the learned policies;* ***d)*** *Each data point represents OOD evaluation performance aggregated over* 4×20 *newly generated tasks, with mean difficulty matching the training tasks.*

The conclusions are similar, despite that the OOD performance gain by addressing delusions is significantly higher than in SSM.

Breakdown of Task Performance In Fig. 6.13, we present the evolution of LEAP variants' performance on the training tasks as well as the OOD evaluation tasks throughout the training process. Note that Fig. 6.12 **d)** is an aggregation of all 4 sources of OOD performance in Fig. 6.13 **b-e)**.



(a) *training,* $\delta = 0.4$ **(b)** *OOD evaluation,* $\delta = 0.25$ **(c)** *OOD evaluation,* $\delta = 0.35$ **(d)** *OOD evaluation,* $\delta = 0.45$ **(e)** *OOD evaluation,* $\delta = 0.55$

Figure 6.13: ***Evolution of OOD Performance of*** LEAP ***Variants on RDS****: All error bars (95%-CI) are established over* 20 *seed runs.*

6.6.2 Rejecting Destabilizing Updates in Background Planning (Exp. 5/8 - Exp. 6/8)

For background planning agents, since they are by design not as good as decision-time planning agents in OOD generalization, we only compare the difference in their training performance.

These two sets of experiments focus on a rollout-based background TAP agent - the classical 1-step Dyna (Sutton, 1991), which uses its learned transition model to generate next states from existing states to construct

(a) *Convergence to Optimal Value*

(b) *Training Performance*

Figure 6.14: Dyna***'s Performance on SSM****: All error bars (95%-CI) are established over* 20 *seed runs. Compared to the baseline Dyna, Dyna+ rejects the updates toward* 1*-infeasible generated states flagged by the evaluator, powered by (E+P+G). **a)** Evolving mean* L_1 *distances between estimated* Q *& optimal values; **b)**: task performance on the* 50 *training tasks & rate of Dyna+ rejecting updates.*

simulated transitions that are used to update the value estimator, *i.e.* a "Dyna update". Jafferjee et al. (2020) demonstrated the benefit when the delusional Dyna updates bootstrapped on hallucinated targets are rejected with an oracle. We replace the oracle using our learned evaluator.

With the same training setup, in Fig. 6.14, we present the empirical results of how target rejection can significantly improve the performance of Dyna on SSM. The rejection rate stabilizes as both the generator and the evaluator learns. These observations are consistent with Exp. $^6/_8$, presented in Sec. 6.6.2.[4]

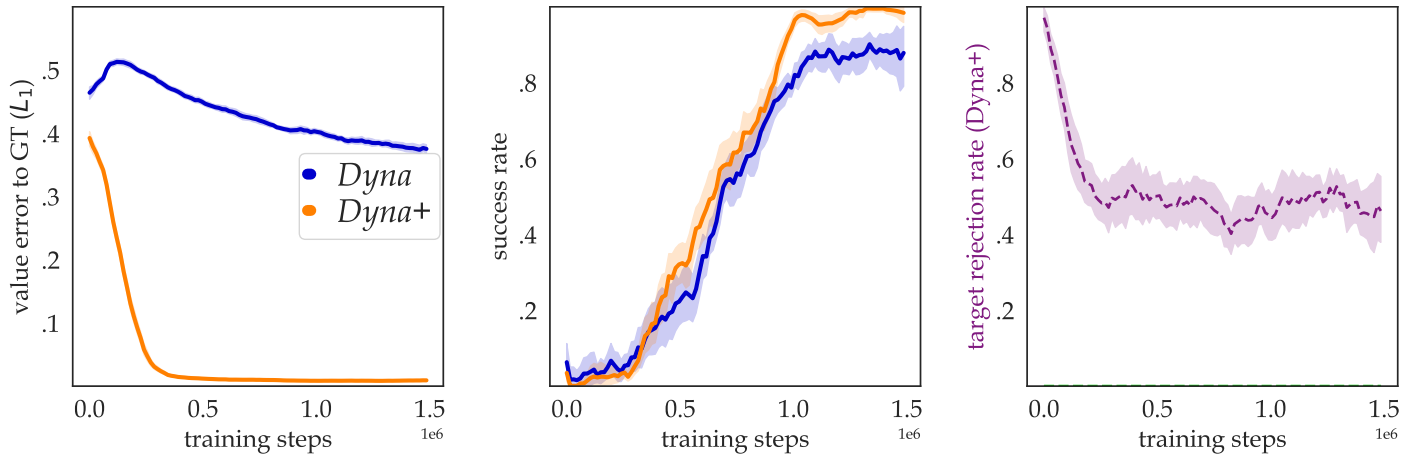
In Fig. 6.15, we present the empirical performance of a Dyna variant with rejection enabled by (E+P+G), which is significantly better than the baseline.

6.6.3 Feasibility Convergence to Non-Singleton Targets (Exp. $^7/_8$ & $^8/_8$)

We test if our implemented feasibility evaluator for Exp. $^1/_8$ - Exp. $^4/_8$ could withstand targets that are non-singleton. In its previous implementation, we use h to enforce the that the targets are singletons. In fact, each $\boldsymbol{g}^{\odot}$ takes the form of a state representation and h is only activated if a state with exactly the same representation is reached. For the non-singleton experiments however, we let h activate when a state is within distance one to the target state, effectively expanding each target set from size 1 to maximally size 5. Given the new termination mechanisms enforced by the new h, each target now, despite still taking the form of a state representation, has a new meaning. This setting mirrors the goal-conditioned path planning agents that seeks to reach certain neighborhoods of the planned waypoints.

With this setting, we can also intuitively analyze the composition of the target set. Specifically, if one of the member state is G.2, then the whole target set are fully made of G.2. If all the 5 states are out of the state space, then the target is fully composed of G.1. For SSM, a target in the temporarily unreachable situation, *e.g.*, $s \in \langle 1, 1\rangle$ with target encoding $s^{\odot} \in \langle 0, 1\rangle$, could be composed of not only G.2 states but also some G.1.

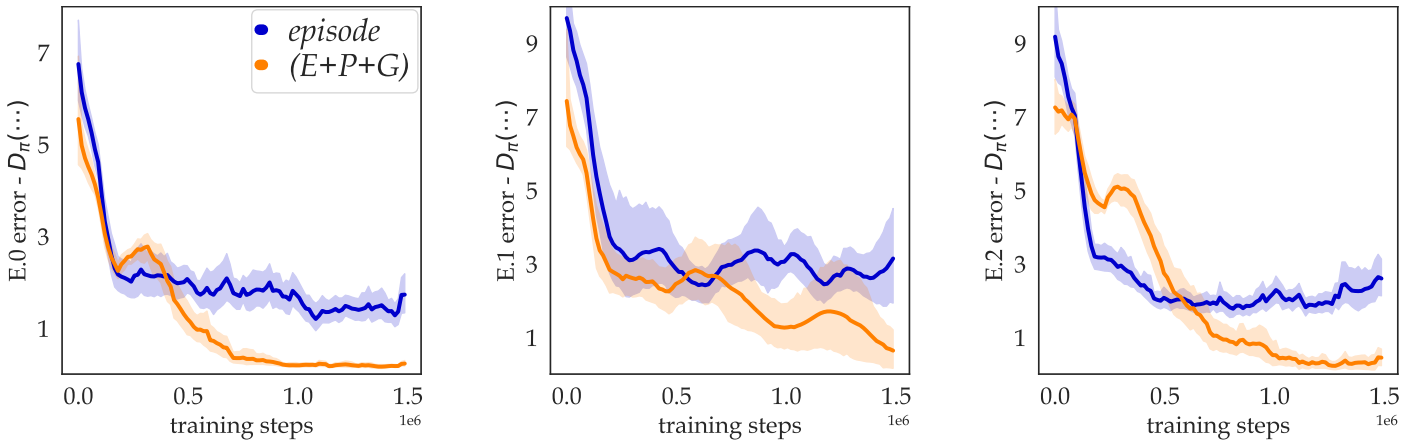
[4]The implementation here can be extended to fixed-horizon rollout agents. In the Sec. 8.3.3, we provide details on how we applied our Dyna solution to DreamerV2 (Hafner et al., 2021).



(a) *Convergence to Optimal Value* **(b)** *Training Performance* **(c)** *Training Performance*

Figure 6.15: Dyna***'s Performance on RDS****: All error bars (95%-CI) are established over* 20 *seed runs.* ***a):*** *Evolving mean* L_1 *distances between estimated* Q *values & ground truth optimals;* ***b):*** *evaluation performance on the* 50 *training tasks;* ***c):*** *rate of rejecting Dyna updates.*

We apply the new h to evaluator training and to the ground truth DP solver, and then compare their differences. As we could observe from Fig. 6.16, the proposed feasibility evaluator, with the help of the two assistive hindsight relabeling strategy, significantly reduces the feasibility errors in all categories.



(a) *Evolution of E.0 Errors* **(b)** *Evolution of E.1 Errors* **(c)** *Evolution of E.2 Errors*

Figure 6.16: ***Feasibility of Non-Singleton Targets on SSM****: All error bars (95%-CI) are established over* 20 *seed runs.* ***a)*** *Evolution of E.0 error;* ***b)*** *Evolution of E.1 error;* ***c)*** *Evolution of E.2 error; The training data is acquired with random walk, since the introduced non-singleton targets do not lead to adequate performances.*

We observe the similar results in RDS, presented in Fig. 6.17.

6.7 Summary

We characterized how generator hallucinations can cause trouble for TAP agents. Then, we proposed to evaluate the feasibility of targets *s.t.* the infeasible hallucinations can be properly rejected during planning.

(a) *Evolution of E.0 Errors*

(b) *Evolution of E.1 Errors*

Figure 6.17: ***Feasibility of Non-Singleton Targets on RDS****: All error bars (95%-CI) are established over* 20 *seed runs.* ***a)*** *Evolution of E.0 error;* ***b)*** *Evolution of E.1 error; The training data is acquired with random walk, since the introduced non-singleton targets do not lead to adequate performances.*

We proposed a combination of learning rules, architectures and data augmentation strategies that leads to robust and accurate output when the proposed evaluator is applied. In experiments, we showed that the evaluator can significantly address the harm of hallucinated targets in various kinds of planning agents.

Chapter 7

Discussions: Conclusions, Limitations & Future Work

In this chapter we re-iterate the contributions of the thesis, discuss the limitations of the work and avenues for future research.

7.1 Summary of Contributions

This thesis explores brain-inspired model-based deep RL agents capable of effectively generalizing their learned skills to new, target application environments. Our research argues that the difficulties of generalization stem from the absence of appropriate reasoning abilities, which are necessary for agents to adapt to novel situations (Kahneman, 2017). To address this, we introduced components inspired by higher cognitive functions involved in human conscious planning, significantly enhancing the agents' zero-shot OOD generalization capabilities (Sylvain et al., 2020). Furthermore, we examined the formation of delusional behaviors in Target-Assisted Planning (TAP) agents, by leveraging insights from the human brain (Kiran and Chaudhury, 2009).

The contributions of the thesis work clearly met the research objectives set before the start of my doctoral study.

We now revisit the main original contributions of this thesis and the implication of the findings.

7.1.1 Conscious Planning: Spatially-Abstract Decision-Time Reasoning

Chap. 4 aimed to address the problem of unsatisfactory generalization in the skills learned by existing RL agents, which are often sensitive to environmental perturbations and OOD problem changes (Ada et al., 2024). To tackle this, we designed a decision-time planning agent capable of dynamically focusing on the most relevant aspects of the state, thereby enhancing OOD generalization - making the first such contribution in the literature at the time when the work was carried out. The agent learns to dynamically focus on relevant (partial) aspects of the state during reasoning, while ignoring irrelevant environmental distractors that might hinder the generalization of its learned skills. Drawing inspiration from conscious decision-making behaviors in humans (Baars, 1993, 2002; Dehaene et al., 2020), we implemented this bottleneck mechanism

using top-down semi-hard attention (Sec. 4.4), and developed a comprehensive system of DRL architectures supported by set-based representations, end-to-end learning, and tree search with Model Predictive Control (MPC). Finished in late 2020, this work was among the first to introduce transformer-based architectures for computational decision-making. In our generalization-focused experimental settings, the proposed bottleneck mechanism enables the agent to selectively focus its computational resources on relevant objects for planning, leading to a significant improvement in OOD performance. This work sparked discussions on how knowledge of human higher-level cognitive functions can be leveraged to enhance the generalization capabilities of computational decision-making systems. The benefits of the focused reasoning induced by the bottleneck algorithm allow CP to generalize better and this probably applies to other learning systems as well.
This work also made some specific original contributions:

- **Set-Based / Object-Oriented Dynamics Model & Other RL Components**: To the best of our knowledge, at the time the preprint of this project was published on arxiv.org, we were the first to successfully implement a set-based (object-oriented) latent space dynamics model for decision-time planning that was trained end-to-end without relying on a reconstruction signal. We developed a robust and effective design that has since been widely acknowledged and adopted, inspiring subsequent methods. Specifically, we addressed the "alignment problem", where positional encoding and object features can create confusion when matching estimates to training targets. Traditionally, this problem was tackled using matching losses, which often resulted in suboptimal performance while incurring significant computational cost. We overcame this challenge by designing an object representation that explicitly separates the dimensions of features and positional encodings. This design allows positional encodings to remain fixed during training, enabling them to be used for matching objects between the estimated set and the update target set. As a result, we could use simple losses, such as L_2, for effective training (Sec. 4.3.3). Additionally, we developed an entire architecture capable of conducting set-based RL end-to-end, including set-based state representation encoder, set-based value estimator, *etc.*.

- **Spatial Abstraction - Top-Down Semi-Hard Attention Bottleneck**: The most significant contribution of this work is the successful implementation of a top-down attention-based bottleneck component that restricts the agent's reasoning abilities to a limited set of object slots (Sec. 4.4). The dynamic selection of a subset of objects within the state set is conditioned on the agent's intention during the tree search of decision-time planning, as well as the relationships between the objects in the state. The use of semi-hard attention enabled backpropagation through a hard-selected continuous bottleneck, circumventing the numerical challenges associated with a fully hard attention bottleneck, where gradients cannot flow directly. Our insights into consciousness in the first sense guided the design of this bottleneck to facilitate generalization, and we successfully integrated this feature into RL (Dehaene et al., 2020).

- **Using Multiple Predictive Losses to Enhance Set-based State Representations**: One key design of the agent is that all the training loss terms collectively shape the state representation (Sec. 4.3.4), a concept traditionally applied to vectorized state representations but not to set-based representations. Our approach aimed to make the state representation capable of predicting values, reward transitions, terminal state status, and latent state dynamics, all at the same time, using all losses to numerically regulate each other.

We validated this methodology and, in the process, discovered that KL-divergences, when used as loss terms, tend to balance each other due to their shared value ranges.

7.1.2 Skipper Framework: Spatio-Temporal Abstractions for Planning

Chap. 5 aimed to push the boundaries of existing decision-time planning methods, enabling them to plan in a way that is both spatially and temporally abstract — similar to how conscious planning in the human brain spans sparse decision points and focuses on relevant aspects of environmental states (Dehaene et al., 2020; Bengio, 2019). To achieve this, we developed a framework called Skipper, which automatically decomposes an overall task into smaller, more manageable steps by leveraging abstractions in both the temporal and spatial dimensions. Skipper employs a constrained form of option-based planning, building on the consciousness-inspired spatial abstraction mechanisms discussed in Chap. 4, particularly when considering each decomposed step (Sec. 5.3.1). The option-based planning in Skipper is conducted over proxy problems, a constrained form of SMDP for learning a problem decomposition of an overall MDP that aligns with the agent's capability to handle each decomposed step during the divide-and-conquer of the given task. Furthermore, we proved that the framework's performance on proxy problems is guaranteed under assumptions that are achievable in practice. This work demonstrates that spatially and temporally abstract planning is not only feasible, but also has performance guarantees, offering a promising direction for option-based planning. A clever way of blending spatio-temporal abstractions is probably a common need for learning systems that seek to tackle complex tasks.
Specific technical contributions are as follows:

- **Proxy Problems to Divide-and-Conquer**: We proposed proxy problems, a constrained form of SMDP, tailored for goal-conditioned planning (Sec. 5.2). We demonstrated that, given a set of checkpoints (a subset of the state space), proxy problems can be used to decompose a complex task into smaller, more manageable steps. Under appropriate assumptions, we proved that, as long as certain values can be estimated, proxy problems can enable optimal decision-making, providing a performance guarantee for planning agents that use them. Proxy problems anchor the Skipper framework in principled theory, thus distinguishing it from heuristic-based methods in temporally-abstracted reasoning.

- **Per-Sample TD-based Learning Rules Converging to Quantities Needed for Proxy Problems**: We designed learning rules that provably enable the agent to learn the essential values for optimal decision-making in proxy problems (Sec. 5.3.1). Additionally, we introduced a simple and general technique that allows an agent to interchangeably estimate the distribution of distances and cumulative discount between states, using C51-style distributional learning (Bellemare et al., 2017). These learning rules obey the technical assumptions regarding estimation accuracy for decision-making with proxy problems.

- **Extending Spatial Abstraction to Convolutional Features**: We extended the spatial abstraction mechanism proposed in Sec. 4.4 to feature maps learned with Convolutional Neural Networks (CNNs) (LeCun et al., 1989), making the approach much more generally applicable (Sec. 5.3.1).

- **Context-Aware Conditional Goal Generator** : We proposed a training loss for the goal generator that optimizes an alternative Evidence Lower BOund (ELBO) to produce goals that are likely to occur within

the same episode, given the context of the current state (Sec. 5.3.2). Compared to predictive models that are widely used in model-based RL, this generator directly proposes goals that be arbitrarily far away from the current state. The generator thus serves as the source for the vertices of proxy problems.

7.1.3 Rejecting Hallucinations: Addressing Delusional Planning Behaviors

Chap. 6 proposes a novel perspective from human psychiatry to analyze a common issue faced by TAP agents, planning agents that use generative models to sample state targets during planning: their tendency to be confused by hallucinated state targets, which undermines their performance and poses safety risks (Bengio et al., 2024). This perspective allowed us to identify the causes of delusional behaviors shared by seemingly different kinds of planning agents and to unify them in one framework, enabling a solution that resembles the coordination between the belief formation and evaluation systems in human psychopathology (Kiran and Chaudhury, 2009). We then proposed a combination of update rules, architecture, and hindsight relabeling strategy to enable learning an add-on feasibility evaluator that can reject proposed targets when they are not trustworthy, *i.e.*, when they are a result of hallucination. After applying the proposed add-on feasibility evaluator to existing agents, we observed a significant reduction in delusional planning behaviors in several types of planning agents, leading to significant improvements in performance. Being able to understand that certain plans are off-limits may be a necessity for any agentic AI composed of learning systems.
More specifically, we make the following technical contributions:

- **TAP Framework**: We abstract seemingly different kinds of existing planning methods that generate state targets into a single framework, Target-Assisted Planning (TAP), whose difficulties in dealing with hallucinations can be addressed with a unified solution. The two key components in TAP agents, the generator, and the evaluator, directly correspond to the belief formation and belief evaluation systems in the human brain, whose coordination is responsible to addressing delusions (Sec. 6.2.1). The TAP framework allows us to clearly highlight the problem of delusional planning behaviors in related RL methods, identify its causes, and propose an effective mitigation strategy: when a state target is a result of hallucination, its evaluation will not be trustworthy, and thus it should be rejected.

- **Categorization of Hallucinations**: Using the TAP framework, by incorporating a temporally-aware perspective, we differentiated two types of hallucinated states that TAP agents are prone to pursue: the permanently unreachable target states and the temporarily unreachable target states (Sec. 6.3). We then used this insight to analyze the more general case, where targets corresponds to sets of states.

- **Strategy of Rejecting Hallucinations**: Inspired by the belief evaluation system in the brain, we proposed to learn a feasibility evaluator, which acts as a firewall to reject targets hallucinated by the generative model in TAP agents. We carefully examined the challenges of learning such an evaluator and proposed a solution that is robust against feasibility delusions. This solution can be directly applied to existing TAP agents as an add-on without the need of changing the baseline RL methods. We systematically validated that our proposed solution leads to a significant reduction in delusional planning behaviors and improved empirical performance of various planning agents (Sec. 6.6).

Having discussed the contributions to original knowledge made in this thesis, we now turn to a discussion of the limitations of the work.

7.2 Limitations

DRL research is still significantly hindered by the sensitivity of existing methods. This thesis aimed to improve generalization in DRL, while simultaneously being shaped by the very challenges it sought to address. Although the research in this thesis primarily focuses on improving the generalization of RL agents to make them more applicable in the real world, a common limitation of the works presented here is that the experiments are somewhat limited in scope. For the sake of experimental rigor, we concentrated on demonstrating our claims using agents with simple neural network architectures in minimalist, carefully controlled environments. These results would be more compelling and impactful if we could extend our experiments to widely-used simulated benchmark suites, such as ProcGen (Cobbe et al., 2020), or, ideally, real-world tasks. However, our efforts in this direction have been hindered by the fact that this research area is still underexplored, making it difficult to identify appropriate baseline methods and experimental settings to validate our ideas convincingly and that such experiments would require computational resources not at our disposal. The primary object of study in the thesis is inductive bias, which is always a tradeoff of effectiveness in some problems for ineffectiveness in others. While our proposed components / framework could be useful for dynamics-consistent zero-shot OOD generalization tasks, OOD challenges are way more vast than being dynamics-consistent: our methods will not be applicable to those distributional shifts with inconsistent / unknown local dynamics. These more difficult OOD challenges would likely require online fusion of learned skills and these possibilities expose areas for future study. Additionally, the thesis argues that decision-time planning is more suited to task generalization than background planning. However, it is also arguably more susceptible to model error or misspecification. I acknowledge this because we did not claim to have found a cure-all.
I will now discuss the limitations of each project in detail.

7.2.1 Limitations of Work on Conscious Planning (Chap. 4)

The conscious planning work presented in Chap. 4 served as a proof-of-concept of an interesting research direction: System-2 DRL. Despite its significant novelty and the usefulness of the core top-down semi-hard attention bottleneck, the approach has several limitations:

- Although, in this project, a form of spatial abstraction is achieved with each search step in planning, the proposed agent still plans over the most atomic timesteps. Thus, as the search depth increases, not only does the number of search nodes grow exponentially, but so does the accumulation of errors due to imperfections in the learned models (Janner et al., 2019). This approach has limited long-term potential because even a comprehensive model that predicts the environment in great detail would struggle, given the search demands required for longer-term planning, which can be further exacerbated by stochasticity in the environment. Incorporating temporally abstract actions, such as options, could help mitigate this problem. While promising, introducing temporal abstraction into model-based RL is a non-trivial task that requires

careful investigation. We committed to exploring this challenge in Skipper (Chap. 5). We also investigated how to deal with imperfect models and hallucinated state targets, and proposed a generic solution in Chap. 6.

- During experimentation, we found that the constant per-step replanning used by the agents imposed significant computational burdens. This suggests that constant replanning may be prohibitive in environments that require rapid reaction, particularly when using a computationally expensive set-based transition model. A more efficient planning strategy could involve controlling when and where the agent performs replanning, potentially by estimating uncertainty. We addressed this challenge in Chap. 5, where temporally abstract planning requires only sparse replanning, triggered either by a timeout or when a target checkpoint is achieved.

- The set-based dynamics model is not yet capable of learning stochastic dynamics. Since the environments in Sec. 4.5 are fully deterministic, we did not extend the design to account for stochasticity. The challenge lies in implementing a latent sample space for an end-to-end trainable set-to-set framework. My intuition is that this can be done via variational approaches (Kingma and Welling, 2014).

Recent years, the newly developed methods of learning a discrete bottleneck has given me more inspirations on approaching this project differently, which I would leave for my future research.

7.2.2 Limitations of Work on Skipper (Chap. 5)

Although the spatio-temporal abstractions used in Skipper offer clear advantages and represent a step closer to human-like planning, Skipper is still undeniably distant from the level of conscious planning exhibited by humans. In my view, there are two major limitations of the current Skipper framework.

- **Planning without Task-Awareness**: While the planning process is spatially and temporally abstract, it is not task-informed. Specifically, future checkpoints are generated randomly by sampling from the partial description space. Despite post-processing steps such as pruning, these checkpoints do not prioritize the most predictable or important states, which are critical for forming a meaningful long-term plan. To plan as efficiently as humans, reasoning agents need to be able to decompose tasks into abstract, task-specific steps.

- **Planning with States**: The framework relies on proxy problems that operate at the state level. In contrast, human planning does not focus on detailed states but rather on high-level transitions or changes in state, often over partial or evolving states. We will discuss this point more in Sec. 7.3.

Additionally, there are several technical limitations in the proposed framework:

- **Continuous State Spaces & Partial Observability**: The current implementation of Skipper is designed for fully observable tasks with discrete state and action spaces. We adopted this minimalist approach to isolate challenges unrelated to the core idea of this work. Skipper is naturally compatible with continuous action spaces, and the only modification required is replacing the baseline agent with one that supports continuous actions, such as TD3 (Fujimoto et al., 2018). However, when it comes to continuous state spaces,

identifying when a checkpoint has been reached becomes more challenging. Specifically, the agent may never exactly fulfill a target checkpoint, which requires us to either use a distance metric to approximate state equivalence or to rely on the equivalence of partial descriptions (as implemented in the current version). Our current implementation sets the partial descriptions as bundles of binary variables, allowing for quick and straightforward comparison across any state space. However, we know that the overwhelming amount of states will pose a tremendous challenge for this design, because each instantiation of a partial description can correspond to a huge set of states. Regarding partial observability, although the current implementation does not incorporate a recurrent mechanism, the framework is compatible with such an approach. Handling partial observability would require augmenting the state encoder with recurrent or memory-based mechanisms, and ensuring the checkpoint generator works directly with the learned state representations. We acknowledge that future work is needed to evaluate Skipper's performance on popular partially observable benchmark suites, which would require incorporating components to handle partial observability and scaling up the architecture for greater expressive power.

- **More Intuitive Theory Boundary**: We do not know the precise boundaries of the proxy problems theory, since it only indicates performance guarantees based on a condition of estimation accuracy, which in turn does not correspond trivially to a set of well-defined problems. We should explore, outside the scope of sparse-reward navigation, how this approach can be used to facilitate better generalization, and at the same time, try to find more powerful theoretical results that can guide us better.

7.2.3 Limitations of Work on Rejecting Hallucinations (Chap. 6)

The limitations of the work presented in Chap. 6 primarily lie in the experiments. If possible, we would like to test our proposed feasibility evaluator on a wider range of TAP methods in more diverse environments, which could demonstrate the benefits of addressing delusional planning behaviors more convincingly. Additionally, our experiments focused on gridworld environments, while we would like to verify the performance gain on environments with more complex features.
Some other planning agents propose "targets" that do not directly correspond to reaching sets of states, but instead, maximize certain signals without verifying if they reached the proposed targets, *i.e.*, without providing h (defined in Sec. 6.2.1). We would like to investigate such agents in future work, to understand how they are impacted by hallucinations.

7.3 Future Work

Abstractions are essential for effective and efficient reasoning. This means that an agent must learn and reason at appropriate levels of detail, recognizing which aspects of the task are important and which are not, both spatially and temporally, as discussed in Chap. 5.
Humans can decompose complex tasks into abstract steps that are topologically sorted, encompassing spatio-temporal abstractions that are inherently task-aware. My goal is to enable RL agents to reason similarly.
From a traditional RL perspective, understanding how a computational agent could achieve abstract planning behavior is critical for studying decision-making, as it could help avoid the challenges associated with rea-

soning over excessive details, among other issues. In essence, this approach shifts the difficulty of learning accurate models to discovering crucial events, *i.e.*, the robustness of the agent's models can be improved if the abstract space in which the agent plans is well-constructed.

For future work, I aim to develop decision-making agents that make abstract plans based on partial changes in states, mimicking human reasoning. When planning a trip to Paris, for instance, experienced humans can easily break down the task into abstract steps without needing to reconstruct all the environmental details. A rough decomposition of the task might include abstract steps such as buying flight tickets, booking hotels, *etc.*. Each of these steps represents a partial change to the state, without the need to consider irrelevant details. For example, one does not need to think about what to have for dinner when planning these steps, and the states corresponding to completing each step can be infinite. The ability to discover and reason with partial information within an efficient abstract planning space allows humans to plan effectively for the future and generalize OOD.

In pursuit of this behavior, I propose that MBRL agents should reason in a space of abstract situations and events, focusing on sets of states defined by partial descriptions and their changes. This space should be discovered by the agent autonomously by trial-and-error.

For over a year now, I have been working on implementing this formulation, which includes abstract planning behaviors based on partial changes in states over extended periods of time. I am currently investigating how a generic credit-assignment mechanism could guide abstract planning.

This is the end of the main parts of this thesis. Thank you very much for reading.

Bibliography

D. Abel, W. Dabney, A. Harutyunyan, M. K. Ho, M. Littman, D. Precup, and S. Singh. On the expressivity of markov reward. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 7799–7812, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/4079016d940210b4ae9ae7d41c4a2065-Paper.pdf.

S. E. Ada, E. Oztop, and E. Ugur. Diffusion policies for out-of-distribution generalization in offline reinforcement learning. *IEEE Robotics and Automation Letters*, 9(4):3116–3123, 2024. URL https://dblp.org/rec/journals/ral/AdaOU24.

S. K. Aithal, P. Maini, Z. Lipton, and J. Z. Kolter. Understanding hallucinations in diffusion models through mode interpolation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 134614–134644, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/f29369d192b13184b65c6d2515474d78-Paper-Conference.pdf.

R. T. Akella, B. Eysenbach, J. Schneider, and R. Salakhutdinov. Distributional distance classifiers for goal-conditioned reinforcement learning. In *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*, 2023. URL https://openreview.net/forum?id=yuLAGjRwk9.

S. Alver and D. Precup. Minimal value-equivalent partial models for scalable and robust planning in lifelong reinforcement learning. In *Conference on Lifelong Learning Agents*, volume 232, pages 548–567. PMLR, 22–25 Aug 2023. URL https://proceedings.mlr.press/v232/alver23a.html.

S. Alver and D. Precup. A look at value-based decision-time vs. background planning methods across different settings, 2024. URL https://arxiv.org/abs/2206.08442.

S. Alver, A. Rahimi-Kalahroudi, and D. Precup. Partial models for building adaptive model-based reinforcement learning agents. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2024.

P. Amortila, D. J. Foster, N. Jiang, A. Krishnamurthy, and Z. Mhammedi. Reinforcement learning under latent dynamics: Toward statistical and algorithmic modularity. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 133007–133091, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/f0262cfa14a20f833fcdae1719b144ec-Paper-Conference.pdf.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf.

S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. VQA: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015. URL https://arxiv.org/abs/1505.00468.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

B. J. Baars. *A cognitive theory of consciousness*. Cambridge University Press, 1993. URL https://www.cambridge.org/universitypress/subjects/life-sciences/neuroscience/cognitive-theory-consciousness.

B. J. Baars. The conscious access hypothesis: Origins and recent evidence. *Trends in Cognitive Sciences*, 6(1): 47–52, 2002. URL https://doi.org/10.1016/S1364-6613(00)01819-2.

B. J. Baars et al. *In the theater of consciousness: The workspace of the mind*. Oxford University Press, USA, 1997. URL https://doi.org/10.1093/acprof:oso/9780195102659.001.1.

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, AAAI'17, page 1726–1734. AAAI Press, 2017. URL https://doi.org/10.1609/aaai.v31i1.10916.

J. Bae, K. Park, and Y. Lee. TLDR: Unsupervised goal-conditioned RL via temporal distance-aware representations. In *8th Annual Conference on Robot Learning (CoRL)*, 2024. URL https://openreview.net/forum?id=deywgeWmL5.

A. Bagaria, J. K. Senthil, and G. Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning (ICML)*, volume 139, pages 521–531. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/bagaria21a.html.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1409.0473.

E. Barnard. Temporal-difference methods and markov models. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(2):357–365, March 1993. URL https://doi.org/10.1109/21.229449.

H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and Chamfer matching: 2 new techniques for image matching. Technical report, SRI International Menlo Park CA Artificial Intelligence Center, 1977. URL https://doi.org/10.1016/0031-3203(77)90011-9.

M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 70, pages 449–458. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/bellemare17a.html.

R. Bellman, R. Corporation, and K. M. R. Collection. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957. URL https://books.google.ca/books?id=wdtoPwAACAAJ.

Y. Bengio. The consciousness prior, 2019. URL https://arxiv.org/abs/1709.08568.

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

Y. Bengio, G. Hinton, A. Yao, D. Song, P. Abbeel, T. Darrell, Y. N. Harari, Y.-Q. Zhang, L. Xue, S. Shalev-Shwartz, et al. Managing extreme AI risks amid rapid progress. *Science*, 384(6698):842–845, 2024. URL https://doi.org/10.1126/science.adn0117.

G. Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, 10(6):849–865, 1988. URL https://doi.org/10.1109/34.9107.

M. Bowling, J. D. Martin, D. Abel, and W. Dabney. Settling the reward hypothesis. In *International Conference on Machine Learning (ICML)*, volume 202, pages 3003–3020. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/bowling23a.html.

N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *Computer Vision – ECCV 2020*, pages 213–229. Springer International Publishing, 2020. URL https://doi.org/10.1007/978-3-030-58452-8_13.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 15084–15097, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/7f489f642a0ddb10272b5c31057f0663-Paper.pdf.

M. Chevalier-Boisvert, L. Willems, and S. Pal. Minimalistic gridworld environment for openai gym, 2018. URL https://github.com/Farama-Foundation/Minigrid. https://github.com/maximecb/gym-minigrid.

M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=rJeXCo0cYX.

M. Chevalier-Boisvert, B. Dai, M. Towers, R. Perez-Vicente, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 73383–73394, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/e8916198466e8ef218a2185a491b49fa-Paper-Datasets_and_Benchmarks.pdf.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/3de568f8597b94bda53149c7d7f5958c-Paper.pdf.

K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 119, pages 2048–2056. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/cobbe20a.html.

R. C. Conant and W. Ross Ashby. Every good regulator of a system must be a model of that system. *International Journal of Systems Science*, 1(2):89–97, 1970. URL https://doi.org/10.1080/00207727008920220.

K. Czechowski, T. Odrzygóźdź, M. Zbysiński, M. Zawalski, K. Olejnik, Y. Wu, L. u. Kuciński, and P. Mił oś. Subgoal search for complex reasoning tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 624–638, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/05d8cccb5f47e5072f0a05b5f514941a-Paper.pdf.

W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. URL https://arxiv.org/abs/1710.10044.

T. Dai, H. Liu, K. Arulkumaran, G. Ren, and A. A. Bharath. Diversity-based trajectory and goal selection with hindsight experience replay. In *PRICAI 2021: Trends in Artificial Intelligence: 18th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2021, Hanoi, Vietnam, November 8–12, 2021, Proceedings, Part III 18*, pages 32–45. Springer, 2021. URL https://arxiv.org/abs/2102.10309.

T. Davchev, O. O. Sushkov, J.-B. Regli, S. Schaal, Y. Aytar, M. Wulfmeier, and J. Scholz. Wish you were here: Hindsight goal selection for long-horizon dexterous manipulation. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=FKp8-pIRo3y.

G. Davidson and B. M. Lake. Investigating simple object representations in model-free deep reinforcement learning, 2020. URL https://arxiv.org/abs/2002.06703.

P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 5. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper_files/paper/1992/file/d14220ee66aeec73c49038385428ec4c-Paper.pdf.

S. Dehaene, H. Lau, and S. Kouider. What is consciousness, and could machines have it? *Science*, 358, 2020. URL https://science.sciencemag.org/content/358/6362/486.

M. P. Deisenroth and C. E. Rasmussen. PILCO: a model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, ICML'11, page 465–472. Omnipress, 2011. URL https://doi.org/10.1177/0278364911427547.

A. Deshpande, S. Sarma, A. Jha, and B. Ravindran. Improvements on hindsight learning, 2018. URL https://arxiv.org/abs/1809.06719.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 4171–4186. Association for Computational Linguistics, 2019. URL https://doi.org/10.18653/v1/N19-1423.

T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000. URL https://doi.org/10.1613/JAIR.639.

Y. Du, T. Lin, and I. Mordatch. Model-based planning with energy-based models. In *Proceedings of the Conference on Robot Learning*, volume 100, pages 374–383. PMLR, 30 Oct–01 Nov 2020. URL https://proceedings.mlr.press/v100/du20a.html.

Y. Duan, W. Mao, and H. Zhu. Learning world models for unconstrained goal navigation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 59236–59265, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/6cca3481ae66707958b824d37df40177-Paper-Conference.pdf.

M. Elsayed, G. Vasan, and A. R. Mahmood. Streaming deep reinforcement learning finally works, 2024. URL https://arxiv.org/abs/2410.14606.

A. Erraqabi, M. C. Machado., M. Zhao, S. Sukhbaatar, A. Lazaric, D. Ludovic, and Y. Bengio. Temporal abstractions-augmented temporally contrastive learning: An alternative to the laplacian in RL. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 180 of *Proceedings of Machine Learning Research*, pages 641–651, 01–05 Aug 2022. URL https://proceedings.mlr.press/v180/erraqabi22a.html.

B. Eysenbach, R. R. Salakhutdinov, and S. Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/5c48ff18e0a47baaf81d8b8ea51eec92-Paper.pdf.

B. Eysenbach, R. Salakhutdinov, and S. Levine. C-learning: Learning to achieve goals via recursive classification. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=tc5qisoB-C.

C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning (ICML)*, volume 80, pages 1515–1528. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/florensa18a.html.

S. L. Frank, W. F. Haselager, and I. van Rooij. Connectionist semantic systematicity. *Cognition*, 110(3):358–379, 2009. URL https://doi.org/10.1016/j.cognition.2008.11.013.

X. Fu, G. Yang, P. Agrawal, and T. Jaakkola. Learning task informed abstractions. In *International Conference on Machine Learning (ICML)*, volume 139, pages 3480–3491. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/fu21b.html.

S. Fujimoto and S. S. Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 20132–20145, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/a8166da05c5a094f7dc03724b41886e5-Paper.pdf.

S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods, 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

D. Ghosh, A. Gupta, and S. Levine. Learning actionable representations with goal conditioned policies. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=Hye9lnCct7.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT Press, 2016. URL https://www.deeplearningbook.org/.

A. Goyal and Y. Bengio. Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A*, 478(2266):20210068, 2022. URL https://doi.org/10.1098/rspa.2021.0068.

A. Gupta, G. Dar, S. Goodman, D. Ciprut, and J. Berant. Memory-efficient transformers via top-k attention. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*. Association for Computational Linguistics, 2021. URL https://arxiv.org/abs/2102.00354.

D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf.

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning (ICML)*, volume 97, pages 2555–2565. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/hafner19a.html.

D. Hafner, T. P. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

D. Hafner, K.-H. Lee, I. Fischer, and P. Abbeel. Deep hierarchical planning from pixels. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 26091–26104, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/a766f56d2da42cae20b5652970ec04ef-Paper-Conference.pdf.

D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models, 2024. URL https://arxiv.org/abs/2301.04104.

J. B. Hamrick, A. L. Friesen, F. Behbahani, A. Guez, F. Viola, S. Witherspoon, T. Anthony, L. H. Buesing, P. Veličković, and T. Weber. On the role of planning in model-based deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=IrM64DGB21.

N. Hansen, H. Su, and X. Wang. TD-MPC2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=Oxh5CstDJU.

H. v. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016. URL https://arxiv.org/abs/1509.06461.

N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/148510031349642de5ca0c544f31b2ef-Paper.pdf.

M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: combining improvements in deep reinforcement learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. URL https://arxiv.org/abs/1710.02298.

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=Sy2fzU9gl.

D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=H1Dy---0Z.

R. A. Howard. *Dynamic Programming and Markov Processes*. John Wiley, 1960. URL https://openlibrary.org/books/OL5798328M/Dynamic_programming_and_Markov_processes.

M. Igl, K. Ciosek, Y. Li, S. Tschiatschek, C. Zhang, S. Devlin, and K. Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/e2ccf95a7f2e1878fcafc8376649b6e8-Paper.pdf.

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=SJ6yPD5xg.

T. Jafferjee, E. Imani, E. Talvitie, M. White, and M. Bowling. Hallucinating value: A pitfall of dyna-style planning with imperfect environment models, 2020. URL https://arxiv.org/abs/2006.04363.

M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/5faf461eff3099671ad63c6f3f094f7f-Paper.pdf.

A. Jesson, N. Beltran Velez, Q. Chu, S. Karlekar, J. Kossen, Y. Gal, J. P. Cunningham, and D. Blei. Estimating the hallucination rate of generative ai. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 31154–31201, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/3791f5fc0e8e43730466afd2bcdb7493-Paper-Conference.pdf.

D. Kahneman. *Thinking, fast and slow*. Farrar, Straus and Giroux, 2017. URL https://www.penguinrandomhouse.com/books/89308/thinking-fast-and-slow-by-daniel-kahneman/.

L. Kaufman and P. Rousseeuw. *Partitioning Around Medoids (Program PAM)*, chapter 2, pages 68–125. John Wiley & Sons, Ltd, 1990. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470316801.ch2.

K. Khetarpal, Z. Ahmed, G. Comanici, and D. Precup. Temporally abstract partial models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 1979–1991, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/0f3d014eead934bbdbacb62a01dc4831-Paper.pdf.

J. Kim, Y. Seo, and J. Shin. Landmark-guided subgoal generation in hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 28336–28349, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/ee39e503b6bedf0c98c388b7e8589aca-Paper.pdf.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014. URL https://openreview.net/forum?id=33X9fd2-9FyZd.

C. Kiran and S. Chaudhury. Understanding delusions. *Industrial psychiatry journal*, 18(1):3–18, 2009. URL https://doi.org/10.4103/0972-6748.57851.

C. A. Knoblock. Automatically generating abstractions for planning. *Artificial intelligence*, 68(2):243–302, 1994. URL https://doi.org/10.1016/0004-3702(94)90069-8.

L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer Berlin Heidelberg, 2006. URL https://doi.org/10.1007/11871842_29.

V. Konda and J. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

G. Konidaris and A. Barto. Efficient skill learning using abstraction selection. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, page 1107–1112. Morgan Kaufmann Publishers Inc., 2009. URL https://www.ijcai.org/Proceedings/09/Papers/187.pdf.

A. R. Kosiorek, H. Kim, and D. J. Rezende. Conditional set generation with transformers, 2020. URL https://arxiv.org/abs/2006.16841.

T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/f442d33fa06832082290ad8544a8da27-Paper.pdf.

L. L. D. Langosco, J. Koch, L. D. Sharkey, J. Pfau, and D. Krueger. Goal misgeneralization in deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 162, pages 12004–12019. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/langosco22a.html.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. URL https://doi.org/10.1162/neco.1989.1.4.541.

J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/lee19d.html.

J. Lee, S. Yun, T. Yun, and J. Park. Gta: Generative trajectory augmentation with guidance for offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 56766–56801, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/67ea314d1df751bbf99ab664ae3049a5-Paper-Conference.pdf.

J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg. Ai safety gridworlds, 2017. URL https://arxiv.org/abs/1711.09883.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016. URL https://jmlr.org/papers/v17/15-522.html.

L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. *AI&M*, 1(2):3, 2006. URL https://anytime.cs.umass.edu/aimath06/proceedings/P21.pdf.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016. URL https://arxiv.org/abs/1509.02971.

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992. URL https://doi.org/10.1007/BF00992699.

S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982. URL https://doi.org/10.1109/TIT.1982.1056489.

C. Lo, K. Roice, P. M. Panahi, S. M. Jordan, A. White, G. Mihucz, F. Aminmansour, and M. White. Goal-space planning with subgoal models. *Journal of Machine Learning Research*, 25(330):1–57, 2024. URL https://jmlr.org/papers/v25/24-0040.html.

S. Löwe, K. Greff, R. Jonschkowski, A. Dosovitskiy, and T. Kipf. Learning object-centric video models by contrasting sets, 2020. URL https://arxiv.org/abs/2011.10287.

A. Manchin, E. Abbasnejad, and A. van den Hengel. Reinforcement learning with attention that works: A self-supervised approach. In *Neural Information Processing*, pages 223–230. Springer International Publishing, 2019. URL https://arxiv.org/abs/1901.07510.

T. A. Mann, S. Mannor, and D. Precup. Approximate value iteration with temporally extended actions. *Journal of Artificial Intelligence Research*, 53:375–438, 2015. URL https://www.jair.org/index.php/jair/article/view/10947.

A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *International Conference on Machine Learning (ICML)*, ICML '01, page 361–368. Morgan Kaufmann Publishers Inc., 2001. URL https://www.cs.iastate.edu/~amy/files/rl-dd_icml2001.pdf.

R. Mendonca, X. Geng, C. Finn, and S. Levine. Meta-reinforcement learning robust to distributional shift via model identification and experience relabeling, 2020. URL https://arxiv.org/abs/2006.07178.

M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961. URL https://doi.org/10.1109/JRPROC.1961.287775.

V. Mnih, N. Heess, A. Graves, and k. kavukcuoglu. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/09c6c3783b4a70054da74f2538ed47c6-Paper.pdf.

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*, 518(7540): 529–533, 2015. URL https://doi.org/10.1038/nature14236.

T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. Model-based reinforcement learning: A survey, 2022. URL https://arxiv.org/abs/2006.16712.

L. Moro, A. Likmeta, E. Prati, and M. Restelli. Goal-directed planning via hindsight experience replay. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=6NePxZwfae.

A. Mott, D. Zoran, M. Chrzanowski, D. Wierstra, and D. Jimenez Rezende. Towards interpretable reinforcement learning using attention augmented agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/e9510081ac30ffa83f10b68cde1cac07-Paper.pdf.

T. Mu, J. Gu, Z. Jia, H. Tang, and H. Su. Refactoring policy for compositional generalizability using self-supervised object proposals. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 8883–8894, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/64dcf3c521a00dbb4d2a10a27a95a9d8-Paper.pdf.

V. Myers, C. Zheng, A. Dragan, S. Levine, and B. Eysenbach. Learning temporal distances: Contrastive successor features can provide a metric structure for decision-making. In *International Conference on Machine Learning (ICML)*, 2024. URL https://openreview.net/forum?id=xQiYCmDrjp.

O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/e6384711491713d29bc63fc5eeb5ba4f-Paper.pdf.

A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/7ec69dd44416c46745f6edd947b470cd-Paper.pdf.

S. Nasiriany, V. Pong, S. Lin, and S. Levine. Planning with goal-conditioned policies. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c8cc6e90ccbff44c9cee23611711cdc4-Paper.pdf.

R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 10. MIT Press, 1997. URL https://proceedings.neurips.cc/paper_files/paper/1997/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

V. Pong, S. Gu, M. Dalal, and S. Levine. Temporal difference models: Model-free deep rl for model-based control, 2020. URL https://arxiv.org/abs/1802.09081.

I. Porada, K. Suleman, A. Trischler, and J. C. K. Cheung. Modeling event plausibility with consistent conceptual abstraction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1732–1743. Association for Computational Linguistics, jun 2021. URL https://aclanthology.org/2021.naacl-main.138/.

M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014. URL https://www.wiley.com/en-us/Markov%2BDecision%2BProcesses%3A%2BDiscrete%2BStochastic%2BDynamic%2BProgramming-p-9781118625873.

J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. *Dataset shift in machine learning*. Mit Press, 2022. URL https://doi.org/10.7551/mitpress/7921.003.0001.

S. Ramstedt and C. Pal. Real-time reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/54e36c5ff5f6a1802925ca009f3ebb68-Paper.pdf.

A. V. Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1): 497–528, 2009. URL https://anilvrao.com/Publications/ConferencePublications/trajectorySurveyAAS.pdf.

A. G. Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005. URL https://doi.org/10.1002/rnc.728. https://dspace.mit.edu/handle/1721.1/28914.

M. B. Ring. Child: A first step towards continual learning. *Machine Learning*, 28(1):77–104, 1997. URL https://doi.org/10.1023/A:1007331723572.

R. Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997. URL https://www.sciencedirect.com/science/article/pii/S0377221796003852.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. URL https://doi.org/10.1038/323533a0.

L. Russell, A. Hu, L. Bertoni, G. Fedoseev, J. Shotton, E. Arani, and G. Corrado. GAIA-2: A controllable multi-view generative world model for autonomous driving, 2025. URL https://arxiv.org/abs/2503.20523.

E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974. URL https://doi.org/10.1016/0004-3702(74)90026-5.

N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=SygwwGbRW.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2016. URL https://arxiv.org/abs/1511.05952.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. P. Lillicrap, and D. Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588, 2019. URL https://www.nature.com/articles/s41586-020-03051-4.

M. Schultheis, C. A. Rothkopf, and H. Koeppl. Reinforcement learning with non-exponential discounting. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 3649–3662, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/178b306c7ee66a66db2171646e17da36-Paper-Conference.pdf.

W. Schultz, P. Dayan, and P. R. Montague. A neural substrate of prediction and reward. *Science*, 275(5306): 1593–1599, 1997. URL https://doi.org/10.1126/science.275.5306.1593.

D. Shah, A. T. Toshev, S. Levine, and brian ichter. Value function spaces: Skill-centric state abstractions for long-horizon reasoning. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=vgqS1vkkCbE.

A. Shams and T. Fevens. Addressing different goal selection strategies in hindsight experience replay with actor-critic methods for robotic hand manipulation. In *2022 2nd International Conference on Robotics, Automation and Artificial Intelligence (RAAI)*, pages 69–73, 2022. URL https://arxiv.org/abs/2205.05055.

A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=HJgLZR4KvH.

L. X. Shi, J. J. Lim, and Y. Lee. Skill-based model-based reinforcement learning. In *Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=iVxy2eO601U.

D. Silver and K. Ciosek. Compositional planning using optimal option models. In *International Conference on Machine Learning (ICML)*, ICML'12, page 1267–1274. Omnipress, 2012. URL https://doi.org/10.1609/aaai.v26i1.8243.

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*, 529(7587):484–489, 2016. URL https://doi.org/10.1038/nature16961.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017a. URL https://doi.org/10.1038/nature24270.

D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, and T. Degris. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning (ICML)*, volume 70, pages 3191–3199. PMLR, 06–11 Aug 2017b. URL https://proceedings.mlr.press/v70/silver17a.html.

D. Silver, S. Singh, D. Precup, and R. S. Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021. URL https://doi.org/10.1016/j.artint.2021.103535.

P. Soulos, H. Conklin, M. Opper, P. Smolensky, J. Gao, and R. Fernandez. Compositional generalization across distributional shifts with sparse tree operations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 112836–112863, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ccfa9ba5a84d0e4c620093d27102b7c5-Paper-Conference.pdf.

R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin*, 2(4):160–163, 1991. URL https://dl.acm.org/doi/10.1145/122344.122377.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

R. S. Sutton, M. C. Machado, G. Z. Holland, D. Szepesvari, F. Timbers, B. Tanner, and A. White. Reward-respecting subtasks for model-based reinforcement learning. *Artificial Intelligence*, 324:104001, 2023. URL https://doi.org/10.1016/j.artint.2023.104001.

T. Sylvain, L. Petrini, and D. Hjelm. Locality and compositionality in zero-shot learning. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=Hye_V0NKwr.

E. Talvitie. Model regularization for stable sample rollouts. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, UAI'14, pages 780–789, 2014. URL https://arxiv.org/abs/1406.0834.

E. Talvitie. Self-correcting models for model-based reinforcement learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, volume 31 of *AAAI'17*. AAAI Press, Feb. 2017. URL https://ojs.aaai.org/index.php/AAAI/article/view/10850.

E. Talvitie and S. Singh. Simple local models for complex dynamical systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 21, 2008. URL https://proceedings.neurips.cc/paper_files/paper/2008/file/f76a89f0cb91bc419542ce9fa43902dc-Paper.pdf.

C. Tang and R. R. Salakhutdinov. Multiple futures prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/86a1fa88adb5c33bd7a68ac2f9f3f96b-Paper.pdf.

Y. Tang, D. Nguyen, and D. Ha. Neuroevolution of self-interpretable agents. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 414–424. Association for Computing Machinery, 2020. URL https://doi.org/10.1145/3377930.3389847.

S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 7. MIT Press, 1994. URL https://proceedings.neurips.cc/paper_files/paper/1994/file/7ce3284b743aefde80ffd9aec500e085-Paper.pdf.

S. Tian, S. Nair, F. Ebert, S. Dasari, B. Eysenbach, C. Finn, and S. Levine. Model-based visual planning with self-supervised functional distances, 2020. URL https://arxiv.org/abs/2012.15373.

A. van den Oord, O. Vinyals, and k. kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf.

R. van Gulick. Consciousness. In *Stanford Encyclopedia of Philosophy*. Stanford: Metaphysics Research Lab, 2004. URL https://plato.stanford.edu/entries/consciousness/.

H. P. van Hasselt, M. Hessel, and J. Aslanides. When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/1b742ae215adf18b75449c6e272fd92d-Paper.pdf.

H. van Seijen and R. Sutton. A deeper look at planning as learning from replay. In *International Conference on Machine Learning (ICML)*, volume 37, pages 2314–2322. PMLR, 07–09 Jul 2015. URL https://proceedings.mlr.press/v37/vanseijen15.html.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 70, pages 3540–3549. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/vezhnevets17a.html.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019. URL https://doi.org/10.1038/s41586-019-1724-z.

T. Wang, R. Liao, J. Ba, and S. Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2018a. URL https://openreview.net/forum?id=S1sqHMZCb.

X. Wang, W. Xiong, H. Wang, and W. Y. Wang. Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*, page 38–55. Springer-Verlag, 2018b. URL https://doi.org/10.1007/978-3-030-01270-0_3.

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992. URL https://doi.org/10.1007/BF00992698.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989. URL https://www.cs.rhul.ac.uk/home/chrisw/neuron/Watkins1989Phd.pdf. https://www.cs.rhul.ac.uk/~chrisw/thesis.html.

M. White and A. White. A greedy approach to adapting the trace parameter for temporal difference learning. In *International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '16, pages 557–565. International Foundation for Autonomous Agents and Multiagent Systems, 2016. URL http://dl.acm.org/citation.cfm?id=2936924.2937006.

K. Xie, H. Bharadhwaj, D. Hafner, A. Garg, and F. Shkurti. Latent skill planning for exploration and transfer. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=jXe91kq3jAq.

S. M. Xie and S. Ermon. Reparameterizable subset sampling via continuous relaxations, 2021. URL https://arxiv.org/abs/1901.10517.

Y. Xing, Y. Li, I. Laptev, and S. Lu. Mitigating object hallucination via concentric causal attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 92012–92035, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/a76ed4a8ef522c823d73925e7fff16d4-Paper-Conference.pdf.

K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning - Volume 37*, ICML'15, page 2048–2057, 2015. URL https://arxiv.org/abs/1502.03044.

Z. Xu, S. Jain, and M. Kankanhalli. Hallucination is inevitable: An innate limitation of large language models, 2025. URL https://arxiv.org/abs/2401.11817.

R. Yang, M. Fang, L. Han, Y. Du, F. Luo, and X. Li. Mher: Model-based hindsight experience replay, 2021. URL https://arxiv.org/abs/2107.00306.

X. Yu, S. Zhang, X. Song, X. Qin, and S. Jiang. Trajectory diffusion for objectgoal navigation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 110388–110411, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/c72861451d6fa9dfa64831102b9bb71a-Paper-Conference.pdf.

T. Yun, S. Yun, J. Lee, and J. Park. Guided trajectory generation with diffusion models for offline model-based optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 83847–83876, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/98904db124b2a2463e8c59ec33fc7150-Paper-Conference.pdf.

A. Zadaianchuk, M. Seitzer, and G. Martius. Self-supervised visual reinforcement learning with object-centric representations. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=xppLmXCbOw1.

M. Zadem, S. Mover, and S. M. Nguyen. Reconciling spatial and temporal abstractions for goal representation. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=odY3PkI5VB.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf.

A. Zhang, C. Lyle, S. Sodhani, A. Filos, M. Kwiatkowska, J. Pineau, Y. Gal, and D. Precup. Invariant causal prediction for block MDPs. In *International Conference on Machine Learning (ICML)*, volume 119, pages 11214–11224. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/zhang20t.html.

D. Zhang, B. Lv, H. Zhang, F. Yang, J. Zhao, H. Yu, C. Huang, H. Zhou, C. Ye, and c. jiang. Focus on what matters: Separated models for visual-based rl generalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 116960–116986, 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d42523d621194ba54dda098669645f91-Paper-Conference.pdf.

L. Zhang, G. Yang, and B. C. Stadie. World model as a graph: Learning latent landmarks for planning. In *International Conference on Machine Learning (ICML)*, volume 139, pages 12611–12620. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/zhang21x.html.

X. Zhang, H. Huang, D. Zhang, S. Zhuang, S. Han, P. Lai, and H. Liu. Generalization vs. hallucination, 2024b. URL https://arxiv.org/abs/2411.02893.

M. Zhao, S. Luan, I. Porada, X.-W. Chang, and D. Precup. Meta-learning state-based eligibility traces for more sample-efficient policy evaluation. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 1647–1655. International Foundation for Autonomous Agents and Multiagent Systems, 2020. URL https://arxiv.org/abs/2003.08254.

M. Zhao, Z. Liu, S. Luan, S. Zhang, D. Precup, and Y. Bengio. A consciousness-inspired planning agent for model-based reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 1569–1581, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/0c215f194276000be6a6df6528067151-Paper.pdf.

M. Zhao, S. Alver, H. van Seijen, R. Laroche, D. Precup, and Y. Bengio. Consciousness-inspired spatio-temporal abstractions for better generalization in reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=eo9dHwtTFt.

M. Zhao, T. Sylvain, R. Laroche, D. Precup, and Y. Bengio. Rejecting hallucinated state targets during planning. In *International Conference on Machine Learning (ICML)*, 2025. URL https://openreview.net/forum?id=40gBawg6LX.

G. Zhou, H. Pan, Y. LeCun, and L. Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning, 2025. URL https://arxiv.org/abs/2411.04983.

Łukasz Kaiser, M. Babaeizadeh, P. Miłos, B. Osiński, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski. Model based reinforcement learning for atari. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=S1xCPJHtDB.

Chapter 8

APPENDICES

In this chapter, the assistive details of the thesis will be provided.

8.1 Technical Auxiliaries for Chap. 4

The source code of our experiments of this chapter can be found at https://github.com/mila-iqia/conscious-planning.

8.1.1 Additional Experimental Insights

Integer Observations

For MiniGrid environments (Chevalier-Boisvert et al., 2018), the observations consist of integers encoding the object and the status of the grids. We found that for the UP models with these integer observations, the transformer layers are not sufficiently capable to capture the dynamics. Such problem can be resolved after increasing the depth of the FC layer depth by another hidden layer. This is one of the reasons why we prioritized on using CP models for the observation-level learning of Dyna, *i.e.*, CP models can handle integer features without deepening.
Similarly, we have tested the effect of increasing the depth of the linear transformations in SA layers. We did not observe significance in the enhancement of the performance, in terms of model learning or RL performance.

Addressing Memorization with Noisy Shift

We discovered a generic trick to enforce better generalization based on our state-set encoding: if we use fixed integer-based positional tails which correspond to the absolute coordinates of the objects, we can add a global noise to all the x and y components in a set whenever one is encoded. By doing so, the coordinate systems would be randomly shifted every time the agent updates itself. Such shifts would prevent the agent from memorizing based on absolute positions. This trick could potentially enhance the agents' understanding of the dynamics even if in a classical static RL setting, under which the environments are fixed.

8.1.2 Experiment Configurations

The source code for this project is implemented with TensorFlow 2.x and open-source at https://github.com/mila-iqia/Conscious-Planning.

Multi-Processing: we implement a multiprocess configuration similar to that of Ape-X Horgan et al. (2018), where 8 explorers collect and sends batches of 64 training transitions to the central buffer, with which the trainer trains. A pause signal is introduced when the trainer cannot consume fast enough *s.t.*, the uni-process and the multiprocess implementation have approximately the same performance, excluding the wall time.

Feature Extractor: We used the Bag-Of-Word (BOW) encoder suggested in Chevalier-Boisvert et al. (2019). Since the experiments employ a fully-observable setting, we did not use frame stack. In MiniGrid-BabyAI environments, a grid is represented by three integers, and three trainable embeddings are created for the BOW representation. For each object (grid), each integer feature would be first independently transformed into embeddings, which are then mean-pooled to produce the final feature. The three embeddings are learnable and linear (with biases).

Stop criterion: Each run stops after 2.5×10^6 agent-environment interactions.

Replay Buffer: We used Prioritized Experience Replay (PER) of size 10^6 (Schaul et al., 2016), the same as in Hessel et al. (2018). We do not use the weights on the model updates, only the TD updates.

Optimization: We have used Adam Kingma and Ba (2015) with learning rate 2.5×10^{-4} and epsilon 1.5×10^{-4}. The learning rate is the same as in Mnih et al. (2015). Our tests show that using 6.25×10^{-5}, as suggested in Hessel et al. (2018), would be too slow. The batch size is the same for both value estimator training and model training, 64. The training frequency is the same as in Hessel et al. (2018): every 4 agent-environment interactions.

γ: Same as in Hessel et al. (2018). 0.99.

In the experiments, we wanted functional architectures with minimal sizes for all the components. Thus, globally for the set-input architectures, we have limited the depth of the transformer layers to be $N = 1$ wherever possible. The FC components are MLPs with 1-hidden layer of width 64. Exceptionally, we find that the effectiveness of the value estimator needs to be guaranteed with at least 3-transformer layers. For the distributional output, while the value estimator has an output of 4 atoms, the reward estimator has only 2.

Transformers: For the SA sub-layers, we have used 8 heads globally. For the FC sub-layers, we have used 2-layer MLP with 64 hidden units globally. All the transformer related components have only one transformer layer except for that of the value estimator, which has three transformer layers before the pooling. We found that the shallower value estimators exhibit unstable training behaviors when used in the non-static settings.

Set Representation: The length of an object in the state set has length 32, where the feature is of length 24 and the positional embedding has length 8. Note that the length of objects must be divisible by the number of heads in the attentions. The positional embeddings are trainable, however their initial values are constructed by the absolute xy coordinates from each corner of the gridworld ($4 \times 2 = 8$). We found that without such initialization, the positional embedding would collapse.

Action Embedding: Actions are embedded as one-hot vectors with length 8.

Planning Steps: for each planning session, the maximum number of simulations based on the learned transition model is 5.

Exploration: ϵ takes value from a linear schedule that decreases from 0.95 to 0.01 in the course of 10^6 agent-environment interactions, same as in Hessel et al. (2018). For evaluation, ϵ is fixed to be 10^{-3}.
Distributional Outputs: We have used distributional outputs Bellemare et al. (2017) for the reward and value estimators. 2 atoms for reward estimation (mapping the interval of $[0, 1]$) and 4 atoms for value estimation (mapping the interval of $[0, 1]$).
Regularization: We find that layer norm is crucial to guarantee the reproducibility of the performance with set-representations. We apply layer normalization Ba et al. (2016) in the sub-layers of transformers as well as at the end of the encoder and model dynamics outputs. This applies for the NOSET baseline as well.
modelfree baseline: We did not use the full Rainbow agent Hessel et al. (2018) as the baseline, because we want to keep our agent as minimalist as possible. The agent does not need the dueling head and the noisy net components to perform well, according to our preliminary ablation tests.

8.2 Technical Auxiliaries for Chap. 5

The source code of our experiments of this chapter can be found at https://github.com/mila-iqia/skipper.

8.2.1 Skipper

Training

The agent is based on a distributional prioritized double DQN. All the trainable parameters are optimized with Adam at a rate of 2.5×10^{-4} (Kingma and Ba, 2015), with a gradient clipping by value (maximum absolute value 1.0). The priorities for experience replay sampling are equal to the per-sample training loss.

Full State Encoder

The full-state encoder is a 2-layer residual block (with kernel size 3 and doubled intermediate channels) combined with the 16-dimensional bag-of-words embedder of BabyAI (Chevalier-Boisvert et al., 2019).

Partial State Selector (Spatial Abstraction)

The selector σ is implemented with single-head (not multihead, thus the output linear transformation of the default multihead attention implementation in PyTorch is disabled.) top-4 attention, with each local perceptive field of size 8×8 cells. Layer normalization (Ba et al., 2016) is used before and after the spatial abstraction.

Estimators

The estimators, which operate on the partial states, are 3-layered MLPs with 256 hidden units.
An additional estimator for termination is learned, which instead of taking a pair of partial states as input, takes only one, and is learned to classify terminal states with cross-entropy loss. The estimated distance from terminal states to other states would be overwritten with ∞. The internal γ for intrinsic reward of π is 0.95, while the task γ is 0.99

The estimators use C51 distributional TD learning (Dabney et al., 2018). That is, the estimators output histograms (`softmax` over vector outputs) instead of scalars. We regress the histogram towards the targets, where these targets are skewed histograms of scalar values, towards which KL-divergence is used to train. At the output, there are 16 bins for each histogram estimation (value for policy, reward, distance).

Checkpoint Generator

Although Skipper is designed to have the generator work on state level, that is, it should take learned state representations as inputs and have state representations as outputs, in our experiments, the generator actually operates on observation inputs and outputs. This is because of the preferred compactness of the observations and the equivalence to full states under full observability in our experiments.
The context extractor $\mathcal{E}_c$ is a 32-dimensional BabyAI BOW embedder. It encodes an input observation into a representation of the episodic context.
The partial description extractor $\mathcal{E}_z$ is made of a 32-dimensional BabyAI BOW embedder, followed by 3 aforementioned residual blocks with 3×3 convolutions (doubling the feature dimension every time) in between, ended by global maxpool and a final linear projection to the latent weights. The partial descriptions are bundles of 6 binary latents, which could represent at most 64 "kinds" of checkpoints. Inspired by VQ-VAE (van den Oord et al., 2017), we use the argmax of the latent weights as partial descriptions, instead of sampling according to the `softmax`-ed weights. This enables easy comparison of current state to the checkpoints in the partial description space, because each state deterministically corresponds to one partial description. We identify reaching a target checkpoint if the partial description of the current state matches that of the target.
The fusing function first projects linearly the partial descriptions to a 128-dimensional space and then uses deconvolution to recover an output which shares the same size as the encoded context. Finally, a residual block is used, followed by a final $1x1$ convolution that downscales the concatenation of context together with the deconv'ed partial description into a 2D weight map. The agent's location is taken to be the `argmax` of this weight map.
The whole checkpoint generator is trained end-to-end with a standard VAE loss. That is the sum of a KL-divergence for the agent's location, and the entropy of partial descriptions, weighted by 2.5×10^{-4}, as suggested in https://github.com/AntixK/PyTorch-VAE. Note that the per-sample losses in the batches are not weighted for training according to priority from the experience replay.
We want to mention that if one does not want to generate non-goal terminal states as checkpoints, we could also seek to train on reversed $\langle S^{\odot}, S_t \rangle$ pairs. In this case, the checkpoints to reconstruct will never be terminal.

HER

Each experienced transition is further duplicated into 4 hindsight transitions at the end of each episode. Each of these transitions is combined with a randomly sampled observation from the same trajectory as the relabelled "goal". The size of the hindsight buffer is extended to 4 times that of the baseline that does not learn from hindsight accordingly, that is, 4×10^6.

Planning

As introduced, we use value iteration over options (Sutton et al., 1999) to plan over the proxy problem represented as an SMDP. We use the matrix form $Q = R_{S\times S} + \Gamma V$, where R and Γ are the estimated edge matrices for cumulative rewards, respectively. Note that this notation is different from the ones we used in the manuscript. The checkpoint value V, initialized as all-zero, is taken on the maximum of Q along the checkpoint target (the actions for μ) dimension. When planning is initiated during decision time, the value iteration step is called 5 times. We do not run until convergence, since with low-quality estimates during the early stages of the learning, this would be a waste of time. The edges from the current state towards other states are always set to be one-directional, and the self-loops are also removed. This means the first column as well as the diagonal elements of R and Γ are all zeros. Besides pruning edges based on the distance threshold, as introduced in the main paper, the terminal estimator is also used to prune the matrices R and Γ: the rows corresponding to the terminal states are all zeros.
The only difference between the two variants, *i.e.*, Skipper-once and Skipper-regen is that the latter variant would discard the previously constructed proxy problem and construct a new one every time the planning is triggered. This introduces more computational effort while lowering the chance that the agent gets "trapped" in a bad proxy problem that cannot form effective plans to achieve the goal. If such a situation occurs with Skipper-regen, as long as the agent does not terminate the episode prematurely, a new proxy problem will be generated to hopefully address the issue. Empirically, as we have demonstrated in the experiments, such variant in the planning behavior results in generally significant improvements in terms of generalization abilities at the cost of extra computation.

Hyperparameter Tuning

Some hyperparameters introduced by Skipper can be located in the pseudocode in Alg. 3.

Timeout and Pruning Threshold Intuitively, we tied the timeout to be equal to the distance pruning threshold. The timeout kicks in when the agent thinks a checkpoint can be achieved within *e.g.*, 8 steps, but already spent 8 steps yet still could not achieve it.
This leads to how we tuned the pruning (distance) threshold: we fully used the advantage of our experiments on DP-solvable tasks: with a snapshot of the agent during its training, we can sample many $\langle$ starting state, target state $\rangle$ pairs and calculate the ground truth distance between the pair, as well as the failure rate of reaching from the starting state to the target state given the current policy π, then plot them as the x and y values respectively for visualization. We found such curves to evolve from high failure rate at the beginning, to a monotonically increasing curve, where at small true distances, the failure rates are near zero. We picked 8 because the curve starts to grow explosively when the true distances are more than 9.

k for k-medoids We tuned this by running a sensitivity analysis on Skipper agents with different k's, whose results are presented previously in this Appendix.
Additionally, we prune from 32 checkpoints because 32 checkpoints could achieve (visually) a good coverage of the state space as well as its friendliness to NVIDIA accelerators.

Size of local Perception Field We used a local perception field of size 8 because our baseline model-free agent would be able to solve and generalize well within 8×8 tasks, but not larger. Roughly speaking, our spatial abstraction breaks down the overall tasks into 8×8 sub-tasks, which the policy could comfortably solve.

Model-free Baseline Architecture The baseline architecture (distributional, Double DQN) was heavily influenced by the architecture used in the previous work (Zhao et al., 2021), which demonstrated success on similar but smaller-scale experiments (8×8). The difference is that while then we used computationally heavy components such as transformer layers on a set-based representation, we replaced them with a simpler and effective local perception component. We validated our model-free baseline performance on the tasks proposed in Zhao et al. (2021).

8.2.2 LEAP

Adaptation for Discrete Action Spaces

The LEAP baseline has been implemented from scratch for our experiments, since the original open-sourced implementation[1] was not compatible with environments with discrete action spaces. LEAP's training involves two pretraining stages, that are, generator pretraining and distance estimator pretraining, which were originally named the VAE and RL pretrainings. Despite our best effort, that is to be covered in detail, we found that LEAP was unable to get a reasonable performance in its original form after rebasing it on a discrete model-free RL baseline.

Replacing the Model

We tried to identify the reasons why the generalization performance of the adapted LEAP was unsatisfactory: we found that the original VAE used in LEAP is not capable to handle even few training tasks, let alone generalize well to the evaluation tasks. Even by combining the idea of the context / partial description split (still with continuous latents), during decision time, the planning results given by the evolutionary algorithm (Cross Entropy Method, CEM, Rubinstein (1997)) almost always produce delusional plans that are catastrophic in terms of performance. This was why we switched into LEAP the same conditional generator we proposed in the paper, and adapted CEM accordingly, due to the change from continuous latents to discrete.
We also did not find that using the pretrained VAE representation as the state representation during the second stage helped the agent's performance, as the paper claimed. In fact, the adapted LEAP variant could only achieve decent performance after learning a state representation from scratch in the RL pretraining phase. Adopting Skipper's splitting generator also disables such choice.

Replacing TDM

The original distance estimator based on Temporal Difference Models (TDM) also does not show capable performance in estimating the length of trajectories, even with the help of a ground truth distance function

[1] https://github.com/snasiriany/leap

(calculated with DP). Therefore, we switched to learning the distance estimates with our proposed method. Our distance estimator is not sensitive to the sub-goal time budget as TDM and is hence more versatile in environments like that was used in the main paper, where the trajectory length of each checkpoint transition could highly vary. Like for Skipper, an additional terminal estimator has been learned to make LEAP planning compatible with the terminal lava states. Note that this LEAP variant was trained on the same sampling scheme with HER as in Skipper.

The introduced distance estimator, as well as the accompanying full-state encoder, are of the same architecture, hyperparameters, and training method as those used inSkipper. The number of intermediate subgoals for LEAP planning is tuned to be 3, which close to how many intermediate checkpoints Skipper typically needs to reach before finishing the tasks. The CEM is called with 5 iterations for each plan construction, with a population size of 128 and an elite population of size 16. We found no significant improvement in enlarging the search budget other than additional wall time. The new initialization of the new population is by sampling a ϵ-mean of the elite population (the binary partial descriptions), where $\epsilon = 0.01$ to prevent the loss of diversity. Because of the very expensive cost of using CEM at decision time and its low return of investment in terms of generalization performance, during the RL pretraining phase, the agent performs random walks over uniformly random initial states to collect experience.

8.2.3 Director

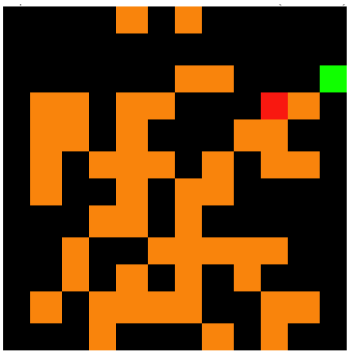


Figure 8.1: ***An Example for Simplified Observations for*** Director

Adaptation

We based our experiments of Director (Hafner et al., 2022) on the publicly available code (https://github.com/danijar/director) released by the authors. Except for a few changes in the parameters, which are depicted in Tab. 8.1, we have used the default configuration provided for Atari environments. Note that as the Director version in which the worker receives no task rewards performed worse in our tasks, we have used the version in which the worker receives scaled task rewards (referred to as "Director (worker task reward)" in Hafner et al. (2022)). This agent has also been shown to perform better across various domains in Hafner et al. (2022).

Encoder Unlike Skipper and LEAP agents, the Director agent receives as input a simplified RGB image of the current state of the environment (see Fig. 8.1). This is because we found that Director performed better with its original architecture, which was designed for image-based observations. We removed all textures to simplify the RGB observations.

Table 8.1: ***Changed Hyperparameters of*** Director

Parameter	Value
replay_size	2M
replay_chunk	12
imag_horizon	8
env_skill_duration	4
train_skill_duration	4
worker_rews	{extr: 0.5, expl: 0.0, goal: 1.0}
sticky	False
gray	False

Failure Modes: Bad Generalization, Sensitive to Short Trajectories

Training Performance We investigated why Director is unable to achieve good training performance (Fig. 5.6). As Director was designed to be trained solely on environments where it is able to collect long trajectories to train a good enough recurrent world model (Hafner et al., 2022), we hypothesized that Director may perform better in domains where it is able to interact with the environment through longer trajectories by having better recurrent world models (*i.e.*, the agent does not immediately die as a result of interacting with specific objects in the environment). To test this, we experimented with variants of the used tasks, where the lava cells are replaced with wall cells, so the agent does not die upon trying to move towards them (we refer to this environment as the "walled" environment). The corresponding results on 50 training tasks are depicted in Fig. 8.2. As can be seen, the Director agent indeed performs better within the training tasks than in the environments with lava.

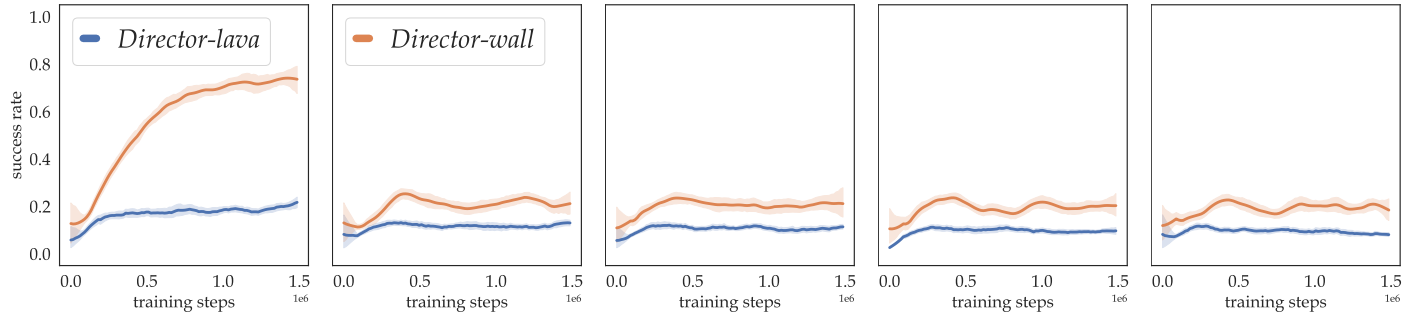
Generalization Performance We also investigated why Director is unable to achieve good generalization (Fig. 5.6). As Director trains its policies solely from the imagined trajectories predicted by its learned world model, we believe that the low generalization performance is due to Director being unable to learn a good enough world model that generalizes to the evaluation tasks. The generalization performances in both the "walled" and regular environments, depicted in Fig. 8.2, indeed support this argument. Similar to what we did in the main paper, we also present experimental results for how the generalization performance changes with the number of training environments. Results in Fig. 8.3 show that the number of training environments has little effect on its poor generalization performance.

8.3 Technical Auxiliaries for Chap. 6

The source code of our experiments of this chapter can be found at https://github.com/mila-iqia/delusions.

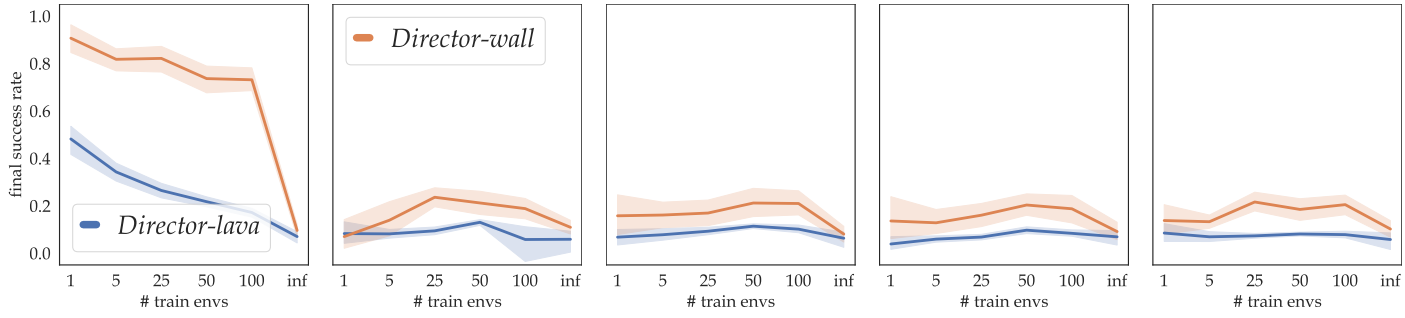
8.3.1 Implementation of PERTASK

PERTASK takes the advantage of the fact that training is done on limited number of fixed task instances. We give each task a unique task identifier. At relabeling time, PERTASK samples observations among all the transitions marked with the same identifier as the current training task instance. This can be trivially implemented



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 8.2: ***Results of Director on Tasks with Lavas v.s. on Tasks with Walls****: the results are obtained with* 50 *training tasks. The results for Director-lava (same as in the main paper) are obtained from* 20 *independent seed runs.*



(a) *training*, $\delta = 0.4$ **(b)** *OOD*, $\delta = 0.25$ **(c)** *OOD*, $\delta = 0.35$ **(d)** *OOD*, $\delta = 0.45$ **(e)** *OOD*, $\delta = 0.55$

Figure 8.3: ***Generalization Performance of Agents on Different Numbers of Training Tasks (while*** Director ***runs on the walled environments)****: besides Director, each data point and corresponding error bar (95% confidence interval) are processed from the final performance from* 20 *independent seed runs. Director-wall's results are obtained from* 20 *runs.*

with individual auxiliary experience replays that store only the experienced states with memory-efficient pointers to the buffered x_t's in the main HER.

8.3.2 Implementation Details for Experiments

Skipper

Our adaptation of Skipper over the original implementation[2] in Zhao et al. (2024) is minimal. We have additionally added two simple vertex pruning procedures before the vertex pruning based on k-medoids. These two procedures include: 1) prune vertices that are duplicated, and 2) prune vertices that cannot be reached from the current state with the estimated connectivity.

We implemented a version of generator that can reliably handle both `RDS` and `SSM` with the same architecture. Please consult `models.py` in the submitted source code for its detailed architecture.

For `SSM` instances, since the state spaces are 4-times bigger than those of `RDS`, we ask that Skipper generate twice the number of candidates (both before and after pruning) for the proxy problems.

[2] https://github.com/mila-iqia/Skipper

All other architectures and hyperparameters are identical to the original implementation.
For better adaptability during evaluation and faster training, Skipper variants in Chap. 5 keeps the constructed proxy problem for the whole episode during training and replanning only triggers a re-selection, while during evaluation, the proxy problems are always erased and re-constructed.
The quality of our adaptation of the original implementation can be assured by the fact the E variant's performance matches the original on RDS.

LEAP

LEAP's training involves two pretraining stages, that are, generator pretraining and distance estimator training.
We improved upon the adopted discrete-action space compatible implementation of LEAP (Nasiriany et al., 2019) from Zhao et al. (2024). We gave LEAP additional flexibility to use fewer subgoals along the way to the task goal if necessary. Also, we improved upon the Cross-Entropy Method (CEM), such that elite sequences would be kept intact in the next population during the optimization process. We increased the base population size of each generation to 512 and lengthened the number of iterations to 10.
For RDS 12×12 and SSM 8×8, at most 3 subgoals are used in each planned path. We find that employing more subgoals greatly increases the burden of CEM and lower the quality of the evolved subgoal sequences, leading to bad performance that cannot be effectively analyzed.
We used the same generator architecture and hyperparameters as in Skipper. All other architectures and hyperparameters remain unchanged.
Similarly for LEAP, for better adaptability during evaluation, the planned sequences of subgoals are always reconstructed whenever planning is triggered. While in training, the sequence is reused and only a subgoal selection is conducted.
The quality of our adaptation of the original implementation can be assured by the fact the E variant's performance matches the original on RDS.

8.3.3 Applying the Evaluator on Dreamerv2

To demonstrate that our approach functions effectively in more generalist settings, such as those with continuous state and action spaces and partial observability, and to illustrate its application to a modern TAP agent, we integrated our proposed evaluator into Dreamerv2 (Hafner et al., 2021). The evaluator filters out potentially delusional values from infeasible states that might distort the λ-returns derived from imagined trajectories. Given the technical complexity ahead, we suggest readers familiarize themselves with Dreamerv2 before continuing (Hafner et al., 2021).
Although Dreamerv2's stochastic states are discrete and could theoretically support similarity assessments, their design ensures they rarely repeat due to numerous possibilities, making them too random for our similarity function h. Consequently, we rely on the deterministic state representations $\boldsymbol{s}$, which also prompt us to more thought-provoking discussions.
Dreamerv2 operates as a Dyna-like method, employing fixed-horizon rollouts with autoregressively imagined states as targets. Lacking a built-in similarity function h, it provides an opportunity to showcase how we construct h in our approach. Our method incorporates various realism aspects to assess state similarity

between the next state and the target state, influencing the branching in Eq. 6.2 during evaluator updates (Russell et al., 2025).

How to craft h: Observational Realism

Observational realism, *i.e.*, the similarity in terms of state representations is the first obvious criteria for h. Theoretically, one might simplistically assume state equivalence by defining an ϵ-ball around the target state. However, in practice, an ϵ-ball based on L_2 distances proves inadequate due to varying representation scales. Instead, we employ Mahalanobis distances, which better accommodate the representations' distributional variations.
To be more precise, we use an Exponential Moving Average (EMA) of the covariance of concatenated current-next deterministic state pairs $[\boldsymbol{s}_t, \boldsymbol{s}_{t+1}]$ to calculate the Mahalanobis distances between the next state pair $[\boldsymbol{s}_t, \boldsymbol{s}_{t+1}]$ and target state pair $[\boldsymbol{s}_t, \hat{\boldsymbol{s}}_{t+1}]$.

How to craft h: Behavioral Realism

The second focus is behavioral realism: does the agent exhibit similar behavior (*e.g.*, in value, reward, and discount estimations) across the states ($\boldsymbol{s}_{t+1}$ & $\hat{\boldsymbol{s}}_{t+1}$)?
Here, we apply Mahalanobis distances to pairs of current and future values, rewards, and discounts, ensuring the states appear similar from the agent's perspective.
Caution is required with action-realism. Naively applying our method to one-hot encoded discrete actions could result in a singular covariance matrix for the ϵ-ball computation.
Preliminary Atari experiments suggest setting distinct ϵ values for different components—state representations, value estimations, reward estimations, and discount estimations.

How to Relabel: Just-In-Time (JIT) Construction

Since Dreamerv2 samples sub-trajectories and computes state representations autoregressively, we forgo a separate HER for storing source-target pairs, opting instead for Just-In-Time (JIT) construction. Designed for single-environment training and evaluation, Dreamerv2 allows us to implement a (E+G) variant on agent-sampled sub-trajectories. Initial tests indicate a balanced mix of EPISODE (within sub-trajectories) and GENERATE performs effectively.

How to Reject: Three Criteria for λ-returns

Dreamerv2 leverages its model to imagine future states and values, using these, along with intermediate rewards and discounts, to compute λ-returns for each origin state.
For such strategy, we implemented the following 3 criteria for rejecting the imagined states:

1. **Transition-wise Rejection**: If a next state seems unlikely to follow from the current state, its value is deemed untrustworthy. This process is repeated for all imagined transitions. Notably, a state rejected as infeasible in one transition might still be reachable elsewhere, so subsequent states are not automatically discarded.

2. **Point-to-Point (P2P) Rejection for Targets**: Starting from a replay-sampled base state, we assess whether each imagined state is reachable, regardless of steps taken. This counters hallucinated targets from accumulated errors over the imagination horizon (Talvitie, 2017), excluding such states from value estimation targets.

3. **P2P Rejection for Current States**: Entirely unreachable states are excluded as current states in multi-step value updates, though subsequent states may remain viable.

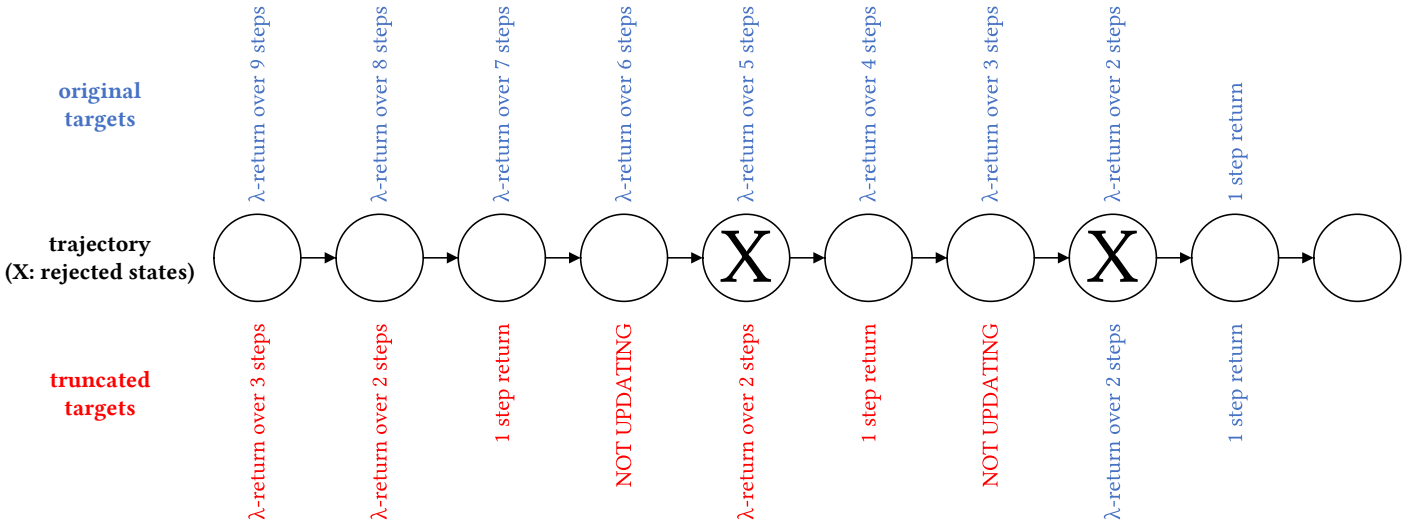


Figure 8.4: ***Truncated λ-Returns with Rejected States for*** Dreamer*: the original λ-returns are illustrated in the top row, while the truncated returns are illustrated in the bottom row with the differences marked in red. Our strategy ensures that the critic targets in the trajectories can be maximally preserved for updates. The states right before the rejected ones will have no trustworthy critic targets and are thus not updated. Starting from the last rejected state, all critic targets remain the same as the originals.*

The first two criteria yield a binary mask to truncate λ-returns in sampled sub-trajectories, excluding untrustworthy values while preserving horizons for reliable ones. Our repository offers an efficient implementation, maintaining the complexity as the original, un-truncated λ-returns. The behavior of the (critic) target-based rejection is presented in Fig. 8.4.

The third criterion masks updates to wholly infeasible imagined states. By examining the rejection rate by the horizon index, the evaluator can also be used to understand how long the imagined trajectories are likely to be trustworthy and thus adjust the associated hyperparameters.

We developed a standalone, user-friendly evaluator (implemented in PyTorch) that integrates seamlessly into TAP agents like Dreamerv2, employing its own optimizer and target networks for robust learning when activated. Please check `evaluator.py` in the source code repository.

We tuned the hyperparameters using the `Atari` environments and found that both the autoregressive estimations of distances and the P2P distances (towards the target states) in the sampled and imagined trajectories roughly converge to the estimated ground truth values, which are deduced from their time indices. This is the best we can do for environments without ground truth access.

Regrettably, our Atari100k preliminary results with 10^5 interactions show negligible performance gains over the baseline (Łukasz Kaiser et al., 2020). This is likely because the state representations of Dreamer usually

takes a significant portion of training to stabilize and for the evaluator to adapt to. The differences are expected to show with prolonged experiments where a significant number of updates will be made after the state representations stabilize. Limited computational resources prevented our extended experiments, and we invite those with greater capacity to investigate further.

Our Dreamer implementations can be found in the source code repository.