Learning and Planning with Structured Abstraction for Embodied Decision-Making

by

Linfeng Zhao

Submitted to the Khoury College of Computer Sciences
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at

NORTHEASTERN UNIVERSITY

September 2025

Author — Linfeng Zhao
Khoury College of Computer Sciences
September 2025

Thesis Advisor — Lawson L.S. Wong
Associate Professor
Northeastern University

Committee Member — Robert Platt
Associate Professor
Northeastern University

Committee Member — Robin Walters
Assistant Professor
Northeastern University

Committee Member — Hao Su
Associate Professor
UC San Diego

Committee Member — Leslie Pack Kaelbling
Professor
MIT

Acknowledgements

I first want to thank my advisor Lawson Wong, who has been the most important person in my academic life so far: during the early and pandemic years he was often my only real support, and from him I learned so much—from how to decompose chaos into solvable parts to very practical rules about how to actually get things done. He not only teaches me these principles, but also shows me how he himself approaches problems and solves them, explaining his detailed "chain-of-thought" reasoning and letting me see how he thinks.

I am deeply grateful to my committee—Hao Su, Robin Walters, Rob Platt, and Leslie Pack Kaelbling. My journey into research started with Hao, who was my advisor at UCSD and later joined my thesis committee; in my defense I used him as the "Mr. Ping" metaphor, because he gave me my first real experience of research and quietly set me onto the path that eventually led to this thesis. Robin and Rob have been like two other masters in my own Kung Fu Panda world: Robin as a kind of "secret advisor" who kept the math and definitions precise, especially around symmetry and equivariance, and Rob as someone whose work and students in manipulation gave me a much clearer sense of the depth and importance of that field. Leslie, together with Tomas and the LIS group, built what I think of as a "Village of Learning and Planning," and I am very grateful both for her support of my collaboration with the "villagers" at LIS and for her very crisp view on problem formulation and the literature—always pushing me to think hard about what question we are really trying to answer in a piece of research before we run any experiments.

At Northeastern, I was lucky to work closely with many people. At Northeastern GRAIL, I thank Shuo Jiang, Seth Peth, Yu Qi, Chengguang Xu and many others for the lab environment and daily support, and in the broader Northeastern robotics / RL community and joint reading group (with Rob's and Chris's groups) I thank Dian Wang, David Klee, Mingxi Jia, Boce Hu, Haibo Zhao, Haotian Liu, Willy Lin, Tarik Kelestemur, Nichols Crawford Taylor, Misha Lvovsky, Yaoyao "Freax" Qian, Yijian Huang, and many others for paper discussions and robot-related chaos. Among my close collaborators, I especially thank Haojie Huang for many long-running conversations on systems, decision-making, and robots; Owen Howell for being a key thinking partner on math, theory, and life; Ondrej Biza for for many discussions and method- and experiment-level collaborations; Hongyu Li for close collaboration and shared ski-and-work time during his master's; Xupeng Zhu for joint work on manipulation algorithms and experiments; Xueguang Luke Lyu for lots of day-to-day debates,

brainstorming, and life events; and John Park for many project discussions and shared topics over the years.

I also fortunately have other collaborators and friends in Boston. Across the river at MIT, I also closely collaborated with local friends and colleagues while visiting to work on the Spot robot and related projects. In particular, I thank Nishanth Kumar for discussions and feedback that helped refine ideas and experiments; Willie McClinton for many conversations about research and life and for iterating on ideas and experiments together; and Will Shen for being both a colleague and ski partner, with many ski runs and lift rides turning into moving research meetings.

Outside Northeastern and MIT, internships and external labs also played an important role. At the Boston Dynamics AI Institute (BDAI), I am especially grateful to Jenny Barry, my internship mentor, for her support and for making that internship a very positive experience working with real systems and teams. I also spent a lot of time interacting with many others at BDAI, including Ondrej Biza, Tao Pang, Sebastian Castro, Shenli Yuan, and many more, and appreciated having a place where it was natural to think out loud about robots. I also had internships at Amazon and Meta, and I thank Kari and Dhruv and my many other colleagues there for their support and discussions during those periods, even if I do not name everyone individually here.

I have been fortunate to collaborate and have useful discussions with many other researchers. I especially thank David Abel, Xuezhou Zhang, Rose Yu, Weiyu Liu, Jiayuan Mao, Tom Silver, and others for their joint work and conversations, and I thank the students and mentees I worked with—Mingxi Jia (Brown), Hongyu Li, Zhewen Zhang, Yue Yang, Lingzhi Kong, Neel Sortur, Fan Xie, and others—for their energy and contributions to ideas and experiments.

Many friends made life in Boston and the broader journey much better. I thank Peng Cao, Lujie Yang, Yongchao Chen, Julie Zhu, Yue Meng, Oliver Wang, Shen Li, and other local friends at Northeastern, MIT, and nearby universities for shared meals and late-night chats that made the city feel livable rather than just a place to work. I also thank friends connected through skiing, tennis, and board games for turning pressure into something digestible and sometimes even fun, including Tongzhou Mu, Zhan Ling, Brown friends like Mingxi and Hongyu, and others from earlier and later stages.

I am grateful for candid conversations and timely advice from many others, whose perspectives helped at key junctures when I needed to choose directions or interpret results. I also thank the staff and administrators at Northeastern and so on for the often invisible work that makes research possible.

Finally, to my family—my constant power supply—thank you for letting curiosity be a full-time job, for supporting moves and uncertainty, and for backing this path even when it was not always clear where it was leading. And to colleagues and friends whose names do not appear here but should: thank you.

Abstract

A general-purpose embodied agent must perceive its environment, reason about it, and act to achieve goals. While deep learning has driven major advances in language modeling, perception, and control, current learning-only systems often excel mainly on their training distribution, which is inadequate for complex decision-making in unseen settings. Humans, by contrast, leverage compositional abstractions—combining familiar concepts in new ways. We borrow that intuition but aim it at decision-making related abstractions.

This thesis studies how to build and use structured abstractions for embodied decision-making to enable systematic generalization and long-horizon behaviors. The key idea is to decompose decision-making into reusable building blocks that compose into complex behavior. We use high-level, composable actions (options) as the units of action abstraction. The state abstraction is designed or learned to support planning with these options, with units such as objects, relations, spatial symmetries, and learned embeddings. At decision time, the agent allocates additional compute for *planning* and *reasoning* in this abstract space (rather than over raw observations and low-level actions), so it can tackle more challenging tasks.

Taken together, we present a framework that couples learning with abstraction and high-level planning, enabling generalization to unseen environments and tasks and providing core ingredients for general-purpose embodied agents.

Contents

Chapter 1

Introduction

A long-term goal of artificial intelligence and robotics is to build general-purpose embodied agents that sense, decide, and act in the physical world [Brooks, 1991, Kaelbling et al., 1998a], which need to handle multimodal input (camera vision input, language instructions, proprioception, and optionally touch and other sensors) and reliably complete long-horizon tasks in potentially unseen environments, such as new goals, objects, skills, instructions, scenes.

Consider **mobile manipulation** as a canonical example: a household robot that must navigate to different rooms, identify and manipulate various objects (dishes, tools, furniture), and accomplish complex goals like "prepare dinner" or "clean the living room." Such tasks require coordinating navigation through cluttered spaces with precise object manipulation, handling uncertainty about object properties and locations, and composing basic skills (grasp, place, navigate) into long action sequences that achieve high-level goals specified in natural language.

We want a scalable computational pipeline where more compute and data reliably help: the system learns the right high-level abstraction and reasons at decision time to stitch them together, instead of memorizing fixed scripts.

Two Primary Paradigms. There are two primary paradigms towards a general purpose embodied agent: learning-based and planning-based. We briefly position ourselves against pure learning and planning, discussing the assumptions and why they are not sufficient.

A pure **learning** approach aims end-to-end learning a mapping from observations (and instructions) directly to actions. It may learn from experience (reinforcement learning) or demonstrations (imitation learning [Pomerleau, 1991, Ross et al., 2011]). It implicitly assumes that more data or parameters will cover the needed diversity of input and output space. Why is it so effective on some problems? It learns effective intermediate representations without hand engineering and can model really complex mapping from data. Recent approaches follow language modeling style [Brown et al., 2020, Vaswani et al., 2017, Achiam et al., 2023], which tokenize the raw pixels, actions, instructions, and goals and represent with distributed representations.

What are the problems? Such raw representations do not really have reusable/-

compositional structure of like the unit of words in LLMs. Thus, it is hard to (1) understand the enormous combinations of the units and (2) reason about the various combinations of long action sequences (aka plans) [Sutton et al., 1999]. The agent has loose perception-action coupling, making it hard to plan to gather new information and reason when to replan [Kaelbling et al., 1998a].

A pure **planning** paradigm assumes a full model exists, such as a symbolic model with state abstraction (object symbols and predicates) and action abstraction (operators with preconditions and effects [Fikes and Nilsson, 1971, Ghallab et al., 2004]). It has explicit abstraction and thus excellent compositionality, which enables strong long-horizon reasoning capability.

Why they are not enough? It assumes handcrafted units, which doesn't scale well for new problems. The symbols and operators are not well transferrable across open settings and embodiments [Garrett et al., 2021]. The grounding of predicates (abstract state) and operators (abstract action) is limited by perception and interaction [Konidaris, 2019]. The planning quality and efficiency also depend on heuristics, and it also lacks learned components to improve estimates or adapt to new settings. It is also hard to learn the abstraction, because the planning algorithms are not designed to work well with abstractions that are approximate [Abel et al., 2020a, Ferns et al., 2004].

The Emerging Third Paradigm. Since this work began, a third paradigm has emerged: large pre-trained models (also sometimes referred as foundation models) that leverage massive scale and self-supervised learning to achieve remarkable capabilities in perception, language understanding, and even language-level reasoning [Brown et al., 2020, Achiam et al., 2023]. These models seem to challenge the need for explicit structure and planning. However, they exhibit critical weaknesses: lack of systematic generalization to novel compositions, inability to perform reliable long-horizon planning, and absence of geometric understanding. This thesis provides exactly what foundation models lack—structured reasoning algorithms that can sit on top of powerful perceptual models. Imagine a household robot powered by GPT-5 for language understanding and vision, but using the planning algorithms from this thesis to ensure dinner actually gets prepared correctly. The foundation model understands what "prepare dinner" means; our algorithms ensure it happens reliably through systematic planning.

Scope and Assumptions. We aim to build agents that benefit from both learning and planning, but we first state the scope and simplifying assumptions.

(1) We typically use **abstract actions**. They can be modeled by *options* [Sutton et al., 1999], which has initial and termination conditions and an option policy. Examples include (i) movement directions or map locations, (ii) object-centric actions, or (iii) PDDL operators with preconditions and effects that are object parameterized. In this case, we assume the high-level plans are **downward refinable**. This means that a high-level plan that satisfies constraints (e.g., options) always has a low-level exe-

cutable plan, so we do not deal with integrated search between high-level and low-level, as studied in integrated task and motion planning (TAMP) [Garrett et al., 2021].

(2) We primarily assume the environment is **fully observable**, besides a work on studying on planning for information gathering. In that case, we assume observations are accurate and no information loss. Under these assumptions, the **state abstraction** can be produced via the current observation or the history, and the high-level planning happens over the abstract state.

Even with these simplifications, the challenges still exist and are nontrivial: choosing the proper state abstraction at design time, estimating and updating the state accurately at decision time, composing plans for long-horizon tasks, handling unseen tasks with unknown information, etc.

Key Principles To deal with the problem, we explain the key principles of the thesis.

Abstraction *= I/O interface.* We treat abstraction as studying of the representation units or the basic building blocks of the system. They are like words in language modeling, where the units can be composed to represent complicated meaning. This includes state abstraction (e.g., objects and scenes) and action abstraction (e.g., abstract actions and plans).

Compositionality *= rules.* Once the units are defined, we can compose them together via certain rules. Objects can be composed together to representation a full scene. A sequence of abstract actions forms a plan. The compositionality of geometric transformations forms symmetry groups.

Planning *(and reasoning) at decision time.* Intuitively, harder (decision-making) problems require more compute (test-time/inference-time scaling). This is realized via planning, i.e., search of good action sequences in a proper state space. However, the major difficulty of long-term planning for embodied decision-making is to build an abstract state and action space to search with a compatible planning algorithm.

Learning *for abstraction and planning.* Analogous to language modeling, the interface (abstraction) and rules (compositionality) enable easier **learning** for embodied decision-making. Language modeling succeeds by (1) learning distributed representations of words in vectors and (2) learning how these vectors interact. Learning of abstraction is analogous to learning distributed representation of words, and learning for high-level planning is analogous to learning how abstract actions and states interact.

With these components, unseen tasks can be modeled by new compositions of familiar parts, enabling (compositional) **generalization**. In summary, we consider abstraction with compositionality as the backbone, learning and planning as engines, and compute as fuel to scale.

This thesis provides systematic planning algorithms and compositional reasoning frameworks—capabilities that complement modern perception models. We work under specific assumptions (downward refinability, structured abstractions) but provide formal guarantees for compositional generalization and geometric reasoning that pure learning approaches cannot achieve.

1.1 Overview of Thesis

This thesis develops learning approaches for compositional abstractions for efficient planning and studies how to integrate these abstractions and planning together into robot decision-making. We demonstrate how learning structured representations—objects, symmetries, and spatial maps—enables efficient planning algorithms that generalize to novel scenarios through compositional reasoning. The work is organized around two main themes: **learning with compositional abstractions** (Part I) and **planning and agent integration** (Part II).

Part I: Learning with Compositional Abstractions. Part I focuses on **learning the right representations and structural priors for planning**. We develop methods to learn object-centric world models, discover and exploit symmetries, and build compositional representations that enable data-efficient generalization across novel scenarios. The emphasis is on how learning can discover and leverage compositional structure to make planning tractable.

Chapter 3: Object-Oriented World Modeling. This chapter establishes the foundational abstraction for the thesis: how to learn compositional representations from object-oriented environments. We introduce the Homomorphic Object-oriented World Model (HOWM) that learns to bind actions to objects and model their interactions through slot-based representations. This addresses the fundamental binding problem in neural networks [Greff et al., 2020] by learning structured representations where objects can be composed in novel ways while preserving dynamics. The chapter provides the compositional "vocabulary" (object slots and factorized actions) that subsequent chapters will exploit for efficient planning.

Linfeng Zhao, Lingzhi Kong, Robin Walters, Lawson L.S. Wong. "Toward Compositional Generalization in Object-Oriented World Modeling." International Conference on Machine Learning (ICML), 2022. Long Presentation (2.1%).

Chapter 4: Symmetry as Structure for Planning. Building on the object-centric representations from Chapter 3, we now inject geometric structure into planning algorithms. This chapter introduces symmetry-aware differentiable planning, where planning networks respect the inherent geometric symmetries of the environment (rotations, translations). We develop SymVIN, which uses steerable convolutions to make Value Iteration Networks equivariant to SE(2) transformations. This enables planning that generalizes across different orientations and positions of the same spatial layout, providing our first connection between learned abstractions and structure-aware planning.

Linfeng Zhao, Xupeng Zhu, Lingzhi Kong*, Robin Walters, Lawson L.S. Wong. "Integrating Symmetry into Differentiable Planning with Steerable Convolutions." International Conference on Learning Representations (ICLR), 2023.*

Chapter 5: E(2)-Equivariant Graph Planning. This chapter extends the symmetry-aware planning from Chapter 4 to more complex spatial representations. Moving beyond regular grids, we develop E(2)-equivariant planning on geometric graphs, enabling navigation in environments represented as spatial scene graphs. We introduce

MPVN (Message Passing Value Networks) that combine the symmetry principles from Chapter 4 with graph neural networks, bridging between discrete graph structure and continuous geometric symmetries. This demonstrates how compositional abstractions can be applied across different spatial representations while preserving their geometric structure.

Linfeng Zhao, Hongyu Li*, Taskin Padir, Huaizu Jiang, Lawson L.S. Wong. "E(2)-Equivariant Graph Planning for Navigation." IEEE Robotics and Automation Letters (RA-L), 2024, co-presented in Oral at IROS 2024.*

Chapter 6: Extensions and Variations. This chapter explores several extensions that broaden the scope of compositional abstractions developed in Chapters 3-5. We cover: (1) equivariant action sampling for continuous control that extends symmetry principles to action generation, (2) spatial maps as an alternative compositional abstraction to object-centric representations, and (3) selected collaborative work on structural representation learning. These extensions demonstrate the flexibility and broader applicability of the compositional abstraction framework beyond the core object-symmetry pipeline.

Key work: Linfeng Zhao, Owen Howell, Xupeng Zhu, Jung Yeon Park, Zhewen Zhang, Robin Walters, Lawson L.S. Wong. "Equivariant Action Sampling for Reinforcement Learning and Planning." Workshop on Algorithmic Foundations of Robotics (WAFR), 2024.

Part II: Planning and Integration with Agent. Part II develops **scalable planning algorithms that exploit the compositional abstractions from Part I and integrates them into complete robotic systems**. We address three key challenges: making planning algorithms trainable and scalable (Chapter 7), integrating learned spatial abstractions with end-to-end perception (Chapter 8), and handling uncertainty and partial observability through belief-space planning (Chapter 9). Together, these chapters demonstrate how the compositional abstractions from Part I can be deployed in practical robotic systems.

Chapter 7: Scaling Differentiable Planning. This chapter addresses the fundamental challenge of making the planning algorithms from Part I practical for real applications. Building on the structured representations and symmetric priors from Chapters 3-6, we develop implicit differentiation techniques that stabilize training and enable scaling to larger planning horizons and more complex scenes. This provides the algorithmic foundation needed to deploy compositional abstractions in real robotic systems with computational constraints.

Linfeng Zhao, Huazhe Xu, Lawson L.S. Wong. "Scaling up and Stabilizing Differentiable Planning with Implicit Differentiation." International Conference on Learning Representations (ICLR), 2023.

Chapter 8: Belief-Space Planning with Foundation Models. This chapter addresses planning under uncertainty and partial observability. Building on the compositional abstractions from Part I and the scalable planning from Chapter 7, we integrate Vision-Language Models for belief estimation with strategic information-

gathering through planning. The system reasons about what it doesn't know and plans to observe what matters for task completion, handling real-world uncertainty through structured belief representations.

Linfeng Zhao, Willie McClinton, Aidan Curtis*, Nishanth Kumar, Tom Silver, Leslie Kaelbling, Lawson L.S. Wong. "Seeing is Believing: Belief-Space Planning with Foundation Models as Uncertainty Estimators." arXiv, 2025.*

Chapter 9: Extensions and Variations (Part II). This chapter presents two complementary extensions. First, collaborative work on learning skill parameter policies [Kumar et al., 2024a] shows how planning can guide autonomous skill improvement during robot deployment. Second, map-based navigation demonstrates zero-shot generalization using spatial abstractions as an alternative to object-centric representations, learning implicit correspondences between 2D maps and 3D environments.

Nishanth Kumar, Tom Silver, Willie McClinton, Linfeng Zhao, Stephen Proulx, Tomás Lozano-Pérez, Leslie Pack Kaelbling, Jennifer Barry. "Practice Makes Perfect: Planning to Learn Skill Parameter Policies." Robotics: Science and Systems (RSS), 2024.

Linfeng Zhao, Lawson L.S. Wong. "Learning to Navigate in Mazes with Novel Layouts Using Abstract Top-down Maps." Reinforcement Learning Conference (RLC), 2024.

Chapters 10: Discussion and Future Directions. Overview and discussion of the entire thesis and what's next.

Chapter 2

Background

2.1 Decision-Making Setup

At time step $t \in \mathbb{N}$ the agent observes multimodal input o_t (vision, language, proprioception, touch), chooses an abstract action a_t in the form of an option, and then receives the next observation o_{t+1}. Low-level controllers still issue primitive controls $u_t \in \mathcal{A}$; the planner reasons at the option level [Sutton et al., 1999, Precup, 2000]. Tasks are specified by goal predicates or sparse rewards, and success is evaluated over abstract facts rather than pixels.

This framework naturally applies to **mobile manipulation** scenarios, where a robot must coordinate navigation through an environment with object manipulation tasks. For example, a mobile robot might need to navigate to a kitchen, identify objects like cups or plates, manipulate them (grasping, placing, stacking), and transport them to different locations—all while handling uncertainty about object properties and environmental layout.

We model the physical world as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \gamma)$ with states $s_t \in \mathcal{S}$, primitive controls $u_t \in \mathcal{A}$, transition kernel $T(s' \mid s, u)$ and reward $R(s, u)$. Observations $o_t = \mathcal{O}(s_t)$ are generated by an observation function $\mathcal{O} : \mathcal{S} \to \Omega$. Because we assume the setting is typically **fully observable**, we take this observation to reveal the low-level state unless noted otherwise. Planning operates over an abstract decision space $\tilde{\mathcal{S}}$ with abstract states $\tilde{s}_t \in \tilde{\mathcal{S}}$ produced by an estimator $\tilde{s}_t = \psi(o_{1:t})$, optionally accompanied by predicate-belief vectors b_t when uncertainty is represented explicitly.

We use **options** [Sutton et al., 1999] as temporally extended actions that abstract over low-level control. Each option $a = \text{option}(\boldsymbol{\varphi})$ has parameters $\boldsymbol{\varphi}$ and a contract with preconditions $\text{pre}(a)$, effects $\text{eff}(a)$, and termination conditions $\text{term}(a)$. Examples include grasp(obj), navigate(location), or place(obj, target). Planning finds sequences $\pi = (a_1, \ldots, a_T)$ that achieve goal predicates while minimizing expected cost.

Key assumptions. We assume **downward refinability**: if option contracts in a high-level plan hold and their predicates are believed true, low-level controllers or motion planners can realize each step, so we do not address integrated task–motion search. The abstraction interface is treated as sufficiently observable to evaluate option

contracts; when this fails, the agent inserts information-gathering options and belief updates. We also assume the **availability of action abstractions** in the form of parameterized options (not necessarily a fixed library).

Why problems remain challenging under these assumptions. Even with these simplifying assumptions, significant challenges persist. Consider a mobile manipulation scenario where a robot must "clear the dining table by moving dishes to the kitchen sink":

- **Compositional generalization**: The robot must handle novel object combinations (different dish types, table layouts) and compose known skills (grasp, navigate, place) in new ways. Even with downward refinability, determining *which* composition works for *which* novel scenario requires learning compositional structure.
- **Representation learning**: Objects must be identified and tracked across viewpoints and partial occlusion. The state estimator $\psi(o_{1:t})$ must learn to extract stable object representations and predict how option executions will change the scene—a non-trivial learning problem even when we assume sufficient observability.
- **Long-horizon coordination**: Multi-step tasks like clearing an entire table involve coordinating dozens of primitive actions across navigation and manipulation. Even with option-level planning, the temporal credit assignment and exploration challenges grow exponentially with horizon length.
- **Planning under uncertainty**: The robot may not know if a dish is fragile, heavy, or dirty without inspection. Strategic information-gathering becomes essential: should it examine each object first, or attempt conservative manipulation and adjust when failures occur? This requires reasoning about expected information value and failure costs.

For planning under uncertainty, we adopt the **open-world assumption** [Reiter, 1978, Genesereth and Nilsson, 1987], where unobserved predicates (or queries to the environment) in an abstract state are assumed **unknown** (triggering perception to resolve uncertainty), contrasting with the **closed-world assumption** where unobserved predicates are simply treated as false. We study world modeling and planning in an **abstract space** (e.g., objects, predicates) rather than raw observation space (e.g., pixels), which raises **grounding challenges**: lossless abstractions preserve optimality but require careful state estimation, while lossy abstractions enable tractability but need learned models to bridge the abstraction gap. Pixel-level prediction, continuous motion planning inside options, and full TAMP remain out of scope.

2.2 Abstraction

We treat abstraction as the agent's I/O interface: what is represented (state) and what can be done (action) at the level where planning occurs. This mirrors the introduction's

principle that the right units plus the right rules enable composition and generalization.

2.2.1 State Abstraction

The state abstraction serves as a decision-level summary of the world: it keeps just enough structure to decide which option to run next while hiding low-level sensor detail. We denote the abstract state space as $\tilde{\mathcal{S}}$ and a state as $\tilde{s}$. Crucially, we do not prescribe a specific representation for these compositional abstract states—they can be:

- *Implicitly defined*: Internal states of Vision-Language Models, learned embeddings from neural networks, or latent vectors that capture scene structure without explicit symbolic grounding
- *Explicitly defined*: Compositional abstractions like objects and their relations, spatial regions and connectivity, or symbolic predicates over typed entities

The key requirement is that these abstract states support planning for options—they must contain sufficient information to evaluate option preconditions, predict effects, and assess goal achievement. What makes them *compositional* is that we can define algebraic operations on these states. For set-based representations (e.g., sets of objects), we have natural operations like union, intersection, and replacement that preserve planning structure. For continuous representations, group operations (rotations, translations, permutations) define the compositional algebra. This algebraic structure enables systematic generalization: if we know how options transform individual components, we can predict their effects on novel compositions.

This formulation naturally connects with modern Vision-Language Models. VLMs can serve as the state estimator $\psi : \Omega^* \to \tilde{\mathcal{S}}$, mapping observations to abstract states. They excel at the perceptual aspects—recognizing objects, evaluating spatial relations, answering questions about scene content. However, they lack the systematic planning capabilities to reason about how sequences of options transform these states toward goals. Our planning algorithms provide this missing capability, operating on whatever abstract state representation the perceptual system provides.

Canonical state abstraction frameworks provide theoretical foundations: model minimization and bisimulation metrics [Dean and Givan, 1997, Li et al., 2006b, Ferns et al., 2004] for lossless compression, object-oriented MDPs [Diuk et al., 2008] for factored representations, and option-aware abstractions [Konidaris, 2019, Abel et al., 2020b,a] for hierarchical decision-making. In practice, Chapter 3 implements this through slot-based representations where objects appear as learned embeddings, predicates are evaluated through attention mechanisms, and uncertainty is tracked explicitly when needed.

2.2.2 Action Abstraction via Options

Intuitively, an *option* represents a mapping from a subset of states to another subset of states—a reusable skill that transforms the world in predictable ways. For example,

grasp(obj) maps from states where the gripper is empty and near the object to states where the object is held. This subset-to-subset view captures the essential planning structure: we chain options together when the target subset of one overlaps with the source subset of the next.

At the abstract planning level, each option is summarized by a *contract*:

- **Preconditions**: The source subset of states where the option can initiate
- **Effects**: The target subset of states likely after successful execution
- **Parameters**: Variables that specialize the option (which object to grasp, where to navigate)

Typical options include grasp(obj), place(obj, target), navigate(location), and inspect(region) for information gathering. The planner's job is to find a sequence of options whose composed state transformations achieve the goal—without needing to reason about the low-level control within each option.

More formally, an option is defined as

$$\omega = (\mathcal{I}_\omega, \pi_\omega, \beta_\omega, \boldsymbol{\varphi}) \tag{2.1}$$

where $\mathcal{I}_\omega \subseteq \tilde{\mathcal{S}}$ is the initiation set (source subset), π_ω is the internal policy that executes the skill, β_ω is the termination condition, and $\boldsymbol{\varphi}$ are parameters. The abstract contract

$$\mathcal{C}(\omega) = (\text{pre}_\omega, \text{eff}_\omega, \text{term}_\omega) \tag{2.2}$$

exposes these components to the planner. This framework naturally encompasses classical planning operators: STRIPS/PDDL actions [Fikes and Nilsson, 1971, Haslum et al., 2019] are options with deterministic contracts, while learned skills are options with probabilistic execution but structured contracts.

This framework naturally integrates with modern foundation models (2023-2025). VLMs excel at providing the state estimator ψ and evaluating predicates—they can answer "what do I see?" and "is the cup on the table?" However, they lack systematic ways to plan over options—they cannot reliably answer "what sequence of actions achieves my goal?" The planning algorithms in this thesis provide that missing capability, transforming foundation model perception into reliable action sequences. Chapter 8 demonstrates this concretely by using VLMs for state estimation while our belief-space planner selects information-gathering and task-completion options.

2.2.3 Fidelity: Lossless vs. Lossy Abstraction

This thesis explores two complementary approaches to abstraction that reflect a fundamental trade-off in planning: optimality preservation versus computational efficiency.

Lossless Abstractions preserve optimality while reducing computational complexity through structured compression. These abstractions maintain strict equivalences that guarantee optimal solutions in the abstract space correspond to optimal solutions in the original space. Two primary sources provide such structure:

Symmetry-based abstraction. When a symmetry group G acts on the state and action spaces, states within the same orbit $[s]_G = \{g \cdot s : g \in G\}$ are control-equivalent if the dynamics and rewards are G-invariant:

$$T(g \cdot s, g \cdot a, g \cdot s') = T(s, a, s'), \quad R(g \cdot s, g \cdot a) = R(s, a) \tag{2.3}$$

The quotient space $\hat{\mathcal{S}} = \mathcal{S}/G$ yields a smaller MDP with preserved optimality via MDP homomorphisms [Ravindran and Barto, 2003, Li et al., 2006b]. Chapter 4 exploits D(4) symmetry for 8× reduction in grid navigation, while Chapter 5 extends to E(2) symmetry for continuous environments.

Object-centric factorization. States can factorize into object descriptors $s = (f_1(s), \ldots, f_n(s))$ following object-oriented MDPs [Diuk et al., 2008], enabling sparse transitions $T(s, a, s') \approx \prod_i T_i(f_i(s), a, f_i(s'))$ and compositional rewards $R(s, a) \approx \sum_i R_i(f_i(s), a)$. Chapter 3's HOWM combines both structures: Σ_N permutation symmetry with object slots, where equivariance error $\lambda = \max_{g \in \Sigma_N} \|f(g \cdot s) - g \cdot f(s)\|$ measures compositional consistency.

Lossy Abstractions employ more aggressive simplifications that intentionally discard information to achieve greater computational efficiency and broader applicability. While they may produce suboptimal solutions, they enable tractability in complex environments where lossless abstractions are insufficient.

Map-based representations compress continuous environments into discrete spatial abstractions (Chapter 9), trading geometric precision for navigational efficiency. *Symbolic representations* use predicate logic and typed entities to capture high-level relationships while abstracting away low-level dynamics. *Language-grounded operators* leverage natural language descriptions as option contracts, enabling generalization across semantic domains. *Belief abstractions* represent uncertainty through three-valued logic (True/False/Unknown) rather than full probability distributions, as demonstrated in Chapter 8's belief-space planning.

Flexible Fidelity. Crucially, the same abstraction *types* can be implemented in both lossless and lossy ways depending on system requirements. Object representations can maintain full geometric precision (lossless) or use coarse spatial discretizations (lossy). Symmetry can be enforced exactly through equivariant architectures (lossless) or approximated through data augmentation (lossy). The same option contracts (pre/eff/term) enable swapping between fidelity levels: higher fidelity is invoked where grounding demands it, while lossy models supply tractability and generalization.

This thesis demonstrates that structured abstractions—whether lossless or lossy—enable systematic generalization beyond pure learning approaches while maintaining computational efficiency through compositional design. The choice of fidelity becomes a design decision based on task requirements and computational constraints rather than a fundamental limitation.

2.3 Compositionality

We formalize composition rules for actions, state, and belief so that long-horizon behavior emerges from short parts through principled combination. The goal is that if

the agent understands how components fit together, it can solve longer tasks without relearning everything from scratch.

2.3.1 Action Composition

Options combine through their contract structure (preconditions, effects, termination conditions) to form longer plans [Sutton et al., 1999, Bacon et al., 2017]. The key composition patterns are: **Sequential composition** chains options when effects satisfy next preconditions (navigate(kitchen) $\rightarrow$ grasp(cup) $\rightarrow$ place(cup, dishwasher)), **choice composition** selects among alternatives based on conditions (open(door) choosing turn-handle vs push by door type), **parallel composition** executes independent options simultaneously when they don't interfere, and **iterative composition** repeats actions until termination (clear-table repeating grasp(dish) $\rightarrow$ navigate(sink) $\rightarrow$ place(dish, sink) until table empty).

For mobile manipulation, these patterns enable complex task execution: a robot clearing a dining table might sequentially navigate to each object, grasp it, navigate to appropriate destinations (sink for dishes, trash for napkins), and place items correctly. Choice composition handles different object types (fragile vs. sturdy dishes requiring different grasp strategies), while iterative composition continues until all objects are cleared. This compositional structure enables systematic construction of complex behaviors from simpler building blocks [Vezhnevets et al., 2017].

2.3.2 State Composition

Building on the state abstraction framework (Section 2.2.1), we formalize how abstract states compose and transform under option execution. The key insight is that compositional structure in the abstract state space $\tilde{\mathcal{S}}$ enables local updates and systematic generalization.

Object-centric composition. When abstract states are explicitly structured as objects and relations, options typically affect only local subsets. For instance, grasp(obj) modifies the state of obj and the gripper while leaving other objects unchanged. This locality enables efficient state updates where only affected components change while the rest of the scene structure is preserved. Chapter 3's HOWM exploits this through slot-based representations where each slot tracks an object independently, enabling local updates without recomputing the entire state.

Relational composition. Predicates like $\text{on}(o_1, o_2)$ or $\text{near}(o_1, o_2)$ capture relationships that persist across object substitutions. The same place option works whether placing a cup on a table or a book on a shelf—the relational structure transfers. This enables generalization: learning how options transform relations on training objects predicts their effects on novel objects.

Symmetry-based composition. Group symmetries define equivalence classes in state space. States differing only by rotations, translations, or object permutations share compositional structure. Chapters 4-5 formalize this through MDP homomorphisms,

where symmetry group actions preserve transition dynamics. This reduces the effective state space by the symmetry group's order while preserving optimal policies.

Implicit composition. Even when states are implicit (e.g., VLM embeddings), compositional structure can emerge through learning. The state estimator $\psi : \Omega^* \to \tilde{\mathcal{S}}$ learns to map observations to representations that support option-level planning. Modern foundation models excel at this perceptual abstraction—recognizing objects, evaluating spatial relations, answering questions about scenes—providing the state representation our planning algorithms operate on.

Belief composition under uncertainty. When predicates are uncertain, we extend to belief states $b \in \Delta(\tilde{\mathcal{S}})$. Chapter 8's approach uses three-valued logic $\{\text{true}, \text{false}, \text{unknown}\}$ for tractable belief representation, where unknown predicates trigger information-gathering options. This compositional belief structure enables efficient belief updates after observations.

2.4 Planning at Decision Time

Planning, broadly defined, is the process of finding a sequence of actions that transforms the current state into a goal state [LaValle, 2006a]. This encompasses a spectrum from classical symbolic planning (STRIPS, PDDL [Fikes and Nilsson, 1971, Ghallab et al., 2004]) to trajectory optimization, model predictive control, and learned planning in neural networks [Tamar et al., 2016a]. In our framework, planning selects options to reach a goal region (a subset of the state space) while respecting the option contracts defined above.

We assume **downward refinability**: our focus is on high-level planning only, where valid abstract plans can be delegated to low-level controllers without additional task–motion reasoning. Planning therefore queries the option-level model to find a finite sequence $\pi = (a_1, \ldots, a_T)$ that minimizes expected cumulative cost (or equivalently maximizes success probability); practical implementations approximate this objective through heuristic search or learned guidance.

The behavior of $\tilde{T}$ and $\tilde{R}$ depends on the abstraction in use. In lossless settings—symmetry-aware planners such as SymVIN or slot-based planners such as HOWM—the abstract model preserves optimality and exploits equivariance or object factoring. In lossy settings—map-conditioned navigation or query-based perception planners—the model is approximate, and learning supplies outcome predictors, heuristics, and proposal distributions that make option selection effective despite the coarser representation.

2.5 Learning for Abstraction and Planning

The key idea is to identify the **basic building blocks** for embodied decision-making, analogous to how language modeling identifies words as the fundamental units of language. Language modeling succeeds through a principled learning framework: (1) distributed representations of words as vectors and (2) learning how these vectors interact

through attention and composition. We apply the same principle to decision-making by identifying objects, predicates, and options as our basic building blocks.

Distributed representations as the key insight. Rather than localist encoding (one-hot vectors), language models learn distributed representations where each concept is encoded by many features, and each feature participates in representing many concepts [Rumelhart et al., 1986, Smolensky, 1990, Plate, 1995]. This enables: (1) **feature sharing** across related concepts, reducing sample complexity; (2) **geometric structure** where similar vectors behave similarly; (3) **compositional operations** through vector arithmetic and attention [Vaswani et al., 2017]; and (4) **differentiable optimization** through shared features. Neural language models learn these representations jointly with prediction through co-training that ties geometry to next-token objectives [Bengio et al., 2003].

Adapting to planning. We apply this framework by treating objects, predicates, and options as decision-making tokens. Attention mechanisms bind objects to predicates, select relevant options, and compose multi-step plans—enabling differentiable composition throughout the planning process.

Toward joint learning of abstractions and planning. A central challenge in decision-making is ensuring that learned abstractions $\psi : \Omega^* \to \tilde{\mathcal{S}}$ are not only predictive of future observations but also **effective for planning**. This connects to the broader principle that representations should be optimized for their downstream use—a lesson from language modeling where embeddings and prediction objectives are co-trained [Bengio et al., 2003].

Traditional pipelines train perception, world modeling, and planning sequentially—optimizing each component independently. This can yield representations that predict well but plan poorly. Joint learning instead optimizes all components toward a shared planning objective that measures task success rather than just prediction accuracy. This approach has been demonstrated across multiple paradigms: model-based approaches like World Models [Ha and Schmidhuber, 2018] train controllers within learned "dreams," while MuZero [Schrittwieser et al., 2020] learns planning-relevant world models and the Dreamer series [Hafner et al., 2020b, 2021, 2023] achieves state-of-the-art performance through imagination rollouts; transformer approaches like Decision Transformer [Chen et al., 2021] treat planning as sequence modeling; foundation model approaches integrate large-scale pretraining with robotic control [Brohan et al., 2022, 2023, Ahn et al., 2022a]; and object-centric approaches like our HOWM work [Zhao et al., 2022a] jointly train object detection, dynamics, and planning with equivariant representations.

The key insight across these approaches is that **planning-centric objectives** often yield different representations than prediction-centric ones. While we explore this principle in several works—differentiable planning (Chapter 4), belief-space planning (Chapter 6), and spatial abstraction (Chapter 5)—the full realization of end-to-end joint training remains an active research direction where abstractions, world models, and planning components are optimized together toward downstream task performance.

Part I

Learning with Compositional Abstractions

Chapter 3

Object-Oriented World Modeling

Environments with objects are a natural setting for embodied decision-making, where scenes are composed of distinct entities (objects) with individual properties and dynamics [Diuk et al., 2008]. Recently, there has been significant interest in learning world models for such environments from visual observations [Burgess et al., 2019, Watters et al., 2019, Kipf et al., 2019, Kossen et al., 2019, Lin et al., 2020, Veerapaneni et al., 2020, Locatello et al., 2020]. These models learn object-centric representations that factorize scenes by their constituent objects, enabling more structured reasoning about environment dynamics.

This chapter addresses a form of generalization that has been less formally studied in action-conditioned world modeling: *compositional generalization*. When a robot trained on scenes with familiar objects encounters novel combinations, it should recognize individual objects and predict their effects appropriately. For example, if trained on scenes containing {▲, ▼} and {◀, ▶}, the model should generalize to novel scenes {▲, ◀} without additional training. To study this systematically, we introduce Object Library environments featuring K objects drawn from a library of N objects ($K < N$), where the K objects remain consistent during each episode. Although visually distinct, these environments are isomorphic to each other, with the isomorphism factorized by their constituent objects. While compositional generalization has been extensively studied in natural language [Johnson et al., 2017, Lake and Baroni, 2018, Bahdanau et al., 2019, Gordon et al., 2019, Keysers et al., 2020], this capability requires new formulations for environments where actions have state-dependent effects.

The thesis principle of *Abstraction = I/O interface* guides our approach. Objects serve as the basic representational units that can be composed according to systematic rules [Diuk et al., 2008]. We formalize compositional generalization as equivariance to object-replacement operations—the ability to generalize from one scene to another with replaced objects. This bridges behavioral generalization (predicting dynamics of novel combinations) with functional implementation (equivariant models). Two paths emerge for achieving this: (1) exact compositional generalization with intensive resource usage, or (2) learning soft object and action binding in latent space [Burgess et al., 2019, Watters et al., 2019, Kipf et al., 2019, Kossen et al., 2019, Lin et al., 2020, Veerapaneni et al., 2020, Locatello et al., 2020]. The primary challenge for the latter

path is not merely learning object representations, but correctly *binding actions to the right objects*—determining which objects are affected by which actions in the absence of canonical object ordering in visual observations.

The technical contribution introduces the Homomorphic Object-oriented World Model (HOWM), which addresses the soft compositional generalization path. HOWM deploys an Action Attention module to learn which latent object slots are affected by each action, and can be trained differentiably with the Aligned Loss. *We prove that if the model learns correct action-slot binding, this approach achieves perfect compositional generalization*—matching exact methods but with computational scaling from library size to scene size ($\mathcal{O}(N^2) \to \mathcal{O}(K^2)$), where N is the library size and K is the number of objects present in each scene.

This chapter is based on our paper "Toward Compositional Generalization in Object-Oriented World Modeling" published at ICML 2022.

3.1 Object Library: A Framework for Studying Compositional Generalization

To study compositional generalization systematically, we design a class of object-oriented environments called **Object Library**, illustrated in Figure 3.1. These environments feature K objects drawn from a library of N objects, where the K objects remain consistent during each episode. Although visually distinct, these environments are isomorphic to each other. The isomorphism is factorized by constituent objects, enabling models to generalize to new scenes (novel object combinations) if constituent objects have been observed during training.

For example, if during training we observe scenes containing {▲, ▼} and {◀, ▶}, models should accurately predict dynamics in novel scenes {▲, ◀} and {▼, ▶}.

We formally define compositional generalization as the ability to generalize from one scene to another with *replaced* objects—equivalently, *equivariance to object-replacement operations*. This bridges behavioral generalization with functional implementation through equivariant models.

3.2 Background

Our environments are modeled as Markov decision processes. A Markov decision process (MDP) is a 5-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $T\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$, reward function $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1]$.

MDP homomorphisms. In this paper, we will study families of related MDPs using the framework of MDP homomorphisms [Ravindran and Barto, 2004, van der Pol et al., 2020a]. An *MDP homomorphism* $h\colon \mathcal{M} \to \overline{\mathcal{M}}$ is a mapping from one MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ to another $\overline{\mathcal{M}} = \langle \overline{\mathcal{S}}, \overline{\mathcal{A}}, \overline{T}, \overline{R}, \gamma \rangle$. The map h consists of a tuple

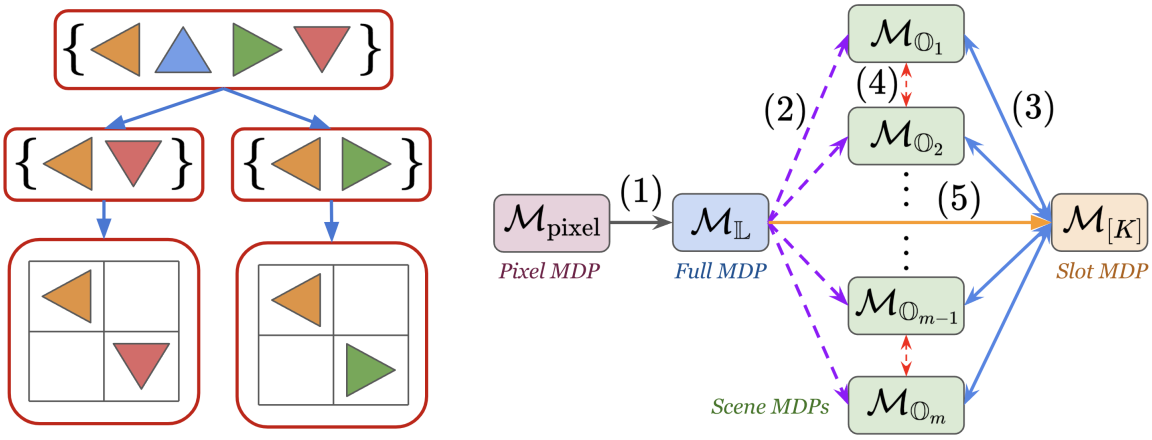

Figure 3.1: **(Left)** An example of our Object Library environment: Rush Hour, described in Section 3.4.1. Each scene features $K = 2$ objects from a library of $N = 4$ objects. **(Right)** A family of MDPs for modeling Object Library, as explained in Section 3.3.3.

of surjective maps $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\}\rangle$, where $\phi\colon \mathcal{S} \to \overline{\mathcal{S}}$ is the state mapping and $\alpha_s\colon \mathcal{A} \to \overline{\mathcal{A}}$ is the *state-dependent* action mapping. The mappings are constructed to satisfy the following conditions for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$:

$$\begin{aligned} \overline{R}\left(\phi(s), \alpha_s(a)\right) &= R(s,a), \\ \overline{T}\left(\phi\left(s'\right) \mid \phi(s), \alpha_s(a)\right) &= \sum_{s'' \in \phi^{-1}(\phi(s'))} T\left(s'' \mid s, a\right). \end{aligned} \tag{3.1}$$

We call the *reduced* MDP $\overline{\mathcal{M}}$ the *homomorphic image* of $\mathcal{M}$ under h. If $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\}\rangle$ has *bijective* maps ϕ and $\{\alpha_s\}$, we call h an *MDP isomorphism*.

Symmetry groups and equivarance. In Section 3.4, we will formalize compositional generalization in object-oriented environments as a type of "object-replacement" symmetry. In mathematics, a *symmetry group* G is an algebraic concept, denoting the collection of all symmetric transformations of an entity, satisfying the axioms of associativity, identity, inverse, and closure. A (left) *group operation* (action) of a group G on a set $\mathcal{X}$ is defined as the mapping $(g, x) \mapsto g.x$. If f is a function $f : \mathcal{X} \to \mathcal{Y}$ and G acts on $\mathcal{X}$ and $\mathcal{Y}$, then f is an *equivariant map* if it commutes with the operation of the group: $g.f(x) = f(g.x), \forall g \in G, \forall x \in \mathcal{X}$. The function f is considered *G-equivariant*. If instead $f(x) = f(g.x)$ holds, then f is considered *G-invariant*.

MDP homomorphisms with symmetry. The above concepts can be connected together. Given group G, an MDP homomorphism h is said to be *group structured* if any state-action pair (s, a) and its transformed counterpart $g.(s, a)$ are mapped to the same abstract state-action pair: $(\phi(s), \alpha_s(a)) = (\phi(g.s), \alpha_{g.s}(g.a))$, for all $s \in \mathcal{S}, a \in \mathcal{A}, g \in G$. For convenience, we denote $g.(s, a)$ as $(g.s, g.a)$, where $g.a$ implicitly[1]

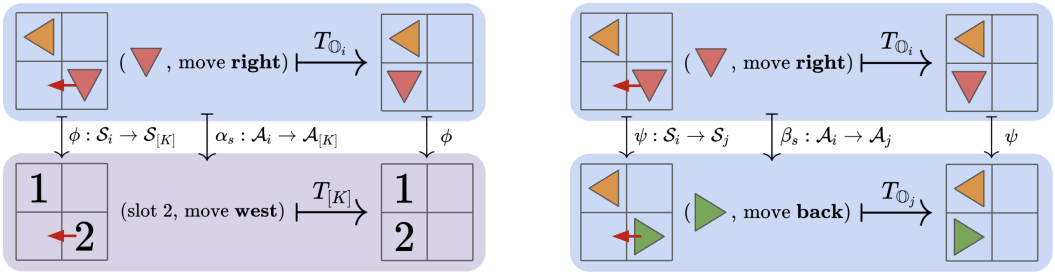

Figure 3.2: An illustrative commutative diagram. **(Left)** For any scene MDP $\mathcal{M}_{\mathbb{O}_i}$, we can bind its objects and actions to the representative slot MDP $\mathcal{M}_{[K]}$. Note that the *object-relative* action "`right`" needs to be converted to the *absolute representative* action "`west`" in $\mathcal{M}_{[K]}$. **(Right)** Since the binding between objects (and their actions) and slots is one-to-one, we can relate two scene MDPs by composing the binding $h_{i\to j} = h_i \circ h_j^{-1}$; the two MDPs are isomorphic.

depends on state s. Applied to the transition and reward functions, the transition function T is G-equivariant if T satisfies $T(g.s'|g.s, g.a) = T(s'|s, a)$, and reward function R is G-invariant if $R(g.s, g.a) = R(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}, g \in G$.

This section is intentionally brief; see van der Pol et al. [2020a] and Ravindran and Barto [2004] for a more in-depth account of MDPs with symmetries. To keep the exposition simple in the main text, we focus on predicting transition dynamics only and assume that transitions are deterministic. The derivations extend naturally to stochastic environments and rewards.

3.3 Object-Oriented Environments

In this section, we define the concept of object-oriented environments (OOE), and introduce a specific family of OOEs, Object Library, for studying composition generalization. We model these environments as MDPs, and in the process we will define multiple related MDPs for Object Library.

3.3.1 Image-based environments and factorization

We study image-based environments with multiple objects. At each time step, the agent observes an image $s_t \in \mathcal{S}$, consisting of multiple objects that fully describe the current *scene*, and takes an action $a_t \in \mathcal{A}$ to control a single object. We model this as an MDP with image-based states, $\mathcal{M}_{\text{pixel}}$. What distinguishes $\mathcal{M}_{\text{pixel}}$ from an arbitrary high-dimensional MDP with a high-dimensional state space is that the environment actually has low-dimensional factorized structure, in that the environment is fully determined

[1]The group operation acting on action space $\mathcal{A}$ *depends on state*, since G actually acts on the *product space* $\mathcal{S} \times \mathcal{A}$, $(g, (s, a)) \mapsto g.(s, a)$, while we denote it as $(g.s, g.a)$ for consistency with $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\}\rangle$. As a bibliographical note, in van der Pol et al. [2020a], the group acting on state and action space is denoted as state transformation $L_g \colon \mathcal{S} \to \mathcal{S}$ and *state-dependent* action transformation $K_g^s \colon \mathcal{A} \to \mathcal{A}$.

solely by the objects in the scene and their relations [Ravindran and Barto, 2004, Diuk et al., 2008].

More formally, we define $\mathcal{M}_{\text{pixel}}$ to be an object-oriented environment (OOE) if there exists an MDP homomorphism $\mathcal{M}_{\text{pixel}} \to \overline{\mathcal{M}}$ that maps the image-based MDP to a factored MDP $\overline{\mathcal{M}}$ [Guestrin et al., 2003]. $\overline{\mathcal{M}}$ has factorized state and action spaces $\overline{\mathcal{S}} = \overline{\mathcal{S}}_1 \times \ldots \times \overline{\mathcal{S}}_N, \overline{\mathcal{A}} = \overline{\mathcal{A}}_1 \times \ldots \times \overline{\mathcal{A}}_N$, where each factor is meant to model one of N objects, and ideally has significantly lower dimension. We use a factorized action space where each action component corresponds to controlling a specific object.

3.3.2 Object Library

In this paper, we consider compositional generalization within a family of OOEs, which we call *Object Library*. Object Library is an OOE augmented with a library of all possible objects $\mathbb{L} = \{o_1, \ldots, o_N\}$, of which only K objects are present in any given scene, where $1 < K < N$. Note that the library is not given and is meant to be learned from images; we merely introduce it as a conceptual tool.

In each episode, the K objects are persistent, but between episodes a different set of K objects may be chosen from $\mathbb{L}$. This allows us to generate different scenes with different object sets during training and evaluation, thus enabling us to measure the performance discrepancy when faced with different *object compositions*. Compositional generalization will be formally defined in the next section.

3.3.3 Modeling Object Library as a family of MDPs

We now define several MDPs for modeling Object Library, illustrated in Figure 3.1.

- Pixel MDP $\mathcal{M}_{\text{pixel}}$: This is the original environment with image-based states.
- Full MDP $\mathcal{M}_{\mathbb{L}}$: We define $\mathcal{M}_{\mathbb{L}}$ to be the latent factored MDP, with one state/action factor per object: $\mathcal{S}_{\mathbb{L}} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_N$, and likewise for $\mathcal{A}_{\mathbb{L}} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$. Since there are only K objects in the scene, each $\mathcal{S}_i$ contains a special null state ε that indicates the object is not present.
- Scene MDP $\mathcal{M}_{\mathbb{O}}$: Since the K objects persist in any particular instance of the Object Library environment, both $\mathcal{M}_{\text{pixel}}$ and $\mathcal{M}_{\mathbb{L}}$ are partitioned into $\binom{N}{K}$ connected components, or *sub-MDPs* [Ravindran and Barto, 2004]. It is convenient to consider each of these sub-MDPs, which we call the *scene MDP* $\mathcal{M}_{\mathbb{O}}$. More formally, let $\mathbb{O} = \{i_1, \ldots, i_K\}$ be the indices of the K present objects. We construct $\mathcal{M}_{\mathbb{O}}$ by projecting $\mathcal{S}_{\mathbb{L}}$ and $\mathcal{A}_{\mathbb{L}}$ to $\mathbb{O}$: $\mathcal{S}_{\mathbb{O}} = \mathcal{S}_{i_1} \times \ldots \times \mathcal{S}_{i_K}$, and likewise for $\mathcal{A}_{\mathbb{O}}$.
- Slot MDP $\mathcal{M}_{[K]}$: We will eventually show that all scene MDPs are isomorphic to each other. Key to this construction is the existence of a *representative* factored MDP among all scenes, which we denote as the *slot MDP* $\mathcal{M}_{[K]}$. $\mathcal{M}_{[K]}$ has K object "slots": $\mathcal{S}_{[K]} = \mathcal{S}_1 \times \ldots \mathcal{S}_K$, and likewise for $\mathcal{A}_{[K]}$. Ideally, we can "bind" every scene MDP $\mathcal{M}_{\mathbb{O}}$ to $\mathcal{M}_{[K]}$ *without* any loss of information, since a scene MDP has K objects and the slot MDP can represent the K objects using its K slots.

The relationship between these MDPs are shown in Figure 3.1 right. There exists an MDP homomorphism from the high-dimensional $\mathcal{M}_{\text{pixel}}$ to the low-dimensional factored $\mathcal{M}_{\mathbb{L}}$ (black arrow). Depending on the K objects present in the scene, $\mathcal{M}_{\mathbb{L}}$ can be projected onto one of $\binom{N}{K}$ scene MDPs $\mathcal{M}_{\mathbb{O}_i}$ (purple dashed arrow). Each of these $\mathcal{M}_{\mathbb{O}_i}$ is isomorphic to $\mathcal{M}_{[K]}$ (blue arrow, double headed), and therefore are also isomorphic to each other (red dashed arrow, double headed). Our analysis in Section 3.4 also assumes the existence of an MDP homomorphism from $\mathcal{M}_{\mathbb{L}}$ to $\mathcal{M}_{[K]}$ (orange arrow).

3.4 Compositional Generalization in OOEs

In OOEs, at least two types of generalization exist: (1) generalizing to *unseen objects*, and (2) generalizing to *unseen combinations* (scenes) of *known objects*. We consider type (2) as *compositional generalization*. This is similar to recent work on compositional generalization in natural language [Gordon et al., 2019], where a sentence is viewed as an ordered set of words. In the context of Object Library, the object library $\mathbb{L}$ is our "vocabulary", and selecting novel sets of objects $\mathbb{O}$ to form scenes is similar to composing unseen sentences from novel combinations of words.

We first illustrate this key idea using a simple instance of Object Library. Next, we formally define our notion of compositional generalization. Finally, we provide a set of conditions that we prove are sufficient to achieve compositional generalization in the Object Library family of OOEs.

3.4.1 An illustrative example

We use an instance of Object Library, Rush Hour, to illustrate our ideas (see Figures 3.1 and 3.2). Rush Hour is motivated by a grid game, where multiple cars are in different absolute orientations: `(north, south, east, west)`. Each car has an action space `(forward, backward, left, right)` that moves it relative to its orientation. For example, if a car is facing `south`, moving `right` means moving `west` in the world. Consider an instance with $N = 4$ possible objects in the library $\mathbb{L}$ described by shape and orientation: {▲, ▼, ◀, ▶}. Each scene consists of $K = 2$ objects, so there are 6 possible scenes: { {▲, ▼}, {▲, ◀}, {▲, ▶}, {▼, ◀}, {▼, ▶}, {◀, ▶} }.

Object-replacement symmetry. We formalize the notion of mapping between scenes (with different object combinations) using permutation symmetry. The symmetry group $\Sigma_N = \Sigma_4$ acts naturally on $\mathbb{L}$ with 4 cars, by replacing a car with another; it induces an operation acting on scenes. For example, if we have a permutation mapping $\sigma(▼) = ▶, \sigma \in \Sigma_4$, and other cars remain the same, the induced operation on the scene would be $\sigma(\{◀, ▼\}) = \{◀, ▶\}$. If our learned transition model is equivariant to Σ_N, then we can generalize to novel scenes (e.g., Figure 3.2 bottom right) by appropriate permuting (replacing objects) from training scenes (Figure 3.2 top right).

Lifting from the slot MDP. We can observe that scenes with K different present objects are structurally similar to each other, motivating us to not consider all N objects independently, but to share knowledge between scenes. Furthermore, since all

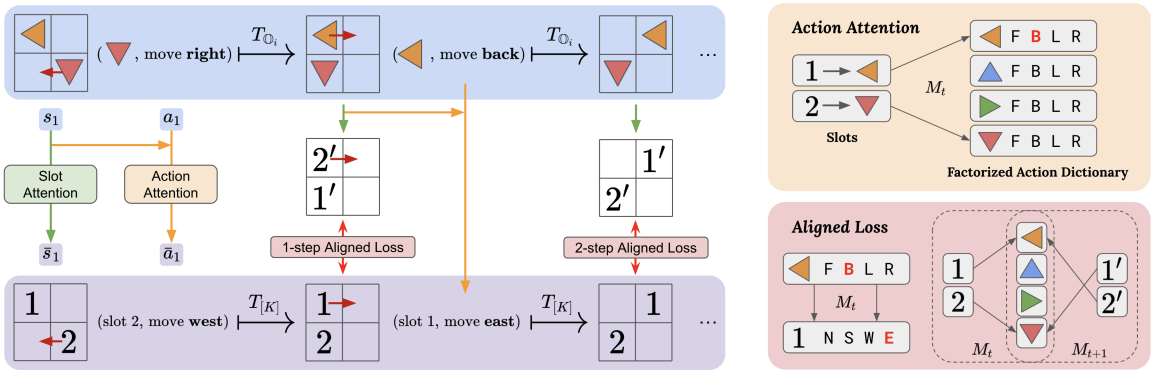

Figure 3.3: **(Left)** Overview of the world model prediction: the upper blue sequence is a ground pixel MDP with some objects $\mathbb{O}_i$, and the lower one is the slot MDP. We emphasize two facts: (1) encoded object slots in different steps may have different ordering (marked as $1'$ and $2'$), and (2) the transition model is equivariant in slot ordering, i.e., consistent across time steps (in 1 and 2), thus the loss computation needs alignment of slots (between $1, 2$ and $1', 2'$). **(Top right)** Action Attention learns to bind actions from interaction (action-object correspondence is *unknown* and learned). **(Bottom right)** In the Aligned Loss, the learned binding matrices M_t and M_{t+1} are used to lift slots in t and $t+1$ to a canonical space (full MDP).

K-object scenes are similar, we could find a *representative* "scene" where all scenes are similar to it, and convert actions to consistent meaning (from relative `up` to absolute `north`, Figure 3.2 bottom left). We call it the *slot MDP* $\mathcal{M}_{[K]}$ (Figure 3.2 bottom left), where all scene MDPs are called *isomorphic* to it. In other words, by learning a world model in the slot MDP, we should automatically get good model for all scenes, or "lifting", and thus generalize.

Measuring compositional generalization. If we train on scene MDPs $\{\mathcal{M}_{\{\blacktriangle,\blacktriangledown\}}, \mathcal{M}_{\{\blacktriangleleft,\blacktriangleright\}}\}$, the model should ideally generalize to $\{\mathcal{M}_{\{\blacktriangle,\blacktriangleleft\}}, \mathcal{M}_{\{\blacktriangledown,\blacktriangleright\}}\}$. By training on the former and evaluating transition-function prediction errors on the latter, we can measure generalization error. We formalize this by defining the notion of *equivariance error* in the next section. This measurement approach is detailed in the original paper.

3.4.2 Defining exact compositional generalization

Our definition is based on equivariance to object replacement in OOEs. Specifically, if the (learned) transition model $\hat{T}_{\mathbb{L}}$ for $\mathcal{M}_{\mathbb{L}}$ is equivariant to any object-set replacement, then $\hat{T}_{\mathbb{L}}$ can generalize to all object sets $\mathbb{O}$, including unseen combinations, as long as all individual objects in the library $\mathbb{L}$ have been observed in training scenes. This equivariance definition connects the behavioral perspective (how a system with compositional generalization should behave) and functional perspective (achieving by permutation equivariant networks). More formally, we define the object-replacement operation as follows:

Definition 3.1 (Object-replacement operation $p_{\mathbb{L}}$)**.** Let Σ_N denote the permutations of N elements, which acts naturally on the object-library set $\mathbb{L}$. The object-replacement operation is the group operation $\rho_{\mathbb{L}} \colon \Sigma_N \times \mathbb{L} \to \mathbb{L}$ where Σ_N acts on the indices in $\mathbb{L}$ to replace the identity of objects.

For example, using disjoint cycle notation, an object replacement operation $\pi = (123)(45)$ acts on $\mathbb{L}$ by sending $\pi(3) = 1$ and $\pi(4) = 5$. This induces a group operation on scenes, p_K, that maps a set of objects to another set, e.g., $p_K(\pi, \{3, 4\}) = \{1, 5\}$. Thus, Σ_N permutes the scene MDPs $\{\mathcal{M}_{\mathbb{O}}\}$.% TODOIL REMOVED We use $p_{\mathbb{L}}$ to define the measure of compositional generalization:

Definition 3.2 (Equivariance error of $\hat{T}_{\mathbb{L}}$ on $\mathcal{M}_{\mathbb{L}}$)**.** Let $\hat{T}_{\mathbb{L}} : \mathcal{S}_{\mathbb{L}} \times \mathcal{A}_{\mathbb{L}} \times \mathcal{S}_{\mathbb{L}} \to \mathbb{R}_+$ be a (learned) transition model of $\mathcal{M}_{\mathbb{L}}$. The sample equivariance error of $T_{\mathbb{L}}$ at $(s, a, s') \in \mathcal{S}_{\mathbb{L}} \times \mathcal{A}_{\mathbb{L}} \times \mathcal{S}_{\mathbb{L}}$ with respect to $\sigma \in \Sigma_N$ is defined

$$\lambda^{\sigma}_{\mathbb{L}} \triangleq \left| \hat{T}_{\mathbb{L}}(s' \mid s, a) - \hat{T}_{\mathbb{L}}(\sigma.s' \mid \sigma.s, \sigma.a) \right| . \tag{3.2}$$

The equivariance error of $T_{\mathbb{L}}$ is then defined as the expectation $\lambda_{\mathbb{L}} = \mathbb{E}_{s,a,s',\sigma}[\lambda^{\sigma}_{\mathbb{L}}]$.

The magnitude of λ measures the failure of $\hat{T}_{\mathbb{L}}$ to be Σ_N-equivariant, hence the name *equivariance error*. For a perfectly Σ_N-equivariant transition function $T_{\mathbb{L}}$, the error is guaranteed to be 0 by the definition of equivariance. Note that for clarity, we only consider *transition* prediction in the above definition. A complete version of homomorphism also preserves the reward structure, thus preserving optimal values and policies.

3.4.3 Achieving soft compositional generalization

Ideally, we can achieve compositional generalization according to the above metric by simply learning a Σ_N-equivariant transition model and correct binding of objects *and* actions. However, this is difficult in practice for large N (size of object library), as we show in our experiments. In this section, we derive an alternative path to Σ_N-equivariance that only requires us to learn a Σ_K-equivariant model, which is significantly more tractable since we expect $K \ll N$.

Instead of directly learning a Σ_N-equivariant model, which is difficult for large N, we can instead learn a Σ_K-equivariant model for $\mathcal{M}_{[K]}$, and "lift" it to achieve Σ_N-equivariance. We need to first define what is "good" binding, which intuitively means that we can project N objects to a subspace of K slots, while we are still able to identify them, i.e., how K-slot identities are permuted.

Definition 3.3 (Projection property)**.** Let h be an MDP homomorphism from the full MDP to the slot MDP, $h : \mathcal{M}_{\mathbb{L}} \to \mathcal{M}_{[K]}$. The homomorphism $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\} \rangle$ satisfies the projection property if, for any $\sigma \in \Sigma_N$, $s \in \mathcal{S}_{\mathbb{L}}$, $a \in \mathcal{A}_{\mathbb{L}}$, there exists $\overline{\sigma} \in \Sigma_K$ such that

$$\overline{\sigma}.\phi(s) = \phi(\sigma.s) \quad \text{and} \quad \overline{\sigma}.\alpha_s(a) = \alpha_{\sigma.s}(\sigma.a) \ . \tag{3.3}$$

In other words, the K present objects in $s \in \mathcal{S}_{\mathbb{L}}$ are *bound* to the K slots in $\phi(s) \in \mathcal{S}_{[K]}$ in a *specific* order. It implicitly assumes the binding of present objects in $\mathcal{M}_{\mathbb{L}}$ to slots in $\mathcal{M}_{[K]}$ is in some *temporally consistent* order: $\mathcal{S}_{i_k} \mapsto \mathcal{S}_k$.

By definition of MDP homomorphisms, given a model $\hat{T}_{\mathbb{L}}$ for $\mathcal{M}_{\mathbb{L}}$, h induces a model $\hat{T}_{[K]}$ for $\mathcal{M}_{[K]}$. If $\hat{T}_{\mathbb{L}}$ has equivariance error $\lambda_{\mathbb{L}}$, we can show that $\hat{T}_{[K]}$ will have the following equivariance error [2]:

Proposition 3.4. *Let $h : \mathcal{M}_{\mathbb{L}} \to \mathcal{M}_{[K]}$ be an MDP homomorphism satisfying the projection property. Suppose $\hat{T}_{\mathbb{L}}$ is a (learned) transition model of $\mathcal{M}_{\mathbb{L}}$ with equivariance error $\lambda_{\mathbb{L}}$. Then $\hat{T}_{[K]}$, the induced transition model of $\mathcal{M}_{[K]}$ under $h = \langle \phi, \{\alpha_s \mid s \in \mathcal{S}\}\rangle$, has sample equivariance error at $(\phi(s), \alpha_s(a), \phi(s')) \in \mathcal{S}_{[K]} \times \mathcal{A}_{[K]} \times \mathcal{S}_{[K]}$ and $\overline{\sigma} \in \Sigma_K$:*

$$\lambda^{\overline{\sigma}}_{[K]} \triangleq \Big[\Big|\hat{T}_{[K]}(\phi(s') \mid \phi(s), \alpha_s(a)) - \hat{T}_{[K]}(\overline{\sigma}.\phi(s') \mid \overline{\sigma}.\phi(s), \overline{\sigma}.\alpha_s(a))\Big|\Big] = C \cdot \lambda^{\sigma}_{\mathbb{L}}\,, \tag{3.4}$$

where $C = \binom{N}{K}$ is the number of K-slot scenes given an N-object library, $\phi : \mathcal{S}_{\mathbb{L}} \to \mathcal{S}_{[K]}$ and $\alpha_s : \mathcal{A}_{\mathbb{L}} \to \mathcal{A}_{[K]}$. The equivariance error is then $\lambda_{[K]} = \mathbb{E}_{s,a,s',\overline{\sigma}}[\lambda^{\overline{\sigma}}_{[K]}] = C \cdot \lambda_{\mathbb{L}}$.

Therefore, if $\hat{T}_{\mathbb{L}}$ has perfect compositional generalization (equivariance error $\lambda_{\mathbb{L}} = 0$), and homomorphism h satisfies the projection property, then the induced model $\hat{T}_{[K]}$ is Σ_K-equivariant.Conversely, since the proposition holds with equality, if we have a Σ_K-equivariant model $\hat{T}_{[K]}$ in $\mathcal{M}_{[K]}$, and $\mathcal{M}_{[K]}$ is a homomorphic image of $\mathcal{M}_{\mathbb{L}}$, then we can lift the model to a Σ_N-equivariant model $\hat{T}_{\mathbb{L}}$ in $\mathcal{M}_{\mathbb{L}}$, which allows us to simplify Σ_N-equivariant models.

Corollary 3.5 (Lifted model is Σ_N-equivariant)**.** *If (1) $\hat{T}_{[K]}$ is Σ_K-equivariant, and (2) there exists a homomorphism $h : \mathcal{M}_{\mathbb{L}} \to \mathcal{M}_{[K]}$ satisfying the projection property, then $\hat{T}_{\mathbb{L}}$ is Σ_N-equivariant.*

3.4.4 Practically measuring compositional generalization

Even though we formally define the compositional generalization error as an expectation $\mathbb{E}_{s,a,s',\overline{\sigma}}[\lambda^{\overline{\sigma}}_{[K]}]$ over all transition tuples and all permutations, directly computing it in *learned latent space* is not practical. The reason is that we only assume there exists correct object and action binding ϕ, α_s (by the projection property), while they are *learned* by models and *not* necessarily correct.

In the definition, several sources of error exist: (1) *prediction error* $(s, a) \to s'$, (2) *binding error* $(\phi(s), \alpha_s(a)) \to \phi(s')$, (3) *compositional error* $(\sigma.\phi(s), \sigma.\alpha_s(a)) \to \sigma.\phi(s')$. Alternatively, we need a practical metric to bypass (2) and measure (1)+(3). To this end, motivated by generating compositional natural language data [Keysers et al., 2020], we propose to train and test on disjoint set of scenes, and directly measure

[2] However, from the computational perspective, the information of binding to latent space is unknown, so the numerical values are hard to compute in practice.

the prediction error on test scenes, which avoids explicitly "replacing" object in latent space $\sigma.\phi(s)$.

Specifically, we need to follow two rules analogously: (1) *Disjoint scene object set.* Training and test set of object scenes should be *disjoint*: $|\mathbb{O}| = K$, $\{\mathbb{O}_{\text{train}}\} \cap \{\mathbb{O}_{\text{test}}\} = \emptyset$. (2) *Similar object distribution:* $i_n \in \mathbb{L}$. Training and test datasets should have similar object frequency (uniform), to ensure the error is not biased by errors in object detection. The training set should also contain *all* objects in library $\bigcup \mathbb{O}_{\text{train}} = \mathbb{L}$. In the experiment section, we then use this strategy to collect data and measure prediction error on test set as a proxy of compositional generalization error.

3.5 Compositionally Generalizable WMs

In the previous section, we provided the mathematical foundations for achieving compositional generalization via Σ_N- and Σ_K-equivariant transition models in $\mathcal{M}_{\mathbb{L}}$ and $\mathcal{M}_{[K]}$ respectively. Now, we provide a practical approach, based on the construction in Section 3.4.3, for achieving soft Σ_N-equivariance in only Σ_K latent space. The overview of the model is provided in Figure 3.3.

We focus on *end-to-end* learning a world model in latent space. The key challenge is that the object binding is unknown and no canonical ordering can be determined purely from images, so the model needs to infer it from data. We assume the action space is factorized by objects [Guestrin et al., 2003, Kipf et al., 2019], where each action component controls a specific object.While the factorization $\mathcal{A}_{\mathbb{L}} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$ is fixed, the model must use interaction data (s, a, s') to *infer* which action controls which object.

Stage 1: Object Extraction. Object extraction, or object binding Greff et al. [2020], learns object-structured representations, where each present object is represented by a latent vector $\mathbb{R}^D$, or *slot*. All slots form a latent state s_t, where no canonical order exists and the order differs in different time steps. This stage implicitly requires that *all* objects in $\mathbb{L}$ must have been observed during *training*; at generalization time we assume it can provide representations for any individual object. We use Slot Attention [Locatello et al., 2020] trained with reconstruction loss.

Stage 2: Action Binding. In learning the transition model $T(s, a) = s'$, we need to correctly align factorized actions a and object slots s, i.e., understand which action is controlling the object in each slot. This corresponds to the projection property required by the homomorphism $h : \mathcal{M}_{\mathbb{L}} \to \mathcal{M}_{[K]}$. We design an attention module, named **Action Attention**, that learns this purely from interaction (s, a, s'), as shown in Figure 3.3 (top right).

The attention matrix $M \in \mathbb{R}^{K \times N}$ is a (jointly learned) binding matrix from slots[3] s_t (at each time step) to some object identity, represented as an identity matrix $\text{Id}_{N \times N}$ in our case. This implements the *state-dependent* action transformation $\alpha_s(a)$, by *binding* K *actions* corresponding to the object slots: $\alpha_{s_t}(a_t) = M_t(s_t)a_t = \bar{a}_t$, and M_t is:

$$M_t(s_t) = \underset{K}{\text{softmax}}\left(\frac{1}{\sqrt{D}} k\left(\text{Id}_{N \times N}\right) \cdot q\left(s_t\right)^T\right) , \tag{3.5}$$

where k, q are linear projections mapping to some dimension D, and a_t acts as the value v without further transformation. Intuitively, the object identity specifies a "name" to factorized actions and can be viewed as a preference for some canonical ordering of objects in the full MDP $\mathcal{M}_{\mathbb{L}}$.

Stage 3: Σ_K-equivariant Transition Modeling. Proposition 3.4 implies that we only need a Σ_K-equivariant transition function for modeling the slot MDP (instead of Σ_N for the full MDP), hence we use a message-passing GNN [Kipf et al., 2019]. Crucially, we only require $K \ll N$ nodes, and since this leads to message-passing complexity of $O(K^2)$ instead of $O(N^2)$, we expect our model to be significantly more efficient. We also assume that interactions between any objects (slots) can be modeled by a one-edge network. This can be viewed as an OO-MDP [Diuk et al., 2008] with only one object class.

Training: Aligned Loss. The binding matrix $M_t \in \mathbb{R}^{K \times N}$ in Action Binding relates slots at time t to specific object identities. Thus, we use it to *align* slots s_t and s_{t+1} to the canonical full MDP space as $s_t^{\uparrow}$ and $s_{t+1}^{\uparrow}$ (Figure 3.3 (bottom right)) using the pseudoinverse: $s_t^{\uparrow} = M_t^{+}\bar{s}_t$, $s_{t+1}^{\uparrow} = M_{t+1}^{+}\bar{s}_{t+1}$, where $M_t^{+} = (M_t M_t^{\top})^{-1} M_t^{\top} \in \mathbb{R}^{N \times K}$, and $M_t^{+} T(\bar{s}_t, \bar{a}_t) \approx M_{t+1}^{+}\bar{s}_{t+1}$. We *jointly* train the model with the *aligned* loss between K slots, using the object-structured contrastive loss [Kipf et al., 2019]; the positive part is:

$$\mathcal{L}^{+}(s_t^{\uparrow}, s_{t+1}^{\uparrow}) = \left\| \texttt{NG}(M_{t+1}^{+})\bar{s}_{t+1} - \texttt{NG}(M_t^{+})T(\bar{s}_t, \bar{a}_t) \right\|^2 , \tag{3.6}$$

where $\texttt{NG}(\cdot)$ is *no-gradient* operation to prevent the shortcut to change the binding matrix M_t outside Action Attention. This forms a differentiable loss and allows us to train the transition model in latent space, which we call **Homomorphic Object-oriented World Model** (HOWM).

Inference: Multi-step prediction in latent space. For multi-step prediction at inference for evaluation, illustrated in Figure 3.3, all slots need to be aligned with their corresponding actions with *learned* binding M_t, since their ordering is potentially different at all time steps. If aligning slots sequentially with learned M as $T(T(\bar{s}_t, \bar{a}_t), M_t M_{t+1}^{+}\bar{a}_{t+1}) \approx M_t M_{t+2}^{+}\bar{s}_{t+2}$ and compute difference, it would suffer from compounding error. Thus, we instead align *actions*: $T(s_t, M_{t+1}^{+} M_t a_t) = \hat{s}_{t+1}$, which gives preceding slots $\hat{s}_{t+1}$ (or telescoping for $t+k$) in the same order as s_t. Implementation details are discussed in the following sections.

We study how compositional generalization is achieved in practice, by using Object Library and generalization metrics.

3.5.1 Experimental Setup

Environments. We designed two instances of the Object Library, ObjLib, environment. They are built upon the 2-D shape version of the Block Pushing environment Kipf et al. [2019]. (1) Basic **Shapes**. In the basic case, we equip the environment with a library of objects $\mathbb{L}$, where objects are different in shape, color, and size. The action

[3] In practice, we use $K+1$ slots to also model the background, as in Slot Attention [Locatello et al., 2020].

Table 3.1: Results for all methods on the Shapes environment with $K = 5$ and $N = 5, 10, 20, 30$ (four numbers in each cell). "OM" stands for out of GPU memory, where we limit the usage to 10GB. We report for memory usage on $N = 20$.

CG Type	**Env=Shapes**	**Eval MRR** (%, 1-step)	**Eval MRR** (%, 5-step)	**Train MRR** (%, 5-step)	**Gap** (MRR %, 5-step)	Memory
(1. Exact CG)	Σ_N-CSWM	100. 100. 99.9 *OM*	99.9 99.9 99.9 *OM*	100. 100. 100. *OM*	0.0 0.0 0.1 *OM*	8.1GB
(2. No guaranteed CG)	Σ_K-CSWM	100. 80.4 70.8 74.1	99.9 43.7 32.2 29.4	100. 100. 100. 100.	0.1 55.3 67.8 70.6	1.5GB
	Σ_K-CSWM(CA)	95.0 72.0 71.4 80.9	85.0 26.8 22.7 24.3	96.3 96.5 96.1 98.0	11.3 69.7 73.4 73.7	1.6GB
	C-WM(N)	83.6 81.4 25.8 12.9	61.8 55.0 11.2 7.6	88.0 96.6 41.5 33.9	26.2 41.6 30.3 26.2	1.3GB
	MONet(N)+BM	12.6 73.9 35.9 *OM*	2.0 20.2 55.9 *OM*	7.0 64.5 84.8 *OM*	5.0 44.3 29.0 *OM*	9.3GB
(3. Soft CG)	**HOWM (ours)**	99.2 98.5 99.7 99.7	92.3 84.2 75.1 81.8	97.0 96.9 98.0 98.1	4.7 12.7 22.9 16.3	3.7GB

Figure 3.4: Example transitions of Shapes (left pair) and Rush Hour environment (right pair).

space of each object is *identical*: `(north, south, east, west)`. (2) **Rush Hour**. To verify the importance of *action binding*, we set each object to have object-specific action space. This variant is motivated by the game *Rush Hour*, where each object is a car and has a (fixed) orientation (only using *triangles*). It can move relatively to its orientation: `(forward, backward, left, right)`. For example, if a car faces `east`, `right` will move `south`. For all variants, the action space is factorized by object and has *fixed* order across all scenes.

Methods. We study the compositional generalization (CG) performance of 6 approaches, under 3 categories:

1. Exact CG: Σ_N-equivariant methods with correct action binding should achieve perfect CG. One such method is **Σ_N-CSWM**, the model from Kipf et al. [2019] with N object masks, one for each library object. Object masks and actions are bound by having the same ordering.

2. No guaranteed CG: To demonstrate the necessity of the three stages in Section 3.5, we consider methods that either do not have action binding or a Σ_N-equivariant transition model. In **C-WM(N)**, we break Σ_N-equivariance by replacing the N-slot GNN and shared encoder with a flat MLP. **MONet(N)+BM** builds on the model from Burgess et al. [2019], which only extracts objects into masks (slots); we use bipartite matching to align slots s_t and s_{t+1}, but *not* to actions, i.e., there is no action binding. We also include two variants of CSWM that uses only K slots (instead of N), but without an explicit binding mechanism: **Σ_K-CSWM** assigns each of the K relevant action factors to slots (but the action may not actually control the object in the slot), whereas **Σ_K-CSWM(CA)** makes all action factors available to all slots.

3. Soft CG: Our method, **HOWM**, achieves soft CG using a K-slot approach by relying on the learned action binding (see Section 3.5).

Training and evaluation setup. We follow the setup in Kipf et al. [2019], using 1K episodes for training (consisting of 10 episodes of length 100, for 100 different scenes), and 10K episodes of length 10 for evaluation. Additionally, to evaluate compositional generalization, we ensure that (1) the combinations of objects in training dataset are different from those of evaluation dataset, and (2) the training data contains all N objects in the library.

Similar to Kipf et al. [2019], we measure the dynamics prediction error using two ranking metrics: Hits at Rank 1 (H@1) and Mean Reciprocal Rank (MRR) Kipf et al. [2019], averaged over 3 runs. For space, we only report MRR here.

3.5.2 Results and analysis

We compare all methods on the Basic Shapes environment in terms of their generalization performance and scalability, shown in Table 3.1. With sufficient resources, Σ_N-CSWM should be the *upper bound* of performance, since it can achieve perfect CG, as verified by our results (near-perfect eval MRR, near-zero gap). Both Σ_K-CSWM and Σ_K-CSWM(CA) also achieve excellent training performance since they can memorize the correct action binding in each scene during training; however, when presented with new scenes in evaluation, the actions may not be bound correctly, leading to worse eval MRR and a large gap. C-WM(N) lacks Σ_N-equivariance and is significantly worse during both training and evaluation, even with more parameters. MONet(N)+BM performs even worse, once again indicating the importance of action binding. Interestingly, it seems to perform better with larger N, but also uses much more resources due to its Σ_N-equivariant model; like Σ_N-CSWM, it exceeds our memory budget for $N = 30$.

HOWM strikes a reasonable middle ground – the training performance (train MRR) is near-perfect, and generalization performance (eval MRR) is still quite high, though clearly not perfect. However, in contrast to all other methods except the Σ_N-CSWM upper bound, generalization is maintained as N increases, and the gap is significantly smaller. These results demonstrate that our proposed approach is able to achieve good generalization for intermediate $N \leq 20$ while consuming significantly fewer resources, and can scale to $N \geq 30$, unlike the exact Σ_N-CSWM method.

One limitation we observed is that for long-term prediction (≥ 5 steps), it is still quite challenging to achieve action binding and compositional generalization for a large library of objects. One reason is that the learned representation module (Slot Attention in our case [Locatello et al., 2020]) does not guarantee perfect object representations across time. This affects both action binding learning (if extracted slots are inaccurate, actions cannot be bound correctly) and long-term evaluation (if objects are missed at some step, predictions for all following steps are likely wrong).

In the Rush Hour environment, shown in Table 3.2, the action-binding problem is more challenging and requires more sophisticated modeling of object-dependent transition dynamics. We report the evaluation MRR and the generalization gap. As before, Σ_N-CSWM acts as an upper bound and performs near-perfectly with near-zero gap; however, as before, we expect that it will not scale beyond $N = 20$ with our resource

Table 3.2: Results on Rush Hour with relative action space, for $N = 5, 10, 20$ (three numbers in each cell).

Env=Rush Hour	**MRR** (1-step)			**MRR** (5-step)			**Gap** (MRR, 5-step)		
Σ_N-CSWM	100.	100.	99.9	99.9	99.8	99.8	0.1	0.2	0.1
Σ_K-CSWM	100.	66.9	47.6	99.9	29.5	13.7	0.1	66.0	85.0
Σ_K-CSWM(CA)	99.2	55.4	80.2	94.9	15.5	15.8	4.2	84.5	84.2
C-WM(N)	55.5	67.8	80.6	23.7	24.5	45.3	48.8	70.7	54.1
HOWM (ours)	96.7	94.3	98.7	84.3	63.2	65.3	11.2	31.1	31.3

budget. All other methods, including HOWM, have worse performance compared to Shapes. However, HOWM still generalizes significantly better to new scenes for $N = 10$ and $N = 20$, whereas performance of other methods declines more steeply.

3.5.3 Visualization

We visualize a model on a sampled transition (s_t, a_t, s_{t+1}) with $N = 10$ and $K = 5$. Action attention matrices are computed using the states: $M_t(s_t), M_{t+1}(s_{t+1}) \in \mathbb{R}^{N\times(K+1)}$. The visualized M align with expectation: (1) K present objects are bound to K slots. (2) the absent $N - K$ objects are randomly assigned to slots (reasonable since not forced by loss), but they are consistent in t and $t + 1$ (as forced by loss to keep N-object $s_t^{\uparrow}, s_{t+1}^{\uparrow}$ embeddings close). Also, the lifted embeddings $M_t^{+}\bar{s}_t, M_{t+1}^{+}\bar{s}_{t+1}$ are well aligned and have little difference. See the complete paper version for detailed visualizations.

3.5.4 Comprehensive Experimental Evaluation

Experimental Setup

We design two instances of the Object Library environment to systematically evaluate compositional generalization:

(1) Shapes Environment: The basic case equips the environment with a library of objects differing in shape, color, and size. Each object has an identical action space: `(north, south, east, west)`. This tests compositional generalization with uniform action semantics.

(2) Rush Hour Environment: To verify the importance of action binding, each object has an object-specific action space. Motivated by the Rush Hour game, each object is a car with fixed orientation that can move relative to its orientation: `(forward, backward, left, right)`. For example, if a car faces east, `right` moves south. This tests compositional generalization with heterogeneous action semantics.

Baseline Comparisons

We evaluate compositional generalization performance across three categories of methods:

Table 3.3: Compositional generalization results on Shapes environment. HOWM achieves strong performance with efficient memory usage.

Method	1-step MRR (%)	5-step MRR (%)
Σ_N-CSWM (Exact)	100.0	99.9
Σ_K-CSWM	74.1	29.4
C-WM(N)	12.9	7.6
HOWM (Ours)	**99.7**	**81.8**

Exact CG: Σ_N-equivariant methods with correct action binding should achieve perfect compositional generalization. We compare against Σ_N-CSWM, which uses N object masks with actions bound by ordering.

No Guaranteed CG: Methods that lack either action binding or Σ_N-equivariant transition models:

- **C-WM(N)**: Breaks Σ_N-equivariance by replacing the N-slot GNN with a flat MLP
- **MONet(N)+BM**: Uses object extraction into slots but no action binding, relying on bipartite matching for slot alignment
- **Σ_K-CSWM variants**: Use only K slots without explicit binding mechanisms

Soft CG: Our HOWM method achieves soft compositional generalization using learned action binding with K slots.

Results

Table 3.3 shows results on the Shapes environment with $K = 5$ and varying $N \in \{5, 10, 20, 30\}$. HOWM demonstrates strong compositional generalization, achieving 99.7% accuracy on 1-step prediction and 81.8% on 5-step prediction for $N = 30$, while maintaining efficient 3.7GB memory usage compared to the 8.1GB required by exact methods that run out of memory for large N.

The results demonstrate that HOWM achieves nearly perfect 1-step prediction accuracy and maintains strong multi-step prediction performance while being significantly more memory-efficient than exact methods. The learned action binding mechanism successfully enables compositional generalization with $K \ll N$, confirming our theoretical predictions.

On the Rush Hour environment, HOWM's action binding mechanism proves crucial for handling object-specific action spaces, further validating the importance of correctly associating actions with objects for compositional generalization.

Object-oriented representations are crucial in artificial intelligence and robotics [Diuk et al., 2008, Wong, 2016]; our framework is related to OO-MDPs [Diuk et al., 2008] consisting of a single class. Recently, a line of works studies learning to *discover*

objects and their representations end-to-end. MONet [Burgess et al., 2019] applies sequential attention with VAEs to learn factorized representations. GSWM [Lin et al., 2020] builds generative models and learns end-to-end with variational inference. Slot Attention [Locatello et al., 2020] proposes attention at the pixel level, which can be viewed as soft clustering of pixels of objects. The order of object slots is decided by randomly initialized cluster centroids.

A object-oriented world model can be learned jointly or separately with object representation, using pixel reconstruction (COBRA [Watters et al., 2019], OAT [Creswell et al., 2021]), variational inference (STOVE [Kossen et al., 2019], OP3 [Veerapaneni et al., 2020]), or contrastive loss (C-SWM [Kipf et al., 2019], NPS [Didolkar et al., 2021]). Furthermore, in jointly learning object representations and world models, a key problem is *object binding* [Greff et al., 2020]: binding object *slots* between time steps, which is typically solved by bipartite matching. In our setup, we emphasize the importance of correctly *binding actions* to objects. In video prediction such as [Kipf et al., 2021], action is not considered and thus has no action binding issue, which is our key focus and brings the major challenge. Biza et al. [2022] use location-based action space and learn attention to factorize monolithic actions. It keeps the setup in CSWM with $N = K$ and does not do compositional generalization in our $N > K$ formalism.

Compositional generalization has been widely studied in deep language models [Johnson et al., 2017]. It also known as *systematic generalization* [Bahdanau et al., 2019], and *systematicity* [Lake and Baroni, 2018]. Similar to us, Gordon et al. [2019] also use permutation equivariance to model compositional generalization in supervised learning on language data. Keysers et al. [2020] propose principles of measuring compositional generalization on sentences. A similar concept, *compositionality*, is usually referred to in disentangled representation learning. However, it typically focuses on disentanglement between individual attributes of distributed representations [Higgins et al., 2018, Caselles-Dupré et al., 2019], while we are more interested in factorization between (attributes of) objects. There is also work on measuring compositionality [Andreas, 2019, Chaabouni et al., 2020]. Beyond language, Locatello et al. [2020], Veerapaneni et al. [2020] learn object-oriented representations, whose compositional generalization usually refers to factorization by objects. Furthermore, no prior work considers the difference between present objects K and its "library" N objects, i.e., they can only use Σ_N-equivariant model and thus struggle in scaling up.

Symmetries and MDP homomorphisms. Symmetry widely exists in various real-world data [Bronstein et al., 2021a], and also in MDPs [Ravindran and Barto, 2004]. A line of works in equivariant networks studies equivariance properties of existing network architectures, such as permutation equivariance in graph neural networks (GNNs) [Keriven and Peyré, 2019], while some other works study equivariant constraints to other symmetry groups [Bronstein et al., 2021a]. In RL, symmetries in MDPs can be modeled by MDP homomorphisms [Ravindran and Barto, 2004], which have nice properties such as *optimal value equivalence* and can be exploited in policy learning [van der Pol et al., 2020a].

3.6 Chapter Discussion

This chapter introduced object-centric representations as a foundation for compositional planning. The Homomorphic Object-oriented World Model (HOWM) addresses the binding problem—determining which objects are affected by which actions—through learned attention mechanisms. By factorizing scenes into slots and learning equivariant transitions, the approach enables generalization to novel object combinations without retraining. The computational scaling from library size to scene size ($\mathcal{O}(N^2) \to \mathcal{O}(K^2)$) demonstrates how structured representations can make combinatorial problems tractable. This principle—that abstracting to the right level enables systematic generalization—recurs throughout the thesis in different forms.

Chapter 4

Symmetry as Structure for Planning

This thesis demonstrates how different forms of compositional structure enable systematic generalization. While Chapter 3 examined object permutations, this chapter shows that *geometric transformations provide compositional structure for spatial planning*. The insight: when environments exhibit rotational or reflectional symmetry, solutions are related by the same geometric transformations. A path through a rotated room is simply the rotated path through the original room. Exploiting this symmetry reduces planning complexity—we effectively search a space reduced by the symmetry group size (e.g., 8× reduction for D(4) symmetry).

Model-based planning algorithms can struggle with complex problems, but applying planning in a structured and reduced space offers a solution [Sutton and Barto, 2018, Li et al., 2006a, Ravindran and Barto, 2004]. When tasks exhibit symmetry, this structure can effectively reduce the search space. However, existing planning algorithms often assume perfect dynamics knowledge and require building equivalence classes, limiting their applicability [Fox and Long, 1999, 2002, Pochter et al., 2011]. Classical algorithms like A* require searching symmetric states (NP-hard) with known dynamics [Pochter et al., 2011, Narayanamurthy and Ravindran, 2008].

Consider path planning on a 2D grid map M. If we rotate the map by 90° to get $g \cdot M$, the optimal solution also rotates by 90°. This means $\texttt{SymPlan}(g \cdot M) = g \cdot \texttt{SymPlan}(M)$ for rotation $g \in D_4$ (the dihedral group of rotations and reflections). Concretely, the action in the northwest corner of the original solution equals the action in the southwest corner of the rotated solution, after rotating the action direction by 90°. This symmetry can be observed *before* solving the task, without requiring knowledge of optimal policies. Using rotation and reflection symmetry (D(4) group) reduces the search space by a factor of 8.

Recent work has studied symmetry in model-free deep RL [Mondal et al., 2020, van der Pol et al., 2020b, Wang et al., 2021]. However, these approaches lack long-horizon planning ability and only handle pixel-level symmetry (flipping or rotating observations and actions together). This motivates combining both approaches: *can we enable end-to-end differentiable planning algorithms to exploit environment symmetry?*

We propose Symmetric Planning (SymPlan), which enables planning with symmetry in an end-to-end differentiable manner while avoiding explicit construction of equivalence classes. Our framework draws inspiration from equivariant networks and geometric deep learning [Bronstein et al., 2021a, Cohen et al., 2020, Kondor and Trivedi, 2018], which view geometric data as signals over a base space. For instance, an RGB image is a signal mapping $\mathbb{Z}^2 \to \mathbb{R}^3$. Equivariant networks inject symmetry through equivariant operations like convolutions, ensuring symmetry properties without explicitly considering "symmetric pixels"—avoiding the need to search symmetric states.

Our technical contribution introduces Symmetric Value Iteration Networks (SymVIN). **We prove that value iteration for path planning is a type of steerable convolution**—enabling us to implement SymVIN by replacing standard 2D convolutions with steerable convolutions [Cohen and Welling, 2016b]. By embedding D(4) symmetry constraints directly into the network architecture, SymVIN guarantees that rotated/reflected inputs produce correspondingly transformed outputs. This architectural constraint provides geometric guarantees—navigation strategies work regardless of orientation—while significantly improving sample efficiency and generalization to previously unseen random maps.

This chapter is based on our paper "Integrating Symmetry into Differentiable Planning with Steerable Convolutions" published at ICLR 2023.

4.1 Background

Markov decision processes. We model the path-planning problem as a Markov decision process (MDP) [Sutton and Barto, 2018]. An MDP is a 5-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability function $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$, reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1]$. Value functions $V : \mathcal{S} \to \mathbb{R}$ and $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ represent expected future returns. The core component behind dynamic programming (DP)-based algorithms in reinforcement learning is the *Bellman (optimality) equation* [Sutton and Barto, 2018]: $V^*(s) = \max_a R(s, a) + \gamma \sum_{s'} P(s'|s, a)V^*(s')$. Value iteration is an instance of DP to solve MDPs, which iteratively applies the Bellman (optimality) operator until convergence.

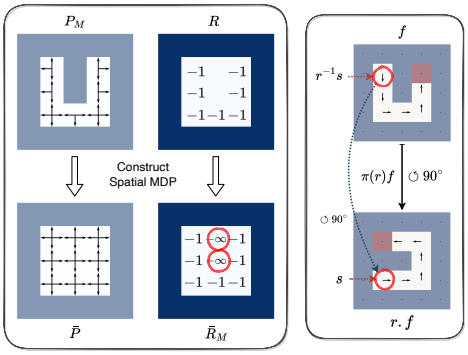

Figure 4.1: **(Left)** Construction of spatial MDPs from path-planning problems, enabling the use of G-invariant transition models. **(Right)** A demonstration of how an action (arrow in red circle) is rotated when a map is rotated.

Path planning. The objective of the path-planning problem is to find optimal actions from every location to navigate to the target in shortest time. However, the original path-planning problem is *not equivariant* under *translation* due to obstacles. VINs [Tamar et al., 2016b] implicitly construct an equivalent problem with an *equivariant transition function*, thus CNNs can be used

to inject translation equivariance. We visualize the construction of an equivalent "spatial MDP" in Figure 4.1 (left), where the key idea is to encode obstacle information in the *transition function* from the map (top left) into the *reward function* in the constructed spatial MDP (bottom right) as "trap" states with $-\infty$ reward. Further details about this construction are in Zhao et al. [2023c]. In Figure 4.1 (right), we provide a visualization of the *representation* $\pi(r)$ of a rotation r of $\circlearrowleft$ 90°, and how an action (arrow) is rotated $\circlearrowleft$ 90° accordingly.

Value Iteration Network. Tamar et al. [2016b] proposed Value Iteration Networks (VINs) that use a convolutional network to parameterize value iteration. It jointly learns in a latent MDP on a 2D grid, which has the latent reward function $\bar{R} : \mathbb{Z}^2 \to \mathbb{R}^{|\mathcal{A}|}$ and value function $\bar{V} : \mathbb{Z}^2 \to \mathbb{R}$, and applies value iteration on that MDP:

$$\bar{Q}^{(k)}_{\bar{a},i',j'} = \bar{R}_{\bar{a},i',j'} + \sum_{i,j} W^V_{\bar{a},i,j} \bar{V}^{(k-1)}_{i'-i,j'-j} \qquad \bar{V}^{(k)}_{i,j} = \max_{\bar{a}} \bar{Q}^{(k)}_{\bar{a},i,j} \tag{4.1}$$

The first equation can be written as: $\bar{Q}^{(k)} = \bar{R}^a + \texttt{Conv2D}(\bar{V}^{(k-1)}; W^V_{\bar{a}})$, where the 2D convolution layer `Conv2D` has parameters W^V.

We intentionally omit the details on equivariant networks and instead focus on the core idea of integrating symmetry with equivariant networks. We present the necessary group theory background and our full framework and theory in Zhao et al. [2023c].

4.2 Method: Integrating Symmetry into Planning by Convolution

This section presents an algorithmic framework that can provably leverage the inherent symmetry of the path-planning problem in a *differentiable* manner. To make our approach more accessible, we first introduce Value Iteration Networks (VINs) [Tamar et al., 2016b] as the foundation for our algorithm: Symmetric VIN. In the next section, we provide an explanation for why we make this choice and introduce further theoretical guarantees on how to exploit symmetry.

How to inject symmetry? VIN uses regular 2D convolutions (Eq. 4.1), which has *translation equivariance* [Cohen and Welling, 2016a, Kondor and Trivedi, 2018]. More concretely, a VIN will output the same value function for the same map patches, up to 2D translation. Characterization of translation equivariance requires a different mechanism and does not *decrease* the search space nor *reduce* a path-planning MDP to an easier problem. We provide a complete description in Zhao et al. [2023c].

Beyond translation, we are more interested in *rotation* and *reflection* symmetries. Intuitively, if we find the optimal solution to a map, it automatically **generalizes** the solution to all 8 transformed maps (4 rotations times 2 reflections, including identity transformation). This can be characterized by *equivariance* of a planning algorithm `Plan`, such as value iteration `VI`: $g.\texttt{Plan}(M) = \texttt{Plan}(g.M)$, where M is a maze map, and g is the symmetry group D_4 under which 2D grids are invariant.

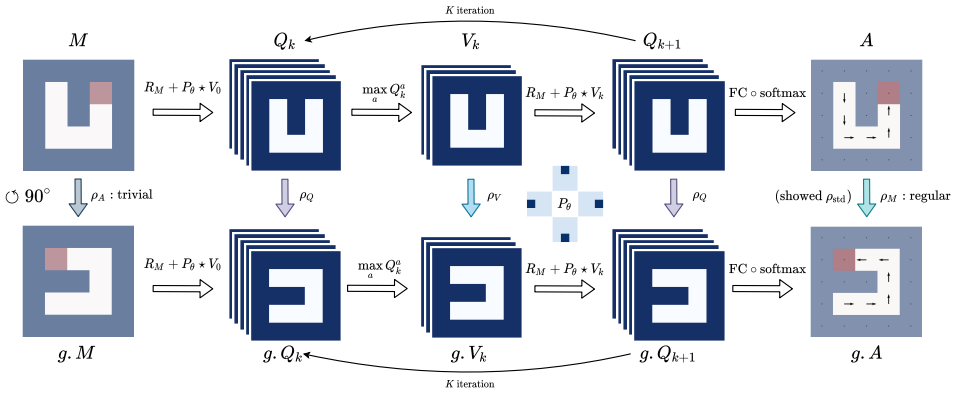

Figure 4.2: The commutative diagram of Symmetric Value Iteration Network (SymVIN). Every *row* is a full computation graph of VIN. Every *column* rotates the field by ↻ 90°.

More importantly, symmetry also helps **training** of differentiable planning. Intuitively, symmetry in path planning poses additional constraints on its search space: if the goal is in the north, go up; if in the east, go right. In other words, the knowledge can be shared between symmetric cases; the path-planning problem is effectively reduced by symmetry to a smaller problem. This property can also be depicted by equivariance of Bellman operators $\mathcal{T}$, or one step of value iteration: $g.\mathcal{T}[V_0] = \mathcal{T}[g.V_0]$. If we use $\texttt{VI}(M)$ to denote applying Bellman operators on arbitrary initialization until convergence $\mathcal{T}^\infty[V_0]$, value iteration is also equivariant:

$$g.\texttt{VI}(M) \equiv g.\mathcal{T}^\infty[V_0] = \mathcal{T}^\infty[g.V_0] \equiv \texttt{VI}(g.M) \tag{4.2}$$

We formally prove this equivariance in Theorem 4.1 in next section. In Theorem 4.2, we theoretically show that value iteration in path planning is a specific type of convolution: *steerable convolution* [Cohen and Welling, 2016b]. Before that, we first use this finding to present our pipeline on how to use *Steerable CNNs* [Cohen and Welling, 2016b] to integrate symmetry into path planning.

Pipeline: SymVIN. We have shown that VI is equivariant given symmetry in path planning. We introduce our method *Symmetric Value Iteration Network* (SymVIN), that realizes equivariant VI by integrating equivariance into VIN with respect to *rotation* and *reflection*, in addition to *translation*. We use an instance of Steerable CNN: $E(2)$-Steerable CNNs [Weiler and Cesa, 2021] and their package `e2cnn` for implementation, which is equivariant under D_4 rotation and reflection, and also $\mathbb{Z}^2$ translation on the 2D grid $\mathbb{Z}^2$. In practice, to inject symmetry into VIN, we mainly need to replace the translation-equivariant `Conv2D` in Eq. 4.1 with `SteerableConv`:

$$\bar{Q}_{\bar{a}}^{(k)} = \bar{R}_{\bar{a}} + \texttt{SteerableConv}(\bar{V}; W^V) \qquad \bar{V}^{(k)} = \max_{\bar{a}} \bar{Q}_{\bar{a}}^{(k)} \tag{4.3}$$

We visualize the full pipeline in Figure 4.2. The map and goal are represented as signals $M : \mathbb{Z}^2 \to \{0, 1\}^2$. It will be processed by another layer and output to the core value iteration loop. After some iterations, the final output will be used to predict the actions and compute cross-entropy loss.

Figure 4.2 highlights our injected equivariance property: if we *rotate* the map (from M to $g.M$), in order to guarantee that the final policy function will also be *equivalently*

rotated (from A to $g.A$), we shall guarantee that every *transformation* (e.g., $Q_k \mapsto V_k$ and $V_k \mapsto Q_{k+1}$) in value iteration will also be *equivariant*, for every *pair of columns*. We formally justify our design in the section below and provide more technical details in Zhao et al. [2023c].

Extension: Symmetric GPPN. Based on same spirit, we also implement a symmetric version of Gated Path-Planning Networks (GPPN [Lee et al., 2018]). GPPNs use LSTMs to alleviate the issue of unstable gradient in VINs. Although it does not strictly follow value iteration, it still follows the spirit of steerable planning. Thus, we first obtained a fully convolutional variant of GPPN, called ConvGPPN, which incorporates translational equivariance. It replaces the MLPs in the original LSTM cell with convolutional layers, and then replaces convolutions with equivariant steerable convolutions, resulting in SymGPPN that is equivariant to translations, rotations, and reflections. See Zhao et al. [2023c] for details.

4.3 Theory: Value Iteration is Steerable Convolution

In the last section, we showed how to exploit symmetry in path planning by equivariance from convolution via intuition. The goal of this section is to (1) connect the theoretical justification with the algorithmic design, and (2) provide intuition for the justification. Even through we focus on a specific task, we hope that the underlying guidelines on integrating symmetry into planning are useful for broader planning algorithms and tasks as well. The complete version is in Zhao et al. [2023c].

Overview. There are numerous types of symmetry in various planning tasks. We study symmetry in **path planning** as an example, because it is a straightforward planning problem, and its solutions have been intensively studied in robotics and artificial intelligence [LaValle, 2006b, Sutton and Barto, 2018]. However, even for this problem, symmetry has *not* been *effectively* exploited in existing planning algorithms, such as Dijkstra's algorithm, A*, or RRT, because it is NP-hard to find symmetric states [Narayanamurthy and Ravindran, 2008].

Why VIN-based planners? There are two reasons for choosing value-based planning methods.

1. The expected-value operator in value iteration $\sum_{s'} P(s'|s,a)V(s')$ is (1) *linear* in the value function and (2) *equivariant* (shown in Theorem 4.1). Cohen et al. [2020] showed that any *linear equivariant operator* (on homogeneous spaces, e.g., 2D grid) is a group-convolution operator.

2. Value iteration, using the Bellman (optimality) operator, consists of only maps between signals (steerable fields) over $\mathbb{Z}^2$ (e.g., value map and transition function map). This allows us to inject symmetry by enforcing equivariance to those maps. For example, the 4 corner states of a square map are symmetric under transformations in D_4. Equivariance enforces those 4 states to have the same value if we rotate or flip

the map. This avoids the need to find if a new state is symmetric to any existing state, which is shown to be NP-hard [Narayanamurthy and Ravindran, 2008].

Our framework for integrating symmetry applies to any value-based planner with the above properties. We found that VIN is conceptually the simplest differentiable planning algorithm that meets these criteria, hence our decision to focus primarily on VIN and its variants.

Symmetry *from* tasks. If we want to exploit inherent symmetry in a task to improve planning, there are two major steps: (1) characterize the symmetry in the task, and (2) incorporate the corresponding symmetry into the planning algorithm. The theoretical results in Zhao et al. [2023c] mainly characterize the symmetry and direct us to a feasible planning algorithm.

The *symmetry in tasks* for MDPs can be specified by the equivariance property of the transition and reward function, studied in Ravindran and Barto [2004], van der Pol et al. [2020a]:

$$\bar{P}(s' \mid s, a) = \bar{P}(g.s' \mid g.s, g.a), \quad \forall g \in G, \forall s, a, s' \tag{4.4}$$

$$\bar{R}_M(s, a) = \bar{R}_{g.M}(g.s, g.a), \quad \forall g \in G, \forall s, a \tag{4.5}$$

Note that how the group G *acts* on states and actions is called *group representation*, and is decided by the space $\mathcal{S}$ or $\mathcal{A}$, which is discussed in Zhao et al. [2023c]. We emphasize that the equivariance property of the reward function is different from prior work [Ravindran and Barto, 2004, van der Pol et al., 2020a]: in our case, the reward function encodes obstacles as well, and thus depends on map input M. Intuitively, if a position s is rotated to $g.s$, in order to find the correct original reward R before rotation, the input map M must also be rotated $g.M$. See Zhao et al. [2023c] for more details.

Symmetry *into* planning. As for exploiting the *symmetry in planning algorithms*, we focus on value iteration and the VIN algorithm. We first prove in Theorem 4.1 that value iteration for path planning respects the *equivariance* property, motivating us to incorporate symmetry with equivariance.

Theorem 4.1 (informal)**.** *If transition is G-invariant, expected-value operator $\sum_{s'} P(s'|s,a)V(s')$ and value iteration are equivariant under translation, rotation, reflection on the 2D grid.*

We visualize the equivariance of the central value-update step $R + \gamma P \star V_k$ in Figure 4.3. The upper row is a value field V_k and its rotated version $g.V_k$, and the lower row is for Q-value fields Q_k and $g.Q_k$. The diagram shows that, if we input a rotated value $g.V_k$, the output $R + \gamma P \star g.V_k$ is guaranteed to be equal to rotated Q-field $g.Q_k$. Additionally, rotating Q-field $g.Q_k$ has two components: (1) spatially rotating each grid (a feature channel for an action $Q(\cdot, a)$) and (2) cyclically permuting the channels (black arrows). The red dashed line points out how a specific grid of a Q-value grid $Q_k(\cdot, \text{South})$ got rotated and permuted. We discuss the theoretical guarantees in Theorem 4.1. For full proofs and detailed derivations, please refer to our ICLR 2023 publication [Zhao et al., 2023c].

However, while the first theorem provides intuition, it is inadequate since it only shows the equivariance property for *scalar-valued* transition probabilities and value functions, and does not address the implementation of VINs with *multiple feature channels* in CNNs. To address this gap, the next theorem further proves that value iteration is a general form of *steerable convolution*, motivating the use of steerable CNNs by Cohen and Welling [2016b] to replace regular CNNs in VIN. This is related to Cohen et al. [2020] that proves steerable convolution is the most general linear equivariant map on homogeneous spaces.

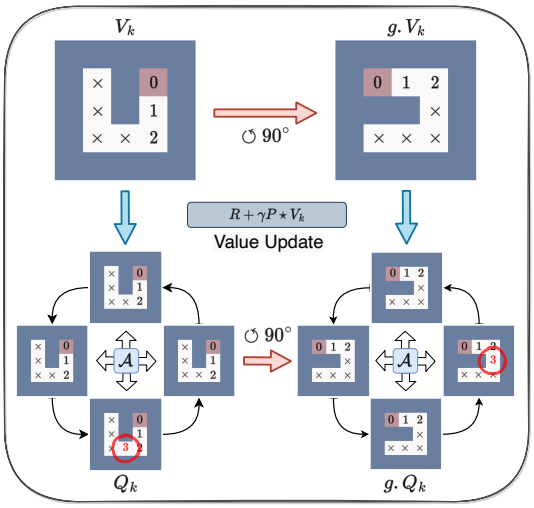

Figure 4.3: Commutative diagram of a single step of value update, showing equivariance under rotations. Each grid in the Q-value field corresponds to all the values $Q(\cdot, a)$ of a single action a.

Theorem 4.2 (informal)**.** *If transition is G-invariant, the expected-value operator is expressible as a steerable convolution $\star$, which is equivariant under translation, rotation, and reflection on 2D grid. Thus, value iteration (with* max, $+$, $\times$*) is a form of steerable CNN [Cohen and Welling, 2016b].*

We provide a complete version of the framework and the proofs in Zhao et al. [2023c]. This justifies why we should use Steerable CNN [Cohen and Welling, 2016b]: the VI itself is composed of steerable convolution and additional operations (max, $+$, $\times$). With equivariance, the value function $\mathbb{Z}^2 \to \mathbb{R}$ is D_4-invariant, thus the planning is effectively done in the quotient space reduced by D_4.

Summary. We study how to inject symmetry into VIN for (2D) path planning, and expect the task-specific technical details are useful for two types of readers. *(i) Using VIN.* If one uses VIN for differentiable planning, the resulting algorithms SymVIN or SymGPPN can be a plug-in alternative, as part of a larger end-to-end system. Our framework generalizes the idea behind VINs and enables us to understand its applicability and restrictions. *(ii) Studying path planning.* The proposed framework characterizes the symmetry in path planning, so it is possible to apply the underlying ideas to other domains. For example, it is possible to extend to higher-dimensional continuous Euclidean spaces or spatial graphs [Weiler et al., 2018, Brandstetter et al., 2021]. Additionally, we emphasize that the *symmetry in spatial MDPs* is different from *symmetric MDPs* [Zinkevich and Balch, 2001, Ravindran and Barto, 2004, van der Pol et al., 2020b], since our reward function is *not* G-invariant (if not conditioning on obstacles). We further discuss this in Zhao et al. [2023c].

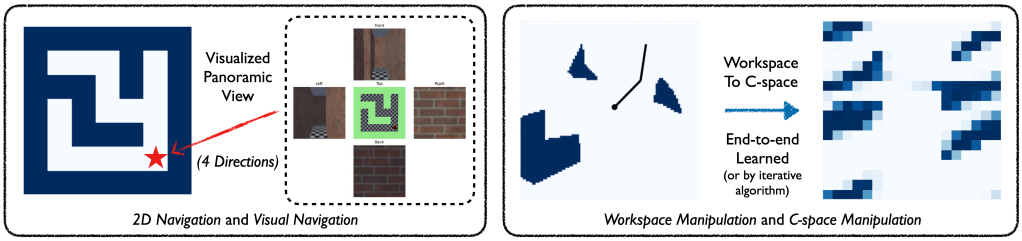

Figure 4.4: **(Left)** We randomly generate occupancy grid $\mathbb{Z}^2 \to \{0, 1\}$ for **2D navigation**. For **visual navigation**, each position provides $32 \times 32 \times 3$ egocentric panoramic RGB images in 4 directions for each location $\mathbb{Z}^2 \to \mathbb{R}^{4\times32\times32\times3}$. One location is visualized. **(Right)** The top-down view (left) is the *workspace* of a 2-DOF manipulation task. For **workspace manipulation**, it is converted by a mapper layer to configuration space, shown in the right subfigure. For **C-space manipulation**, the ground-truth C-space is provided to the planner.

4.4 Experiments

We experiment with VIN, GPPN and our SymPlan methods on four path-planning tasks, using both *given* and *learned* maps. Additional experiments and ablation studies are provided in our ICLR 2023 publication [Zhao et al., 2023c].

Environments and datasets. We demonstrate the idea in four path-planning tasks: (1) **2D navigation**, (2) **visual navigation**, (3) 2 degrees of freedom (2-DOF) **configuration-space (C-space) manipulation**, and (4) **2-DOF workspace manipulation**. For each task, we consider using either *given* (2D navigation and 2-DOF configuration-space manipulation) or *learned* maps (visual navigation and 2-DOF workspace manipulation). In the latter case, the planner needs to jointly learn a mapper that converts egocentric panoramic images (visual navigation) or workspace states (workspace manipulation) into a map that the planners can operate on, as in [Lee et al., 2018, Chaplot et al., 2021]. In both cases, we randomly generate training, validation and test data of $10K/2K/2K$ maps for all map sizes, to demonstrate data efficiency and generalization ability of symmetric planning. Due to the small dataset sizes, test maps are unlikely to be symmetric to the training maps by any transformation from the symmetry groups G. For all environments, the planning domain is the 2D regular grid as in VIN, GPPN and SPT $\mathcal{S} = \Omega = \mathbb{Z}^2$, and the action space is to move in 4 $\circlearrowleft$ directions[1]: $\mathcal{A} =$ `(north, west, south, east)`.

Methods: planner networks. We compare five planning methods, two of which are our SymPlan methods. Our two equivariant methods are based on *Value Iteration Networks* (**VIN**, [Tamar et al., 2016b]) and *Gated Path Planning Networks* (**GPPN**, [Lee et al., 2018]). Our equivariant version of VIN is named **SymVIN**. For GPPN, we first obtained a *fully convolutional* version, named **ConvGPPN**, and furthermore obtain **SymGPPN** by replacing standard convolutions with steerable CNNs. All methods use (equivariant) convolutions with *circular padding* to plan in configura-

[1]Note that the MDP action space $\mathcal{A}$ needs to be *compatible* with the group action $G \times \mathcal{A} \to \mathcal{A}$. Since the E2CNN package [Weiler and Cesa, 2021] uses *counterclockwise* rotations $\circlearrowleft$ as generators for rotation groups C_n, the action space needs to be *counterclockwise* $\circlearrowleft$.

Table 4.1: Averaged test success rate (%) for using 10K/2K/2K dataset for all four types of tasks.

Method	Navigation				Manipulation		
(10K Data)	15×15	28×28	50×50	Visual	18×18	36×36	Workspace
VIN	66.97	67.57	57.92	50.83	77.82	84.32	80.44
SymVIN	98.99	98.14	86.20	95.50	99.98	99.36	**91.10**
GPPN	96.36	95.77	91.84	93.13	2.62	1.68	3.67
ConvGPPN	99.75	99.09	97.21	98.55	99.98	99.95	89.88
SymGPPN	**99.98**	**99.86**	**99.49**	**99.78**	**100.00**	**99.99**	90.50

tion spaces for the manipulation tasks, except GPPN which is not fully convolutional.

Training and evaluation. We report success rate and training curves over 3 seeds. The training process (on given maps) follows [Tamar et al., 2016b, Lee et al., 2018], where we train 30 epochs with batch size 32, and use kernel size $F = 3$ by default. The default batch size is 32. GPPN variants need smaller number because LSTM consumes much more memory.

4.4.1 Planning on given maps

Environmental setup. In the **2D navigation** task, the map and goal are randomly generated, where the map size is $\{15, 28, 50\}$. In **2-DOF manipulation** in configuration space, we adopt the setting in Chaplot et al. [2021] and train networks to take as input "maps" in configuration space, represented by the state of the two manipulator joints. We randomly generate 0 to 5 obstacles in the manipulator workspace. Then the 2 degree-of-freedom (DOF) configuration space is constructed from the workspace and discretized into 2D grids with sizes $\{18, 36\}$, corresponding to bins of 20° and 10°, respectively. All methods are trained using the same network size, where for the equivariant versions, we use *regular* representations for all layers, which has size $|D_4| = 8$. We keep the same parameters for all methods, so all equivariant convolution layers with *regular* representations will have higher embedding sizes. Due to memory constraints, we use $K = 30$ iterations for 2D maze navigation, and $K = 27$ for manipulation. We use kernel sizes $F = \{3, 5, 5\}$ for $m = \{15, 28, 50\}$ navigation, and $F = \{3, 5\}$ for $m = \{18, 36\}$ manipulation.

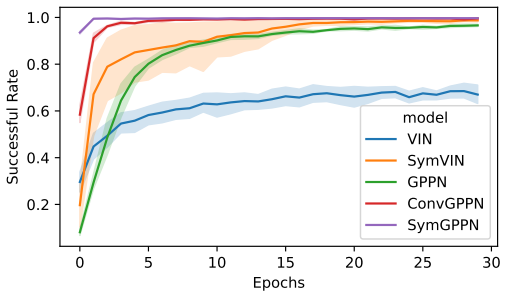

Figure 4.5: Training curves on 2D navigation with 10K of 15×15 maps. Faded areas indicate standard error.

Results. We show the averaged test results for both 2D navigation and C-space manipulation tasks on generalizing to unseen maps (Table 4.1) and the training curves

for 2D navigation (Figure 4.5). For VIN-based methods, SymVIN learns much faster than standard VIN and achieves almost perfect asymptotic performance. For GPPN-based methods, we found the translation-equivariant convolutional variant ConvGPPN works better than the original one in [Lee et al., 2018], especially in learning speed. SymVIN fluctuates in some runs. We observe that the first epoch has abnormally high loss in most runs and quickly goes down in the second or third epoch, which indicates effects from initialization. SymGPPN further boosts ConvGPPN and outperforms all other methods and does not experience variance. One exception is that standard GPPN learns poorly in C-space manipulation. For GPPN, the added circular padding in the convolution encoder leads to a gradient-vanishing problem.

Additionally, we found that using regular representations (for D_4 or C_4) for state value $V : \mathbb{Z}^2 \to \mathbb{R}^{C_V}$ (and for Q-values) works better than using trivial representations. This is counterintuitive since we expect the V value to be scalar $\mathbb{Z}^2 \to \mathbb{R}$. One reason is that switching between regular (for Q) and trivial (for V) representations introduces an unnecessary bottleneck. Depending on the choice of representations, we implement different max-pooling operations tailored to preserve equivariance. We also empirically found using FC only in the final layer $Q_K \mapsto A$ helps stabilize the training. Additional ablation studies are described in [Zhao et al., 2023c].

Remark. Our two symmetric planners are both significantly better than their standard counterparts. Notably, we did not include any symmetric maps in the test data, which symmetric planners would perform much better on. There are several potential sources of advantages: (1) SymPlan allows parameter sharing across positions and maps and implicitly enables planning in a reduced space: every (s, a, s') generalizes to $(g.s, g.a, g.s')$ for any $g \in G$, (2) thus it uses training data more efficiently, (3) it reduces the hypothesis-class size and facilitates generalization to unseen maps.

4.4.2 Planning on learned maps: simultaneously planning and mapping

Environmental setup. For **visual navigation**, we randomly generate maps using the same strategy as before, and then render four egocentric panoramic views for each location from 3D environments produced with *Gym-MiniWorld* [Chevalier-Boisvert, 2018], which can generate 3D mazes with any layout. For $m \times m$ maps, all egocentric views for a map are represented by $m \times m \times 4$ RGB images. For **workspace manipulation**, we randomly generate 0 to 5 obstacles in workspace as before. We use a mapper network to convert the 96×96 workspace (image of obstacles) to the $m \times m$ 2 degree-of-freedom (DOF) configuration space (2D occupancy grid). In both environments, the setup is similar to Section 4.4.1, except here we only use map size $m = 15$ for visual navigation (training for 100 epochs), and map size $m = 18$ for workspace manipulation (training for 30 epochs).

Methods: mapper networks and setup. For **visual navigation**, we implemented an equivariant mapper network based on Lee et al. [2018]. The mapper network converts every image into a 256-dimensional embedding $m \times m \times 4 \times 256$ and

then predicts the map layout $m \times m \times 1$. For **workspace manipulation**, we use a U-net [Ronneberger et al., 2015] with residual-connection [He et al., 2015] as a mapper. For detailed training procedures and hyperparameters, see [Zhao et al., 2023c].

Results. The results are shown in Table 4.1, under the columns "Visual" (navigation, 15×15) and "Workspace" (manipulation, 18×18). In visual navigation, the trends are similar to the 2D case: our two symmetric planners both train much faster. Besides standard VIN, all approaches eventually converge to near-optimal success rates during training (around 95%), but the validation and test results show large performance gaps. Complete training curves and detailed analysis are available in [Zhao et al., 2023c]. SymGPPN has almost no generalization gap, whereas VIN does not generalize well to new 3D visual navigation environments. SymVIN significantly improves test success rate and is comparable with GPPN. Since the input is raw images and a mapper is learned end-to-end, it is potentially a source of generalization gap for some approaches. In workspace manipulation, the results are similar to our findings in C-space manipulation, but our advantage over baselines were smaller. We found that the mapper network was a bottleneck, since the mapping for obstacles from workspace to C-space is non-trivial to learn.

4.4.3 Results on generalization to larger maps

To demonstrate the generalization advantage of our methods, all methods are trained on small maps and tested on larger maps. All methods are trained on 15×15 with $K = 30$. Then we test all methods on map size 15×15 through 99×99, averaging over 3 seeds (3 model checkpoints) for each method and 1000 maps for each map size. Iterations K were set to $\sqrt{2}M$, where M is the test map size (x-axis). The results are shown in Figure 4.6.

Results. SymVIN generalizes better than VIN, although the variance is greater. GPPN tends to diverge for larger values of K. ConvGPPN converges, but it fluctuates for different seeds. SymGPPN shows the best generalization and has small variance. In conclusion, SymVIN and SymGPPN generalize better to different map sizes, compared to all non-equivariant baselines.

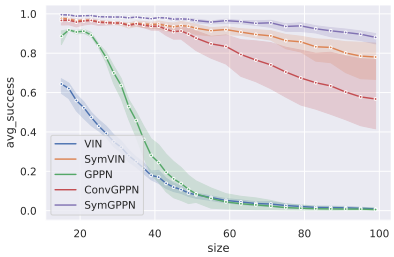

Figure 4.6: Results for testing on larger maps, when trained on size 15 map. Our methods outperform all baselines.

Remark. In summary, our results show that the SymPlan models demonstrate end-to-end planning and learning ability, potentially enabling further applications to other tasks as a differentiable component for planning. Additional results and ablation studies are in [Zhao et al., 2023c].

4.5 Chapter Discussion

This chapter demonstrated how geometric symmetries provide compositional structure for planning. Symmetric Value Iteration Networks (SymVIN) show that value iteration itself can be made equivariant through steerable convolutions, not just the representations. The approach exploits the insight that symmetry constraints reduce the hypothesis space while preserving optimality—the quotient MDP framework guarantees that symmetric states share the same optimal value, enabling generalization to rotated and reflected environments without additional training.

Chapter 5

E(2)-Equivariant Graph Planning for Navigation

Chapter 4 demonstrated D(4)-equivariant planning on regular grids, but real-world navigation requires handling continuous spaces with irregular obstacles and complex topologies. This chapter extends symmetry-aware planning from discrete lattices to continuous environments through E(2)-equivariant message passing on geometric graphs.

The key challenge is maintaining equivariance guarantees when transitioning from regular grids to irregular graph structures. While grid-based methods use translation-equivariant 2D convolutions [Tamar et al., 2016b], geometric graphs require different approaches. We address this using geometric graphs [Bronstein et al., 2021b, Brandstetter et al., 2021] where nodes represent states at arbitrary positions $\boldsymbol{x}_i \in \mathbb{R}^2$ and edges represent feasible transitions. The Euclidean group $\mathrm{E}(2) = \mathbb{R}^2 \rtimes \mathrm{O}(2)$ captures both translations and rotations/reflections that preserve navigation optimality.

Our technical contribution introduces Message Passing Value Iteration Networks (MP-VIN) that maintain E(2)-equivariance through steerable message passing [Weiler and Cesa, 2021, Cohen and Welling, 2016b]. For translation equivariance, we use relative positions $\boldsymbol{x}_i - \boldsymbol{x}_j$ in message computation. For rotation/reflection equivariance, we employ steerable kernels that satisfy $K(gx) = \rho_{\text{out}}(g) \circ K(x) \circ \rho_{\text{in}}(g)^{-1}$ for group elements $g \in \mathrm{O}(2)$. This ensures that rotating the environment produces correspondingly rotated value functions and policies.

A practical challenge is multi-camera fusion: robots typically have cameras at fixed angles (e.g., 4 cameras at 90° intervals), but continuous rotation symmetry requires handling arbitrary angles. We develop learnable equivariant layers that lift discrete camera features to continuous group representations using the induced representation $\mathrm{Res}^G_{C_K}[\rho^G_{\text{reg}}]$. This enables E(2)-equivariant planning even with discrete sensor configurations.

Experiments on grid worlds, graph worlds, and Miniworld environments demonstrate that MP-VIN with $\mathbb{R}^2 \rtimes D_8$ symmetry achieves 91.5% success on 15×15 grids and 96.96% on graph-based navigation, significantly outperforming non-equivariant baselines. The approach scales from structured grids to unstructured geometric graphs while preserving symmetry guarantees.

This chapter is based on our paper "E(2)-Equivariant Graph Planning for Navigation" published in IEEE Robotics and Automation Letters 2024.

5.1 Related Works

Geometric deep learning. Geometric deep learning and equivariant networks extend the study of classic 2D translation-equivariant convolution neural networks into more symmetry groups and spaces [Bronstein et al., 2021b, Cohen and Welling, 2016a]. Cohen and Welling [2016a] propose group convolution network (G-CNN), a pioneer work that studies rotation symmetry, followed by an extension to steerable convolution, Steerable CNN [Cohen and Welling, 2016b]. It has also been extended to the 3D case [Weiler et al., 2018] and supported by a library in E(2) [Weiler and Cesa, 2021]. For graphs, equivariant message passing uses equivariant multilayer perceptrons (MLPs) to propagate geometric quantities between nodes to preserve the symmetry [Satorras et al., 2021, Brandstetter et al., 2021]. Different from E(3)-equivariant message passing in [Brandstetter et al., 2021], we work on E(2) case. Additionally, the relationship between geometric graphs and dynamic programming or value iteration has been discussed in [Xu et al., 2019, Dudzik and Veličković, 2022]. The theory on equivariant networks has also been well developed based upon representation theory of symmetry groups [Kondor and Trivedi, 2018, Cohen et al., 2020, Lang and Weiler, 2020]. In practice, equivariant networks enable sharing parameters and reduce the number of parameters.

Equivariance in RL and planning. Symmetry and equivariance have been studied in reinforcement learning and planning before and in the era of deep learning [Fox and Long, 1999, 2002, Pochter et al., 2011, Shleyfman et al., 2015, Sievers et al., 2015, Sievers, Winterer et al., Röger et al., 2018, Sievers et al., 2019, Fiser et al., 2019]. Invariance of the optimal value function and equivariance of the optimal policy of a Markov Decision Process (MDP) with symmetry have been shown in Ravindran and Barto [2004], Ravindran and Barto, Zinkevich and Balch [2001]. When using function approximation, equivariant policy networks and invariant value networks have been used to improve training efficiency in model-free RL [van der Pol et al., 2020a, Mondal et al., 2020, Wang et al., 2021], and equivariance also helps in transition model and model-based RL [Zhao et al., 2022a, Park et al., 2022, Zhao et al., 2022b, 2023a].

Learning to navigate. In real-world robotic applications, navigation within a 2D Euclidean space involves a significant consideration of symmetry. This is due to the property that the distance to a goal remains constant regardless of the robot's orientation. Consequently, it follows that the optimal course of action, with respect to orientation, remains consistent. In this paper, we aim to investigate a particular class of navigation algorithms that rely on Value Iteration Network (VIN) [Tamar et al., 2016b] and its variants [Lee et al., 2018, Zhao et al., 2022b, 2023b, Zhao and Wong, 2020]. The selection of VINs is motivated by the fact that value iteration is fully differentiable and inherently encompasses an equivariant convolution [Zhao et al., 2022b]. The closest works to ours are from Ishida and Henriques [2022] and Gupta et al. [2017], which

leverage VIN to address robot navigation. Ishida and Henriques [2022] propose several improvement techniques of the original VIN to better fit real-world deployment. Gupta et al. [2017] solves mapping and planning simultaneously, and achieve proof-of-concept on a real-world Turtlebot. While these approaches contribute to adapting VIN to real-world applications, they still operate on a structured grid by discretizing the continuous state space. This sacrifice the agility of navigation and makes it hard to scale to the larger environment due to memory and computational constraints. Prior to us, Zhao et al. [2022b] improve VIN with symmetry on the 2D grid $\mathbb{Z}^2$. In this paper, we extend the symmetry-based planning to the 2D plane $\mathbb{R}^2$, enabling navigation in more realistic unstructured environments.

5.2 Problem Formulation: Navigation as Geometric Graphs

In this section, we give a definition of the navigation problem studied in this work and discuss symmetry in the definition.

Overview. We construct a *geometric graph* [Bronstein et al., 2021b, Brandstetter et al., 2021] to represent the navigation MDP. The goal is to learn a planner that reaches targets on the graph: $\boldsymbol{a}_t = \texttt{plan}_\theta(\boldsymbol{s}_t)$. We focus on global planning: given navigation task (and all image observations) as a feature field/map M, we output action field $A = \texttt{Plan}_\theta(M)$. The implementation of equivariance is introduced in the next section.

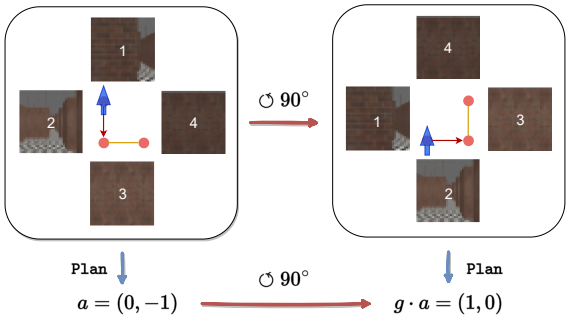

Figure 5.1: Illustration of rotation equivariance. We use the blue arrow to show the orientation of the robot. Rotating the robot ↻ 90° is equivalent to rotating the world frame ↺ 90°. When camera views are cyclically permuted, action output (red arrow) is transformed by rotation matrix.

Definition. We approach navigation as a 2D continuous path planning problem, an extension to the 2D discrete grid version introduced in [Tamar et al., 2016b, Zhao et al., 2022b]. We extend 2D grids to the use of *geometric graphs* (spatial graphs) $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ on 2D Euclidean space $\mathbb{R}^2$. In navigation tasks, the agent observes a state $\boldsymbol{s}_t \in \mathcal{S}$ at each step, and the action is to move on the 2D plane $\boldsymbol{a}_t \in \mathcal{A} = \mathbb{R}^2$. States can represent a 2D position in $\mathbb{R}^2$ or egocentric panoramic images in $\mathbb{R}^{K \times H \times W}$ (where K denotes the number of images of resolution $H \times W$). To convert the task into a geometric graph, each node $v_i \in \mathcal{V}$ corresponds to a state $\boldsymbol{s} \in \mathcal{S}$ and is associated with a *node feature* $\boldsymbol{h}_i$ (such as images) and has a position $\boldsymbol{x}_i \in \mathbb{R}^2$. Similarly, each edge $e_{ij} \in \mathcal{E}$ corresponds to a state-action transition $(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A}$ and has an *edge feature* $\boldsymbol{a}_{ij} \in \mathbb{R}^{c_e}$.

Geometric Structure. Such MDP naturally exists geometric structure: the MDP, modeled as a geometric graph, "lives" in a 2D Euclidean space, and we may *transform* the graph via 2D Euclidean transformations that preserve distances (*isometric*) and

do not affect optimal solution of the MDP [Ravindran, van der Pol et al., 2020b, Zhao et al., 2022b]. The set of all such transformations in 2D is called *Euclidean group* E(2), which can be uniquely decomposed into translation part $\mathbb{R}^2$ and rotation/reflection part O(2), denoted as semi-direct product $\mathrm{E}(2) = \mathbb{R}^2 \rtimes \mathrm{O}(2)$ [Cohen and Welling, 2016a, Weiler and Cesa, 2021]. We only require the node features $\boldsymbol{h}$ and edge features $\boldsymbol{a}$ are transformable by $G \leq \mathrm{E}(2)$. Following the notation in [Weiler and Cesa, 2021], we denote the rotation/reflection symmetry part as compact symmetry group $G \leq \mathrm{GL}(2)$, because translation group is *not compact* and many useful theorems do not hold. In implementation, translation equivariance is achieved by using *relative position*. For any subgroups G of rotation/reflection, its equivariance needs *group convolution* [Cohen and Welling, 2016a] or *steerable convolution* [Cohen and Welling, 2016b].

Value Iteration and Symmetry. Our planning is based on value iteration, a dynamic programming algorithm on solving optimal value and policy functions in MDPs [Sutton and Barto, 2018]. When symmetry appears in an MDP, the value and policy functions are equivariant [Ravindran, Ravindran and Barto, van der Pol et al., 2020b, Zhao et al., 2022b]. Abstractly, we can write value iteration as iteratively applying the Bellman operator $\mathcal{T}$, defined by:

$$Q(\boldsymbol{s}, \boldsymbol{a}) := R(\boldsymbol{s}, \boldsymbol{a}) + \int_{\mathbb{R}^2} d\boldsymbol{s}' P(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) V(\boldsymbol{s}'), \quad V(\boldsymbol{s}) = \max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a}) \tag{5.1}$$

where the input and output of the Bellman operator are both value function $V : \mathcal{S} \to \mathbb{R}$. Zhao et al. [2022b] explore the equivariance of it for 2D grid case:

$$\texttt{VI}(M) = \mathcal{T}_M^{\infty}[V_0], \quad g \cdot \texttt{VI}(M) \equiv g \cdot \mathcal{T}^{\infty}[V_0] = \mathcal{T}^{\infty}[g \cdot V_0] \equiv \texttt{VI}(g \cdot M) \tag{5.2}$$

Symmetry Transformations. The equivariance constraints enforce equivalence between transformed and original input/output [Cohen and Welling, 2016a,b, Lang and Weiler, 2020]. Thus, we need to understand how to transform feature maps (fields) of the geometric graphs. Under group transformation g, a (left) *regular* representation transforms a feature map with c_{out}-dimensional vector (vector field) $f : X \to \mathbb{R}^{c_{\text{out}}}$ as [Cohen and Welling, 2016b, Lang and Weiler, 2020, Bronstein et al., 2021b]:

$$[L_g f](x) = \left[f \circ g^{-1}\right](x) = \rho_{\text{out}}(g) \cdot f\left(g^{-1} x\right), \tag{5.3}$$

where ρ_{out} is the G-representation associated with output $\mathbb{R}^{c_{\text{out}}}$. For the *scalar* value map, ρ_{out} is identity, or trivial representation. For example, for *action* feature map $A : \mathbb{R}^2 \to \mathbb{R}^2$ (i.e., every position $\boldsymbol{x} \in \mathbb{R}^2$ is associated with a relative movement). There are several useful functions in RL and planning that can be written as graph features, where functions on $\mathcal{S}$ are node features and functions on $\mathcal{S} \times \mathcal{A}$ are edge features. Note that M and A are *vector* map thus their fiber (vector) needs additional transformation. When $\mathcal{A}$ is continuous action, $\rho_{\mathcal{A}}$ is rotation matrices. For image-input

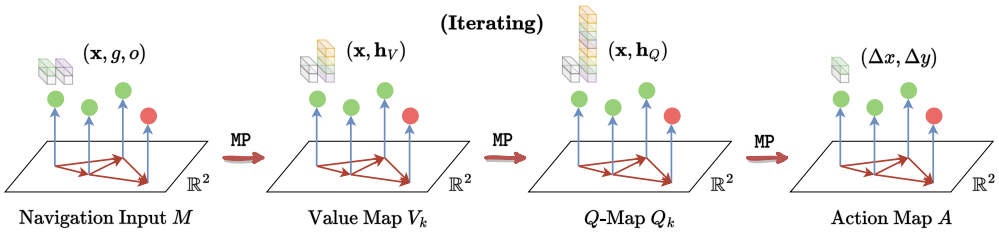

Figure 5.2: Overview of the pipeline. The planner first takes input of the navigation task, representation by feature map M, and apply value iteration to iterate between Q_k and V_k until convergence, and output action map A (relative translation for each node).

case of M, $\rho_{\text{camera}}(g)$ means cyclically permuting cameras, discussed next section.

$$\text{Navigation Input} \quad M : \mathcal{S} \to \mathbb{R}^{K \times H \times W} : \quad \rho_{\text{camera}}(g) \cdot M(\boldsymbol{s}_t) = M(\rho_{\mathcal{S}}(g) \cdot \boldsymbol{s}_t) \tag{5.4}$$

$$\text{Reward Map} \quad R : \mathcal{S} \times \mathcal{A} \to \mathbb{R} : \quad R(\boldsymbol{s}_t, \boldsymbol{a}_t) = R(\rho_{\mathcal{S}}(g) \cdot \boldsymbol{s}_t, \rho_{\mathcal{A}}(g) \cdot \boldsymbol{a}_t) \tag{5.5}$$

$$Q\text{-Map} \quad Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R} : \quad Q(\boldsymbol{s}_t, \boldsymbol{a}_t) = Q(\rho_{\mathcal{S}}(g) \cdot \boldsymbol{s}_t, \rho_{\mathcal{A}}(g) \cdot \boldsymbol{a}_t) \tag{5.6}$$

$$\text{Value Map} \quad V : \mathcal{S} \to \mathbb{R} : \quad V(\boldsymbol{s}_t) = V(\rho_{\mathcal{S}}(g) \cdot \boldsymbol{s}_t) \tag{5.7}$$

$$\text{Action Map} \quad A : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|} : \quad \rho_{\mathcal{A}}(g) \cdot A(\boldsymbol{s}_t) = A(\rho_{\mathcal{S}}(g) \cdot \boldsymbol{s}_t) \tag{5.8}$$

5.3 Methodology: Equivariant Message Passing for Value Iteration

Following the spirit of VIN [Tamar et al., 2016b], we build a geometric message passing (MP) network and extend to learning value iteration on geometric graphs: $A = \texttt{Plan}_\theta(M)$. As it takes as input feature map M and outputs action map A that are both transformable under the group, we enforce equivariance constraints throughout the MP network: $g \cdot A = \texttt{Plan}_\theta(g \cdot M)$. The visualization of equivariance is shown in Figure 5.1, and the overview of the pipeline is shown in Figure 5.2.

Discretization to Graph. In regular grid, we can use regular 2D convolution to perform value iteration, as done in VIN [Tamar et al., 2016b]. However, for irregular graph, grid does not fit anymore. A prior work [Niu et al., 2017] use an earlier version of graph convolution, but it only has $\mathbb{R}^2$ translation equivariance and did not consider rotation or reflection O(2) symmetry. Here, we derive from first principles using the original continuous form of value iteration. The integral term in VI can be written as a mapping Φ

$$\boldsymbol{h}'(\boldsymbol{x}) = \Phi[h](\boldsymbol{x}) = \int_{\mathbb{R}^2} K(\boldsymbol{x}, \boldsymbol{x}')\boldsymbol{h}(\boldsymbol{x}'), \tag{5.9}$$

where $K : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}}$, $\boldsymbol{h} : \mathbb{R}^2 \to \mathbb{R}^{c_{\text{in}}}$ and $\boldsymbol{h}' : \mathbb{R}^2 \to \mathbb{R}^{c_{\text{out}}}$ are input and output feature map. If translation equivariance is desired, the kernel can be further simplified from two-argument to one-argument case, and the mapping is *convolution*

[Cohen et al., 2020, Zhao et al., 2022b]. The continuous steerable convolution is defined (via cross-correlation) by [Cohen and Welling, 2016b, Weiler and Cesa, 2021, Lang and Weiler, 2020]:

$$\boldsymbol{h}'(\boldsymbol{x}) = \Phi[h](\boldsymbol{x}) = [K \star \boldsymbol{h}] = \int_{\mathbb{R}^2} K(\boldsymbol{x}' - \boldsymbol{x})\boldsymbol{h}(\boldsymbol{x}'), \tag{5.10}$$

where $K : \mathbb{R}^2 \to \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}}$ is a (steerable) kernel.

If we sample nodes in $\mathbb{R}^2$ and construct edges by transition $\mathcal{S} \times \mathcal{A}$, the continuous convolution on $\mathbb{R}^2$ can be discretized, which is similar to strategy of PointConv [Brandstetter et al., 2021] [1]. We use *nonlinear message passing* to replace linear convolution. We use two MLP for computing messages ($\texttt{propagate}_\theta$) and updating node features ($\texttt{update}_\theta$), and the basic form is given by

$$\boldsymbol{m}_{ij} = \texttt{propagate}_\theta\left(\boldsymbol{h}_i, \boldsymbol{h}_j, \boldsymbol{x}_i, \boldsymbol{x}_j\right), \quad \boldsymbol{h}_i' = \texttt{update}_\theta\left(\boldsymbol{h}_i, \sum_{j \in \mathcal{N}(i)} \boldsymbol{m}_{ij}\right). \tag{5.11}$$

Implementation of Equivariance. We implement E(2)-equivariant message passing on the graph that is equivariant under two parts:

- *Translation* $\mathbb{R}^2$. The translation group is non-compact because it is not bounded. There are several possible ways: (1) using relative position [Brandstetter et al., 2021], (2) induced representation from $\mathrm{SO}(d)$ to $\mathrm{SE}(d)$ [Cohen and Welling, 2016b, Cohen et al., 2020, Lang and Weiler, 2020], (3) canonalization to a fixed origin [Weiler and Cesa, 2021, Lang and Weiler, 2020], etc. We use relative position between nodes $\boldsymbol{x}_i - \boldsymbol{x}_j$ as input, which is analogous to translation-equivariant 2D convolution:

$$\boldsymbol{m}_{ij} = \texttt{propagate}_\theta\left(\boldsymbol{h}_i, \boldsymbol{h}_j, \boldsymbol{x}_i - \boldsymbol{x}_j\right). \tag{5.12}$$

- *Rotation and Reflection* $\mathrm{O}(2)$. We use steerable equivariant network to implement $\mathrm{O}(2)$-equivariance [Cohen and Welling, 2016b,a, Weiler and Cesa, 2021, Brandstetter et al., 2021, Cohen et al., 2020]. The $\mathrm{O}(2)$ group is compact and thus its representations are decomposable into *irreducible representations* [Weiler and Cesa, 2021, Lang and Weiler, 2020], thus convolution can perform on Fourier domain and be more efficient. We use it to build equivariant MLPs of `propagate` and `update` (effectively 1×1 convolution). The kernel K of G-steerable convolution needs to satisfy constraint [Weiler and Cesa, 2021, Lang and Weiler, 2020], where G can be any (discrete) subgroup of $\mathrm{O}(2)$:

$$K(gx) = \rho_{\text{out}}(g) \circ K(x) \circ \rho_{\text{in}}(g)^{-1} \quad \forall g \in G \leq \mathrm{O}(2), x \in \mathbb{R}^2. \tag{5.13}$$

Equivariance of the message passing version of value iteration can be shown (see the complete paper version for the formal proof).

[1] Brandstetter et al. [2021] discuss other strategy for 3D steerable messsage passing, which expands the feature maps to spherical harmonics. Analogously, it is possible to expand the features to cyclic harmonics.

Theorem 5.1. *The Bellman operator is equivariant under Euclidean group* E(2).

Processing Multi-camera Images. One technical challenge to consider equivariance in real-world navigation is that, normally, a robot is equipped with evenly distributed cameras in 360°, such as 4-camera in 90° or 6-camera in 60°. However, the symmetry group we consider is not necessarily "compatible" with the images. For example, if the planner $g \cdot A = \texttt{Plan}_\theta(g \cdot M)$ is required to be G-equivariant, such as $G = C_8$ (45° rotation group), a 45° rotation of action needs to correspond to a 45° rotation of robot orientation [2]. However, we do not have camera images for such case without interpolation. As shown in Figure 5.1, we can only cyclically permute four camera views.

More concretely, recall the number of cameras/images for each position $\boldsymbol{x} \in \mathbb{R}^2$ in M is K, so the multi-view image $I = (I_1, \ldots, I_K)$ belongs[3] to $\mathbb{R}^{K \times H \times W}$. For example, if $K = 4$, we write it as $I = (I_1, I_2, I_3, I_4)$. For this particular case, rotating a robot can only be $90° \cdot k$ rotations from C_4 (cyclic group with $k \cdot 2\pi/4$ rotations), which corresponds to cyclically permuting images in I (by *regular representation*) of C_4: $\rho_{\text{reg}}(\circlearrowleft 90°) \cdot I = \rho_{\text{reg}}(\circlearrowleft 90°) \cdot (I_1, I_2, I_3, I_4) = (I_4, I_1, I_2, I_3)$.

However, this blocks us from using higher-order symmetry G in later planning network. To this end, we propose using a *trainable equivariant layer* to *lift* the H-features to G-features, which is to *induce* how multi-view images $I = (I_1, \ldots, I_K)$ are transformed under a larger group[4] $G \geq H$. Such *induction* layer needs to satisfy the steerable kernel constraint with input and output representation:

$$\rho_{\text{in}} = \rho_{\text{reg}}^{C_K}, \quad \rho_{\text{out}} = \text{Res}_{C_K}^{G}\left[\rho_{\text{reg}}^{G}\right], \quad K(0) = \text{Res}_{C_K}^{G}\left[\rho_{\text{reg}}^{G}\right](g) \cdot K(0) \cdot \rho_{\text{reg}}^{C_K}(g^{-1}), g \in C_K, \tag{5.14}$$

where ρ_{in} is (e.g., regular) representation of C_K and ρ_{out} is restricted representation of (e.g., regular) representation of SO(2) to C_K. The restricted representation limits the equivariant constraints to only the subgroup $H = C_K$. See the complete paper version for mathematical details.

Intuitively, this layer only enforces H-equivariant ($H = C_K$) but not G: cyclically permuting images $\circlearrowleft 90°$ should result in $\circlearrowleft 90°$ rotated output. However, it outputs G-equivariant features, thus proceeding layers are able to apply G-equivariance. In implementation, we use discrete subgroup such as $D_8 \leq \text{O}(2)$, since continuous groups do not have finite-dimensional regular representation to permute $I = (I_1, \ldots)$.

5.4 Experiments

We evaluate our proposed approach MP-VIN and baselines on four different tasks. Among these tasks, we perform point goal navigation under different environments:

[2] One solution for D_4 group is to use *quotient* representations, but it not generally applicable for higher-degree rotations such as D_8 or infinitesimal rotations SO(2).

[3] We omit image channel for notation simplicity, such as RGB channels.

[4] This requires C_K to be a subgroup of G, which is always the case for $G = \text{SO}(2)$ and for $G = C_{K'}$ or $D_{K'}$ when K' is divisible by K.

known structured environments (`Grid World`), known unstructured environments (`Graph World`), unknown structured environments (`Miniworld`), and unknown unstructured environments (`Miniworld-Graph`). For more general details regarding our experiment setup, please see the complete paper version.

Methods. We experiment four variants of our methods, with or without translation ($\mathbb{R}^2$) or rotation/reflection (using $G = D_8 \leq \mathrm{O}(2)$) equivariance: No-Sym, D_8, $\mathbb{R}^2$, and $\mathbb{R}^2 \rtimes D_8$. We use two grid-based methods: VIN Tamar et al. [2017] and SymVIN Zhao et al. [2023c] (with D_4-equivariance). These methods are grid-based; therefore, we apply several modifications, including pre-processing and post-processing, to ensure fair comparisons. These modifications are detailed in the later sections. We also replace and compare our message passing module with the Graph Convolutional Networks Kipf and Welling [2017] (GCN-VIN) and Graph Attention Networks Veličković et al. [2018] (GAT-VIN).

5.4.1 Planning on known maps: Grid World

Setting. In this task, we randomly generate synthetic mazes with size $m \times m$ (`Grid World`). We validate the performance on two different sizes $m \in \{15, 27\}$. Each cell on the maze map is represented as occupied (0) or unoccupied (1). There are four actions available for each cell on the map: north, east, west, and south. We randomly select a goal on the map and generate a goal map, where the cell containing the goal is marked as 1. Each cell is labeled by the ground-truth action toward the goal using Dijkstra's algorithm.

To adapt the grid world setting to our approach, we transform the grid representation into the graph representation Niu et al. [2017], in which each cell in the grid corresponds to a node in the graph. Concretely, each node contains a four-dimensional feature vector $\mathbb{R}^4$. The first two elements represent the location of the corresponding cell in the grid world. The third element represents whether the node is an obstacle, and the fourth element represents whether the node is the goal. This could be viewed as a graph version of the maze map and the goal map in the grid-based methods. The edge connections between nodes follow the adjacency in the grid, whereas each node is connected with its neighbors on the grid.

Results. MP-VIN with $\mathbb{R}^2 \rtimes D_8$ symmetry demonstrates faster learning efficiency than its graph-based variants and VIN (the left of Fig. 5.3). We surprisingly find that MP-VIN with $\mathbb{R}^2 \rtimes D_8$ has much smoother learning curves than its variants without D_8 symmetry ($\mathbb{R}^2$ and No-Sym). This indicates that injecting Euclidean symmetry may improve the loss landscape. In terms of absolute performance gain, by adding D_8 symmetry to MP-VIN with $\mathbb{R}^2$, we obtain another 0.69% and 12.07% success rate on the 15×15 and 27×27 mazes, respectively. However, it is still outperformed by Sym-VIN, which uses a convolutional neural network (CNN) to process the input. We believe the CNN takes advantage of the structured environment (grid) and is more efficient in such scenarios than a GNN, which operates in a more expressive unstructured environment (graph).

When the map size increases, MP-VIN with $\mathbb{R}^2 \rtimes D_8$ symmetry demonstrates the

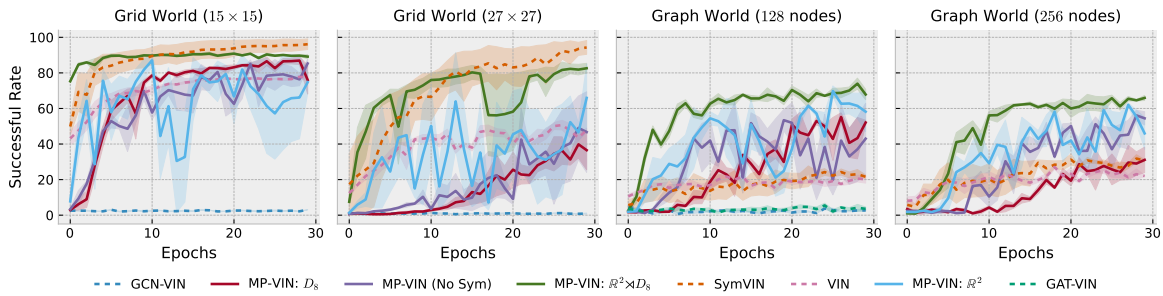

Figure 5.3: Learning curves on the Grid World experiments (left two) and the Graph World experiments (right two). The shadow area shows the standard error. Dashed lines are for non-MP-VIN methods (VIN, SymVIN, GCN-VIN, and GAT-VIN).

second-least performance degradation. Therefore, leveraging Euclidean symmetry also leads to better generalization to larger maps.

5.4.2 Planning on known graphs: Graph World

Setting. We validate the performance of our approach in unstructured graph environments (`Graph World`). We follow the setup of Niu et al. [2017] to generate the random graphs. We randomly generate N nodes, each with coordinates between $(0, 0)$ and (m, m). These nodes are connected using a KNN graph. We randomly select some nodes as the obstacle nodes, and one node as the goal node.

To verify the performance of grid-based approaches in Graph World, we discretize the environment Niu et al. [2017]. We round down the coordinates of each node to map it to a cell on the grid. The obstacle feature is carried over to the grid. Ultimately, we verify its performance on the graph by transforming the discrete actions into continuous actions (represented in $[x, y]$ coordinate). For example, `north` is transformed into $[1, 0]$, and `east` is transformed into $[0, 1]$. See the complete paper version for more details regarding discretization and action representations.

Results. MP-VIN with $\mathbb{R}^2 \rtimes D_8$ symmetry demonstrates the strongest performance in this task (Fig. 5.3). We observe a similar faster learning efficiency and smoother learning curve in this task as in the Grid World setting. Since the graph world is more complex than the grid world, all methods encounter performance degradation. Grid-based methods encounter the largest drop since CNN struggles to process the unstructured environments.

5.4.3 Mapping and planning under unknown maps: Miniworld

Setting. We compare different methods in a more challenging visual environment (`Miniworld`), where the models learn mapping and planning simultaneously. We leverage the Miniworld simulator Chevalier-Boisvert [2018] to render the randomly generated maze into a 3D visual environment. Different from the Grid World, we are not

Table 5.1: Averaged test success rate (%) and standard deviation for our experiments. The best result is **bolded**. The second-best result is underlined.

Method	Grid World		Graph World		Miniworld	
	15×15	27×27	128 nodes	256 nodes	Grid	Graph
VIN [Tamar et al., 2016b]	78.51±1.81	50.15±3.94	18.75±1.95	20.09±6.87	57.14±8.92	18.90±2.87
SymVIN [Zhao et al., 2022b]	**95.85**±5.02	**93.73**±7.33	24.40±2.11	27.53±4.73	**91.67**±2.58	27.98±4.34
MP-VIN (No Sym)	87.07±4.53	55.99±39.56	63.10±17.26	54.76±3.29	–	–
MP-VIN: D_8	87.19±2.78	38.53±16.17	52.38±5.02	32.89±3.47	–	–
MP-VIN: $\mathbb{R}^2$	90.81±1.02	72.45±31.94	70.24±2.69	58.33±5.93	79.76±24.06	96.58±2.46
MP-VIN: $\mathbb{R}^2 \rtimes D_8$	91.50±1.04	84.52±6.04	**72.17**±5.08	**61.90**±5.33	90.89±1.63	**96.96**±1.00

given a map in this experiment. Instead, we use egocentric RGB observations. For each cell in the maze environment, we obtain the RGB images from the cameras facing four orientations (0°, 90°, 180°, 270°). We transform the dataset into a graph using the same approach as mentioned in Sec. 5.4.1. In order to estimate the map of the environment, we encode the visual observations into occupancy features using a mapper network Lee et al. [2018], Chaplot et al. [2021], Zhao et al. [2022b]. The details regarding the mapping network are in the complete paper version.

Results. Since this task is based on a 15×15 grid using visual observations, the experiment results are similar to the Grid World. MP-VIN with $\mathbb{R}^2 \rtimes D_8$ symmetry demonstrates higher learning efficiency than MP-VIN with only $\mathbb{R}^2$ symmetry and VIN. However, every method faces a performance drop due to the mapping uncertainty. We observe that the performance gap of MP-VIN with $\mathbb{R}^2 \rtimes D_8$ symmetry (0.61%) is lower than that of SymVIN (4.18%). Therefore, the performance gap between MP-VIN with $\mathbb{R}^2 \rtimes D_8$ symmetry and SymVIN becomes narrower.

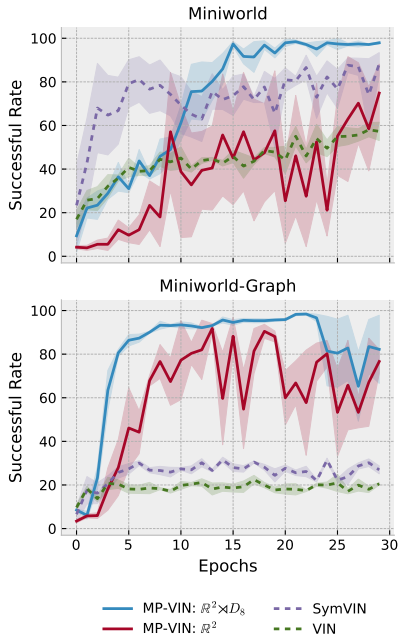

Figure 5.4: Learning curves on the `Miniworld` experiment (top) and `Miniworld-Graph` experiment.

5.4.4 Mapping and planning under unknown graphs: Miniworld-Graph

While Miniworld is a 3D-rendered visual environment, the state and action spaces are still discrete (grid-based). To show real-world feasibility, we aim at a more realistic setting in this experiment.

Setting. We sample random navigation graphs (256 nodes) in the Miniworld environment. The edges between nodes represent navigability. Like the Miniworld experiment in Section 5.4.3, each node contains a panoramic egocentric RGB observation facing four directions. We use a similar mapper to estimate the map from the visual observations. Differently, we estimate the occupancy graph instead of the occupancy grid. More details are in the complete

paper version.

Results. Similar to the Miniworld experiment on grid representation, we observe the MP-VIN with $\mathbb{R}^2 \rtimes D_8$ symmetry has higher learning efficiency than MP-VIN with only $\mathbb{R}^2$ symmetry. Grid-based approaches suffer in this task since it is hard for CNN to process the expressive unstructured environment, also indicated in the previous `Graph World` experiment. In both Miniworld experiments, we observe that MP-VIN with $\mathbb{R}^2 \rtimes D_8$ symmetry has a much smoother learning curve and lower variance than MP-VIN with only $\mathbb{R}^2$ symmetry. This indicates that by adding $\mathbb{R}^2$ symmetry, we could further optimize the network with better stability.

5.5 Conclusion and Discussion

In this paper, we explored the applicability of exploiting Euclidean symmetry within the context of a navigation planner. We contributed a novel equivariant differentiable planner. The effectiveness of the proposed approach is extensively assessed across four distinct tasks involving structured and unstructured environments, with known and unknown maps. The empirical findings demonstrate a significant enhancement in learning efficiency when Euclidean symmetry is integrated into 2D navigation planning. Furthermore, the results indicate that leveraging Euclidean symmetry yields more stable optimization and yields superior overall performance. In the future, we hope to extend our work to navigation task that has higher dimension, such as semantic navigation Liang et al. [2021], Chaplot et al. [2020], Chang et al. [2020] and Vision-and-Language navigation Anderson et al. [2018], Tellex et al. [2020].

5.6 Limitation

While we focus on study equivariance in the graph variant of VIN, there exists limitations. Inheriting from VIN, our message passing planner also considers only fully-observable states and take observations of all states as input at once Tamar et al. [2016b], Lee et al. [2018], Zhao et al. [2022b], which is impractical in large-scale real-world navigation. One potential solution is to partial observation, as done in [Ishida and Henriques, 2022]. However, we develop based on VIN variants because we derive that value iteration is a form of (graph) convolution network [Zhao et al., 2022b]. We believe the understanding of symmetry in VIN-based navigation is a valuable step before moving to more realistic navigation algorithms.

5.7 Chapter Discussion

This chapter addressed the limitation that real environments rarely conform to regular grids. Message Passing Value Networks (MPVN) extend the symmetry-aware planning of Chapter 4 to geometric graphs with irregular connectivity. The approach combines graph neural networks with E(2) equivariance, enabling planning over spatial

structures like building layouts and road networks. The multi-camera fusion component shows how equivariant principles handle practical sensor configurations—different camera arrangements viewing the same scene should produce consistent plans. This progression from grids to graphs demonstrates that compositional abstractions with symmetry constraints apply across different spatial representations, from discrete lattices to continuous geometric structures.

Chapter 6

Extensions and Variations (Part I)

This chapter demonstrates the ideas of learning for decision-making with structure through extensions that push to more generalized settings.

The first extension tackles continuous action spaces through equivariant sampling. While the previous chapters applied symmetry to planning with discrete states, real robots must select from continuous action distributions. We show that the same geometric principles—equivariance and group structure—apply to sampling-based control. This generalizes the idea beyond discrete actions.

The second extension explores spatial maps as an alternative to object-centric abstractions. Not all environments decompose naturally into objects—sometimes the space itself is the abstraction. Maps provide hierarchical compositional structure where local features compose into rooms, rooms into buildings, and buildings into environments. This complements our object-based approach and points toward hybrid representations that combine both perspectives—a key direction for integrating with foundation models that excel at both object recognition and spatial understanding.

These extensions demonstrate how structured learning principles adapt across different problem settings. As foundation models show increasing capability in spatial reasoning and perception, structured approaches provide formal frameworks to guarantee properties like equivariance and compositional generalization that pure learning approaches cannot ensure.

6.1 Extending to Continuous Control: Equivariant Action Sampling

While the previous chapters explored symmetry-aware planning in discrete environments and deterministic policies, modern model-based reinforcement learning algorithms operate in continuous action spaces where sampling-based optimization becomes essential. The transition from discrete to continuous control presents a fundamental challenge: how can we maintain the equivariance guarantees that proved so valuable in discrete settings when actions must be selected through sampling-based optimization?

This section addresses a critical gap in extending symmetry-aware planning to

model-based RL with continuous actions. Unlike the discrete action spaces of Chapters 4 and 5 where we could enumerate all possibilities, continuous control requires sampling-based methods like Model Predictive Path Integral (MPPI) control and Cross-Entropy Method (CEM) to explore the infinite action space. These sampling-based approaches are at the heart of state-of-the-art model-based RL algorithms like TD-MPC, yet their integration with symmetry principles has remained largely unexplored.

The fundamental challenge with sampling-based action selection lies in the finite-sample regime. While existing sampling methods can theoretically maintain symmetry in the limit of infinite samples, practical implementations with finite samples only approximate these symmetries. This approximation breaks the exact equivariance guarantees that are essential for leveraging the benefits of symmetric reinforcement learning methods, such as improved sample efficiency and generalization.

In this work, we address this challenge by developing a novel sampling strategy that preserves symmetry properties exactly, even with finite samples. We formulate the action optimization problem as a two-step procedure where neural networks first estimate the performance of state-action pairs through energy functions or Q-values, and then sampling is employed to identify optimal actions. Our investigation reveals the equivariance properties inherent in this formulation and demonstrates how they can be preserved through principled sampling techniques.

6.1.1 Geometric MDPs and Problem Definition

To formalize our approach, we introduce the concept of Geometric Markov Decision Processes (GMDPs). A GMDP extends the standard MDP framework by explicitly embedding the state and action spaces within geometric spaces such as $\mathbb{R}^2$ or $\mathbb{R}^3$. These embedded spaces naturally exhibit symmetries under Euclidean groups $E(2)$ or $E(3)$, which include translations, rotations, and reflections.

As a foundational case study, we examine the coordinate regression problem where an agent must select coordinates (x, y) in a 2D space to minimize the distance to a target location. This problem naturally exhibits $E(2)$ equivariance: if we rotate the target location, the optimal action should undergo the same rotation. This simple yet illustrative example captures the essence of the symmetry preservation challenge in continuous control.

The action optimization process in our framework consists of two distinct steps. First, a neural network estimates an energy function $E(s, a)$ that quantifies the quality of state-action pairs. Second, we employ sampling to identify the action that minimizes this energy function. For the overall procedure to maintain equivariance, both steps must individually preserve the symmetry properties of the underlying problem.

For the motivating coordinate regression task, we require that the sampling algorithm be equivariant:

$$a_0 = \arg\min_a E(s_0, a) \tag{6.1}$$

In other words, if we rotate the state $s_0 \to g \cdot s_0$, the selected action should also be rotated $a_0 \to g \cdot a_0$.

The Finite Sampling Challenge

Traditional sampling approaches face a fundamental limitation when applied to symmetric problems. Consider a standard finite sampling procedure where we sample N independent actions $\{a_1, a_2, ..., a_N\}$ and select the optimal action as $a^* = \arg\min_i E(s, a_i)$. While this approach is statistically sound, it fails to respect the symmetries of the underlying problem when the number of samples is finite.

The root of this issue lies in the random nature of sampling. Even if the energy function $E(s, a)$ is perfectly equivariant, the finite set of sampled actions may not exhibit the same symmetries. This asymmetry in the sampling process propagates to the final action selection, breaking the end-to-end equivariance of the optimization procedure.

Orbit Sampling: A Symmetry-Preserving Solution

We propose orbit sampling as a principled solution to the finite sampling challenge. Instead of sampling N completely independent actions, our approach samples only $N/|G|$ base actions, where $|G|$ denotes the size of the symmetry group. For each base action a_i, we generate its complete orbit under the group action: $\{g \cdot a_i : g \in G\}$. The optimal action is then selected from the union of all orbit elements across all base actions.

Thus, the G-augmented sample set is $G\mathcal{A} = \{g \cdot a_i \mid g \in G\}_{i=1}^{N}$. This orbit sampling strategy provides a crucial theoretical guarantee: it ensures exact equivariance regardless of the finite sample size. The equivariance condition for the optimization can be written as:

$$g \cdot a_0 = g \cdot \arg\min_{a \in G\mathcal{A}} E(s_0, a) = \arg\min_{a \in G\mathcal{A}} E(g \cdot s_0, a) \tag{6.2}$$

By construction, the set of candidate actions always exhibits the same symmetries as the underlying problem, preserving equivariance throughout the optimization process. This property holds for any finite number of base actions, not just in the infinite sample limit.

6.1.2 Extension to Model Predictive Path Integral Control

The principles of orbit sampling extend naturally to multi-step trajectory optimization in model-based RL, where the challenges of maintaining symmetry become even more pronounced. Model Predictive Path Integral (MPPI) control represents a particularly important class of sampling-based algorithms that can benefit from our symmetry-preserving approach. MPPI operates by sampling multiple trajectory rollouts and weighting them according to their cumulative returns, making it a natural candidate for our orbit sampling methodology. This is crucial for modern model-based RL algorithms like TD-MPC that rely on MPPI for action selection in continuous spaces.

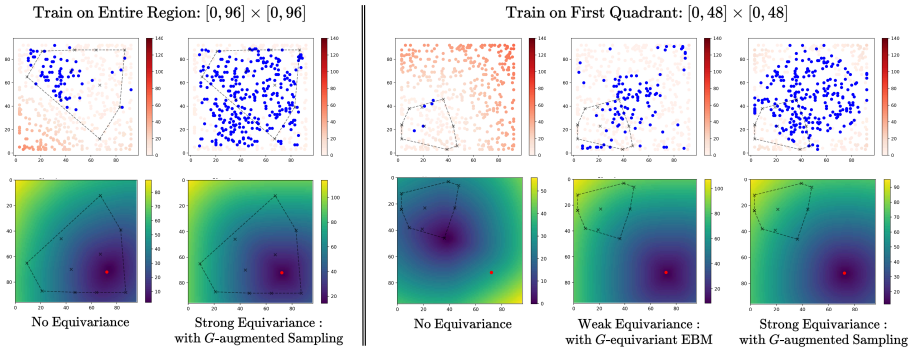

Figure 6.1: Coordinate regression results comparing standard sampling vs. orbit sampling. Top row shows spatial generalization with blue markers indicating accurate predictions. Bottom row shows energy landscapes. Orbit sampling maintains equivariance and achieves superior generalization, especially when training is limited to the first quadrant.

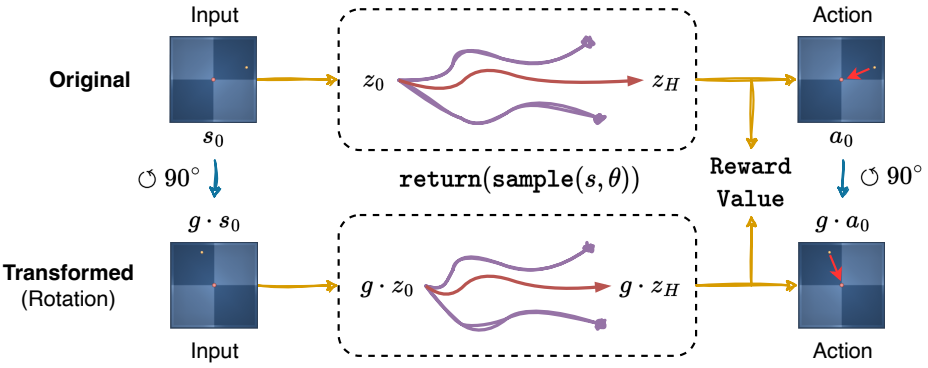

Figure 6.2: The equivariant sampling-based planning algorithm $a_0 = \texttt{plan}(s_0)$. If the input state is rotated, the output action should be rotated accordingly. This requires (1) the learned functions to be G-equivariant or G-invariant networks and (2) our specialized orbit sampling strategy to maintain equivariance in the finite sample regime.

Requirements for Equivariant MPPI

For MPPI to maintain equivariance properties, several components of the system must satisfy specific symmetry constraints. The dynamics model must exhibit equivariance, meaning that applying a group transformation to the state and action should result in the same transformation applied to the next state: $f(g \cdot s, g \cdot a) = g \cdot f(s, a)$ for all group elements g. Additionally, the reward function should be invariant under group transformations: $r(g \cdot s, g \cdot a) = r(s, a)$, ensuring that symmetric states and actions receive identical rewards.

The return computation for a trajectory τ in MPPI is given by:

$$\texttt{return}(\tau) = \mathbb{E}_\tau \left[\gamma^H Q_\theta(s_H, a_H) + \sum_{t=0}^{H-1} \gamma^t R_\theta(s_t, a_t) \right] \tag{6.3}$$

where H is the planning horizon, γ is the discount factor, and Q_θ is the learned value function. This return function is G-invariant when all its components satisfy the appropriate symmetry constraints.

The integration of orbit sampling into MPPI requires careful consideration of how to generate trajectory samples. Rather than sampling completely independent action sequences, we sample base trajectories and generate their orbits under the symmetry group. This approach ensures that the trajectory distribution always respects the underlying symmetries of the problem, maintaining equivariance throughout the optimization process.

Implementation with TDMPC

We demonstrate the practical application of our approach by integrating orbit sampling with Temporal Difference Model Predictive Control (TDMPC), a state-of-the-art model-based reinforcement learning algorithm that exemplifies modern continuous control methods. TDMPC combines the benefits of model-free and model-based approaches, using learned dynamics models for planning while maintaining the sample efficiency of temporal difference learning. Critically, TDMPC uses MPPI for action selection, making it an ideal testbed for our equivariant sampling methodology.

The integration of orbit sampling into TDMPC addresses a fundamental limitation in model-based RL: while we can design equivariant dynamics models and value functions, the sampling-based action selection breaks equivariance in practice. Our equivariant version of TDMPC preserves all the algorithmic benefits of the original method while adding the symmetry-awareness that enables improved generalization and sample efficiency. The key insight is that by augmenting the action samples with their group orbits, we ensure that the planner considers all symmetric variants of each sampled trajectory, maintaining exact equivariance even with finite samples.

This implementation demonstrates that symmetry-aware planning is not limited to discrete action spaces or deterministic policies, but can be effectively extended to the continuous control setting that dominates modern model-based RL.

6.1.3 Experimental Validation

Our experimental evaluation demonstrates the effectiveness of orbit sampling across multiple domains, from simple coordinate regression tasks to complex continuous control environments. The results consistently show that preserving symmetry through our sampling strategy leads to improved performance and better generalization.

Coordinate Regression Results

In the coordinate regression domain, orbit sampling demonstrates significant improvements over standard sampling approaches. The performance gains scale directly with the degree of symmetry present in the problem, validating our theoretical predictions.

Table 6.1: Performance comparison on symmetric continuous control tasks.

Environment	Standard MPPI		Equivariant MPPI	
	Success Rate	Sample Efficiency	Success Rate	Sample Efficiency
Pushing Task	72.3 ± 5.2	$1.2\times$	$\mathbf{84.7 \pm 3.1}$	$\mathbf{2.3}\times$
Navigation	68.9 ± 4.7	$1.0\times$	$\mathbf{79.4 \pm 2.9}$	$\mathbf{1.8}\times$

When the underlying problem exhibits strong symmetries, our approach achieves substantially better accuracy with fewer samples, highlighting the importance of respecting these geometric properties.

The visualization of energy landscapes reveals why orbit sampling succeeds where traditional methods struggle. Standard sampling often produces asymmetric action distributions that fail to capture the underlying problem structure, while orbit sampling maintains perfect symmetry by construction. This structural preservation translates directly into improved optimization performance.

Continuous Control Environments

Our evaluation on symmetric manipulation and navigation tasks reveals three key advantages of equivariant MPPI over traditional approaches. First, the algorithm achieves faster convergence during training, requiring fewer episodes to reach optimal performance. This improvement stems from the reduced effective search space that results from properly leveraging symmetries.

Second, equivariant MPPI demonstrates superior sample efficiency, particularly in data-limited regimes where the benefits of structural bias become most apparent. The algorithm learns more quickly because it can generalize knowledge across symmetric configurations, effectively increasing the value of each training sample.

Third, the approach shows remarkable improvements in generalization to rotated and reflected versions of the training environments. While standard MPPI often requires retraining when confronted with transformed environments, our equivariant version maintains performance across all symmetric variants of the original problem.

6.2 Spatial Maps as Compositional Abstractions

While Chapters 3-5 focused on object-centric representations, not all environments decompose naturally into objects. Warehouse navigation, for instance, focuses on spatial layout (corridors, intersections) rather than individual items. Spatial maps provide an alternative compositional abstraction where local features compose hierarchically into rooms, buildings, and environments.

This section provides a brief preview. The complete technical treatment appears in Chapter 9 (Extensions Part II), where we develop end-to-end map-based planning for zero-shot navigation from abstract 2D maps to 3D environments [Zhao and Wong, 2024].

6.3 Chapter Discussion

This chapter presented two extensions beyond the core framework. Equivariant sampling (Section 6.1) shows that geometric symmetries apply to continuous action generation—orbit sampling maintains SO(2) equivariance while preserving the stochastic exploration needed for sampling-based control. Spatial maps (Section 6.2) demonstrate compositional structure beyond objects—hierarchical spatial representations where local features compose into rooms and buildings. The complete technical treatment appears in Chapter 9.

Part I: Learning with Structure. Part I demonstrated that compositional structure—whether from objects, symmetries, or spatial hierarchies—enables systematic generalization. The progression from object-centric planning (Ch 3) to symmetry-aware planning (Ch 4-5) to continuous control (Ch 6) shows these principles apply across different settings and representational choices.

Part II

Planning and Agent Integration

Chapter 7

Scaling Differentiable Planning with Implicit Differentiation

A core thesis principle is that harder problems require more compute at decision time—not more training data, but more search at inference. Differentiable planning implements this principle by learning compact representations that enable efficient value iteration [Tamar et al., 2016b, Schrittwieser et al., 2019, Oh et al., 2017]. Like traditional planning algorithms that allocate arbitrary computation to find solutions, differentiable planners should run more iterations for harder tasks without retraining. For instance, learning to navigate mazes or play Atari games with minimal supervision demonstrates how learned models can preserve value equivalence while discovering abstract state representations.

However, current differentiable planning faces a fundamental bottleneck that undermines this principle. Training requires backpropagating gradients through all planning iterations, coupling forward computation with backward passes. This limits practical planning to approximately 20-30 steps due to memory constraints and gradient instability [Tamar et al., 2016b, Oh et al., 2017]—far too few for the long-horizon robotic tasks that Part I's compositional abstractions (object-centric world models, symmetry-aware planners, geometric graph networks) enable. Without solving this scalability challenge, the structured representations from Chapters 3-6 cannot deploy at realistic planning horizons.

This chapter addresses the scalability problem through implicit differentiation. The core challenge in both planning and modern language modeling is similar: how to enable deep iterative computation without proportional training cost. Language models face analogous issues—transformer architectures stack many layers to build sophisticated representations, but backpropagation through all layers couples training memory to model depth. Deep Equilibrium Models (DEQ) apply implicit differentiation to transformers, treating the network as computing a fixed point rather than explicitly stacking layers [Bai et al., 2019]. This enables training transformers with constant memory regardless of effective depth, successfully demonstrated on large-scale language modeling benchmarks like WikiText-103.

For planning, instead of unrolling the entire iterative value iteration process, we

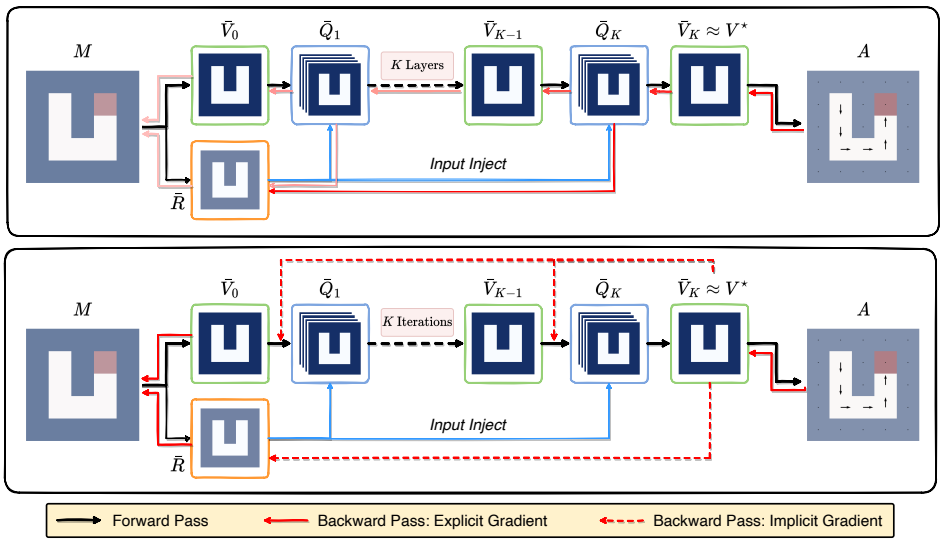

Figure 7.1: An overview of VIN, a planner with algorithmic differentiation, and ID-VIN, our proposed planner with implicit differentiation. Lighter colors for the backward pass with algorithmic differentiation (solid red arrows) indicate larger backpropagation depth. For backward passes with implicit differentiation, the dashed red arrows start from the solved equilibrium $V^\star$ and end at each forward layer (black arrows).

compute gradients directly through the fixed-point conditions that planning algorithms satisfy—the Bellman equations. This decouples forward and backward passes: models can plan for 100+ steps at inference time while maintaining constant-cost gradient computation during training, regardless of planning horizon [Bai et al., 2019, Amos and Yarats, 2019]. The principle parallels modern trends in large language models where test-time compute can scale independently of training cost—models use more reasoning steps for harder problems without retraining. Here, planning algorithms similarly allocate more iterations for complex tasks while training remains efficient.

We apply this approach to Value Iteration Networks and their variants [Tamar et al., 2016b, Lee et al., 2018, Zhao et al., 2022b], developing Implicit Differentiable Planning (IDP). The decoupling brings several benefits: forward passes can scale to 100+ iterations while backward passes maintain constant cost, models are no longer constrained to differentiable solvers, and intermediate computations can be reused across passes. Experiments demonstrate that IDP enables training on 50×50 grids (compared to the previous 15×15 limit), reduces backward pass time by 5-10×, and achieves superior performance on long-horizon navigation and manipulation tasks.

This chapter is based on our paper "Implicit Differentiable Planning" published at ICLR 2023.

7.1 Related work

Differentiable Planning In this paper, we use *differentiable planning* to refer to planning with neural networks, which can also be named *learning to plan* and may be

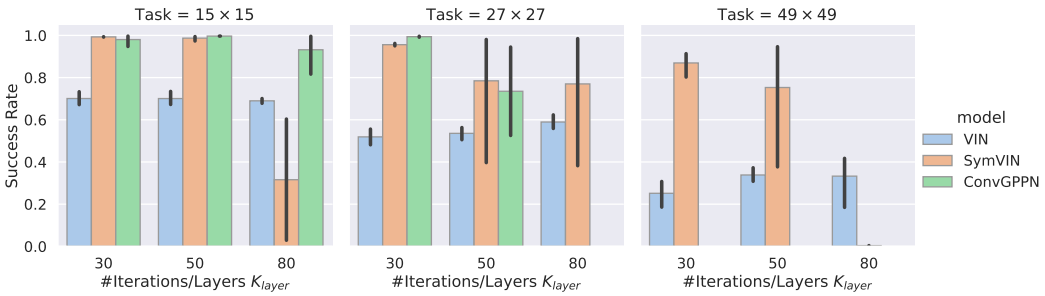

Figure 7.2: Demonstration of differentiable planners with algorithmic differentiation, which cannot scale up to large tasks or more iterations, due to coupled forward and backward pass.

viewed as a subclass of *integrating planning and learning* [Sutton and Barto, 2018]. It is promising because it can be integrated into a larger differentiable system to form a closed loop. Grimm et al. [2020, 2021] propose to understand model-based planning algorithms from value equivalence perspective. Value iteration network (VIN) [Tamar et al., 2016b] is a representative work that performs value iteration using convolution on lattice grids, and has been further extended [Niu et al., 2017, Lee et al., 2018, Chaplot et al., 2021, Deac et al., 2021] and Abstract VIN [Schleich et al., 2019]. Other than using convolution network, the work on combining learning and planning includes [Oh et al., 2017, Karkus et al., 2017, Weber et al., 2018, Srinivas et al., 2018, Schrittwieser et al., 2019, Amos and Yarats, 2019, Wang and Ba, 2019, Guez et al., 2019, Hafner et al., 2020a, Pong et al., 2018, Clavera et al., 2020].

Implicit Differentiation Beyond computing gradients by following the forward pass layer-by-layer, the gradients can also be computed with *implicit differentiation* to bypass differentiating through some advanced root-find solvers. This strategy has been used in a body of recent work [Chen et al., 2019, Bai et al., 2019, Amos and Kolter, 2019, Ghaoui et al., 2020]. Particularly related, Bai et al. [2019] propose *Deep Equilibrium Models (DEQ)* that decouples the forward and backward pass and solve the backward pass iteratively also through a fixed-point system. Winston and Kolter [2021] study the convergence of fixed point iteration in a specific type of deep network. Amos and Kolter [2019] formalize optimization as a layer, and Amos and Yarats [2019] further apply the idea to iterative LQR. Gehring et al. [2021] theoretically study gradient dynamics of implicit parameterization of value function through the Bellman equation. Nikishin et al. [2021] similarly use implicit differentiation, while they explicitly solve the backward pass and only work on small-scale tasks because explicit solving is not scalable. Bacon et al. [2019] instead focus on a Lagrangian perspective. Our work is focused on scalability and convergence stability on differentiable planning, and experiments with challenging tasks in simulation to empirically justify the approach.

7.2 Differentiable Planning with algorithmic differentiation

Background: Value Iteration Networks. Value iteration is an instance of the dynamic programming (DP) method to solve Markov decision processes (MDPs). It iteratively applies the Bellman (optimality) operator until convergence, which is based on the following Bellman (optimality) equation: $Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')$ and $V(s) = \max_a Q(s,a)$.

Tamar et al. [2016b] used a convolution network to parameterize value iteration, named Value Iteration Network (VIN). VINs jointly learn and plan in a latent MDP on the 2D grid, which has the latent reward function $\bar{R} : \mathbb{Z}^2 \to \mathbb{R}^{|\mathcal{A}|}$ and has transition probability $\bar{P}$ represented as $W^V : \mathbb{Z}^2 \to \mathbb{R}^{|\mathcal{A}|}$, which only relies on differences between states. The value function is written as $\bar{V} : \mathbb{Z}^2 \to \mathbb{R}$ and $\bar{Q} : \mathbb{Z}^2 \to \mathbb{R}^{|\mathcal{A}|}$. Value iteration can be written as:

$$\bar{Q}^{(k)}_{\bar{a},i',j'} = \bar{R}_{\bar{a},i,j} + \sum_{i,j} W^V_{\bar{a},i,j} \bar{V}^{(k-1)}_{i'-i,j'-j}, \quad \bar{V}^{(k)}_{i,j} = \max_{\bar{a}} \bar{Q}^{(k)}_{\bar{a},i',j'}. \tag{7.1}$$

If we let $\boldsymbol{f}$ be a single application of the Bellman operator, Eq. 7.1 can be written as:

$$\bar{V}^{(k)} = \boldsymbol{f}(\bar{V}^{(k-1)}, \bar{R}, W^V) \equiv \max_a \bar{R}^a + W^V_{\bar{a}} \star \bar{V}^{(k-1)} \equiv \max_a \bar{R}^a + \texttt{Conv2D}(\bar{V}^{(k-1)}; W^V_{\bar{a}}) \tag{7.2}$$

where convolution $W^V_{\bar{a}} \star V$ is implemented as a 2D convolution layer `Conv2D` with learnable weight W^V. For simplicity, we later use $\boldsymbol{\theta}$ to refer to network weights, and write each iteration as $\boldsymbol{v}_{k+1} = f(\boldsymbol{v}_k, \boldsymbol{r}, \boldsymbol{\theta})$, where $\boldsymbol{r}$ stands for reward map $\bar{R}$ and $\boldsymbol{v}_k$ for value map $\bar{V}^{(k)}$.

Pitfall: Coupled forward and backward pass. The forward computation of VIN iteratively applies the Bellman update $\boldsymbol{f}$. Thus, the optimization needs automatic differentiation: differentiating through multiple layers of forward iterations $\boldsymbol{f} \circ \boldsymbol{f} \circ \ldots \circ \boldsymbol{f}$. Using automatic differentiation for VIN has a major drawback: the forward computation of value iteration ("forward pass") and the computation of its gradients ("backward pass") are *coupled*. That is, if the planning horizon is enlarged, VINs would need larger number of iterations to propagate the value from goals to remaining positions. As used in VIN and GPPN [Tamar et al., 2016b, Lee et al., 2018], it requires intensive memory usage to store every intermediate iterate $V^{(k)}$ to enable automatic differentiation.

To illustrate the effects, we show the performance of *ADP* in Figure 7.2, including three models with increasingly higher memory use and time cost: VIN, SymVIN [Zhao et al., 2022b] and ConvGPPN (a modified version of GPPN [Lee et al., 2018]). They are trained on 15×15, 27×27, and 49×49 path planning tasks. To solve larger mazes, each model consumes more memory while also needing more iterations (x-axes). However, since algorithmic differentiation requires backpropagating gradients layer by layer, with more iterations or larger tasks, some models either diverge or run out of memory (11GB limit). We present further experimental details and analyses in Section 7.4.2.

7.3 Approach: From *Explicit* to *Implicit Differentiation* for Planning

This section introduces a strategy with *implicit differentiation* to resolve the issue by decoupling the forward and backward computation. We first derive implicit differentiation for VIN-based planners, then propose a pipeline for implementing those planners with implicit differentiation. We refer to them as *implicit differentiable planners* (IDP), in contrast to *algorithmic differentiable planners (ADP)* with algorithmic differentiation. We analyze the technical differences and similarities of IDPs vs. ADPs afterward.

7.3.1 Implicit Differentiation for Value Iteration

We derive implicit differentiation for VIN and variants, where each layer f is a step as in value iteration. Since the derivation does not rely on the concrete instantiation of f, we can freely replace f from `Conv2D` with other types of layers. We will introduce these variants in the next subsection.

Implicit differentiation. A fixed-point problem can be solved iteratively by fixed-point iteration and other algorithms. However, as pointed out in [Bai et al., 2019], naively differentiating through the solver would require intensive memory usage, since it needs to store every intermediate iterate $V^{(k)}$ to enable automatic differentiation, as in [Tamar et al., 2016b, Lee et al., 2018]. As also used recently in [Bai et al., 2019, Nikishin et al., 2021, Gehring et al., 2021], another solution is to instead differentiate directly through the fixed point z^* using the implicit function theorem and implicit differentiation. Then, we can skip all of this by decoupling forward (fixed-point iteration as the solver) and backward pass (differentiating through the solver).

We start with the fixed point equation $\boldsymbol{v}^\star = \boldsymbol{f}(\boldsymbol{v}^\star, \boldsymbol{r}, \boldsymbol{\theta})$ from the Bellman optimality equation. Below we use $\boldsymbol{x}$ to stand for either input $\boldsymbol{r}$ or $\boldsymbol{\theta}$. The implicit function theorem tells us that, under some mild conditions of the derivatives ($\boldsymbol{f}$ is continuously differentiable with non-singular Jacobian $\partial \boldsymbol{f}(\boldsymbol{v},\boldsymbol{x}) / \partial \boldsymbol{x}$), $\boldsymbol{v}^\star(\boldsymbol{x})$ is a differentiable function of $\boldsymbol{x}$ locally: $0 = f(\boldsymbol{v}^\star(\boldsymbol{x}), \boldsymbol{x})$.

For fixed point equation, we can assume the fixed point solution $\boldsymbol{v}^\star$ is obtained, thus this can be used to compute the partial derivative w.r.t. to any quantity (input, parameter, etc) $\partial \boldsymbol{v}^\star(\cdot) / \partial(\cdot)$. It avoids backpropagating gradients through the forward fixed-point iteration, which is computationally inefficient and requires considerable memory to store intermediate iteration variables. Additionally, it also allows to even use of non-differentiable operations in the forward pass.

Differentiating both sides of the equation $\boldsymbol{v}^\star = f(\boldsymbol{v}^\star, \boldsymbol{x})$ and applying the chain rule:

$$\frac{\partial \boldsymbol{v}^\star(\cdot)}{\partial(\cdot)} = \frac{\partial f(\boldsymbol{v}^\star(\cdot), \boldsymbol{x})}{\partial(\cdot)} = \frac{\partial f(\boldsymbol{v}^\star, \boldsymbol{x})}{\partial \boldsymbol{v}^\star} \frac{\boldsymbol{v}^\star(\cdot)}{\partial(\cdot)} + \frac{\partial f(\boldsymbol{v}^\star, \boldsymbol{x})}{\partial(\cdot)}, \tag{7.3}$$

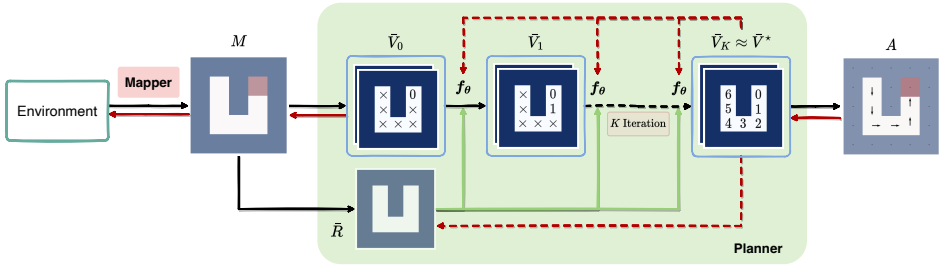

Figure 7.3: The proposed pipeline of implicit differentiable planning by using implicit differentiation.

where we use $(\cdot)$ to denote an arbitrary variable. Rearranging terms:

$$\frac{\partial \boldsymbol{v}^\star(\cdot)}{\partial(\cdot)} = \left(I - \frac{\partial f(\boldsymbol{v}^\star, \boldsymbol{x})}{\partial \boldsymbol{v}^\star}\right)^{-1} \frac{\partial f(\boldsymbol{v}^\star, \boldsymbol{x})}{\partial(\cdot)}. \tag{7.4}$$

Solving backward pass. To integrate into a deep learning framework for automatic differentiation, two quantities are needed: VJP (vector-Jacobian product) and JVP (Jacobian-vector product) [Gilbert, 1992]. Nevertheless, the computation of the inverse term $(I - {}^{\partial f(\boldsymbol{v}^\star, \boldsymbol{x})}/_{\partial \boldsymbol{v}^\star})^{-1}$ can be a major bottleneck due to its dimension ($O(d^2)$ or $O(m^4)$, where $d = m^2$ is the matrix width and m is the map size) [Bai et al., 2019, 2021]. Additionally, when applied to VINs, we concatenate a policy layer that maps the final equilibrium $\boldsymbol{v}^\star \in \mathbb{R}^{m \times m}$ to action logits $\mathbb{R}^{m \times m}$ and compute the cross-entropy loss $\ell(\cdot)$ [Tamar et al., 2016b]. Thus, the derivative of loss is:

$$\frac{\partial \ell}{\partial(\cdot)} = \frac{\partial \ell}{\partial \boldsymbol{v}^\star} \frac{\partial \boldsymbol{v}^\star(\cdot)}{\partial(\cdot)} = \frac{\partial \ell}{\partial \boldsymbol{v}^\star} \left(I - \frac{\partial f(\boldsymbol{v}^\star, \boldsymbol{x})}{\partial \boldsymbol{v}^\star}\right)^{-1} \frac{\partial f(\boldsymbol{v}^\star, \boldsymbol{x})}{\partial(\cdot)}, \tag{7.5}$$

Defining $\boldsymbol{w}$ as follows forms a linear fixed-point system [Bai et al., 2019]:

$$\boldsymbol{w}^\top \triangleq \frac{\partial \ell}{\partial \boldsymbol{v}^\star} \left(I - \frac{\partial f(\boldsymbol{v}^\star, \boldsymbol{x})}{\partial \boldsymbol{v}^\star}\right)^{-1}; \quad \boldsymbol{w}^\top = \boldsymbol{w}^\top \frac{\partial f(\boldsymbol{v}^\star, \boldsymbol{x})}{\partial \boldsymbol{v}^\star} + \frac{\partial \ell}{\partial \boldsymbol{v}^\star}. \tag{7.6}$$

This backward pass fixed-point equation can also be solved by a generic fixed-point solver or root finder. Then, we can substitute the solution $\boldsymbol{w}$ back: ${}^{\partial \ell}/_{\partial(\cdot)} = \boldsymbol{w}^\top {}^{\partial f(\boldsymbol{v}^\star, \boldsymbol{x})}/_{\partial(\cdot)}$. The computation is then purely based on VJP and JVP. In summary, an IDP needs to solve both the (nonlinear) *forward* fixed-point system and the (linear) *backward* fixed-point system, as in DEQ.

7.3.2 A Pipeline of Implicit Differentiable Planning

We can derive variants of VIN using implicit differentiation by abstracting out the implementation of value iteration layer. In this section, we propose a generic implicit planning pipeline to extend our approach to Gated Path Planning Networks (GPPN) [Lee et al., 2018] and Symmetric VIN (SymVIN) [Zhao et al., 2022b]. Spatial Planning Transformers (SPT) [Chaplot et al., 2021] also fits into the pipeline, but it performs less well (see the complete paper version for details).

Figure 7.3 shows the general pipeline, where the network layer $\boldsymbol{f}_\theta$ can be replaced by any single layer that is capable of iterating values. The pipeline follows VIN and GPPN, where for 2D path planning a map $\mathbb{Z}^2 \to \{0, 1\}$ is provided, and the planners' output actions (their logits) for each position $\mathbb{Z}^2 \to \mathbb{R}^{|\mathcal{A}|}$. All these tasks can be represented as taking a form of map "signal" over grid $\mathbb{Z}^2$, as previously been done [Zhao et al., 2022b, Chaplot et al., 2021].

Planner instantiations. We now introduce the instantiations of implicit planners one by one. We focus on the value iteration part (omit map input and action output), and all planners follow the form $\bar{V}^{(k+1)} = f_{\boldsymbol{\theta}}(\bar{V}^{(k)}, \bar{R})$ (bars omitted later). There are two design principles: (1) input inject ($\bar{R}$ must be input, as input $\boldsymbol{x}$) and (2) weight-tied ($\boldsymbol{\theta}$ is shared across layers f), as also used in DEQ [Bai et al., 2019]. Specifically, the purpose of input inject is that fixed-point solution $\bar{V}^\star$ does not depend on initialization $\bar{V}^{(0)}$, so we must pass information of the map through input inject (by $\bar{R}$).

(i) **ID-VIN** uses regular 2D translation-equivariant convolution layer, where $f(V, R) = \max_a R^a + \texttt{Conv2D}(V; \theta)$. *(ii)* **ID-SymVIN** aims to integrate symmetry into planning and uses equivariant $E(2)$-steerable CNN [Weiler and Cesa, 2021]. It has similar form to VIN and just replaces `Conv2D` with `SteerableConv`, thus the form is $f(V, R) = \max_a R^a + \texttt{SteerableConv}(V; \theta)$.

(iii) **ID-ConvGPPN** is based on our modified version of GPPN [Zhao et al., 2022b], where we (1) use GRU since it has a single input with form $z' = \text{GRU}(z, x)$ and is easier to integrate into our current form, (2) replace all fully connected layers with convolution layers, and (3) inject R to every step. The result is that every layer is a ConvGRU, instead of LSTM in GPPN: $f(V, R) = \texttt{ConvGRU}(V, R; \theta)$. Note that the GPPN variants do not have max in each iteration anymore and directly take reward R to the recurrent cell [Lee et al., 2018].

Mapper Layer. We can handle tasks with more challenging input, such as *visual navigation* and *workspace manipulation* [Lee et al., 2018, Chaplot et al., 2021, Zhao et al., 2022b], by learning an additional mapping network (*mapper*) to first map the input to a 2D map. Further details about environments and mapper implementation are in the complete paper version.

Optimization. We build upon the deep equilibrium model (DEQ) and relevant work [Bai et al., 2019, 2020, 2021], which includes several effective techniques. By representing the implementation of value iteration as fixed-point solving, we have the flexibility to use any fixed-point or root solver for $f(\boldsymbol{v}, \boldsymbol{\theta}) = \boldsymbol{v}$ or $\boldsymbol{v} - f(\boldsymbol{v}, \boldsymbol{\theta}) = 0$. A straightforward solver is forward iteration, as used in VIN's feedforward network [Tamar et al., 2016b]. However, recent work has employed Anderson's or Broyden's method [Bai et al., 2019, 2020]. In our experiments, we compare the use of forward iteration and the Anderson solver in both forward and backward passes. Notably, the SymVIN architecture requires the entire forward pass to be equivariant, so extra attention is necessary when designing the forward solver. Further details and results can be found in the complete paper version.

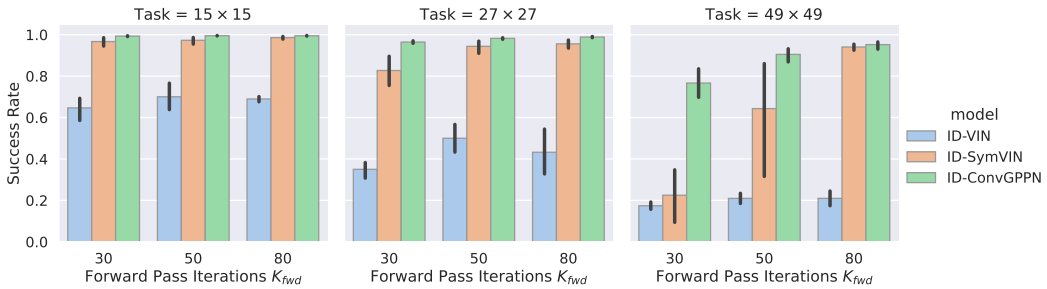

Figure 7.4: Performance of implicit differentiable planners on 2D navigation tasks. Since the forward and backward pass are decoupled, the IDPs can keep consistent runtime and memory cost for backward pass while have forward pass of value iteration converged for large tasks and iterations.

7.3.3 Implicit vs. Explicit Differentiable Planners

Underlying computational similarity. The gradient computation is done by automatic differentiation [Gilbert, 1992]. For *algorithmic differentiation*, the gradients are computed through direct backpropagation and the implementation is also based on efficiently computing vector-Jacobian product (VJP) and Jacobian-vector product (JVP) [Bai et al., 2019]. Christianson [1994] studied automatic differentiation for implicit differentiation of fixed-point system. The only difference is the number of operations required and that implicit differentiation is based on the Jacobian at the equilibrium. The detailed derivation is available in the complete paper version.

Comparison: Implicit vs. algorithmic differentiation. In Figure 7.4, we compare the performance of three IEPs with different numbers of iterations in the forward pass. Unlike the ADPs shown in Figure 7.2, our IDPs converge stably with larger forward-pass iterations (horizontal axis), while ADPs sometimes diverge on large iterations. Note that we should not directly compare IDPs and ADPs with the same number of forward iterations since they refer to different things: IDPs compute an equilibrium and use it for backward pass (more forward iterations would be better), while for ADPs # iterations = # layers (more iterations/layers leads to instability). Further analyses in Section 7.4.2.

Tradeoff: The quality of equilibrium. Theoretically, IDPs have a constant cost of the backward pass with respect to the forward planning horizon. However, the *backward pass* requires the Jacobian of the final equilibrium $\boldsymbol{v}^\star \approx \boldsymbol{v}_K$ from the forward pass. If the equilibrium $\boldsymbol{v}^\star$ is not solved reasonably well, the backward pass in Eq. 7.6 would iterate based on an inaccurate Jacobian $\partial f(\boldsymbol{v}^\star, x)/\partial \boldsymbol{v}^\star$, which would cause poor performance. In contrast, because ADPs compute exact gradients by backpropagation through layers, they do not suffer from this issue. Additionally, fewer iterations (layers) would also alleviate their convergence issues.

Empirically, with fewer *forward iterations* (e.g., 10), we observe a performance drop from IDPs. Although ADPs also perform worse with fewer layers, they have a smaller drop compared to IDPs. However, when scaling up to more forward iterations K_{fwd} (e.g. ≥ 30 iterations, which are necessary for maps $\geq 27 \times 27$) IDPs may solve the equilibrium well enough and are more favorable because of their efficient backward pass.

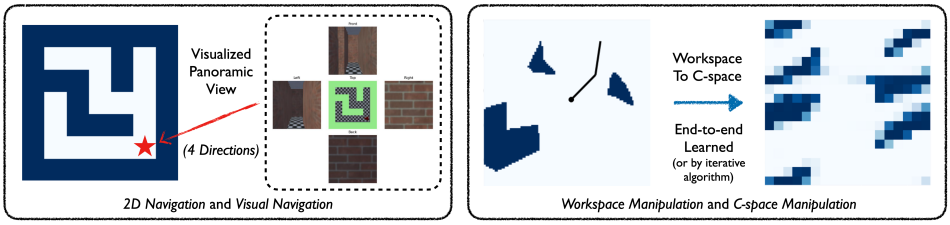

Figure 7.5: **(Left)** We randomly generate occupancy grid $\mathbb{Z}^2 \to \{0, 1\}$ for **2D navigation**. For **visual navigation**, each position provides $32 \times 32 \times 3$ egocentric panoramic RGB images in 4 directions for each location $\mathbb{Z}^2 \to \mathbb{R}^{4\times32\times32\times3}$. One location is visualized. **(Right)** The top-down view (left) is the *workspace* of a 2-DOF manipulation task. For **workspace manipulation**, it is converted by a mapper layer to configuration space, shown in the right subfigure. For **C-space manipulation**, a ground-truth C-space is provided to planners.

Moreover, we find that using around $K_{\text{bwd}} \approx 15$ iterations for backward pass works well enough consistently across different map sizes (15×15 through 49×49). Since both algorithmic and implicit differentiation use a similar amount of vector-matrix products, using more than $K_{\text{layer}} \geq 15 \approx K_{\text{bwd}}$ would consume more resources and favor IDPs.

7.4 Empirical Analysis

We present more results on convergence, scalability, generalization, and efficiency, which extends the study in previous sections on comparing implicit vs. algorithmic differentiable planners.

7.4.1 Environments and Setup

Environments and datasets. We run implicit and algorithmic differentiable planners on four types of tasks: (1) **2D navigation**, (2) **visual navigation**, (3) 2 degrees of freedom (2-DOF) **configuration space manipulation**, and (4) **2-DOF workspace manipulation**. These tasks require planning on either *given* (2D navigation and 2-DOF configuration-space manipulation) or *learned* maps (visual navigation and 2-DOF workspace manipulation), where the maps are 2D regular grid as in prior work [Tamar et al., 2016b, Lee et al., 2018, Chaplot et al., 2021]. To learn maps, a planner needs to jointly learn a mapper that converts egocentric panoramic images (visual navigation) or workspace states (workspace manipulation) into a 2D grid. We follow the setup in [Lee et al., 2018, Chaplot et al., 2021] (see the complete paper version for further details). In both cases, we randomly generate training, validation and test data of $10K/2K/2K$ maps for all map sizes. For all maps, the action space is to move in 4 $\circlearrowleft$ directions: $\mathcal{A} =$ `(north,west,south,east)`.

Training and evaluation. We report success rate and training curves over 5 seeds. The training process (on given maps) follows [Tamar et al., 2016b, Lee et al., 2018, Zhao et al., 2022b], where we train 60 epochs with batch size 32, and use kernel size $F = 3$ by default. We use RTX 2080 Ti cards with 11GB memory for training, thus we use 11GB as the memory limit for all models.

7.4.2 Convergence and Scalability

In the previous sections with Figure 7.2 and 7.4, we have presented quantitive analysis of IDPs and ADPs in terms of convergence with more iterations and scalability on larger tasks. Here, we provide the detailed setup of the experiment and put the attention more on the qualitative side.

Setup. We train all models on 2D maze navigation tasks with map sizes 15×15, 27×27, and 49×49. We use $K_{\text{layer}} = 30, 50, 80$ iterations for *ADPs*, which is effectively the number of layers in their networks. Correspondingly, for *IDPs*, we choose to use *forward iteration* solver for forward pass and Anderson solver for backward pass. We fix the number of iterations of backward solver as $K_{\text{bwd}} = 15$ and of forward solver as $K_{\text{fwd}} = 30, 50, 80$.

Results. We examine results by algorithm and focus on their trend with iteration number (x-axis) and map size (column), not just the absolute numbers. The conclusion already mentioned in the above section: Beyond an intermediate iteration number (around 30-50), IDPs are more favorable because of scalability and computational cost. We present other analyses here.

We start from ConvGPPN and ID-ConvGPPN, which perform the best in ADPs and IDPs class, respectively. They also have the most number of parameters and use greatest time because of the gates in ConvGRU units. As shown in Figure 7.2, this also caused two issues of ConvGPPN: scalability to larger maps/iterations (out of memory for 27×27 80 iterations and 49×49 50 and 80 iterations), and also convergence stability (e.g. 27×27 50 iterations).

For SymVIN and ID-SymVIN, they replace `Conv2D` with `SteerableConv`, with computational cost slightly higher than VIN and much lower than ConvGPPN. Thus, they can successfully run on all tasks and iteration numbers. However, we find that explicit SymVIN may diverge due to bad initialization, and this is more severe if the network is deeper (more iterations), as in Figure 7.2's 50 and 80 iterations. Nevertheless, ID-SymVIN alleviates this issue, since implicit differentiable planning decouples forward and backward pass, so the gradient computation is not affected by forward pass.

Furthermore, VIN and ID-VIN are surprisingly less affected by the number of iterations and problem scale. We find their forward passes tends to converge faster to reasonable equilibria, thus further increasing iteration does not help, nor break convergence as long as memory is sufficient.

Forward and backward runtime. We visualize the runtime of IDPs and ADPs in Figure 7.6. For IDPs, we use the forward-iteration solver for the forward pass and Anderson solver for the backward pass. Note that in the bottom left, we intentionally plot backward runtime vs. forward pass iterations. This emphasizes that IDPs decouple forward and backward passes because the backward runtime does not rely on forward pass iterations. Instead, for ADPs, value iteration is done by network layers, thus the backward pass is coupled: the runtime increases with depth and some runs failed due to limited memory (11GB, see missing dots).

Therefore, this set of figures shows better scalability of IDPs (no missing dots – out of memory – and constant backward time). In terms of absolute time, the forward

Figure 7.6: The runtime (in seconds) on 2D navigation tasks with size 15×15, 27×27, and 49×49, averaged over 5 seeds. The ADPs are on the *left* six figures and the IDPs on the *right*. *Missing dots* are due to out of memory caused by algorithmic differentiation. The *upper* row is for forward pass runtime, and the *lower* row is for backward runtime. The horizontal axes mean differently: (1) ADPs: the number of layers K_{layer}, also number of iterations, and (2) IDPs: the forward pass iterations K_{fwd}.

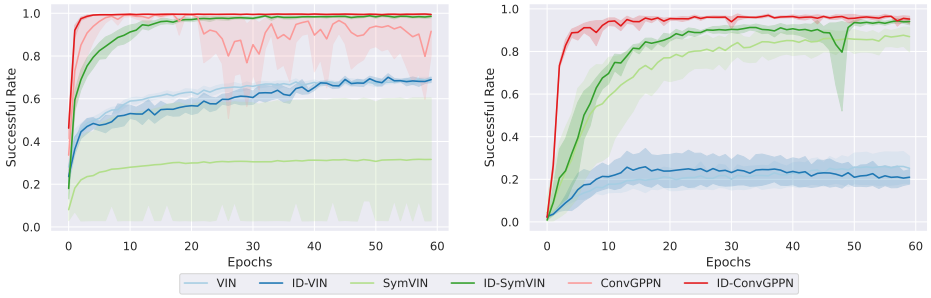

Figure 7.7: **(Left)** Training curves on 2D maze navigation 15×15 maps, with 80 layers for ADPs and 80 iterations for IDPs. **(Right)** Training curves on 49×49 maps, with 30 layers for ADPs (due to scalability issue) and 80 iterations for IDPs.

runtime of IDPs when using the *forward solver* is comparable with successful ADPs.

7.4.3 Training Performance

Setup. Beyond evaluating generalization to novel maps, we compare their training efficiency with learning curves. Each learning curve is aggregated over 5 seeds, which are from the models in the above section. The learning curves are for all planners on 15×15 maps (Figure 7.7 left) and 49×49 maps (Figure 7.7 right, 30 layers for ADPs – due to scalability issue – and 80 iterations for IDPs).

Results. On 15×15 maps, we show $K_{\text{layer}} = 80$ layers for ADPs and $K_{\text{fwd}} = 80$ iterations for IDPs. ID-ConvGPPN performs the best and is much more stable than its ADP counterpart ConvGPPN. ID-SymVIN learns reliably, while SymVIN fails to converge due to instability from 80 layers. ID-VIN and VIN are comparable throughout training. On 49×49 maps, we visualize $K_{\text{layer}} = 30$ layers for ADPs (due to their limited scalability) and $K_{\text{fwd}} = 80$ iterations for IDPs. ConvGPPN cannot run at all even for only 30 layers, while ID-ConvGPPN still reaches a near-perfect success

rate. ID-SymVIN learns slightly better than SymVIN and reaches higher asymptotic performance. ID-VIN has a similar trend to VIN, but performs worse overall due to the complexity of the task.

7.4.4 Performance on More Tasks

Table 7.1: Averaged test success rate (%) over 5 seeds for using 10K/2K/2K dataset on the rest of 3 types of tasks. We highlight entry with *italic* for runs with at least one diverged trial (any success rate $< 20\%$).

Type	Methods	18×18 Mani.	36×36 Mani.	Workspace Mani.	Visual Nav.
Explicit	VIN	89.65±7.97	74.75±8.18	80.98±3.84	66.11±8.91
	SymVIN	*55.15*±49.54	*65.72*±47.11	*82.17*±24.72	96.04±4.24
	ConvGPPN	79.71±20.71	*70.55*±36.13	70.23±19.44	*81.76*±31.04
Implicit (ours)	ID-VIN	80.53±6.98	*56.27*±20.92	77.17±7.24	62.53±15.93
	ID-SymVIN	99.63±0.08	98.53±1.42	*87.60*±24.11	*86.41*±30.34
	ID-ConvGPPN	97.28±0.74	93.60±1.68	92.60±1.83	98.91±0.34

Setup. We run all planners on the other three challenging tasks. For **visual navigation**, we randomly generate 10K/2K/2K maps using the same strategy as 2D navigation and then render four egocentric panoramic views for each location from produced 3D environments with *Gym-MiniWorld* [Chevalier-Boisvert, 2018]. For **configuration-space manipulation** and **workspace manipulation**, we randomly generate 10K/2K/2K tasks with 0 to 5 obstacles in workspace. In configuration-space manipulation, we manually convert each task into a 18×18 or 36×36 map (20° or 10° per bin). The workspace task additionally needs a mapper network to convert the 96×96 workspace (image of obstacles) to an 18×18 2-DOF configuration space (2D occupancy grid). Additional details are provided in the complete paper version.

Results. In Table 7.1, due to space limitations, we average over $K_{\text{layer}} = 30, 50, 80$ for ADPs and $K_{\text{fwd}} = 30, 50, 80$ for IDPs. For each task, we present the mean and standard deviation over 5 seeds times three hyperparameters (see the complete paper version for separated results). We italicize entries for runs with at least one diverged trial (any success rate $< 20\%$).

Generally, IDPs perform much more stably. On 18×18 or 36×36 configuration-space manipulation, ID-SymVIN and ID-ConvGPPN reach almost perfect results, while ID-VIN has diverged runs on 36×36 (marked in *italic*). SymVIN and ConvGPPN are more unstable, while VIN even outperforms them and is also better than ID-VIN. On 18×18 workspace manipulation, because of the difficulty of jointly learning maps and potentially planning on inaccurate maps, most numbers are worse than in configuration-space. ID-ConvGPPN still performs the best, and other methods are comparable. For 15×15 visual navigation, it needs to learn a mapper from panoramic images and is more challenging. ID-ConvGPPN is still the best. ID-SymVIN exhibits some failed runs and gets underperformed by SymVIN in these seeds, and ID-VIN is comparable with VIN.

Across all tasks, the results confirm the superiority of scalability and convergence stability of IDPs and demonstrate the ability of jointly training mappers (with algorithmic differentiation for this layer) even when using implicit differentiation for planners.

7.5 Conclusion

This work studies how VIN-based differentiable planners can be improved from an implicit-function perspective: using implicit differentiation to solve the equilibrium imposed by the Bellman equation. We develop a practical pipeline for implicit differentiable planning and propose implicit versions of VIN, SymVIN, and ConvGPPN, which is comparable to or outperforms their explicit counterparts. We find that implicit differentiable planners (IDPs) can scale up to longer planning-horizon tasks and larger iterations. In summary, IDPs are favorable for these cases to ADPs for several reasons: (1) better performance mainly due to stability, (2) can scale up while some ADPs fail due to memory limit, (3) less computation time. On the contrary, if using too few iterations, the equilibrium may be poorly solved, and ADPs should be used instead. While we focus on value iteration, the idea of implicit differentiation is general and applicable beyond path planning, such as in continuous control where Neural ODEs can be deployed to solving ODEs or PDEs.

This computational efficiency unlocked by IDP is crucial for deploying the equivariant planning methods from Part I in real-world robotics. By decoupling forward and backward computational complexity, we enable 10-100x speedup in gradient computation while maintaining planning quality. This scalability is essential for the practical applications demonstrated in the subsequent chapter on belief-space planning, where robots must plan over long horizons in complex environments. The implicit differentiation framework developed here thus serves as a critical bridge between the theoretical elegance of symmetry-aware planning and the computational demands of real-world deployment.

Chapter 8

Planning with Abstract Belief for Integrated Perception and Manipulation

Part I (Chapters 3-6) developed compositional abstractions—object-centric world models, symmetry-aware planners, geometric graph networks—that enable systematic generalization through structured representations. However, these methods assumed full observability: the robot knows all object properties and environment states. This chapter extends the framework to partially observable environments where uncertainty is fundamental.

Task-level planning is critical to achieving long-horizon mobile manipulation tasks [Lozano-Pérez et al., 1989, Yenamandra et al., 2023]. However, many planning methods assume complete knowledge of the environment, including object properties and relations, rendering them inapplicable to partially observable settings where robots must dynamically discover, perceive, and interact with objects while resolving uncertainties. We address the challenge of operating in partially observable environments characterized by (1) uncertain object properties, (2) unknown object instances (number, type, location), and (3) goals specified via ungrounded natural language. In such domains, information gathering becomes crucial alongside manipulation planning.

Handling partial observability on real robots, especially for long-horizon mobile manipulation tasks, is notoriously difficult. These problems are often modeled as Partially Observable Markov Decision Processes (POMDPs) [Kaelbling et al., 1998b]. Compared to tabletop setups with fixed cameras offering near-complete views, mobile robots face inherently greater uncertainty about the environment state and limitations from onboard, viewpoint-dependent perception. One major strategy for tackling POMDPs involves planning in *belief space*, aggregating information into an explicit belief state representing the agent's knowledge and uncertainty. Our work follows this strategy.

While excellent solutions for belief-space planning exist [Kaelbling and Lozano-Perez, 2012a, Garrett et al., 2021, Kurniawati et al., 2008, Platt Jr et al., 2010], they often require sophisticated algorithms, complex belief representations (e.g., full probability distributions), and carefully hand-engineered perception pipelines. An alter-

Figure 8.1: **Example tasks demonstrating various uncertainty levels.** (1) Cup Pick-Place: a fully observable tabletop manipulation task with multiple cups. (2) Empty Cup Removal: requires inspecting cups from above to determine if they are empty before removal. (3) Drawer Cleaning: involves opening drawers to discover and remove objects inside. (4) Sort Weight: requires weighing sealed boxes on a scale to identify and dispose of empty ones. These tasks demonstrate increasing complexity in information gathering, from fully observable scenarios to those requiring several strategic information-gathering steps.

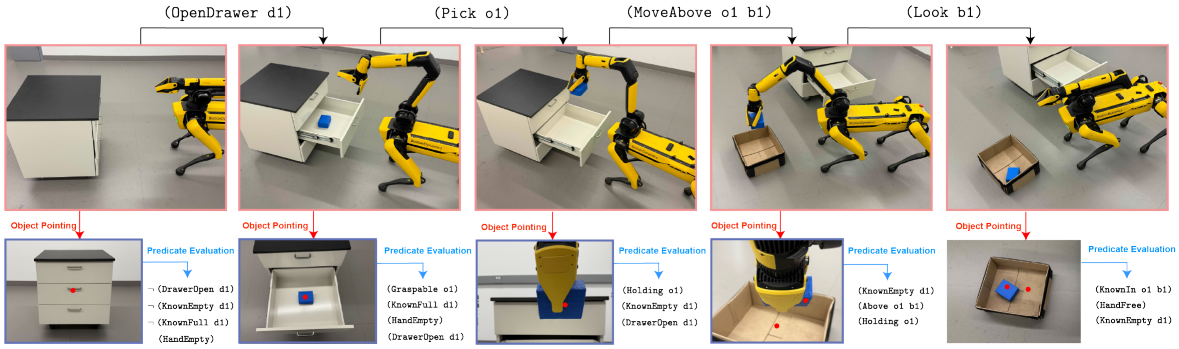

Figure 8.2: **Example plan.** A task to put any object in the drawer into a paper bin. Because the drawer is closed, the robot needs to maintain uncertainty of the environment and plan under uncertainty to achieve the belief goal: `KEmpty+(drawer)` and `Inside(block, box)`. The sequence shows: (1) initial reach to the closed drawer (without knowing if the drawer is empty or not), (2) opening the drawer to reveal a blue block inside and update belief, (3) grasping the block from the drawer, (4) moving the block over the paper bin, and (5) successfully placing the block into the bin. This demonstrates how the robot handles uncertainty through interleaved information gathering (opening drawer to check contents) and manipulation actions (grasping and placing the block).

native strategy involves learning history-conditioned policies directly [Bhattacharya et al., 2020]. Recent foundation model approaches (VLMs, LLMs) typically fall into this category but often struggle with systematic planning under uncertainty and strategic information gathering, especially with limited real-robot data [Ahn et al., 2022b, Huang et al., 2022b].

This chapter illustrates that the core ideas of belief-space planning can be applied relatively straightforwardly to mobile manipulation tasks by leveraging modern, off-the-shelf components: classical planners (adapted via determinization) and Vision-Language Models (VLMs) for evaluating perceptual predicates. We make simplifying assumptions—treating predicate values as True/False/Unknown, assuming perfect local observations, deterministic world-changing actions, and no information loss over time—that enable a lightweight belief representation and planning strategy.

Following Bonet and Geffner [2011], we represent the agent's belief using three-valued logic for predicates, explicitly tracking known vs. unknown states via K-fluents. We formulate planning as finding a sequence of actions to reach a goal belief state (e.g., knowing an object property). Information-gathering actions are modeled systematically. To handle the resulting belief non-determinism tractably, we employ all-outcomes determinization and online replanning [Bonet and Geffner, 2011]. Our integrated pipeline leverages VLMs within this framework as flexible perception modules to evaluate predicate truth values (updating K-fluents) and to ground natural language goals.

Experiments on mobile-manipulation tasks in simulated synthetic environments and on a physical Spot robot demonstrate that the system effectively manages partial observability, dynamically adapts plans, and leverages strategic information gathering more efficiently than end-to-end VLM baselines. This chapter is based on our paper "Seeing is Believing: Belief-Space Planning for Mobile Manipulation with Foundation Models" [Zhao et al., 2025].

8.1 Related Work

8.1.1 Foundation Models for Planning

Recent advances using LLMs and VLMs enable approximate robot planning in diverse environments Ahn et al. [2022b], Hazra et al. [2024], Curtis et al. [2024a], Liang et al. [2023], but evaluations highlight limitations in long-horizon or combinatorial problems when lacking an explicit planning model [Huang et al., 2022a, Silver et al., 2022c]. These foundation models can be used in a "zero shot" way to directly generate plans from natural language goals and visual inputs, but they perform poorly with systematic planning, maintaining long-term context, handling uncertainty, and strategic information gathering [Ahn et al., 2022b, Valmeekam et al., 2023]. These shortcomings are particularly visible in partially observable scenarios requiring multi-step planning and deliberate information gathering.

8.1.2 Belief-Space Planning

Belief-space planning effectively handles uncertainty from partial observability [Kaelbling and Lozano-Perez, 2012b, Kaelbling and Lozano-Pérez, 2013, Kaelbling and Lozano-Pérez, 2017, Garrett et al., 2020, Curtis et al., 2024b, 2022, Kaelbling et al.,

2021]. It models the robot's knowledge as a belief state and plans to achieve belief goals (e.g., "know the lights are off") using both world-changing and information-gathering actions. This is crucial for mobile manipulation in partially known environments with unknown objects or states.

Belief-based systems typically involve a state estimator `SE` for belief updates and a policy π mapping beliefs to actions. Computing a complete policy offline is often intractable since only a tiny subset of possible beliefs will ever be reached. We instead follow an alternative approach that constructs partial plans online and replans if unanticipated observations are obtained [Kaelbling and Lozano-Perez, 2012b]. Beliefs can be modeled as probability distributions or, alternatively, as sets of possible worlds represented compactly using three-valued logic (true, false, unknown) [Ginsberg, 1988, Kleene, 1952, Codd, 1979, Zadeh, 1965, Srivastava et al., 2008], which efficiently extends deterministic models to belief space [Bonet and Geffner, 2011].

Manual specification of planning domains is laborious; predicate and operator invention methods automate learning symbolic representations from data [Silver et al., 2022b, 2023]. Our framework reduces belief-space planning to replanning in a determinized belief state constructed from standard planning operators (Section 8.2). This suggests that techniques for learning standard planning operators might be adaptable to learn the belief-space operators to reduce manual effort.

8.1.3 Object Uncertainty, Search, and Grounding

A key challenge in applying planning methods to partially-observed scenarios is handling uncertainty related to objects, including their existence, properties, and relevance to the task. Typical planning systems often assume a pre-specified domain of objects and properties, limiting their applicability when the set of objects is unknown or properties need active discovery [Kaelbling and Lozano-Pérez, 2013].

One aspect of handling object existence uncertainty is *object search*, where the goal is to locate objects, potentially requiring exploration or manipulation [Wong et al., 2013, 2015]. Much existing work focuses specifically on finding objects of a particular category, often using special-purpose strategies. However, many tasks require gathering information about object *properties* beyond just their location or category (e.g., determining if a container is empty). Classical systems have explored planning to perceive specific properties [Sridharan et al., 2010, Srivastava et al., 2014b, Nie et al., 2016], but integrating general property verification with manipulation planning remains challenging. Our work aims for a more general mechanism for gathering critical property information, integrated within belief-space planning.

Furthermore, interpreting goals and understanding scenes becomes complex when objects are initially unknown. While foundation models like LLMs and VLMs have been used for goal parsing and scene understanding in structured and fully observable environments [Silver et al., 2022c], partially observable settings demand active perception to dynamically discover objects and evaluate their properties [Ahn et al., 2022b]. While Sun et al. [2024] explored LLMs for POMDP planning, their approach relied on leveraging VLM-based perception to implicitly interpret belief-updates and their effects

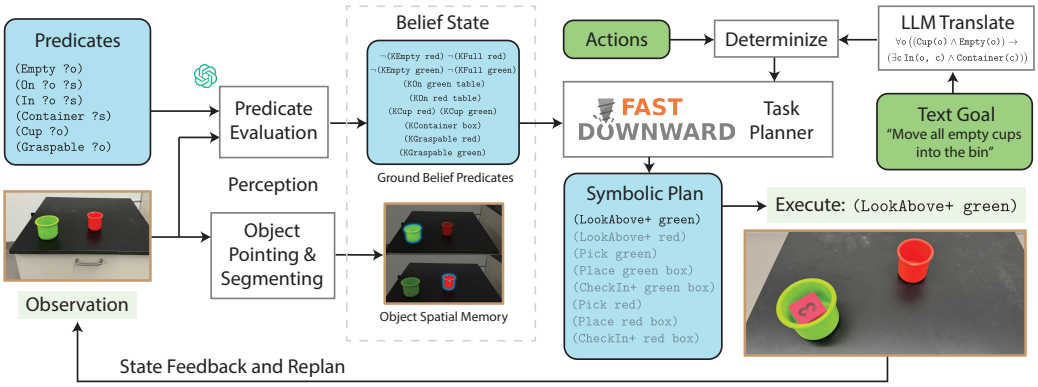

Figure 8.3: **Pipeline overview.** Our system integrates perception, belief-state update, and planning. The example shows a task of moving empty cups to a bin, where the system must evaluate cup properties and plan appropriate manipulation actions. Before runtime, a text goal is first translated into symbolic specifications, which along with actions are determinized for the task planner. During a step of belief state update at runtime, given an observation (images and sensor inputs from a robot), the system performs two parallel processes for: (1) object pointing and segmenting to maintain a spatial memory of objects, and (2) predicate evaluation to ground belief predicates (e.g., `Empty`, `On`). The planner generates a symbolic plan based on the symbolic belief state, and the first action is executed to generate a new observation. The belief state is updated based on the new observation, and the process repeats until the goal is satisfied.

on planning, instead of having an explicit belief space within a belief-space task planning framework. With an explicit representation, one common approach to handling the object uncertainty not addressed in Sun et al. [2024] is using lifted representations (e.g., first-order logic goals like "all empty cups"), which allows generalization of objects of similar type. Each object has a grounded representation (referring to specific object instances like 'cup1'), which require mechanisms for grounding goals and actions as new objects are discovered during execution. Since integrating active perception with belief-space planning is crucial for enabling robots to iteratively gather information and refine their understanding, we chose an explicit representation over relying on VLM to do both belief-state estimation and planning.

8.2 Formulation: Modeling with Uncertainty

The problem of sequential decision-making under uncertainty in partially observable environments, such as mobile manipulation tasks, is typically modeled as a *partially observable Markov decision process (POMDP)* [Kaelbling et al., 1998b].

8.2.1 Modeling the Environment with Uncertainty

We approach this POMDP by converting it to a planning problem in an associated *belief-space MDP*. Our approach involves: (1) explicitly representing the agent's *belief*

state b about the true world state, and (2) performing *online approximate planning* within this belief space, utilizing replanning to handle unexpected observations or outcomes resulting from the discrepancy between the planning model and the execution environment.

We model the partially observable planning problem in terms of: (1) a object-centric belief state b, discussed in §8.2.2; (2) a set of object-parameterized *actions* A that can be executed by the agent, modifying the belief state, discussed in §8.2.4; and (3) a *goal* G, specified as a first-order logical expression over relations that must be `True` in the terminal belief, discussed in §8.2.3.

To make this belief representation and the online planning approach tractable, we adopt the following core simplifying assumptions in our formulation:

- *Perfect Observations:* While observations are partial (local), the information obtained about the observed part of the state is assumed to be accurate and noise-free.
- *Deterministic World Dynamics:* Actions that intentionally change the physical state of the world (world-changing actions, A_w) are assumed to have deterministic effects on the underlying world state, as defined by their models.
- *Static Environment (No Exogenous Changes):* The state of the world only changes due to the robot's actions; there are no external events modifying the environment. Furthermore, we assume no loss of information over time (e.g., forgetting known facts).
- *Separable Action Effects:* We model actions such that their effects can be separated into either changing the physical world state or purely gathering information (updating belief), not both simultaneously.

These assumptions, particularly perfect observations and deterministic dynamics, significantly simplify the belief update and planning process. Unexpected observations and exogenous changes to object locations and properties, which violate these assumptions, are handled at *runtime* via replanning (with updated belief state; see §8.3). Relaxing these assumptions typically requires more complex probabilistic belief representations and planning algorithms [Kaelbling and Lozano-Pérez, 2013, Curtis et al., 2024b].

8.2.2 Belief State Representation

Conceptually, the agent's belief state b represents a set of possible world states consistent with its observations and action history. We explicitly structure this belief state using the following components:

- A set of conjectured *objects* $\mathbb{O}$ currently believed to exist.
- A set of symbolic *predicates* $\mathcal{P}$ (e.g., `Empty(cup)`). They are object-parameterized functions using $\mathbb{O}$, and their values produce the symbolic *state variables*.

- Physical information, such as object memory (e.g., object positions) and robot configuration.

To handle uncertainty, the truth value for any predicate $P \in \mathcal{P}$ applied to objects $\mathbf{o} \subseteq \mathbb{O}$ can be `True` (`T`), `False` (`F`), or `Unknown` (`U`). Following Bonet and Geffner [2011], we represent the agent's knowledge about these three-valued predicates using two binary *K-fluents*. For each belief predicate $P(\mathbf{o})$, we define $K_P(\mathbf{o})$ (known true) and $K_{\neg P}(\mathbf{o})$ (known false). The values regarding $P(\mathbf{o})$ is encoded as: (1) `T` if $K_P(\mathbf{o})$ is true, (2) `F` if $K_{\neg P}(\mathbf{o})$ is true, and (3) `U` if both $K_P(\mathbf{o})$ and $K_{\neg P}(\mathbf{o})$ are false. This allows explicit but determinsitic reasoning about knowns and unknowns. A consequence of our core assumptions (perfect observations, static world, no information loss) is *monotonic knowledge acquisition*: known facts (`T` or `F`) remain known unless explicitly changed by a world-changing action.

8.2.3 Goal Representation

A task goal G is specified as an *achievable* First-Order Logic (FOL) formula over belief-state predicates, typically requiring certain K-fluents to be true (e.g., $K_{\text{Empty}}(\text{cup1})$). Goals can be (1) *grounded*, referring to specific known objects in $\mathbb{O}$, or (2) *lifted*, quantifying over objects possibly including those yet undiscovered, which is crucial for open-world tasks.

The scope of achievable goals is limited by the predicate set $\mathcal{P}$: (1) Goals must be expressible using the defined predicates (or their K-fluents). (2) Goals involving specific unknown objects or properties (e.g., finding a `green_cup` before achieving `Inside(green_cup)`) may implicitly require prerequisite information gathering actions.

8.2.4 Action Modeling and Planning via Determinization

Actions A available to the agent are parameterized operators defined by preconditions and effects on the belief state b. Following our core assumptions (§8.2.1), we separate actions into two types based on their primary effect:

World-Changing Actions (A_w): These actions modify the physical world state deterministically. Their effects on the belief state b update K-fluents to reflect the known outcome. For example, picking up an object `?o`:

```
(:action Pick
 :parameters (?o - object)
 :precondition (and (HandEmpty) (CanGrasp ?o))
 :effects (and ($\lnot$HandEmpty) (Holding ?o)))
```

Information-Gathering Actions (A_i): These actions aim to reduce uncertainty about a predicate $P(\mathbf{o})$ whose value is Unknown ($\neg K_P(\mathbf{o}) \wedge \neg K_{\neg P}(\mathbf{o})$). They only update the belief state by revealing information (making the predicate known True or known False), without directly changing the physical world state. For example, `ObserveEmptiness(?cup)` reveals whether a cup is empty.

Planning Model via Determinization: Information-gathering actions (A_i) introduce non-determinism into the belief state transition, as the observation outcome (True or False) is unknown beforehand. This complicates planning, potentially requiring complex conditional plans. To enable the use of efficient classical planners, we employ an *all-outcomes determinization* approach [Bonet and Geffner, 2011]. This involves (1) *compiling* each information-gathering action A_i (like `ObserveEmptiness`) into two deterministic PDDL actions, A_i^+ and A_i^-, one for each possible observation outcome (e.g., observing True or observing False), and (2) *optimistic planning*, choosing the outcome action (A_i^+ or A_i^-) that appears most beneficial for reaching the goal according to the deterministic model. For instance, `ObserveEmptiness(?o)` is split into two actions:

```
(:action ObserveEmptiness+ ; Outcome is True
 :parameters (?o - object)
 :precondition (and (CanObserve ?o)
                    (not (KEmpty+ ?o)) (not (KEmpty- ?o)))
 :effect (KEmpty+ ?o)) ; Effect: Known True

(:action ObserveEmptiness- ; Outcome is False
 :parameters (?o - object)
 :precondition (and (CanObserve ?o)
                    (not (KEmpty+ ?o)) (not (KEmpty- ?o)))
 :effect (KEmpty- ?o)) ; Effect: Known False
```

Here, `KEmpty+` and `KEmpty-` represent K_{Empty} (known true) and $K_{\neg\text{Empty}}$ (known false), respectively. The precondition `(not (KEmpty+ ?o)) (not (KEmpty- ?o))` ensures the action is only applicable when the emptiness status of `?o` is Unknown. This determinization allows plan generation using a classical planner, but the optimism means that discrepancies between the planned outcome and the actual observation during execution necessitate runtime *replanning* to correct the course of action (§8.3).

Executing Symbolic Plans: We assume that each symbolic action in a generated plan corresponds to an executable skill or program. These skills are responsible for achieving the action's specified effects in the physical world. Internally, they might employ methods like motion planning or utilize learned policies to realize the desired outcome, while their failures will be handled via replanning, similar to [Kumar et al., 2024b].

8.3 Implementation: Planning under Uncertainty

In this section, we detail the runtime pipeline for planning under uncertainty on a mobile-manipulation robot, building upon the formulation described in §8.2. We term our system BKLVA: ***B**elief-space planning with **K**-fluents, **L**anguage-based goal-grounding, **V**LM-based perception and estimation, and information-gathering **A**ctions.* The execution pipeline takes as input (see Algorithm 1): (1) a PDDL domain definition corresponding to the belief-space actions $\mathcal{A}$, (2) a natural language goal g_{text},

Algorithm 1 Belief-Space Planning and Execution (§8.3)

Require: Initial belief state b_0 with initial objects $\mathbb{O}_0$, text goal g_{text}, predicates $\mathcal{P}$, actions $\mathcal{A}$

$g \leftarrow \texttt{Translate}(g_{\text{text}}, \mathcal{P}, \mathbb{O}_0)$ ▷ Translation (§8.3)
$b \leftarrow b_0$
while $\neg\texttt{Satisfied}(g, b)$ **do**
  $p \leftarrow \texttt{Plan}(b, g, \mathcal{A})$ ▷ Generate plan (§8.3.4)
  if $p =$ None **then return** False ▷ Goal is infeasible.
  for a **in** p **do**
    $o \leftarrow \texttt{Execute}(a)$ ▷ Get observation (§8.3.1)
    $b \leftarrow \texttt{BeliefUpdate}(b, a, o)$ ▷ Update (§8.3.2)
    if $\neg\texttt{ExpectedEffects}(a, b)$ **then**
      break ▷ Replan with updated belief (§8.3.4)
    end if
  end for
end while
return True ▷ Goal achieved

(3) pretrained foundation models for perception (detailed in §8.3.1), and (4) an initial belief state b_0.

The execution then proceeds in an observe-update-plan-execute cycle, as outlined in Algorithm 1:

*(1) Perception and (2) Belief Update (**`Observe`**, **`BeliefUpdate`**)* The system processes sensor observations using VLMs to identify objects and evaluate relevant predicates (detailed in §8.3.1). This information is then integrated into the current belief state b through data association and predicate updates, potentially discovering new objects (detailed in §8.3.2).

*(3) Planning (**`Plan`**)* Given the current belief state b and goal G, the symbolic planner (Fast Downward Helmert [2006]) generates a plan p using the determinized PDDL action descriptions derived from our formulation (§8.2). This high-level plan assumes downward refinability into executable skills.

*(4) Execution (**`Execute`**)* The first action $a = p[0]$ of the plan is executed using the corresponding low-level skill. After execution and subsequent observation/belief update, the system checks if the outcome matches the planner's optimistic assumption. If not, replanning is triggered (detailed in §8.3.4).

This cycle continues until either the belief state satisfies the goal ($b \models G$) or the goal is determined to be infeasible. Replanning occurs from the current, updated belief state whenever expectations are violated.

8.3.1 Perception via Vision-Language Models

Our perception system utilizes Vision-Language Models (VLMs) to process visual observations and extract both object detections and predicate evaluations, corresponding

to the `Observe` step in Algorithm 1. The perception pipeline consists of two main components:

Object Detection and Localization

For object detection, we utilize MOLMO [Deitke et al., 2024], a pre-trained model with a pointing feature that enables object localization. Given an observation (e.g., an image $\bar{o}$) and a set of textual prompts describing potential objects (slightly tuned for the objects in tasks), MOLMO identifies objects of interest $\mathbb{O}'$, providing their pixel locations in the image. These detections are then processed using the Segment Anything Model (SAM) [Kirillov et al., 2023] to generate segmentation masks. Combined with depth information, these masks yield the center of mass for each object in a map coordinate frame (maintained by an underlying SLAM-based odometry system).

Predicate Evaluation

For each detected object in $\mathbb{O}'$, we query a VLM (e.g., GPT-4o [OpenAI, 2023]) to evaluate the truth values of relevant predicates $\mathcal{P}$. For example, given an image containing a drawer, we might ask "Is the drawer empty?" to evaluate `EmptyContainer(drawer1)`. The VLM acts as a belief-space grounding classifier, determining whether each predicate is known-true (K_P), known-false ($K_{\neg P}$), or remains unknown ($\neg K_P \wedge \neg K_{\neg P}$). This evaluation is done in a batched manner for all relevant predicates across all camera views at the current time step for efficiency, potentially using a single query to evaluate multiple predicates simultaneously. Details on VLM prompting strategies, batching implementation, and cost/timing considerations are provided in Appendix. While VLM inference can be computationally intensive, calls are typically limited within the execution loop (e.g., once per observation cycle or per skill execution for pre/post-condition checks), making the approach practical.

8.3.2 Belief State Update

The belief-state update process (`BeliefUpdate` in Algorithm 1) integrates the structured observation o (containing $\mathbb{O}'$, L, S, and predicate evaluations from §8.3.1) with the existing belief state b, maintaining consistency in object tracking and predicate evaluations:

Object List Update

When new observations yield object detections $\mathbb{O}'$, we merge them with the existing object set $\mathbb{O}$ in the belief b. This involves data association, primarily based on object unique identifiers (represented in language) across all observations, to match observed objects $\mathbb{O}'$ with known objects $\mathbb{O}$. Unmatched and verified novel detections are added to $\mathbb{O}$. For example, if we detect "cup4" near a previously known "drawer1", and it doesn't match any existing object based on location, we add it to our object set $\mathbb{O}$.

This process ensures consistent object tracking while allowing for the discovery of new objects.

Predicate Value Update

For all objects visible in the current observation, we update the truth values (K-fluents) of associated predicates in the belief b based on the VLM's evaluations from §8.3.1. The belief update mechanism follows a monotonic knowledge acquisition principle, consistent with our formulation (§8.2): unknown predicates ($\neg K_P \wedge \neg K_{\neg P}$) can transition to known states (K_P or $K_{\neg P}$) based on confident VLM outputs, but known predicates never revert to unknown states. This reflects the assumption that observations are perfect (§8.2).

8.3.3 Planning in Belief Space

In the planning process, corresponding to the `Plan` step in Algorithm 1, we follow a procedure similar to [Kumar et al., 2024b] where we are given a task and the robot will plan to generate actions that are likely to achieve the goal from the current state. Planning is decomposed into two levels: skill sequencing and continuous parameter selection. Skill sequencing consists of generating a *skeleton*, e.g., (`MoveTo(box,` θ_1`)`, `Pick(box, floor,` θ_2`)`, `MoveTo(table,` θ_3`)`, `Place(box, table,` θ_4`)`). Given a skeleton, we select continuous parameters using $\theta \sim$ *parameter policies* defined for each skill.

We sample and execute each skills *greedily*. If the skill terminates and does not meet its success condition (as defined by the operator effects), we replan. During the evaluation, the robot continues to plan and execute until a maximum number of actions is reached. We refer the reader to other references Helmert [2006], Fox and Long [2003] for a more formal description of the planning techniques we use in this work.

8.3.4 Skill Plan Execution

This section details the `Execute` step and the replanning logic within our observe-update-plan-execute cycle (§8.3, Algorithm 1).

Execution

The `Execute` step takes the first symbolic action $a = p[0]$ from the current plan (e.g., `PlaceOn(cup1, bin0)`) and executes the corresponding parameterized *skill*. Skills encapsulate the low-level control logic required to perform the action. Some skills may involve internal computations or closed-loop control for robustness, similar to approaches like Kumar et al. [2024b]. The low-level aspects, including grounding symbolic object parameters (like 'cup1') to continuous geometric values (e.g., poses, trajectories) and handling physical constraints via motion planning, are managed by the skill execution module. We assume a skill is a function that can execute and may fail, and their failure will be handled via replanning (§8.2.1). Further details on the skill execution

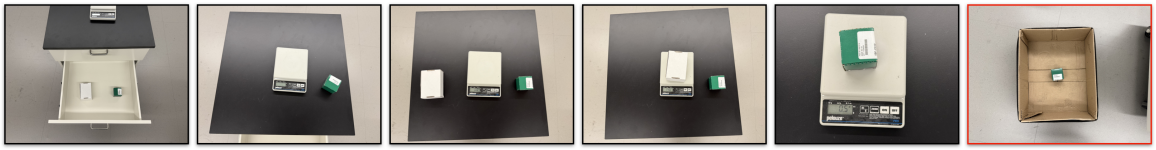

Figure 8.4: *Sort Weight*. An example task in our synthetic environment with real images. The agent needs to open the drawer and retrieve sealed boxes to weigh them. The boxes cannot be opened by the agent but can only be measured indirectly by a scale. The goal is to find empty boxes and remove it to a bin, and a few notable states are shown in the figure. The optimal path takes 14 steps.

Methods \ Tasks	Cup Pick-Place		Drawer Cleaning		Sort Weight	
	Success	SPL	Success	SPL	Success	SPL
Random	0%	0.00 ± 0.00	0%	0.00 ± 0.00	0%	0.00 ± 0.00
VLM End-to-End	30%	0.15 ± 0.24	0%	0.00 ± 0.00	0%	0.00 ± 0.00
VLM (Captioning) + LLM	**100%**	0.49 ± 0.07	10%	0.04 ± 0.11	0%	0.00 ± 0.00
VLM (Labeling) + LLM	90%	0.69 ± 0.28	0%	0.00 ± 0.00	0%	0.00 ± 0.00
BKLVA (Ours)	**100%**	**1.00 ± 0.00**	**80%**	**0.32 ± 0.16**	**70%**	**0.46 ± 0.32**

Table 8.1: Performance comparison across synthetic tasks. Success indicates success rate (%) and SPL indicates average success rate weighted by (normalized inverse) path length (between 0 to 1, with 1 being optimal).

framework, parameter grounding methods, and low-level implementation specifics are provided in Appendix.

Replanning

Replanning is triggered in the following scenarios: (1) when an observation action yields an unexpected outcome (e.g., finding a drawer non-empty when assumed empty by the planner's optimistic determinization), or (2) when new objects are incidentally discovered during perception (§8.3.2). When replanning is triggered, we update the belief state with the new information and generate a new plan from the current state by re-invoking the `Plan` step.

8.4 Experiments

Through our experiments, we aim to answer several key questions: (i) Does our structured approach (using VLMs for belief state estimation for symbolic planning) effectively handle uncertainty? (ii) How efficient is our belief-space planning strategy compared to alternatives? (iii) Can this pipeline extend to a real-world robot scenario with real perception and control?

Environments. We evaluate the approaches on (1) a synthetic environment with real images for mobile manipulation, and (2) a Spot real-robot mobile-manipulation

environment. A synthetic task is defined within a fully-observable transition graph that accepts symbolic actions and returns both images and other non-visual predicates (e.g,. whether the agent is holding an object or reachable to an object), though the images may not reveal the entire system state (for example, a drawer's contents remain hidden from view until it is opened). Consequently, similar to a real-robot scenario, the agent maintains a partially observable belief model to plan effectively. We build synthetic domains using real-world images (taken by phones or robots), where each node in the environment's transition graph represents a distinct viewpoint (represented as images and other non-visual predicates). When the agent selects an action, the environment presents images of the resulting state, allowing the agent to update its belief, which initially may contain uncertain or incomplete knowledge of object properties. This symbolic environment with real images (1) systematically evaluates symbolic belief-space planning that integrates perception and task reasoning, (2) reduces the complexity of physical control, and (3) manages randomness when comparing against multiple baselines. These features enable us to explore long-horizon belief-space planning tasks.

We evaluate on the following tasks:

- *Cup Pick-Place (Synthetic).* This is a fully-observable task that tests the agent's ability to rearrange some cups on a table into a box. A set of information-gathering actions is provided to verify if an agent understands whether uncertainty is present and needed to be reduced.

- *Drawer Cleaning (Synthetic).* In this task, one or more drawers of cabinets may contain various objects. Initially, the robot does not know which drawers hold items. It must open each drawer to observe its contents, dynamically update its belief regarding newly found objects, and then remove those objects to a box. An illustration with one drawer and one block is shown in Figure 8.2. The synthetic task features 2 objects and 1 drawer.

- *Sort Weight (Synthetic).* The agent needs to remove closed boxes by using a scale to measure the weight of the boxes. The boxes are hidden inside cabinets, so the agent needs to find the cabinets and measure their weight. After finding the empty boxes, the agent needs to remove them to a bin.

- *Empty Cup Removal (Robot).* A Spot robot is used to execute this task, where three cups are placed on two tables, while the robot does not know if cups contain contents or are empty. From a normal front-facing view, the robot cannot distinguish whether a cup is empty or not. It must navigate closer, take a camera perspective from above, and inspect the contents. Once the robot identifies an empty cup, it removes it to a bin. We include this setup as a real-robot demo of the system, integrated with real-world object detection, segmentation, and belief-state update. See Figure 8.3 and the supplementary video.

Approaches. We evaluate the performance of our approach in comparison to several baselines across simulation and real-robot environments. The environments

vary in terms of observability and task complexity, including both fully observable and partially observable settings, as well as short-horizon and long-horizon tasks. The approaches are summarized here, where more details are provided in the appendix:

- *Random* planner that selects object-parameterized skills and valid object parameters uniformly at random.
- *VLM End-to-end Planning* uses VLMs for end-to-end planning without explicit handling of uncertainty. It has been explored in tabletop manipulation tasks and web agent literature.
- *VLM State Captioning + LLM Planning* uses VLM to generate a text caption of the history of observations (with images and other non-visual predicates). An LLM then outputs a sequence of actions using the caption.
- *VLM Predicate Labeling + LLM Planning* uses VLM to perceive predicate values and LLM for planning, but does not handle uncertainty explicitly either.
- *BKLVA(Ours)*: An approach that uses belief-space operators integrated with VLM-based belief-state estimation to plan to gather information in order to achieve the goal.

For VLM-based planner or VLM-based state captioning, we provide the history of visual observations to them for handling partial observability. Replanning is used and needed when the expected outcome is not achieved, particularly in partially observable environments where belief-space operators are determinized to an optimistic outcome. See the appendix for more details.

Experimental Setup. For each synthetic task, we run 10 random seeds and report the average results, controlling for variance in the perception system, foundation model calls, planning, low-level control, and other factors. For all LLM- and VLM-based approaches, we use `GPT-4o` with zero-shot prompting, with available objects, operators parameterized by objects, operators' preconditions and effects, current state (LLM-based), and observation (VLM-based) or history (partially observable variants) provided in the context window. We use the following metrics to evaluate the approaches for performance and efficiency in handling uncertainty in completing tasks: (1) task success rate, (2) average number of symbolic actions task plan length required for successful runs of a task weighted by the success rate, also referred as "SPL" (Success rate weighted by Path Length) in visual navigation. The agent succeeds when it reaches the goal state and fails when it reaches the max number of steps, transitions to an illegal state (e.g., pick up full cup), or gets stuck in a state. For *real-robot* experiments, we use the Spot robot and only demonstrate with our approach. We use the same set of symbolic actions (operators) as the synthetic tasks, including the information-gathering and manipulation actions. Each action is additionally associated with a skill that executes on the robot. More details are provided in the appendix.

Results and Discussion. We first evaluate whether our structured approach, which integrates VLM-based predicate estimation with symbolic planning, effectively

handles uncertainty (**Q1**). As shown in Table 8.1, our method consistently outperforms baselines on partially observable tasks like *Drawer Cleaning* and *Sort Weight*. Unlike end-to-end VLM methods or caption-based state representations, our system selectively gathers information only when needed (e.g., opening drawers or weighing boxes if relevant), thereby avoiding redundant actions.

We also compare the efficiency of our belief-space planner against alternatives on gathering information (**Q2**). We notice a few patterns that baselines do not handle well and result in lower success rates and longer SPL: (1) it relies on commonsense knowledge to plan, which may take extra steps based on its commonsense; (2) it does not capture some subtle diffences in state, causing failure or inefficiency on the task; (3) it does not necessarily understand the history and may retry the same actions at same or similar states that occurred before; (4) it does not output correct format of the state. As an example, the *Cup Pick-Place* task provides information-gathering actions while all information is provided, all baselines tend to take extra steps and result in lower SPL, because it does not understand the actions can be directly executed without additional information needed. A hypothetical approach that separates information gathering and manipulation stages by exhaustively removing uncertainty (e.g., opening all drawers first) would be prohibitively time-consuming in large or intricate environments. By focusing on only those uncertainty necessary to fulfill the task goals, our method achieves higher success rates with fewer actions.

Finally, we test whether the proposed pipeline extends to real-robot scenarios on the *Empty Cup Removal* task (**Q3**), with a video provided in the supplementary material. A Spot robot inspects three cups placed on different tables to determine which are empty before disposing of them. The same VLM-based predicate evaluation prompts are used as in synthetic tasks on images. Our demonstration shows that the pipeline of VLM-based belief state estimation and symbolic planning in BKLVA transfers well to real-world perception and control despite additional noise and uncertainty, as seen in the accompanying video. Thus, the real-robot trials confirm that our framework remains effective in physical settings, suggesting strong potential for more complex mobile-manipulation tasks beyond laboratory conditions.

Overall, these results validate each of our three research questions: (1) explicit predicate-based reasoning and belief maintenance of BKLVA enables robust handling of uncertainty, (2) belief-space planning improves efficiency compared to naive or exhaustive approaches, and (3) the BKLVA pipeline scales to a real robot with minimal modification (besides additional real-world perception and control), demonstrating its viability for real-world partial observability.

8.5 Limitations and Discussion

To make belief-space planning tractable and demonstrate its core benefits, our framework incorporates several simplifying assumptions, which also highlight avenues for future enhancement.

First, we simplify perception by assuming *perfect observations* from the VLM and

represent belief using *ternary logic* (True/False/Unknown). This facilitates integration but ignores perceptual noise and graded belief. Future work could increase robustness by adopting *probability-valued fluents* and *Bayesian belief updates* to handle imperfect VLM outputs more naturally. Second, for planning efficiency, we employ *optimistic determinization* of information-gathering actions. This allows the use of classical planners but is a simplification compared to fully *probabilistic planning frameworks* that explicitly reason about stochastic outcomes, which could yield more robust plans at higher computational cost. Third, we assume a *static world* between actions and *monotonic knowledge acquisition* (no information loss). This simplification is reasonable for many short-horizon tasks but limits applicability in dynamic environments or over longer periods. Modeling *exogenous effects* and information decay, at least in the belief update, would extend the framework's realism.

Further simplifications enabling our current focus include: (1) relying on *manually defined action operators*, whereas learning or extracting these could increase adaptability; (2) addressing *perception scope* primarily through VLM predicate grounding, leaving complex object search and long-term data association for future work; and (3) abstracting away low-level *skill execution details*, where tighter integration with planning methods is needed for robust physical deployment.

While these simplifications enabled us to demonstrate the value of VLM-integrated belief-space planning, relaxing them, particularly towards more probabilistic representations and reasoning, constitutes important next steps.

8.6 Conclusion

In this paper, we presented a novel approach to belief-space robot planning that effectively addresses the challenges of partial observability and uncertainty in partially observable environments on mobile-manipulation robots. By integrating belief-space planning with information-gathering actions and leveraging vision-language models (VLMs) as belief-state estimators, our method demonstrated superior performance in handling long-horizon tasks in diverse, partially observable settings. The ability to reason about information gathering using ternary belief-space predicates enabled the robot to systematically reduce uncertainty, leading to higher task-success rates and improved efficiency compared to baseline approaches.

Our results highlight the importance of combining high-level belief-space reasoning with robust perception systems in robotic planning frameworks. Unlike baselines that lacked explicit uncertainty handling or efficient planning for information gathering, our approach demonstrated the advantage of integrating symbolic planning with modern perception systems, particularly VLMs. By automating the process of belief-state estimation with VLMs, we made belief-space planning more feasible in real-world scenarios, addressing the complexity of manually designing predicates for dynamic, unstructured environments.

This work serves as a foundation for future research in belief-space planning. While we focused on leveraging VLMs for belief-state estimation, future extensions could ex-

plore learning belief-space operators, improving sampling strategies, and integrating end-to-end learning to bridge the gap between planning and execution. These advancements could further enhance the practicality of belief-space planning and enable more adaptive and capable robotic systems in partially observed settings.

8.7 Chapter Discussion

This chapter tackled planning under partial observability, revealing a conceptual insight: *information is an action*. The 'look' operation isn't merely sensing—it's a first-class planning primitive that transforms the state space from Unknown to Known. This reframes information gathering as part of the planning problem itself, not a separate perception module.

The approach uses Vision-Language Models (VLMs) to estimate belief states over object properties, representing uncertainty through three-valued logic (true/false/unknown) rather than full probability distributions. By reasoning about which observations would reduce uncertainty, the robot actively seeks information needed for task completion, contrasting with reactive VLM policies that passively accept uncertainty. The 40% action reduction compared to reactive baselines demonstrates that strategic planning with explicit belief tracking outperforms end-to-end approaches, particularly for long-horizon tasks requiring information gathering.

Chapter 9

Extensions and Variations (Part II)

Part II made compositional abstractions deployable through scaled planning (Chapter 7) and belief-space reasoning (Chapter 8). This chapter presents two extensions: using planning to guide skill learning and navigating from abstract spatial maps.

The first extension (Section 9.1) addresses parameterized skill learning—using planning to select which skills to practice during autonomous deployment. The second (Section 9.2) tackles map-based navigation—learning implicit correspondences between 2D maps and 3D environments for zero-shot generalization.

9.1 Long-Horizon Planning to Learning Skill Parameter Policies

This section describes collaborative work with Nishanth Kumar and others on planning to practice parameterized skills, extending our compositional abstractions toward integrated Task and Motion Planning [Kumar et al., 2024a].

9.1.1 Problem Formulation and Background

Long-horizon mobile manipulation tasks require robots to execute sequences of 10-20+ primitive actions to achieve goals like cleaning a kitchen or organizing a workspace. These tasks naturally decompose into a bilevel planning hierarchy. At the high level, discrete task planning determines which skills to execute in what order—this is where classical symbolic planning frameworks like PDDL (Planning Domain Definition Language) operate, reasoning about preconditions, effects, and action sequences using logical predicates. At the low level, continuous parameter selection determines how to execute each skill—where exactly to grasp an object, what velocity to use when sweeping, what trajectory to follow when placing. Classical PDDL planning assumes deterministic action schemas with known outcomes, but real robot skills have varying and initially unknown success rates.

Parameterized skills bridge this symbolic-continuous gap through the formalism $\omega = (\mathcal{I}_i, \Theta_i, \mu_i, \beta_i, \phi_i)$, where $\mathcal{I}_i : \mathcal{S} \to \{0, 1\}$ specifies initiation conditions (when

the skill can be attempted), $\Theta_i \subseteq \mathbb{R}^m$ defines the continuous parameter space (grasp offsets, velocities), $\mu_i(s, \theta)$ implements the low-level controller, β_i specifies termination conditions, and $\phi_i : \mathcal{S} \to \{0, 1\}$ indicates success. These are like PDDL operators but augmented with continuous parameters and probabilistic outcomes. For instance, `Pick(object, surface, θ)` has discrete arguments (which object, which surface) and continuous parameters $\theta \in \mathbb{R}^3$ (grasp position offsets).

The compositional planning frameworks developed throughout this thesis use action abstraction—planning over skills or options rather than primitive motor commands. Chapters 3-5 employ this abstraction, assuming that primitive skills execute reliably once selected. This is the downward refinability assumption: high-level plans can be refined into low-level executions that succeed. In practice, this assumption often fails. Skills start with low competence: a newly deployed robot might grasp objects only 30% of the time and place them correctly only 40%. These low competences compound disastrously over long horizons. If `Pick` has 30% competence and `Place` has 40%, moving one object succeeds with probability $0.3 \times 0.4 = 0.12$. A three-object task has success probability $(0.12)^3 < 0.002$—essentially guaranteed failure.

This work addresses integrated Task and Motion Planning (TAMP) where the downward refinability assumption doesn't hold. Rather than assuming skills work reliably, we track skill competence through experience and use planning to determine which skills need improvement. This transforms the classical TAMP problem—how to sequence skills and select parameters—into a learning problem: how to improve skill competence efficiently during deployment.

The learning challenge is how to improve these skills efficiently during autonomous deployment. Consider a reset-free online learning setting [Thrun, 1995, Gupta et al., 2021] where the robot alternates between task time (executing assigned goals) and free time (autonomous practice without external supervision). During free time, which skills should the robot practice to maximize future task success? This is not standard active learning—collecting practice data for sweeping requires sequential reasoning. The robot may need to execute up to 19 preceding actions just to reach a state where the floor is cluttered and sweeping becomes possible and meaningful. This is meta-planning: using planning to determine what to learn, not just what to do.

This work connects to Chapter 8's belief-space planning framework. Both integrate planning with learning and execution. Chapter 8 uses planning to gather information about unknown world states (is the drawer empty? is the cup full?), reasoning in belief space to strategically reduce uncertainty. Here, we use planning to identify which skills are bottlenecks for task success and to set up diverse practice scenarios. Both demonstrate how compositional planning principles extend beyond nominal execution to handle deployment realities: partial observability in Chapter 8, skill learning here. Together they show that planning is not just for achieving goals but for reasoning about what to learn and what to observe.

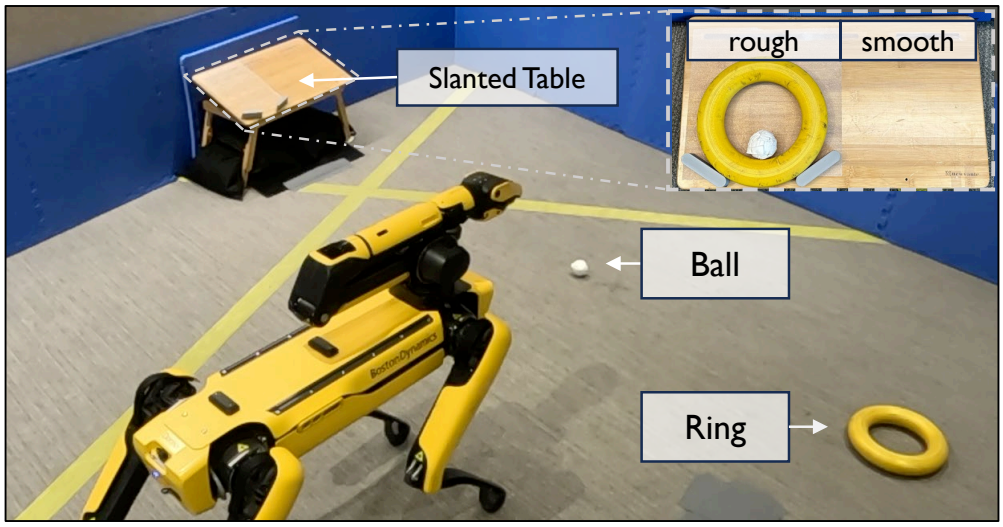

Figure 9.1: **Example: Ball-Ring environment.** The goal is to put the ball on the table. The robot must learn that (1) the ball cannot be placed directly because it will roll off the slanted table; (2) the ring can only be placed on the left side because the right side is smooth; (3) placing the ring on the table and then placing the ball inside the ring accomplishes the goal.

9.1.2 Related Work

This problem sits at the intersection of several research traditions. Classical Task and Motion Planning (TAMP) [Garrett et al., 2021, Srivastava et al., 2014a] addresses the integration of symbolic task planning with continuous motion planning, but typically assumes known skill models and focuses on satisfiability—finding any feasible execution—rather than improving skill competence through practice. Recent work on learning for TAMP uses parameter policies as samplers [Silver et al., 2022a, Mishra et al., 2023] that generate continuous parameters to satisfy symbolic action schemas, but most approaches assume offline learning from demonstrations rather than autonomous online practice during deployment.

The learning problem relates to competence-based exploration in reinforcement learning [Stout and Barto, 2010, Baranes and Oudeyer, 2013, Colas et al., 2019], where agents seek experiences that improve their capabilities. However, these methods typically explore to discover novel states or behaviors without grounding in a known task distribution. Our setting differs critically: we have a distribution of tasks the robot will face, and the question is how to practice efficiently to maximize success on those specific tasks. This task-aware practice is closer to curriculum learning but with the robot autonomously selecting its curriculum through planning.

Parameterized action MDPs [Masson et al., 2016, Dalal et al., 2021, Nasiriany et al., 2022] provide the formal framework for skills with both discrete selections and continuous parameters, extending standard MDPs with hybrid action spaces. Most work in this area focuses on learning joint policies over discrete action selection and continu-

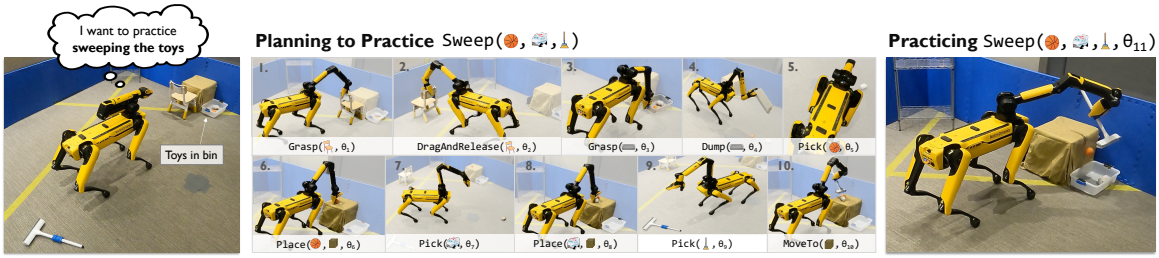

Figure 9.2: **Planning to practice.** To practice a skill, the robot must be in a state where the skill can be initiated. Here, the robot plans to practice sweeping two toys into a bin from its initial state (left) in the Cleanup Playroom environment. This requires chaining up to 19 skills (middle), some omitted for brevity, before practicing (right). This demonstrates meta-planning: using planning to determine what to learn.

ous parameterization. Our approach assumes the discrete skill sequencing problem is solved through planning and focuses instead on rapidly specializing continuous parameter policies through strategic, planned practice scenarios that maximize transfer to the task distribution.

9.1.3 Approach and Results

The Estimate, Extrapolate & Situate (EES) approach addresses skill selection through three integrated components. First, it estimates current skill competence by maintaining Beta-Bernoulli models: after observing n successes and m failures for skill ω, competence is modeled as $c_\omega \sim \text{Beta}(n+1, m+1)$, providing both mean estimates and uncertainty quantification. Second, it extrapolates future improvement by predicting how competence would change with additional practice using a time series model that considers current competence level, historical improvement rate, and predicted competence after k additional practice attempts. Third, it situates skill improvements within the task distribution using cost-aware classical planning [Helmert, 2006] where each skill has cost $\text{cost}(\omega) = -\log(c_\omega)$. By planning over the task distribution with hypothetically improved competences, the system identifies which skill's improvement would most benefit overall task success—that skill is selected for practice.

Once a skill is selected, planning generates diverse practice scenarios. The robot samples random initial environment states, uses planning to reach states where the skill's initiation conditions $\mathcal{I}_\omega$ are satisfied, executes this setup plan, attempts the skill with parameters sampled from the current parameter policy $\theta \sim \pi_\omega(\cdot|s)$, and updates the policy based on success or failure. This planning-to-practice mechanism ensures the same skill gets practiced from different approach angles, with objects in various configurations, under diverse environmental conditions—exactly the systematic variation needed for robust skill learning.

Evaluated on a Boston Dynamics Spot robot, EES achieved 85% task success compared to 45% for random practice baselines in a ball-ring placement task where the robot must discover that balls roll off slanted tables unless placed inside stabilizing

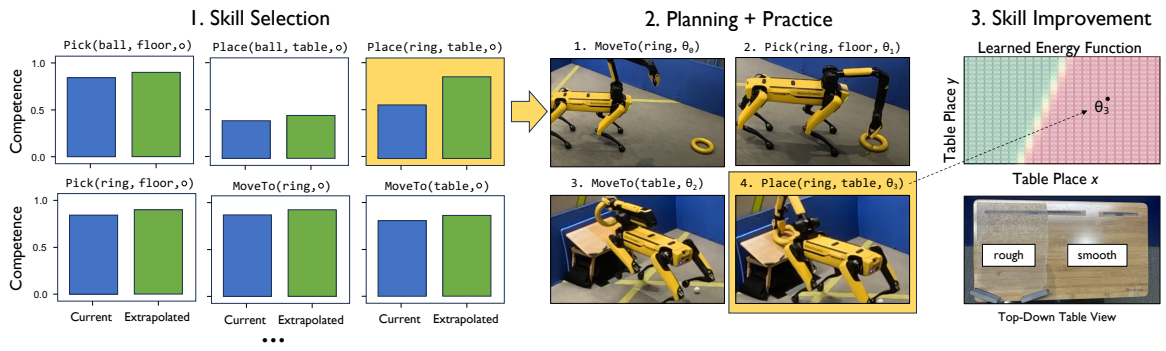

Figure 9.3: **EES pipeline overview.** (1) During free time, the robot selects skills to practice based on which would most improve task success. Here, `Place(ring, table)` is selected. (2) The robot plans to satisfy the skill's initiation condition, then samples a continuous parameter to practice. (3) The success or failure updates the parameter policy.

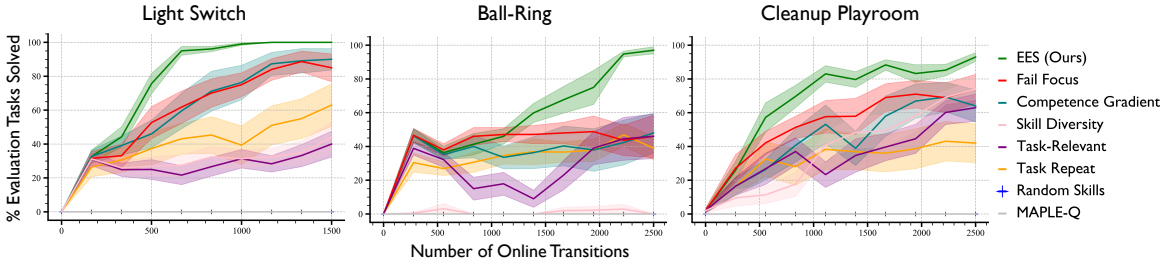

Figure 9.4: **Learning curves across simulated environments.** Percentage of evaluation tasks solved vs. number of online transitions for three environments (1D Grid, Ball-Ring, Kitchen Cleanup). EES (our approach) demonstrates 3-5× sample efficiency compared to baselines, reaching 80% task success with significantly fewer practice attempts. Solid lines show means, shading shows standard error across 10 seeds.

rings. In kitchen cleanup tasks involving pick, place, and sweep operations, task success improved from 38% to 76% after practice, outperforming curiosity-driven baselines at 59%. Across three simulated environments, EES required 3-5× fewer practice attempts than the strongest baselines to reach 80% task success. This sample efficiency stems directly from using planning to identify bottleneck skills—the system practices what matters for the task distribution, not what is merely novel or uncertain. For complete technical details and additional experiments, see the full paper [Kumar et al., 2024a].

9.1.4 Discussion

The option contracts from Chapters 3-5 structure the learning process. Initiation conditions $\mathcal{I}_\omega$ and success conditions ϕ_ω define when skills apply and how to evaluate success—these provide the structure. Parameter policies are what get learned, improving through practice while initiation and success conditions remain fixed. The approach uses planning to determine what to learn, not just what to do. Planning

structure reveals which competences matter—which skills appear in task plans, how frequently, where execution fails.

Compositional factorization enables efficient learning. Improvements to `Pick` generalize across all picking instances, regardless of object or surface. Planning identifies which skill types need improvement by analyzing failures across task distribution. This compositional benefit produces the 3-5× sample efficiency gains. The approach bridges symbolic PDDL-like sequencing with continuous control through parameter policies as learned samplers for motion planning [Garrett et al., 2021, Srivastava et al., 2014a].

This connects to Chapter 8's belief-space planning. Both use planning to handle deployment realities: Chapter 8 for unknown world states (is the drawer empty?), here for skill improvement (which skills need practice?). Together they suggest integrated systems reasoning about what they know, what they can do, and what they should improve—using compositional abstractions across perception, learning, and execution.

9.2 Spatial Abstractions: Map-Based Navigation

This section describes work on end-to-end spatial planning with abstract maps [Zhao and Wong, 2024].

While Part I focused on object-centric representations, navigation presents a fundamentally different compositional structure: spatial hierarchy. Buildings consist of floors, floors contain rooms, rooms have corridors—a compositional organization that doesn't fit neatly into object slots. Novel environments recombine familiar spatial patterns (L-shaped corridors, T-junctions, rectangular rooms) in new configurations.

The key challenge is the correspondence problem: how to execute abstract 2D map plans in 3D environments with first-person observations. Traditional approaches use explicit localization and path following. We instead learn implicit correspondences through a hypermodel architecture—a neural network that takes a 2D map as input and generates environment-specific transition dynamics in a learned latent space. Value iteration in this latent space produces motor commands that can be executed directly without translation.

The approach achieves zero-shot navigation in novel DeepMind Lab mazes without exploration or adaptation. Training on 20 diverse layouts, the system generalizes to completely new environments by recognizing compositional spatial patterns. This demonstrates that compositional abstractions extend beyond objects to spatial hierarchies, enabling systematic generalization through structured representations.

9.2.1 Problem Formulation

Consider an agent placed in a novel 3D maze—a DeepMind Lab environment with multiple rooms, corridors, and dead ends it has never seen before. The agent receives a rough 2D map $M \in \{0,1\}^{64\times64}$ showing walls and walkable space, with current position and target goal marked. It perceives the world through first-person RGB observations $o_t \in \mathbb{R}^{84\times84\times3}$ and must issue continuous turning and forward/backward movement

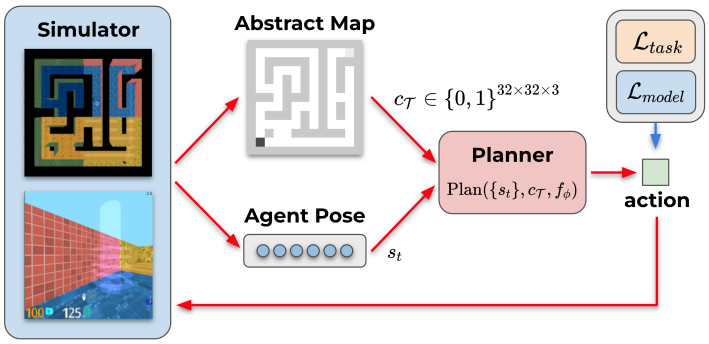

Figure 9.5: **Zero-shot navigation from abstract maps.** The agent navigates unseen DeepMind Lab mazes without prior exploration. Given a top-down 2D occupancy map with start (black dot) and goal (gray dot), the agent must execute the abstract plan in continuous 3D environment with first-person observations, handling unknown map scale, inaccuracies, and noisy localization.

commands. A human could immediately plan a route from the map, but how does the agent execute this abstract plan in the actual 3D environment?

The correspondence problem is fundamental: 2D maps and 3D environments have completely different representations. Scale is unknown—one pixel might represent 1 meter or 10 meters. Perspective differs—maps show top-down layout while the agent sees first-person walls and corridors. Action spaces mismatch—planning on the map suggests "move up 3 cells, turn right" while execution requires "rotate 47.3°, move forward 2.7m". Multiple corridors look identical from first-person view, creating localization ambiguity. Traditional SLAM-based methods require extensive exploration to build accurate maps, incompatible with zero-shot requirements. Direct path following fails due to scale and orientation uncertainty. Reactive policies learn to navigate specific environments but fail on novel layouts.

Formally, given training set $\mathcal{D}_{\text{train}} = \{(M_i, \mathcal{E}_i)\}_{i=1}^N$ of map-environment pairs, we learn a policy $\pi : (o_t, h_{t-1}, M_{\text{test}}, s_0, g) \to (a_t, h_t)$ for novel test environments $(M_{\text{test}}, \mathcal{E}_{\text{test}}) \notin \mathcal{D}_{\text{train}}$, where h_t is learned hidden state. The challenge is zero-shot generalization: navigate successfully in $\mathcal{E}_{\text{test}}$ without gradient updates or exploration, using only the provided map and current observation. Success requires recognizing compositional spatial patterns—straight corridors, L-turns, T-junctions—and translating them into appropriate navigation behaviors.

9.2.2 Solution Approach and Results

Rather than explicitly localizing and following paths, we learn implicit correspondences through a hypermodel that generates environment-specific dynamics from maps. The hypermodel H is a convolutional network taking the entire map M with start/-goal markers and outputting weights $\Theta \in \mathbb{R}^d$ for a transition network: $H : M \in \{0,1\}^{64\times64} \to \Theta$. These weights parameterize a transition model T_Θ predicting next states in learned latent space: $z_{t+1} = T_\Theta(z_t, a_t)$. Crucially, while states z are latent,

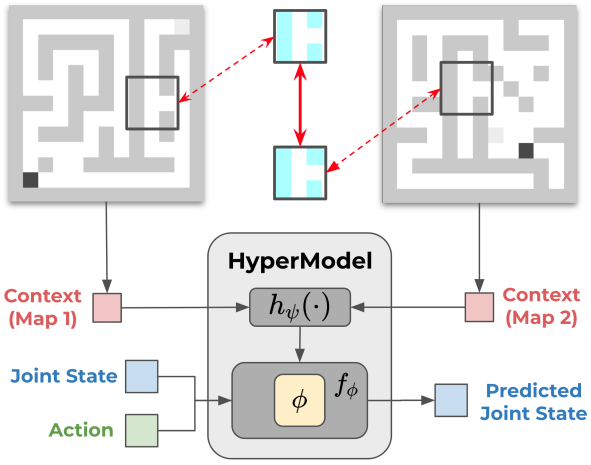

Figure 9.6: **Hypermodel architecture.** The hypermodel H processes maps M_1 and M_2 with goals, outputting transition network weights Θ_1 and Θ_2. Each transition network predicts next states $f(z, a; \Theta_i) = z'$ using its environment-specific weights. Maps sharing local patterns (3×3 patches in light blue) share learned features captured by the hypermodel.

actions a_t are actual motor commands—planned action sequences execute directly without translation.

The system operates end-to-end. Observations encode to latent states $z_t = f_{\text{enc}}(o_t, h_{t-1})$. The hypermodel generates map-specific dynamics $\Theta = H(M, s_0, g)$. Differentiable value iteration runs for K iterations:

$$V^{(k+1)}(z) = \max_a [R(z, a) + \gamma V^{(k)}(T_\Theta(z, a))] \tag{9.1}$$

Policy extraction yields $\pi(z_t) = \arg\max_a Q(z_t, a)$ where Q comes from value iteration. Actions $a_t = \pi(z_t)$ execute directly in the environment.

Training uses hindsight experience replay extended to multi-step trajectories. When the agent attempts goal g but reaches state s' instead, we relabel the trajectory as if s' was the intended goal, providing dense training signal from failed attempts. This extension to n-step trajectories enables learning longer-horizon navigation.

Evaluated in DeepMind Lab 3D mazes, the system trained on 20 diverse layouts with 10 goal positions each, then tested on 10 completely novel layouts. The Map-conditioned Multi-task Navigator (MMN) achieved zero-shot generalization, outperforming reactive CNN policies, explicit map-following planners, and SLAM-based systems, particularly for longer navigation distances. The system showed robustness to map perturbations and scaled to larger environments, succeeding on complex hierarchical tasks spanning multiple rooms without exploration or adaptation in test environments. See the full paper [Zhao and Wong, 2024] for detailed experimental results.

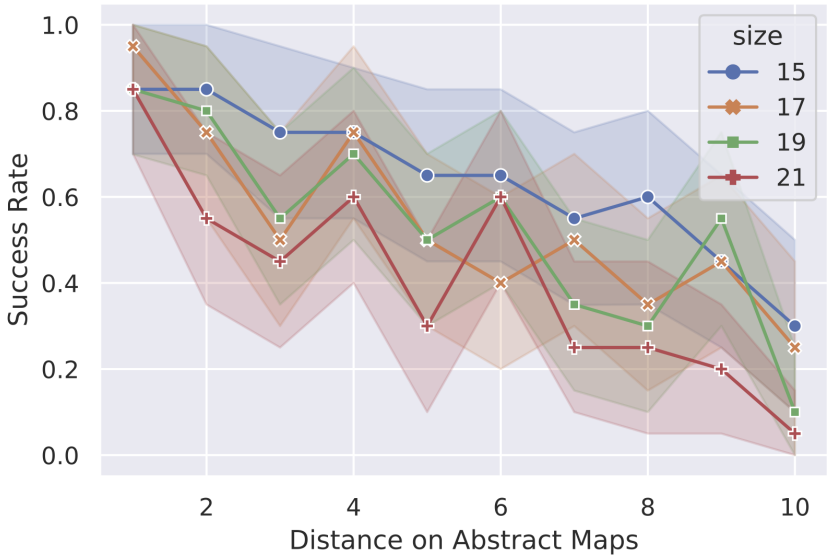

Figure 9.7: **Zero-shot navigation success rates.** Performance on 10 novel test layouts across different navigation distances. MMN (our approach) outperforms reactive CNN baselines and explicit path-following approaches, demonstrating successful compositional generalization to unseen spatial patterns.

9.3 Chapter Discussion

This chapter presented two extensions: planning-guided skill learning (Section 9.1) and map-based navigation (Section 9.2). Both demonstrate how compositional abstractions—whether through option contracts or spatial hierarchies—enable systematic generalization in practical deployment scenarios.

Chapter 10

Conclusion and Future Directions

10.1 Thesis Summary

This thesis set out to build agents that benefit from both learning and planning—systems that could handle the complexity of real-world embodied tasks through *compositional reasoning* (building up complex behaviors from simple parts) rather than memorization. We proposed that **abstraction and compositionality** (structured representations and modular design) would serve as the backbone, with learning and planning as engines powered by compute.

The work developed this vision through discovering a *fundamental tension in AI*: the battle between structure and scale, between understanding and emergence, between guarantees and capabilities. Through this journey, we learned that *structure without learning is brittle* (classical planning fails), *learning without structure is unreliable* (pure neural approaches fail), and the future needs both—but the integration is non-trivial.

We first established how objects can become *modular, interchangeable units* where the system understands that "objectness" itself is a symmetry—when you replace a red cube with a blue sphere, the underlying physics doesn't change, just the specific instance. We then showed how to build spatial reasoning that works regardless of orientation, ensuring that a robot that learns to navigate a room doesn't fail when the room is rotated 90 degrees. Finally, we solved the computational bottleneck that prevents long-horizon planning by recognizing that the computational graph of planning doesn't match how planning actually works—separating finding solutions from computing gradients enables hundred-step horizons where naive approaches fail at twenty.

10.2 Design Principles and Lessons Learned

Through this work, several key design principles have emerged for building robots that plan their actions:

Abstraction as Interface. Treating abstraction as the I/O interface between perception and planning enables modularity and compositional reasoning. Objects, pred-

icates, and options provide a structured vocabulary that can be composed in novel ways, enabling generalization to unseen scenarios through familiar parts.

Compositional Structure as Inductive Bias. Compositional structure and geometric symmetries provide powerful inductive biases that improve data efficiency and generalization. By building equivariance into both representations and planning algorithms, we achieve better sample complexity and robustness to spatial transformations without explicit data augmentation.

Planning at Decision Time. Harder decision-making problems require more compute at inference time. By investing computational resources in planning during execution rather than only during training, systems can handle novel compositions and long-horizon tasks that would be intractable to memorize.

Learning for Abstraction and Planning. While classical planning provides the algorithmic backbone, learning is essential for acquiring the representations, estimating transition models, and adapting to new environments. The integration of differentiable planning with deep learning enables end-to-end optimization while maintaining interpretable structure.

10.3 Retrospective Assessment: What This Work Actually Accomplished

With the benefit of hindsight and the foundation model revolution, we can now see more clearly what this thesis actually accomplished: *not competing with emergent intelligence but providing the structured reasoning it lacks.*

When GPT-4 emerged, it could describe eloquently how to prepare dinner, understanding ingredients, techniques, and sequences at a conceptual level. Yet it cannot *systematically plan* the actual action sequence that would achieve this in a real kitchen. Vision transformers can recognize objects across many viewing angles, yet they cannot *guarantee* that a manipulation plan working for one object configuration will work for another. CLIP can match images across viewpoints, yet it cannot ensure that a navigation strategy that works facing north also works facing east.

This thesis studies *structure* that sits on top of powerful monolistic models. Our equivariance framework ensures *compositional generalization*—when a robot learns to stack red blocks, it immediately knows how to stack blue ones. Our symmetry-aware planning provides *geometric guarantees*—navigation strategies work regardless of orientation. Our belief-space reasoning enables *systematic information gathering* rather than hoping the right observations will occur.

The core insight that emerged from this journey: *intelligence requires both pattern recognition* (what foundation models excel at) *and structured reasoning* (what this thesis provides). Neither alone is sufficient. This isn't about choosing sides in the structure

versus scale debate—it's about recognizing they are *complementary capabilities* that together enable intelligent behavior.

10.4 Reflection of Limitations

This thesis made *critical simplifying assumptions* that enabled progress but also constrained scope. The *downward refinability assumption* allowed us to separate high-level planning from low-level control, but real manipulation tasks often require tighter coupling between symbolic and geometric reasoning. Rather than manually specifying individual abstractions, we *specified compositional structure* that induces abstractions—the symmetries, object-centric representations, and planning frameworks that enable systematic generalization. While this reduces manual effort, autonomous discovery of the structure itself from raw experience remains an open challenge.

Our experimental validation provides strong evidence for the approach's effectiveness, though *further verification on real-world deployments* would strengthen these claims. Real environments present additional challenges—sensor noise, partial observability, dynamic disturbances—that require continued research to fully address.

The Data Question: Is Scale All We Need? A critical question facing AI today is whether data and scale alone suffice for intelligence. This thesis aims to answer this a bit differently: *data is necessary but not sufficient.* Pure scaling can achieve remarkable pattern recognition and even emergent reasoning, but it cannot provide the *systematic guarantees* required for reliable embodied intelligence. When a robot must stack blocks or navigate spaces, we need more than statistical correlations—we need compositional structure that ensures plans work across variations.

Our work shows that *structure amplifies data efficiency.* With compositional representations and symmetric planning, systems can generalize from limited examples to novel scenarios. This isn't about replacing learning with hand-engineering but about identifying the *right inductive biases* that make learning tractable. Foundation models may eventually discover these structures implicitly through massive scale, but explicitly building them in achieves similar capabilities with substantially less data—a critical advantage for robotics where data collection is expensive and safety-critical applications demand guarantees.

10.5 Future Directions: Integrating Structure and Scale

The future of embodied intelligence lies not in choosing between structure and scale but in their principled integration. Foundation models have shown us what massive scale can achieve in perception and language understanding, while this thesis demonstrates what structured reasoning provides for systematic planning and compositional generalization. The path forward involves combining these complementary strengths.

Consider hybrid architectures where foundation models handle perception—understanding what objects are present, what the human is requesting, what the scene contains—while our planning algorithms handle the systematic reasoning about how to achieve goals through sequences of actions. This division of labor plays to each approach's strengths. Foundation models excel at the pattern recognition and common sense that would be nearly impossible to hand-engineer, while our structured algorithms provide the systematic reasoning that ensures plans actually work.

Beyond simple integration, we can inject compositional priors directly into foundation model training. By building in our structured principles from the start, we might achieve similar capabilities with reduced data requirements. This addresses a critical challenge as AI systems scale: the computational and data requirements of pure scaling are becoming prohibitive. Structure offers a path to efficiency without sacrificing capability.

Perhaps most importantly, as robots deploy in human environments, the need for verification and safety becomes paramount. These structured frameworks provide a foundation for guaranteeing properties of learned systems—something pure neural approaches cannot offer. When a robot is handling sharp knives in a kitchen or navigating around children, we need more than statistical confidence; we need structured reasoning that can be verified and validated.

10.6 Closing Remarks

This work demonstrates that embodied agents need both the pattern recognition of modern AI and the systematic reasoning of structured algorithms. The path forward isn't choosing one over the other—it's building systems where each approach handles what it does best.

Structure remains essential even in the age of foundation models. The formal guarantees and systematic generalization developed here address gaps that pure scaling cannot fill, providing verified building blocks for safety-critical applications where statistical confidence isn't enough. Rather than hoping end-to-end learning will solve everything, successful deployment depends more on systematic reasoning than on perfect perception.

Some of the hardest problems now lie at the interfaces: How to maintain formal guarantees while integrating learned components? How to discover compositional structure from raw experience? How to verify hybrid systems that combine neural and symbolic reasoning? These questions need to be answer in the next few years of AI and robotics research, where the impactful work may happen not within individual approaches but at their intersection.

Bibliography

D. Abel, D. Arumugam, L. Lehnert, and M. L. Littman. A theory of abstraction in reinforcement learning. In *Proceedings of ICML*, 2020a.

D. Abel, N. Umbanhowar, K. Khetarpal, D. Arumugam, D. Precup, and M. Littman. Value Preserving State-Action Abstractions. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pages 1639–1650. PMLR, June 2020b. URL https://proceedings.mlr.press/v108/abel20a.html. ISSN: 2640-3498.

J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. 2023.

M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. 2022a.

M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, et al. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022b.

B. Amos and J. Z. Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks. *arXiv:1703.00443 [cs, math, stat]*, Oct. 2019. URL http://arxiv.org/abs/1703.00443. arXiv: 1703.00443.

B. Amos and D. Yarats. The Differentiable Cross-Entropy Method. Sept. 2019. doi: 10/gq5m59. URL https://arxiv.org/abs/1909.12830v4.

P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sunderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-Language Navigation: Interpreting Visually-Grounded Navigation Instructions in Real Environments. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, Salt Lake City, UT, June 2018. IEEE. ISBN 978-1-5386-6420-9. doi: 10.1109/CVPR.2018.00387. URL https://ieeexplore.ieee.org/document/8578485/.

J. Andreas. Measuring compositionality in representation learning. In *International Conference on Learning Representations*, 2019.

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. doi: 10.1609/aaai.v31i1.10916.

P.-L. Bacon, F. Schäfer, C. Gehring, A. Anandkumar, and E. Brunskill. A Lagrangian Method for Inverse Problems in Reinforcement Learning. page 12, 2019.

D. Bahdanau, S. Murty, M. Noukhovitch, T. H. Nguyen, H. de Vries, and A. Courville. Systematic generalization: What is required and can it be learned? In *International Conference on Learning Representations*, 2019.

S. Bai, J. Z. Kolter, and V. Koltun. Deep Equilibrium Models. *arXiv:1909.01377 [cs, stat]*, Oct. 2019. URL http://arxiv.org/abs/1909.01377. arXiv: 1909.01377.

S. Bai, V. Koltun, and J. Z. Kolter. Multiscale Deep Equilibrium Models. *arXiv:2006.08656 [cs, stat]*, Nov. 2020. URL http://arxiv.org/abs/2006.08656. arXiv: 2006.08656.

S. Bai, V. Koltun, and J. Z. Kolter. Stabilizing Equilibrium Models by Jacobian Regularization. Technical Report arXiv:2106.14342, arXiv, June 2021. URL http://arxiv.org/abs/2106.14342. arXiv:2106.14342 [cs, stat] type: article.

A. Baranes and P.-Y. Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 2013. URL http://www.pyoudeyer.com/ActiveGoalExploration-RAS-2013.pdf.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

S. Bhattacharya, S. Badyal, T. Wheeler, S. Gil, and D. Bertsekas. Reinforcement learning for pomdp: Partitioned rollout and policy iteration with application to autonomous sequential repair problems. *IEEE Robotics and Automation Letters*, 5 (3):3967–3974, 2020. doi: 10.1109/LRA.2020.2978451.

O. Biza, R. Platt, J.-W. van de Meent, L. L. Wong, and T. Kipf. Binding actions to objects in world models. *arXiv preprint arXiv:2204.13022*, 2022.

B. Bonet and H. Geffner. Planning under partial observability by classical replanning: theory and experiments. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, page 1936–1941. AAAI Press, 2011. ISBN 9781577355151.

J. Brandstetter, R. Hesselink, E. van der Pol, E. J. Bekkers, and M. Welling. Geometric and Physical Quantities Improve E(3) Equivariant Message Passing. *arXiv:2110.02905 [cs, stat]*, Dec. 2021. URL http://arxiv.org/abs/2110.02905. arXiv: 2110.02905.

A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. 2022.

A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. 2023.

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021a.

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv:2104.13478 [cs, stat]*, Apr. 2021b. URL http://arxiv.org/abs/2104.13478. arXiv: 2104.13478.

R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1-3):139–159, 1991.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

C. P. Burgess, L. Matthey, N. Watters, R. Kabra, I. Higgins, M. Botvinick, and A. Lerchner. MONet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

H. Caselles-Dupré, M. Garcia-Ortiz, and D. Filliat. Symmetry-based disentangled representation learning requires interaction with environments. In *Neural Information Processing Systems*, 2019.

R. Chaabouni, E. Kharitonov, D. Bouchacourt, E. Dupoux, and M. Baroni. Compositionality and generalization in emergent languages. In *Annual Meeting of the Association for Computational Linguistics*, 2020.

M. Chang, A. Gupta, and S. Gupta. Semantic Visual Navigation by Watching YouTube Videos. In *Advances in Neural Information Processing Systems*, volume 33, pages 4283–4294. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/2cd4e8a2ce081c3d7c32c3cde4312ef7-Abstract.html.

D. S. Chaplot, D. P. Gandhi, A. Gupta, and R. R. Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. *Advances in Neural Information Processing Systems*, 33:4247–4258, 2020.

D. S. Chaplot, D. Pathak, and J. Malik. Differentiable Spatial Planning using Transformers. *arXiv:2112.01010 [cs]*, Dec. 2021. URL http://arxiv.org/abs/2112.01010. arXiv: 2112.01010.

L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in Neural Information Processing Systems*, 2021.

R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural Ordinary Differential Equations. Technical Report arXiv:1806.07366, arXiv, Dec. 2019. URL http://arxiv.org/abs/1806.07366. arXiv:1806.07366 [cs, stat] type: article.

M. Chevalier-Boisvert. Miniworld: Minimalistic 3d environment for rl & robotics research. https://github.com/maximecb/gym-miniworld, 2018.

B. Christianson. Reverse accumulation and attractive fixed points. *Optimization Methods and Software*, 3(4):311–326, Jan. 1994. ISSN 1055-6788, 1029-4937. doi: 10/cthcgc. URL http://www.tandfonline.com/doi/abs/10.1080/10556789408805572.

I. Clavera, V. Fu, and P. Abbeel. Model-Augmented Actor-Critic: Backpropagating through Paths. *arXiv:2005.08068 [cs, stat]*, May 2020. URL http://arxiv.org/abs/2005.08068. arXiv: 2005.08068.

E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems (TODS)*, 4(4):397–434, 1979.

T. Cohen, M. Geiger, and M. Weiler. A General Theory of Equivariant CNNs on Homogeneous Spaces. *arXiv:1811.02017 [cs, stat]*, Jan. 2020. URL http://arxiv.org/abs/1811.02017. arXiv: 1811.02017.

T. S. Cohen and M. Welling. Group Equivariant Convolutional Networks. *arXiv:1602.07576 [cs, stat]*, June 2016a. URL http://arxiv.org/abs/1602.07576. arXiv: 1602.07576.

T. S. Cohen and M. Welling. Steerable CNNs. Nov. 2016b. URL https://openreview.net/forum?id=rJQKYt5ll.

C. Colas, P. Fournier, M. Chetouani, O. Sigaud, and P.-Y. Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2019. URL https://proceedings.mlr.press/v97/colas19a/colas19a.pdf.

A. Creswell, R. Kabra, C. Burgess, and M. Shanahan. Unsupervised object-based transition models for 3d partially observable environments. *Advances in Neural Information Processing Systems*, 34:27344–27355, 2021.

A. Curtis, X. Fang, L. P. Kaelbling, T. Lozano-Pérez, and C. R. Garrett. Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances. In *Proceedings of the 2022 IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

A. Curtis, N. Kumar, J. Cao, T. Lozano-Pérez, and L. P. Kaelbling. Trust the proc3s: Solving long-horizon robotics problems with llms and constraint satisfaction, 2024a. URL https://arxiv.org/abs/2406.05572.

A. Curtis, G. Matheos, N. Gothoskar, V. Mansinghka, J. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Partially observable task and motion planning with uncertainty and risk awareness. In *Robotics: Science and Systems (RSS)*, 2024b.

M. Dalal, D. Pathak, and R. R. Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://proceedings.neurips.cc/paper/2021/file/b6846b0186a035fcc76b1b1d26fd42fa-Paper.pdf.

A. Deac, P. Veličković, O. Milinković, P.-L. Bacon, J. Tang, and M. Nikolić. Neural Algorithmic Reasoners are Implicit Planners. Oct. 2021. URL https://arxiv.org/abs/2110.05442v1.

T. Dean and R. Givan. Model minimization in markov decision processes. In *Proceedings of AAAI*, 1997.

M. Deitke, C. Clark, S. Lee, R. Tripathi, Y. Yang, J. S. Park, M. Salehi, N. Muennighoff, K. Lo, L. Soldaini, J. Lu, T. Anderson, E. Bransom, K. Ehsani, H. Ngo, Y. Chen, A. Patel, M. Yatskar, C. Callison-Burch, A. Head, R. Hendrix, F. Bastani, E. VanderBilt, N. Lambert, Y. Chou, A. Chheda, J. Sparks, S. Skjonsberg, M. Schmitz, A. Sarnat, B. Bischoff, P. Walsh, C. Newell, P. Wolters, T. Gupta, K.-H. Zeng, J. Borchardt, D. Groeneveld, C. Nam, S. Lebrecht, C. Wittlif, C. Schoenick, O. Michel, R. Krishna, L. Weihs, N. A. Smith, H. Hajishirzi, R. Girshick, A. Farhadi, and A. Kembhavi. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models, 2024. URL https://arxiv.org/abs/2409.17146.

A. Didolkar, A. Goyal, N. R. Ke, C. Blundell, P. Beaudoin, N. Heess, M. C. Mozer, and Y. Bengio. Neural production systems. *Advances in Neural Information Processing Systems*, 34:25673–25687, 2021.

C. Diuk, A. Cohen, and M. L. Littman. An object-oriented representation for efficient reinforcement learning. In *International Conference on Machine Learning*, 2008.

A. Dudzik and P. Veličković. Graph Neural Networks are Dynamic Programmers. *arXiv:2203.15544 [cs, math, stat]*, Mar. 2022. URL http://arxiv.org/abs/2203.15544. arXiv: 2203.15544.

N. Ferns, P. Panangaden, and D. Precup. Metrics for finite markov decision processes. In *Proceedings of UAI*, 2004.

R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

D. Fiser, A. Torralba, and A. Shleyfman. Operator Mutexes and Symmetries for Simplifying Planning Tasks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7586–7593, July 2019. ISSN 2374-3468. doi: 10/ghkkbq. URL https://ojs.aaai.org/index.php/AAAI/article/view/4751. Number: 01.

M. Fox and D. Long. The Detection and Exploitation of Symmetry in Planning Problems. In *In IJCAI*, pages 956–961. Morgan Kaufmann, 1999.

M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *In Proceedings of AIPS'02*, pages 83–91, 2002.

M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox. Online replanning in belief space for partially observable task and motion problems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5678–5684. IEEE, 2020. URL https://arxiv.org/pdf/1911.04577.pdf.

C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 2021. URL https://www.annualreviews.org/doi/pdf/10.1146/annurev-control-091420-084139.

C. Gehring, K. Kawaguchi, J. Huang, and L. P. Kaelbling. Understanding End-to-End Model-Based Reinforcement Learning Methods as Implicit Parameterization. May 2021. URL https://openreview.net/forum?id=xj2sE--Q90e.

M. R. Genesereth and N. J. Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann, 1987.

M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

L. E. Ghaoui, F. Gu, B. Travacca, A. Askari, and A. Y. Tsai. Implicit Deep Learning. Technical Report arXiv:1908.06315, arXiv, Aug. 2020. URL http://arxiv.org/abs/1908.06315. arXiv:1908.06315 [cs, math, stat] type: article.

J. C. Gilbert. Automatic differentiation and iterative processes. *Optimization Methods and Software*, 1(1):13–21, Jan. 1992. ISSN 1055-6788. doi: 10/dbgqw5. URL https://doi.org/10.1080/10556789208805503. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/10556789208805503.

M. L. Ginsberg. Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4(3):256–316, 1988.

J. Gordon, D. Lopez-Paz, M. Baroni, and D. Bouchacourt. Permutation equivariant models for compositional generalization in language. In *International Conference on Learning Representations*, 2019.

K. Greff, S. van Steenkiste, and J. Schmidhuber. On the binding problem in artificial neural networks. *arXiv preprint arXiv:2012.05208*, 2020.

C. Grimm, A. Barreto, S. Singh, and D. Silver. The Value Equivalence Principle for Model-Based Reinforcement Learning. *arXiv:2011.03506 [cs]*, Nov. 2020. URL http://arxiv.org/abs/2011.03506. arXiv: 2011.03506.

C. Grimm, A. Barreto, G. Farquhar, D. Silver, and S. Singh. Proper Value Equivalence. *arXiv:2106.10316 [cs]*, Dec. 2021. URL http://arxiv.org/abs/2106.10316. arXiv: 2106.10316.

C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

A. Guez, M. Mirza, K. Gregor, R. Kabra, S. Racanière, T. Weber, D. Raposo, A. Santoro, L. Orseau, T. Eccles, G. Wayne, D. Silver, and T. Lillicrap. An investigation of model-free planning. *arXiv:1901.03559 [cs, stat]*, May 2019. URL http://arxiv.org/abs/1901.03559. arXiv: 1901.03559.

A. Gupta, J. Yu, T. Z. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine. Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021. URL https://arxiv.org/pdf/2104.11203.pdf.

S. Gupta, V. Tolani, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive Mapping and Planning for Visual Navigation. *CVPR 2017*, 2017. URL http://arxiv.org/abs/1702.03920. arXiv: 1702.03920.

D. Ha and J. Schmidhuber. World models. *Advances in Neural Information Processing Systems*, 2018.

D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. *arXiv:1912.01603 [cs]*, Mar. 2020a. URL http://arxiv.org/abs/1912.01603. arXiv: 1912.01603.

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning to dream: Model-based reinforcement learning via latent world models. In *International Conference on Machine Learning*, 2020b.

D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021.

D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. 2019.

R. Hazra, P. Z. D. Martires, and L. D. Raedt. Saycanpay: Heuristic planning with large language models using learnable domain knowledge, 2024. URL https://arxiv.org/abs/2308.12682.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)*, 2006. URL https://www.jair.org/index.php/jair/article/view/10457/25068.

I. Higgins, D. Amos, D. Pfau, S. Racaniere, L. Matthey, D. Rezende, and A. Lerchner. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018.

W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning (ICML)*, 2022a.

W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022b.

S. Ishida and J. F. Henriques. Towards real-world navigation with deep differentiable planners. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17306–17315, New Orleans, LA, USA, June 2022. IEEE. ISBN 978-1-66546-946-3. doi: 10.1109/CVPR52688.2022.01681. URL https://ieeexplore.ieee.org/document/9878917/.

J. Johnson, B. Hariharan, L. Van Der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

L. P. Kaelbling and T. Lozano-Perez. Integrated robot task and motion planning in belief space. July 2012a. URL https://dspace.mit.edu/handle/1721.1/71529. Accepted: 2012-07-03T18:00:04Z.

L. P. Kaelbling and T. Lozano-Perez. Unifying perception, estimation and action for mobile manipulation via belief space planning. In *2012 IEEE International Conference on Robotics and Automation*, pages 2952–2959, St Paul, MN, USA, May

2012b. IEEE. ISBN 978-1-4673-1405-3 978-1-4673-1403-9 978-1-4673-1578-4 978-1-4673-1404-6. doi: 10.1109/ICRA.2012.6225237. URL http://ieeexplore.ieee.org/document/6225237/.

L. P. Kaelbling and T. Lozano-Pérez. Pre-image backchaining in belief space for mobile manipulation. In *Robotics Research: The 15th International Symposium ISRR*, pages 383–400. Springer, 2017.

L. P. Kaelbling and T. Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, Aug. 2013. ISSN 0278-3649, 1741-3176. doi: 10.1177/0278364913484072. URL http://journals.sagepub.com/doi/10.1177/0278364913484072.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998a.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, May 1998b. ISSN 0004-3702. doi: 10.1016/S0004-3702(98)00023-X. URL https://www.sciencedirect.com/science/article/pii/S000437029800023X.

L. P. Kaelbling, A. LaGrassa, and T. Lozano-Pérez. Specifying and achieving goals in open uncertain robot-manipulation domains. *arXiv preprint arXiv:2112.11199*, 2021.

P. Karkus, D. Hsu, and W. S. Lee. QMDP-Net: Deep Learning for Planning under Partial Observability. *arXiv:1703.06692 [cs, stat]*, Nov. 2017. URL http://arxiv.org/abs/1703.06692. arXiv: 1703.06692.

N. Keriven and G. Peyré. Universal invariant and equivariant graph neural networks. In *Neural Information Processing Systems*, 2019.

D. Keysers, N. Schärli, N. Scales, H. Buisman, D. Furrer, S. Kashubin, N. Momchev, D. Sinopalnikov, L. Stafiniak, T. Tihon, D. Tsarkov, X. Wang, M. van Zee, and O. Bousquet. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*, 2020.

T. Kipf, E. van der Pol, and M. Welling. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2019.

T. Kipf, G. F. Elsayed, A. Mahendran, A. Stone, S. Sabour, G. Heigold, R. Jonschkowski, A. Dosovitskiy, and K. Greff. Conditional object-centric learning from video. *arXiv preprint arXiv:2111.12594*, 2021.

T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907 [cs, stat]*, Feb. 2017. URL http://arxiv.org/abs/1609.02907. arXiv: 1609.02907.

A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. Segment anything, 2023. URL https://arxiv.org/abs/2304.02643.

S. C. Kleene. *Introduction to Metamathematics*. North-Holland Publishing Company, 1952.

R. Kondor and S. Trivedi. On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups. *arXiv:1802.03690 [cs, stat]*, Nov. 2018. URL http://arxiv.org/abs/1802.03690. arXiv: 1802.03690.

G. Konidaris. On the necessity of abstraction. *Current Opinion in Behavioral Sciences*, 29:1–7, 2019.

J. Kossen, K. Stelzner, M. Hussing, C. Voelcker, and K. Kersting. Structured object-aware physics prediction for video modeling and planning. In *International Conference on Learning Representations*, 2019.

N. Kumar, T. Silver, W. McClinton, L. Zhao, S. Proulx, T. Lozano-Pérez, L. P. Kaelbling, and J. Barry. Practice makes perfect: Planning to learn skill parameter policies. In *Robotics: Science and Systems (RSS)*, 2024a.

N. Kumar, T. Silver, W. McClinton, L. Zhao, S. Proulx, T. Lozano-Pérez, L. P. Kaelbling, and J. Barry. Practice makes perfect: planning to learn skill parameter policies, May 2024b. URL http://arxiv.org/abs/2402.15025. arXiv:2402.15025 [cs].

H. Kurniawati, D. Hsu, and W. S. Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Citeseer, 2008.

B. Lake and M. Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, 2018.

L. Lang and M. Weiler. A Wigner-Eckart Theorem for Group Equivariant Convolution Kernels. Sept. 2020. URL https://openreview.net/forum?id=ajOrOhQOsYx.

S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006a.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, May 2006b. ISBN 978-1-139-45517-6.

L. Lee, E. Parisotto, D. S. Chaplot, E. Xing, and R. Salakhutdinov. Gated Path Planning Networks. *arXiv:1806.06408 [cs, stat]*, June 2018. URL http://arxiv.org/abs/1806.06408. arXiv: 1806.06408.

L. Li, T. J. Walsh, and M. Littman. Towards a Unified Theory of State Abstraction for MDPs. In *AI&M*, 2006a.

L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. In *AI&M*, 2006b.

J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control, 2023. URL https://arxiv.org/abs/2209.07753.

Y. Liang, B. Zhai, Z. Chen, B. Li, and S. Huang. Sscnav: Confidence-aware semantic scene completion for visual semantic navigation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11461–11467. IEEE, 2021.

Z. Lin, Y.-F. Wu, S. Peri, B. Fu, J. Jiang, and S. Ahn. Improving generative imagination in object-centric world models. In *International Conference on Machine Learning*, 2020.

F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf. Object-centric learning with slot attention. In *Neural Information Processing Systems*, 2020.

T. Lozano-Pérez, J. L. Jones, E. Mazer, and P. A. O'Donnell. Task-level planning of pick-and-place robot motions. *Computer*, 22(3):21–29, 3 1989.

W. Masson, P. Ranchod, and G. Konidaris. Reinforcement learning with parameterized actions. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016. URL https://ojs.aaai.org/index.php/AAAI/article/view/10226/10085.

U. A. Mishra, S. Xue, Y. Chen, and D. Xu. Generative skill chaining: Long-horizon skill planning with diffusion models. In *Conference on Robot Learning*, 2023. URL https://openreview.net/pdf?id=HtJE9ly5dT.

A. K. Mondal, P. Nair, and K. Siddiqi. Group Equivariant Deep Reinforcement Learning. *arXiv:2007.03437 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2007.03437. arXiv: 2007.03437.

S. M. Narayanamurthy and B. Ravindran. On the hardness of finding symmetries in Markov decision processes. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 688–695, Helsinki, Finland, 2008. ACM Press. ISBN 978-1-60558-205-4. doi: 10/bkswc2. URL http://portal.acm.org/citation.cfm?doid=1390156.1390243.

S. Nasiriany, H. Liu, and Y. Zhu. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2022. URL https://arxiv.org/pdf/2110.03655.pdf.

X. Nie, L. L. Wong, and L. P. Kaelbling. Searching for physical objects in partially known environments. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5403–5410, 2016. doi: 10.1109/ICRA.2016.7487752.

E. Nikishin, R. Abachi, R. Agarwal, and P.-L. Bacon. Control-Oriented Model-Based Reinforcement Learning with Implicit Differentiation. Technical Report arXiv:2106.03273, arXiv, June 2021. URL http://arxiv.org/abs/2106.03273. arXiv:2106.03273 [cs, stat] type: article.

S. Niu, S. Chen, H. Guo, C. Targonski, M. C. Smith, and J. Kovačević. Generalized Value Iteration Networks: Life Beyond Lattices. *arXiv:1706.02416 [cs]*, Oct. 2017. URL http://arxiv.org/abs/1706.02416. arXiv: 1706.02416.

J. Oh, S. Singh, and H. Lee. Value Prediction Network. *arXiv:1707.03497 [cs]*, Nov. 2017. URL http://arxiv.org/abs/1707.03497. arXiv: 1707.03497.

OpenAI. GPT-4 Technical Report, Mar. 2023. URL http://arxiv.org/abs/2303.08774. arXiv:2303.08774 [cs].

J. Y. Park, O. Biza, L. Zhao, J. W. van de Meent, and R. Walters. Learning symmetric embeddings for equivariant world models. In *International Conference on Machine Learning (ICML)*, pages 17426–17442, 2022. URL https://proceedings.mlr.press/v162/park22a.html.

T. A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641, 1995.

R. Platt Jr, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and systems*, 2010.

N. Pochter, A. Zohar, and J. S. Rosenschein. Exploiting Problem Symmetries in State-Based Planners. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, Aug. 2011. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3732.

D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

V. Pong, S. Gu, M. Dalal, and S. Levine. Temporal Difference Models: Model-Free Deep RL for Model-Based Control. *arXiv:1802.09081 [cs]*, Feb. 2018. URL http://arxiv.org/abs/1802.09081. arXiv: 1802.09081.

D. Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 2000.

B. Ravindran. AN ALGEBRAIC APPROACH TO ABSTRACTION IN REINFORCEMENT LEARNING. page 181.

B. Ravindran and A. G. Barto. Symmetries and Model Minimization in Markov Deision Proesses.

B. Ravindran and A. G. Barto. Smdp homomorphisms: An algebraic approach to abstraction in semi-markov decision processes. In *Proceedings of IJCAI*, 2003.

B. Ravindran and A. G. Barto. An algebraic approach to abstraction in reinforcement learning. *MIT Press*, 2004.

R. Reiter. On closed world data bases. *Logic and Data Bases*, pages 55–76, 1978.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

G. Röger, S. Sievers, and M. Katz. Symmetry-Based Task Reduction for Relaxed Reachability Analysis. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, June 2018. URL https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17772.

V. G. Satorras, E. Hoogeboom, and M. Welling. E(n) Equivariant Graph Neural Networks. *arXiv:2102.09844 [cs, stat]*, Feb. 2021. URL http://arxiv.org/abs/2102.09844. arXiv: 2102.09844.

D. Schleich, T. Klamt, and S. Behnke. Value Iteration Networks on Multiple Levels of Abstraction. May 2019. doi: 10/gq5m5v. URL https://arxiv.org/abs/1905.11068v2.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *arXiv:1911.08265 [cs, stat]*, Nov. 2019. URL http://arxiv.org/abs/1911.08265. arXiv: 1911.08265.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. volume 588, pages 604–609, 2020.

A. Shleyfman, M. Katz, M. Helmert, S. Sievers, and M. Wehrle. Heuristics and Symmetries in Classical Planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), Mar. 2015. ISSN 2374-3468. doi: 10/gq5m5s. URL https://ojs.aaai.org/index.php/AAAI/article/view/9649. Number: 1.

S. Sievers. Structural Symmetries of the Lifted Representation of Classical Planning Tasks. page 8.

S. Sievers, M. Wehrle, M. Helmert, and M. Katz. An Empirical Case Study on Symmetry Handling in Cost-Optimal Planning as Heuristic Search. In S. Hölldobler, R. Peñaloza, and S. Rudolph, editors, *KI 2015: Advances in Artificial Intelligence*, volume 9324, pages 166–180. Springer International Publishing, Cham, 2015. ISBN 978-3-319-24488-4 978-3-319-24489-1. doi: 10.1007/978-3-319-24489-1_13. URL http://link.springer.com/10.1007/978-3-319-24489-1_13. Series Title: Lecture Notes in Computer Science.

S. Sievers, G. Röger, M. Wehrle, and M. Katz. Theoretical Foundations for Structural Symmetries of Lifted PDDL Tasks. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29:446–454, 2019. ISSN 2334-0843. doi: 10/gq5m5t. URL https://ojs.aaai.org/index.php/ICAPS/article/view/3509.

T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Learning neuro-symbolic skills for bilevel planning. In *6th Annual Conference on Robot Learning*, 2022a. URL https://openreview.net/forum?id=OIaJRUo5UXy.

T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Learning neuro-symbolic skills for bilevel planning. In *Proceedings of the 6th Annual Conference on Robot Learning (CoRL)*, 2022b.

T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling. Pddl planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022c.

T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2023. URL https://ojs.aaai.org/index.php/AAAI/article/view/26429/26201.

P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46(1-2):159–216, 1990.

M. Sridharan, J. Wyatt, and R. Dearden. Planning to see: A hierarchical approach to planning visual actions on a robot using pomdps. *Artificial Intelligence*, 174(11): 704–725, 2010.

A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn. Universal Planning Networks. *arXiv:1804.00645 [cs, stat]*, Apr. 2018. URL http://arxiv.org/abs/1804.00645. arXiv: 1804.00645.

S. Srivastava, N. Immerman, and S. Zilberstein. Using abstraction for generalized planning. In *ISAIM*, 2008.

S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE international conference on robotics and automation (ICRA)*, 2014a. URL https://people.eecs.berkeley.edu/~russell/papers/icra14-planrob.pdf.

S. Srivastava, S. J. Russell, P. Ruan, and X. Cheng. First-Order Open-Universe POMDPs. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI'14, pages 742–751, Arlington, Virginia, USA, July 2014b. AUAI Press. ISBN 978-0-9749039-1-0. URL https://www.semanticscholar.org/paper/First-Order-Open-Universe-POMDPs-Srivastava-Russell/da3059ea0db45b39d68ae7b61c59003887208eb7.

A. Stout and A. G. Barto. Competence progress intrinsic motivation. In *IEEE 9th International Conference on Development and Learning (ICDL)*, 2010. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1e9521d28184a344c077edaf780c1205b3e90139.

L. Sun, D. K. Jha, C. Hori, S. Jain, R. Corcodel, X. Zhu, M. Tomizuka, and D. Romeres. Interactive planning using large language models for partially observable robotic tasks. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14054–14061. IEEE, 2024.

R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel. Value iteration networks. *arXiv preprint arXiv:1602.02867*, 2016a.

A. Tamar, Y. WU, G. Thomas, S. Levine, and P. Abbeel. Value Iteration Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016b. URL https://proceedings.neurips.cc/paper/2016/hash/c21002f464c5fc5bee3b98ced83963b8-Abstract.html.

A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel. Value Iteration Networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 4949–4953, Melbourne, Australia, Aug. 2017. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-0-3. doi: 10/ggjfst. URL https://www.ijcai.org/proceedings/2017/700.

S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek. Robots That Use Language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):25–55, 2020. doi: 10.1146/annurev-control-101119-071628. URL https://doi.org/10.1146/annurev-control-101119-071628. _eprint: https://doi.org/10.1146/annurev-control-101119-071628.

S. Thrun. A lifelong learning perspective for mobile robot control. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*,

1995. URL https://www.ri.cmu.edu/pub_files/pub1/thrun_sebastian_1995_3/thrun_sebastian_1995_3.pdf.

K. Valmeekam, S. Sreedharan, M. Marquez, A. Olmo, and S. Kambhampati. On the planning abilities of large language models (a critical investigation with a proposed benchmark). *arXiv preprint arXiv:2302.06706*, 2023.

E. van der Pol, D. Worrall, H. van Hoof, F. Oliehoek, and M. Welling. MDP homomorphic networks: Group symmetries in reinforcement learning. In *Neural Information Processing Systems*, 2020a.

E. van der Pol, D. E. Worrall, H. van Hoof, F. A. Oliehoek, and M. Welling. MDP Homomorphic Networks: Group Symmetries in Reinforcement Learning. *arXiv:2006.16908 [cs, stat]*, June 2020b. URL http://arxiv.org/abs/2006.16908. arXiv: 2006.16908.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, Dec. 2017. URL http://arxiv.org/abs/1706.03762. arXiv: 1706.03762.

R. Veerapaneni, J. D. Co-Reyes, M. Chang, M. Janner, C. Finn, J. Wu, J. Tenenbaum, and S. Levine. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, 2020.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks, Feb. 2018. URL http://arxiv.org/abs/1710.10903. arXiv:1710.10903 [cs, stat].

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*, pages 3540–3549, 2017.

D. Wang, R. Walters, and R. Platt. $\mathrm{SO}(2)$-Equivariant Reinforcement Learning. Sept. 2021. URL https://openreview.net/forum?id=7F9cOhdvfk_.

T. Wang and J. Ba. Exploring Model-based Planning with Policy Networks. June 2019. URL https://arxiv.org/abs/1906.08649v1.

N. Watters, L. Matthey, M. Bosnjak, C. P. Burgess, and A. Lerchner. COBRA: Data-efficient model-based RL through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*, 2019.

T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Hassabis, D. Silver, and D. Wierstra. Imagination-Augmented Agents for Deep Reinforcement Learning. *arXiv:1707.06203 [cs, stat]*, Feb. 2018. URL http://arxiv.org/abs/1707.06203. arXiv: 1707.06203.

M. Weiler and G. Cesa. General $E(2)$-Equivariant Steerable CNNs. *arXiv:1911.08251 [cs, eess]*, Apr. 2021. URL http://arxiv.org/abs/1911.08251. arXiv: 1911.08251.

M. Weiler, M. Geiger, M. Welling, W. Boomsma, and T. Cohen. 3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data. In *NeurIPS*, 2018.

E. Winston and J. Z. Kolter. Monotone operator equilibrium networks. *arXiv:2006.08591 [cs, stat]*, May 2021. URL http://arxiv.org/abs/2006.08591. arXiv: 2006.08591.

D. Winterer, M. Wehrle, and M. Katz. Structural Symmetries for Fully Observable Nondeterministic Planning. page 7.

L. L. Wong. *Object-based world modeling for mobile-manipulation robots*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2016.

L. L. Wong, L. P. Kaelbling, and T. Lozano-Perez. Manipulation-based active search for occluded objects. In *2013 IEEE International Conference on Robotics and Automation*, pages 2814–2819, Karlsruhe, Germany, May 2013. IEEE. ISBN 978-1-4673-5643-5 978-1-4673-5641-1. doi: 10.1109/ICRA.2013.6630966. URL http://ieeexplore.ieee.org/document/6630966/.

L. L. S. Wong, T. Kurutach, T. Lozano-Perez, and L. Kaelbling. Object-Based World Modeling in Semi-Static Environments with Dependent Dirichlet Process Mixtures. Dec. 2015. URL https://www.semanticscholar.org/paper/Object-Based-World-Modeling-in-Semi-Static-with-Wong-Kurutach/3f6adf48071b89d6d2404761ff7948ea345196ee.

K. Xu, J. Li, M. Zhang, S. S. Du, K.-i. Kawarabayashi, and S. Jegelka. What Can Neural Networks Reason About? May 2019. URL https://arxiv.org/abs/1905.13211v4.

S. Yenamandra, A. Ramachandran, M. Khanna, K. Yadav, J. Vakil, A. Melnik, M. Büttner, L. Harz, L. Brown, G. C. Nandi, A. PS, G. K. Yadav, R. Kala, R. Haschke, Y. Luo, J. Zhu, Y. Han, B. Lu, X. Gu, Q. Liu, Y. Zhao, Q. Ye, C. Dou, Y. Chua, V. Kuzma, V. Humennyy, R. Partsey, J. Francis, D. S. Chaplot, G. Chhablani, A. Clegg, T. Gervet, V. Jain, R. Ramrakhya, A. Szot, A. Wang, T.-Y. Yang, A. Edsinger, C. Kemp, B. Shah, Z. Kira, D. Batra, R. Mottaghi, Y. Bisk, and C. Paxton. Towards open-world mobile manipulation in homes: Lessons from the neurips 2023 homerobot open vocabulary mobile manipulation challenge. In *Thirty-seventh Conference on Neural Information Processing Systems: Competition Track*, 2023. URL https://arxiv.org/abs/2407.06939.

L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

L. Zhao and L. L. Wong. Model-based navigation in environments with novel layouts using abstract 2-d maps. In *NeurIPS 2020 Workshop on Deep Reinforcement Learning, Under Conference Review*, 2020.

L. Zhao and L. L. Wong. Learning to navigate in mazes with novel layouts using abstract top-down maps. *Reinforcement Learning Journal*, 5:2359–2372, 2024.

L. Zhao, L. Kong, R. Walters, and L. L. Wong. Toward compositional generalization in object-oriented world modeling. In *International Conference on Machine Learning (ICML)*, pages 26855–26870, 2022a. URL https://proceedings.mlr.press/v162/zhao22g.html.

L. Zhao, X. Zhu, L. Kong, R. Walters, and L. L. S. Wong. Integrating Symmetry into Differentiable Planning. In *ICLR 2023*. ICLR, June 2022b. doi: 10.48550/arXiv.2206.03674. URL http://arxiv.org/abs/2206.03674. arXiv:2206.03674 [cs] type: article.

L. Zhao, J. Y. Park, X. Zhu, R. Walters, and L. L. Wong. `SE(3)` frame equivariance in dynamics modeling and reinforcement learning. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023a.

L. Zhao, H. Xu, and L. L. Wong. Scaling up and stabilizing differentiable planning with implicit differentiation. In *International Conference on Learning Representations (ICLR)*, 2023b. URL https://openreview.net/forum?id=PYbe4MoHf32.

L. Zhao, X. Zhu, L. Kong, R. Walters, and L. L. Wong. Integrating symmetry into differentiable planning with steerable convolutions. In *International Conference on Learning Representations (ICLR)*, 2023c. URL https://openreview.net/forum?id=PYbe4MoHf32.

L. Zhao, W. McClinton, A. Curtis, N. Kumar, T. Silver, L. P. Kaelbling, and L. L. Wong. Seeing is believing: Belief-space planning with foundation models as uncertainty estimators. In submission to RSS 2025, 2025.

M. Zinkevich and T. Balch. Symmetry in Markov decision processes and its implications for single agent and multi agent learning. In *In Proceedings of the 18th International Conference on Machine Learning*, pages 632–640. Morgan Kaufmann, 2001.