

UC Berkeley

UC Berkeley Electronic Theses and Dissertations

Title

Between pixels and policies: Toward interpretable representations for (inter)action

Permalink

<https://escholarship.org/uc/item/24q6v5j1>

ISBN

9798293892969

Author

Stocking, Kaylene Caswell

Publication Date

2025-08-01

Peer reviewed|Thesis/dissertation

Between pixels and policies: Toward interpretable representations for (inter)action

by

Kaylene Caswell Stocking

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Claire Jennifer Tomlin, Chair

Professor Shankar Sastry

Professor Stuart Russell

Professor Alison Gopnik

Summer 2025

Between pixels and policies: Toward interpretable representations for (inter)action

Copyright 2025
by
Kaylene Caswell Stocking

Abstract

Between pixels and policies: Toward interpretable representations for (inter)action

by

Kaylene Caswell Stocking

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Claire Jennifer Tomlin, Chair

Robots and other embodied agents use an internal representation of their surrounding environment to pick actions that are appropriate for their purpose or goal. In robotics, this representation is often computed directly from camera images by using deep learning algorithms to extract relevant high-level features. However, unlike in other computer vision applications, robot representations must support closed-loop interaction, where current actions affect future observations. Furthermore, the need for safety and transparency in robotics motivates closer scrutiny of the contents and limitations of learned representations.

This dissertation argues that performant and interpretable robot representations are a goal of both scientific interest and practical importance. Two possible paths toward these representations are new representation learning algorithms that are engineered with interpretability in mind (“design”), and better tools for improving our understanding of the representations learned by existing algorithms (“interpretation”). However, current techniques fall far short of the goal of both strong learning performance and deep mechanistic understanding. This dissertation describes research that advances the state of the art along both paths. Part I proposes two algorithms that learn symbolic representations that facilitate a robot’s ability to reason about other agents. By focusing on more realistic settings than prior works, these chapters show that interpretable-by-design algorithms need not be limited to simple toy problems. Part II introduces interpretability tools from other disciplines to robotics for the first time, leading to novel insights about the representations learned by a deep end-to-end neural network trained for autonomous driving (similar to human drivers’ representations for low-level vision but not for representations of other agents) and vision-language-action models (deeply semantic despite being fine-tuned to only output actions). A common thread throughout both parts is making connections between robot representations and the representations that support human cognition and action. Whether the design path or the interpretation path ultimately proves more fruitful, this research works toward a holistic understanding of how capable embodied agents represent their environments.

To a universe that sparks curiosity and wonder
and the people in it who let me look

Contents

Contents	ii
1 Two paths to capable AI systems we understand	1
1.1 Robotics, safety, and interpretability	2
1.2 Two paths to meaningful interpretability	3
I Designing interpretable robot representations	6
2 Maximum likelihood constraint inference over continuous state spaces	7
2.1 Introduction	7
2.2 Related Work	10
2.3 Constraint Inference On Continuous State Spaces	10
2.4 Analysis with Pendulum System	16
2.5 Constraint Inference on 1/10 Scale Car	18
2.6 Conclusion	19
3 Maximum likelihood constraint inference with Bayesian posteriors	20
3.1 Introduction	20
3.2 Background and terminology	21
3.3 Estimating the Posterior Probability of a Constraint	22
3.4 Posteriors for Sets of Multiple Constraints	25
3.5 Conclusion	27
4 Linking vision and motion for self-supervised object-centric perception	29
4.1 Introduction	29
4.2 Related work	30
4.3 Approach	31
4.4 Results	35
4.5 Conclusion	39

II	Interpreting robot representations	40
5	Representations in a deep end-to-end driving model predict human brain activity in an active driving task	41
5.1	Introduction	41
5.2	Related work	43
5.3	Voxelwise modeling with deep neural networks for driving	43
5.4	Results and discussion	47
5.5	Conclusion	52
6	Mechanistic interpretability for steering vision-language-action models	55
6.1	Introduction	56
6.2	Related Work	57
6.3	Interpreting VLAs	57
6.4	Steering VLAs	60
6.5	Discussion and Conclusion	65
6.6	Limitations	66
III	Conclusion and end matter	67
7	Final thoughts and future directions	68
	Bibliography	70
8	Appendices	81
8.1	Chapter 2 Appendix: Proofs of theorems	81
8.2	Chapter 4 Appendix: Additional architecture details	83
8.3	Chapter 5 Appendix A: Risk analysis of ridge regression under sparse random projection	84
8.4	Chapter 5 Appendix B: Additional figures	86
8.5	Chapter 5 Appendix C: More details on methodology	86
8.6	Chapter 6 Appendix A: More details on steering interventions	92
8.7	Chapter 6 Appendix B: Interpretability Experiments	93
8.8	Chapter 6 Appendix C: Simulation Experiments	94
8.9	Chapter 6 Appendix D: Hardware Experiments	95

Acknowledgments

I would like to thank the research mentors who generously shared their time and expertise and whose support has been invaluable during my research journey, including Professor Herbert Sauro, Professor Omar Gharbawie, Professor Kenichiro Kamei, Professor Takashi Kozai, Dr. Christopher P. Burgess, Professor Jack Gallant, and of course, the incredible Professor Claire Tomlin. I would also like to thank my many collaborators, from whom I learned a great deal and without whom this work would not be possible. My dissertation committee was a great source of wisdom and encouragement. I also benefited greatly from the support of my friends and colleagues in the Hybrid Systems Lab as well as the greater Berkeley AI Research, EECS, and neuroscience communities, and the Pitt Robotics and Automation Society. I wish I could have written a paper with each and every one of you! Finally, to my family: from the bottom of my heart, thank you for everything.

Chapter 1

Two paths to capable AI systems we understand

Humans, cats, and robots all have something in common: they are embodied agents that need to continuously decide on and carry out physical actions. To do this, they need to be able to both perceive the world around them, and use this perception to decide what to do next. This is simple enough to state, yet it raises profound questions. What does an agent need to know about the world around it in order to make decisions? In other words, what is its *representation* of its environment? And how does it obtain that knowledge from raw perceptual input, whether that input is the firing of neurons¹ or RGB measurements for a set of pixels?

At a low level, the perceptual system of a cat looks nothing like that of a robot equipped with a few cameras. Nonetheless, both biology and computer science have a rich tradition of looking to each other for inspiration, and sometimes even for shared fundamental principles underlying perceptual cognition. What might these principles look like? One influential early theory was that the goal of vision is to build a 3D model of the world that projects a 2D image onto the retina or camera plane, and that this goal can be achieved by detecting low-level visual features such as lines, then finding objects bounded by those lines, and so on. Supporting this theory was the Nobel Prize-winning discovery of individual neurons in the visual cortex that fire when a line of a particular orientation is present in the visual field [1]. Only two years after this discovery, one of the earliest PhD dissertations in computer vision described an algorithm for inferring 3D polygons from simple images of lines [2].

Today, we have powerful deep learning algorithms that can do much more: detect objects, infer depth, and even output control actions for a robot to complete a task. Unfortunately, our understanding of how these algorithms work has not progressed nearly as quickly as their capabilities. They are often referred to as “black boxes”, and while many promising techniques have aimed to interpret their inner workings, these have only lightened the darkness

¹No one tells the brain that a particular incoming signal is e.g. from a cone cell in the retina of the eye—neuronal firing is all there is for us to make sense of.

a little [3]. Things are even worse in biology, where measuring neural activity is orders of magnitude more difficult than it is in artificial systems. The best predictive models we have of higher-order visual cortex are, in fact, the interpretation-resistant features of artificial neural networks [4].

These challenges to deeper understanding are daunting. Meanwhile, rapid progress in deep learning continues, apparently unimpeded by human understanding lagging behind. This progress translates into both success on new and more complicated tasks, as well as greater accuracy and reliability on existing ones. If we are indeed missing something important, the challenge is to articulate exactly what, and why it matters. (Arguably, the principles underlying perceptual cognition are of great scientific interest independent of any particular application. However, I argue below that there are, in fact, very valuable practical applications.) I will briefly zoom out to consider this question more broadly before returning to representations for embodied agents, as well as how this dissertation aims to contribute to this topic.

1.1 Robotics, safety, and interpretability

Roboticians have a long tradition of caring deeply about safety, and especially about mathematical safety guarantees. Guarantees are nice because, if their assumptions are satisfied, we can be absolutely certain about how a robotic system will behave. (For most of the history of the field, a robot making contact with a human would likely cause serious injury—guaranteeing that this won’t happen is obviously desirable.)

As long as robots were mainly confined to factories and other industrial settings, this was a good bargain, because the assumptions underlying the safety guarantees could reasonably be met in such tightly controlled settings. But deep learning is slowly being integrated into various aspects of robotics, from perception to control. Along with it, there is increasing commercial pressure to push robots into less controlled, more “open-world” settings. As a result, providing safety guarantees is becoming much more challenging.

Instead of giving up on safety assurances, these developments should motivate us to double down on making AI systems, including robots, as safe as possible. As autonomous systems become more capable and we integrate them more tightly into our everyday lives, the potential consequences of system failure only become more severe. Furthermore, for systems operating in open-world environments, it’s impossible to rigorously test every possible circumstance in advance. Good performance on benchmarks, therefore, isn’t enough: without a deep understanding of the system, we have no idea whether a strange edge case will cause potentially catastrophic failure after deployment.

Bringing machine learning into robotics does change how we should think about providing guarantees, however. Unlike with classical robotics, where every part of the system could be designed with specific safety specifications in mind, with learned components, we do not know in advance how the final system will solve important parts of the task. For example, the planning algorithm for an autonomous vehicle might be designed to always follow traffic

rules, but rely on accurate state information from a perception system that uses learned components to identify pedestrians and other vehicles. A number of newer works have aimed to provide guarantees even in the presence of these learned components, including by filtering potentially dangerous robot control actions through a “safety filter” [5] or directly verifying learned components [6], [7]. However, there is still a large gap between what is covered by these mathematical tools and real-world systems that are being deployed: assumptions are often too strong, or the systems we can verify are much smaller and simpler than their practically useful counterparts.

Interpretability may be able to help fill this gap. By deepening our understanding of the mechanisms learned components discover, we can improve our ability to make good predictions about how they will behave in various circumstances. This is, after all, exactly how science works: good causal or mechanistic hypotheses allow us to make predictions about complex phenomena ranging from designing buildings that resist earthquakes to proposing novel drug candidates for devastating diseases. Deep learning doesn’t make the quest for mechanistic understanding intractable; instead, it might just make it a little more like biology [8]. And while mathematical safety guarantees are still preferable to even the most well-founded guess, better mechanistic understanding may help with deriving guarantees as well.

1.2 Two paths to meaningful interpretability

If we accept improved mechanistic understanding of learned systems as a valuable goal, there are two broad paths towards achieving it. The first is *design*: constructing the learning system in such a way that its behavior will be easier for humans to understand. The second is *interpretation*: devising new tools that help us better understand existing learned systems, even if they weren’t designed with interpretability in mind.

Outline and contributions

This dissertation describes work that aims to improve the interpretability of robot representations along both of these paths. In the first part, I focus on design. Chapters 2 and 3 discuss how *constraints* can be a powerful way to infer and quickly act on information about surprising behavior from other agents, such as humans. Unlike reward functions, which are famously difficult to interpret and robustly reason about [9], constraints are easy to understand and can be composed in predictable ways. Chapter 2 describes a novel method for inferring the most likely constraints when the state space is continuous, a more realistic setting than small discrete spaces (the focus of most prior work). Then, chapter 3 extends this approach to infer multiple constraints and estimate their posterior likelihoods. Together, these chapters contribute to making constraint-based representations learned from data more practical and powerful.

However, this constraint inference work assumes that a great deal of reliable state information, such as the positions and velocities of other agents, is already available. Chapter 4

looks at the problem of learning to infer such information using only a self-supervised variational inference objective. This method has two main advantages: first, the system itself learns how best to organize information about its perceptual inputs, instead of relying on human-provided labels that are expensive to collect and may be suboptimal for the downstream robotic task. And second, a disentanglement-promoting inductive bias encourages the final learned representation to consist of disentangled factors such as position and color, which may be easier to interpret. These features suggest that novel representations learned from data need not be black boxes that we struggle to understand.

In the second part of the dissertation, I change tacks and focus on interpretation of representations learned by existing state-of-the-art robotics algorithms. First, in chapter 5, I discuss the first effort to map similarity between a deep learning algorithm and human brain activity across the human cortex for an interactive task. Our experimental setup examines internal representations during interactive driving and allows us to identify specific parts of a deep learning driving model that appear to be more and less aligned with brain activity across multiple human drivers. As mentioned above, better alignment is likely to be helpful for interpretability, and this methodology could help us pinpoint issues and measure progress in representational alignment. Furthermore, since driving is an inherently social task, better models of human drivers is directly beneficial for autonomous driving algorithms [10]. Finally, in chapter 6, I focus on interpretation of emerging “vision-language-action” (VLA) models in robotics. Our work is the first application of mechanistic interpretability techniques, originally developed for large language models, to VLAs. Our findings are suggestive of a causal connection between the VLA’s internal semantic representations of concepts like “up” and “slow”, initially acquired during language-only pre-training, and corresponding physical robot actions with real-world grounding. This exciting result illustrates the potential of better interpretability methods to yield meaningful safety-relevant insights about highly complex learned systems.

Beyond robotics

Throughout all of the work described in this dissertation, a recurring theme is making connections between robotics and fields that aim to understand representation learning in biological systems, including neuroscience, developmental psychology, and philosophy of mind. Beyond the usual benefits of crossing disciplinary lines, there are at least two good reasons to believe that making these connections may be especially fruitful. First, while machine intelligence need not necessarily resemble that of humans, increasingly capable AI systems seem to be converging on more humanlike representations [11], [12]. Whether this is because biological inspiration has been genuinely helpful for designing these systems, because they are trained on increasing amounts of human-produced data, or simply because learned representations tend to reflect patterns in the outside world in consistent ways, this empirical trend suggests that we can learn more by bringing these fields together than by studying them in isolation.

Second, it seems plausible that intentionally making AI systems more humanlike in targeted ways will be helpful for understanding them. Among biological intelligences, we often

have an easier time understanding those that are closest to us, whether within humans (shared cultural backgrounds vs. different ones) or across other species (we find octopus intelligence much more mysterious than that of chimpanzees). As an example in AI, the similarity of large-language-model chain-of-thought traces to familiar human reasoning has led to many important insights [13], [14].

Therefore, while this dissertation is primarily about representations and representation learning in robotics, I point out connections to other fields when possible (and in the case of chapter 5, explicitly study one kind of connection). It is my hope that these references are genuinely helpful in working towards a shared understanding of representation learning across biological and artificial systems - an ambition that is not only fascinating, but potentially critical for building autonomous systems we can trust to be safe.

This dissertation was only possible because of the significant contributions of many wonderful co-authors, collaborators, and mentors, including Claire Tomlin, Tianjiao Zhang, D. Livingston McPherson, Jack Gallant, Christopher P. Burgess, Bear Häon, Robert P. Matthew, Christopher Strong, Jingqi Li, Alison Gopnik, Ian Chuang, Zak Murez, and Vijay Badrinarayanan. They have my deepest gratitude and appreciation. Any remaining mistakes are my own!

Part I

Designing interpretable robot representations

Chapter 2

Maximum likelihood constraint inference over continuous state spaces

This chapter is adapted from the previously published paper “Maximum likelihood constraint inference on continuous state spaces,” presented at the 2022 IEEE International Conference on Robotics and Automation, and included with permission from my coauthors D. Livingston McPherson, Robert P. Matthew, and Claire Tomlin.

2.1 Introduction

The behavior of an agent is an important source of information about their goals and surroundings. For example, inverse reinforcement learning estimates the reward function an agent appears to be optimizing, assuming the dynamics are known [15]. In many practical situations, however, we already have a good idea of the reward function an agent may be optimizing. Drivers usually want to stay near the center of their lane, and a person walking across a room typically chooses shorter paths over longer ones. In cases like these, even though the reward function is already known, an agent’s actions can still provide useful information. Consider a driver who temporarily swerves out of their lane to avoid an obstacle. Even if we can’t directly observe the obstacle, a reasonable inference is that something is preventing the driver from taking the expected path. The driver’s swerving behavior is surprising if no obstacle is present, but becomes much more likely when we incorporate a possible obstacle into our model. This intuition can be formalized as maximum likelihood constraint inference (MLCI), first developed in [16]. MLCI uses the maximum entropy framework [15] to identify which constraint in a constraint hypothesis set provides the best explanation for unexpected demonstrator behavior. The constraint hypothesis set can be obtained, for example, by dividing the state space into an even grid. It can also be designed specifically for a particular application, with possible constraints chosen based on their utility or known prevalence. Choosing the hypothesized constraints is analogous to selecting features to be used in the learned reward function in standard inverse reinforcement learning formulations

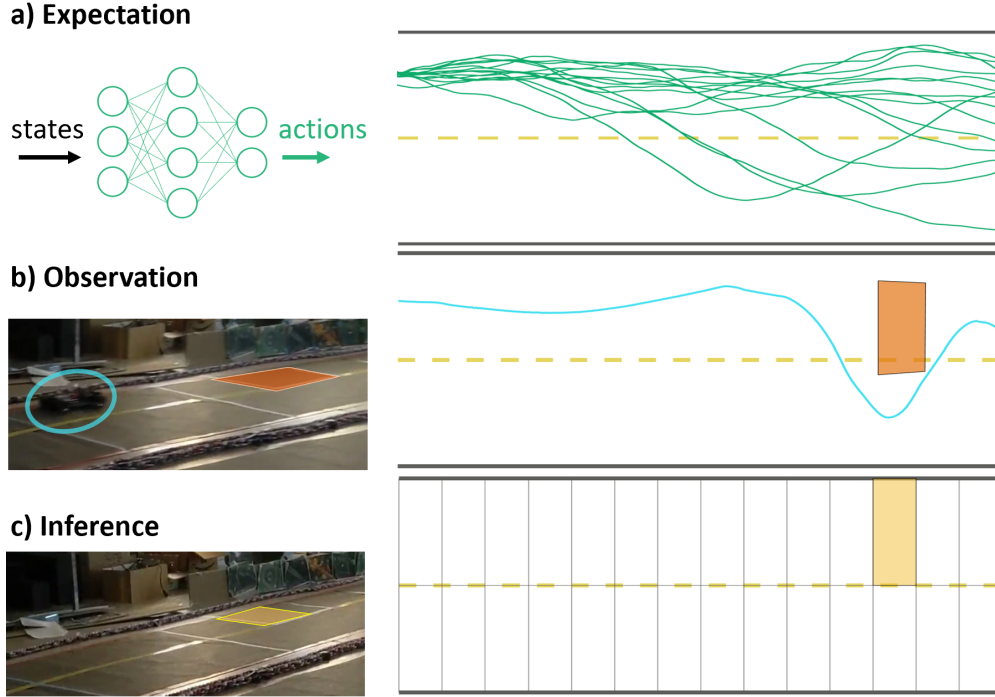


Figure 2.1: Our proposed method allows for fast inference of the most likely constraints on demonstrators. Here, we illustrate our approach on a demonstration from a human-controlled model car driving along a racetrack. Given a known reward function and dynamical model, deep reinforcement learning is used to learn an implicit representation of the maximum entropy trajectory distribution, and trajectories are sampled from this distribution (a). These samples provide an estimate of the expected frequency a demonstrator will violate each constraint in a constraint hypothesis set. When an unusual demonstration is observed as the car swerves around an obstacle on the road (b), its constraint violations can be compared with the pre-computed expectations, and our proposed algorithm identifies the constraints that provide the best explanation of the demonstrator’s behavior (c).

[15]. Although using a constraint hypothesis set limits which constraints it is possible to learn, it also acts as a source of inductive bias and means the results of MLCI tend to be easy to interpret.

Unfortunately, the MLCI algorithm presented in [16] only works for system models with discrete, finite state-action spaces. While it is possible to approximate continuous dynamical systems for use with the algorithm, this process introduces significant error into the dynamics and suffers from an exponential increase in the number of discrete states required as the number of state dimensions in the dynamics increases (often called the “curse of dimensionality”) [17]. In this chapter, we present a novel MLCI method that works directly with continuous state spaces, allowing for a much closer approximation to continuous real-world dynamics and avoiding the curse of dimensionality in the state-space. Our method first learns an implicit representation of the expected maximum entropy distribution over demonstrator trajectories by leveraging deep reinforcement learning (RL). Then, we sample from this distribution to obtain Monte Carlo estimates of expected constraint violations, which allows us to perform MLCI. Our method retains several key advantages from [16], including working with sub-optimal demonstrations and allowing for most calculations to be pre-computed before any demonstrations are observed.

Our work is motivated in part by the observation that humans also seem to utilize a form of constraint inference to reason about the behavior of others. The driving scenario mentioned above is one example of this, but it has also been supported in the psychology literature. In one insightful experiment, Gergely, Bekkering, and Király [18] show 14-month (preverbal) infants a box with a large button on it. An experimenter pushes the button with her forehead, and then the box is given to the infant to freely interact with. The infant typically imitates the experimenter, and tries pushing the button with their forehead as well. But in a twist, the experimenter sometimes wears a blanket that trapped her arms underneath it while pushing the button. In this condition, infants push the button with their hands instead of with their forehead, suggesting that they were able to infer that the experimenter would have done the same if her hands hadn’t been trapped by the blanket. In other experiments, young children spontaneously help adults by handing them objects they unsuccessfully reach for [19] or opening doors for them [20], suggesting they understand the difference between the adult’s underlying goal (reward function) and constraints preventing them from achieving it. Together, these experiments suggest that humans employ sophisticated reasoning about the goals and constraints from others, even before they acquire language.

The remainder of this chapter is laid out as follows. In Section 2.2, we present prior work and discuss the trade-offs between various constraint inference methods. In Section 2.3, we briefly overview the mathematical background for MLCI and describe our algorithm in more detail. Section 2.4 presents analysis on a 2-dimensional pendulum model with demonstrations generated by an optimal control algorithm. Finally, Section 2.5 provides an example of successful real-world constraint inference on the trajectory of a 1/10-scale car, driven by a human, that must swerve to avoid an obstacle.

2.2 Related Work

Different perspectives on constraint inference have yielded a variety of methods which differ in what kinds of systems can be modeled and what assumptions are made about the demonstrator. Our work is most directly inspired by the maximum likelihood method introduced by [16], which identifies the most likely constraint(s) from a hypothesis set even when the demonstrator may behave sub-optimally with respect to their goal. This is achieved by using the maximum entropy framework, which is also leveraged by [21] to learn task specifications. Specifications can be thought of as a generalization of Markovian (memoryless) constraints to complex multi-step behaviors. Both of these methods require discrete state-action spaces, although [17] examines how continuous dynamical systems might be approximated with discrete spaces for the purposes of constraint inference.

Many other methods work directly with continuous dynamics, but drop the maximum likelihood feature. Chou et al. [22] assume that all possible trajectories that could earn higher reward than the demonstration must be constrained in some way. Other methods use heuristics about the demonstrator’s behavior rather than an explicit reward function. For example, Pais et al. [23] assume that constrained behaviors will have high within-demonstration variance and low between-demonstration variance, and Li et al. [24] uses the heuristic that maintaining a robot end effector in the same orientation throughout a demonstration suggests a constraint. Lin et al. [25] present a kinematics-based approach for learning constraints that affect how a nominal policy is executed in different environments. An approach proposed by Mehr et al. [26] is specialized for online constraint inference in the context of shared autonomy, where mis-identified constraints can be corrected by the user.

Malik et al. [27] recently presented an approach that works with continuous state-action spaces and retains the maximum likelihood framework. They use deep reinforcement learning to identify a constraint function that will allow an agent to avoid violating the constraint(s) affecting the demonstrator. The method we propose similarly uses deep RL to handle continuous state spaces, but focuses instead on learning constraints from a pre-specified constraint hypothesis set. Because of this difference in perspective, our method enables checking for specific possible constraints and leveraging their presence (or absence) to make decisions beyond executing a modified policy. Our method can also leverage prior knowledge of the dynamics and reward function to perform fast online inference. Therefore, our work is likely to be especially useful in situations where the system is known before observing demonstrations.

2.3 Constraint Inference On Continuous State Spaces

Markov Decision Dynamics

Our approach relies on a system model formulated as a deterministic Markov Decision Process (MDP). The MDP is a tuple of four elements:

- A state space $\mathcal{S}$ to navigate and a set of actions $\mathcal{A}$ that can be taken at each state. Both $\mathcal{S}$ and $\mathcal{A}$ may be either continuous or discrete.
- A transition kernel $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ that describes how $a_{t-1} \in \mathcal{A}$ affects s_t . The repeated application of this transition kernel generates a sequence of states over a time horizon $t \in [0 : T]$ given a sequence of action choices. The couple of state sequence and action sequence is a trajectory $\xi = (s_0, a_0, \dots, a_{T-1}, s_T)$ and the space of all possible trajectories is Ξ .
- An objective metric $R : \Xi \rightarrow \mathbb{R}$ (also called a reward function) that measures the quality of trajectories.

If the state and action spaces are both discrete and finite, we say that the MDP is tabular. The MLCI approach developed by [16] only works for tabular MDPs. The method we propose in this section, however, admits MDPs with discrete or continuous state spaces. For many continuous dynamical systems, this allows for a model that is a much closer approximation to the true dynamics. Note that the MDP formulation requires discrete time steps; however, continuous dynamics can be approximated well with a sufficiently small δt using standard simulation methods [28].

Maximum Entropy Likelihood on Trajectories

After observing a demonstrator's behavior, we suspect that they may be avoiding a constraint we don't know about when their trajectory is surprising, in the sense of accumulating significantly less reward than should be possible. We can formalize this intuition by defining a distribution of expected demonstrator behavior. Following [15], we adopt the maximum entropy likelihood distribution, under which the probability of a trajectory on a tabular MDP is defined as:

$$P(\xi) = \frac{1}{Z} e^{\beta R(\xi)} = \frac{1}{Z} q(\xi) \quad (2.1)$$

$$Z = \sum_{\xi' \in \Xi} e^{\beta R(\xi')} \quad (2.2)$$

where β is a temperature parameter that reflects how noisy (sub-optimal) the demonstrator is, $q(\xi)$ is the unnormalized trajectory probability, and Z is a normalizing constant. For continuous state-action spaces, this distribution can be generalized to a probability density:

$$p(\xi) = \frac{e^{\beta R(\xi)}}{\int_{\xi' \in \Xi} e^{\beta R(\xi')} d\xi'} = \frac{1}{Z} q(\xi) \quad (2.3)$$

To remain agnostic as to whether the state-action space is continuous or discrete, we define the maximum entropy distribution as $\pi(\xi) = \frac{1}{Z} q(\xi)$ for both cases.

Constraints invalidate any trajectory that would otherwise enter the constrained region of the state-action space, rendering its probability zero. Therefore, when we augment a deterministic MDP with a constraint c , we obtain a new maximum entropy distribution:

$$\pi_c(\xi) = \begin{cases} \frac{1}{Z_c} q(\xi), & (s_t, a_t) \notin c \ \forall t \in [0 : T] \\ 0, & \text{otherwise} \end{cases} \quad (2.4)$$

Where Z_c is a new, smaller normalizing constant that reflects the trajectory probability mass removed by imposing the constraint. Let Z_0 be the normalizing constant for the original, unconstrained MDP. Assuming the demonstrator doesn't violate c , their behavior is more likely under the new distribution because $Z_c < Z_0$. In fact, given a uniform belief prior over each constraint in a constraint hypothesis set $\mathcal{C}$, the most likely constraint is the one that yields the smallest Z_c . Therefore, to perform MLCI, it is sufficient to calculate the ratio Z_{c_i}/Z_0 for each $c_i \in \mathcal{C}$ and identify the smallest value. If subsequent demonstrations violate new constraints, these are simply removed from the hypothesis set, and the smallest remaining Z_{c_i}/Z_0 indicates the likeliest constraint.

Theorem 1. *Let $\mathbb{1}_{c_i}(\xi)$ be the indicator function that trajectory ξ does not violate constraint c_i at any $t \in [0 : T]$. We have:*

$$\frac{Z_{c_i}}{Z_0} = \mathbb{E}_{\xi \sim \pi_0}[\mathbb{1}_{c_i}(\xi)]$$

(A proof is provided in the appendix to this chapter in section 8.1.)

Theorem 1 suggests that a MLCI algorithm requires two components: a representation of the unconstrained maximum entropy distribution $\pi_0(\xi)$, and a way to calculate expected constraint violation under this distribution. For tabular MDPs, both components can be obtained via dynamic programming [16]. Although a method for approximating continuous dynamical systems with tabular MDPs was presented in [17], it is desirable to work directly with continuous state spaces and avoid the approximation errors inherent to the discretization process. This motivates a new approach.

A Sampling-based Method for Continuous MLCI

In this section we describe our algorithm for constraint inference on MDPs with continuous state spaces.

Deep Reinforcement Learning of $\pi_0(\xi)$

As we will see in the following section, a direct representation of the maximum entropy distribution $\pi_0(\xi)$ isn't necessary for MLCI; simply being able to sample trajectories from this distribution is sufficient. This motivates us to approximate $\pi_0(\xi)$ by leveraging the

policy gradient method in reinforcement learning (RL). Following [27], consider the KL-divergence between the true maximum entropy distribution $\pi_0(\xi)$ and a learned policy $\pi_\theta(\xi)$ parameterized by θ .

$$\begin{aligned} D_{KL}(\pi_\theta || \pi_0) &= \mathbb{E}_{\xi \sim \pi_\theta} [\log \pi_\theta(\xi) - \log \pi_0(\xi)] \\ &= \mathbb{E}_{\xi \sim \pi_\theta} [\log \pi_\theta(\xi) - \beta R(\xi) + \log Z_0] \end{aligned} \quad (2.5)$$

Since $\log Z_0$ is a constant, we can minimize the divergence by maximizing the following modified expected reward function:

$$J(\pi_\theta) = \mathbb{E}_{\xi \sim \pi_\theta} [R(\xi) - \frac{1}{\beta} \log \pi_\theta(\xi)] \quad (2.6)$$

This objective can be used with any policy gradient method. In our experiments, we use proximal policy optimization (PPO) due to its good performance and stability [29].

The parameterization of π_θ is an important limiting factor in how good of an approximation can be achieved. Most RL algorithms working with continuous action spaces learn a Gaussian distribution over actions at each state (e.g. see [30]). However, this is unsuitable for our application because it results in a policy that commits to a single “global” strategy rather than simultaneously exploring multiple strategies with probability determined by their expected reward. Therefore, we handle continuous action spaces by discretizing the range of possible action inputs. A categorical action policy can then be used to learn action distributions with an arbitrary shape at each state.

Sampling-Based Approximation of Z_{c_i}/Z_0

The learned policy π_θ doesn’t directly yield estimates of the trajectory probability distribution $\pi_0(\xi)$. However, for a potential constraint c_i of interest, we can obtain a Monte Carlo estimate of $\mathbb{E}_{\xi \sim \pi_0} [\mathbb{1}_{c_i}(\xi)]$ by simulating policy executions many times and checking whether each sampled trajectory $\xi \sim \pi_\theta$ violates c_i . Since the constraint avoidance indicator $\mathbb{1}_{c_i}(\xi)$ for $\xi \sim \pi_\theta$ is a Bernoulli random variable, Hoeffding’s inequality tells us that the Monte Carlo estimate will be very close to the true constraint violation probability (the probability that the absolute error δ is larger than ϵ decays exponentially with the number of samples n : $P(\delta > \epsilon) \leq 2e^{-2\epsilon^2 n}$). By Theorem 1, calculating the fraction of trajectories that violate each hypothesized constraint is sufficient for determining the most likely constraint. If constraint regions are determined by actions (for example, a constraint on the amount of torque that can be applied with a motor), extra care may need to be taken with determining how the actions are discretized and constraint violation is determined for sampled trajectories. In this work we focus on state-based constraints.

Real-Time Constraint Inference

Once we have learned the policy π_θ and calculated expected constraint violation via Monte Carlo sampling, we are ready to perform constraint inference. This last step simply consists

of sorting by magnitude the estimated Z_{c_i}/Z_0 for each hypothesized constraint, then checking this ranking against the constraints violated by the demonstrations. The smallest Z_{c_i}/Z_0 that is not violated by any demonstration is the most likely constraint, as described previously in this section. It is important to note that prior to this final step of checking which constraints are violated by the demonstrations, no information from the demonstrations is used by our algorithm. Therefore, even though the policy learning and sampling steps described in this section may be time-consuming, they can be pre-computed when the MDP and constraint hypothesis set are known in advance. This means that constraint inference can occur in real-time as demonstrations are collected, a key difference from prior work that requires demonstrations to be available throughout the deep learning process [27].

Highly Optimal Demonstrators

One drawback to a Monte-Carlo approach to calculating expected constraint violation is that if the demonstrator behaves close to optimally (i.e. with a large β parameter), the expected unconstrained trajectories will be tightly clustered around the most optimal path(s). This makes accurate constraint inference difficult due to the paucity of trajectory samples with lower rewards under the unconstrained distribution, since these lower-reward trajectories would only occur when acting under a constraint. We can circumvent this problem by learning a π_θ distribution with a lower β than the demonstrator, and then using techniques from importance sampling to correct our results for the true reward function. To formalize this idea, consider the case where we sample from $\pi_{s,0}(\xi) = \frac{1}{Z_{s,0}} e^{\beta_s R(\xi)}$, but the true reward function induces the distribution $\pi_{d,0}(\xi) = \frac{1}{Z_{d,0}} e^{\beta_d R(\xi)}$. If we can determine $Z_{d,c_i}/Z_{s,0}$ for each $c_i \in \mathcal{C}$, this allows us to determine which Z_{d,c_i} is smallest and therefore identify the likeliest constraint. Since

$$\frac{Z_{d,c_i}}{Z_{s,0}} = \frac{Z_{d,c_i}}{Z_{s,c_i}} \frac{Z_{s,c_i}}{Z_{s,0}} \quad (2.7)$$

and we can use Theorem 1 to estimate $Z_{s,c_i}/Z_{s,0}$, all that remains is to calculate $Z_{d,c_i}/Z_{s,c_i}$. The following theorem allows us to do this:

Theorem 2.

$$\frac{Z_{d,c_i}}{Z_{s,c_i}} = \mathbb{E}_{\xi \sim \pi_{s,c_i}} [e^{(\beta_d - \beta_s) R(\xi)}]$$

(A proof is provided in the appendix.) Since we assume deterministic dynamics, sampling from π_{s,c_i} can be easily achieved by sampling from $\pi_{s,0}$ and discarding trajectories that violate constraint c_i .

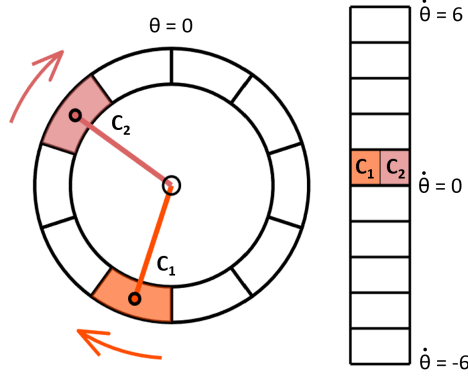


Figure 2.2: This figure illustrates the ground truth constraints for our pendulum dynamics analysis. The constraint hypothesis space evenly divides the state space into 100 cells, 10 along the angle axis and 10 along angular velocity. The two constraints used in our experiments, C_1 and C_2 , are shown here in different shades of orange. The constraints cover different angle regions but the same angular velocity.

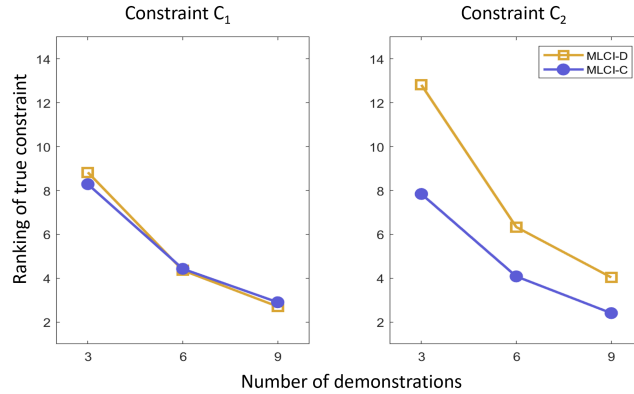


Figure 2.3: Constraint inference accuracy for our method (MLCI-Continuous, blue circles), with the performance of the discretized state space method from [17] (MLCI-Discrete, gold squares) as a comparison. The y-axis measures the likelihood ranking of the ground truth constraint among all 100 hypothesized constraints, on average. A score of 1 means that the true constraint is identified as the most likely.

2.4 Analysis with Pendulum System

Pendulum Dynamics

The pendulum model consists of a 2-dimensional state space (angle θ and angular velocity $\dot{\theta}$). The 1-dimensional control input is the normalized torque applied at the base of the pendulum:

$$\ddot{\theta} = \frac{g}{l} \cdot \sin(\theta) + u \quad (2.8)$$

Where the gravitational constant g and the length of the pendulum l are both assumed to be 1 for simplicity. The constraint hypothesis set is an evenly spaced 10-by-10 grid of non-overlapping cells that cover the state space of $\theta \in [0, 2\pi]$ and $\dot{\theta} \in [-6, 6]$, for a total of 100 possible constraints. We assign a uniform prior likelihood to every constraint. The demonstrator wants to arrive at a particular goal state $\hat{s}_T$ at the end of a $T = 5$ s period while minimizing the total squared torque and avoiding the true constraint region, K . In practice, this is achieved by optimizing the reward function

$$\begin{aligned} u^* = \max_{u \in \mathcal{U}} & -[\alpha_1 \|s(T) - \hat{s}_T\|_2^2 + \alpha_2 * \int_0^T u(t)^2 dt] \\ \text{s.t. } & s(t) \notin K \quad \forall t \in [0, T] \end{aligned} \quad (2.9)$$

using the constrained optimal control method described in [17]. $\alpha_1 = 80$ and $\alpha_2 = 0.05$ were chosen so that most trajectories terminate in a small neighborhood around the goal state. Note that since the pendulum dynamics are nonlinear, the demonstrations generated with this method are not guaranteed to be globally optimal and may correspond to local maxima in the reward function.

Experimental Design

We examined two possible ground truth constraints, shown in Fig. 2.2. We generated 65 demonstration trajectories with random start and goal points for each constraint. The MDP for this system was formulated using a timestep of $\delta t = 1/60$ s and 5 discrete action choices linearly distributed between $u = -1$ and $u = 1$. A separate deep RL policy was trained for each of the 65 goal states by inserting the reward function in equation (2.9) into the objective derived in equation (2.6). Using $\beta = 0.67$ was found to provide the best fit to the demonstrations. Random starting states were used during training. After training, sampling was performed by using the same starting state as the demonstrator and performing 10,000 independent policy rollouts on the simulated pendulum environment.

Constraint Inference Results

To analyze constraint inference performance on the pendulum system, we pick demonstrations at random (across 200 trials) and use the average constraint violations from the cor-

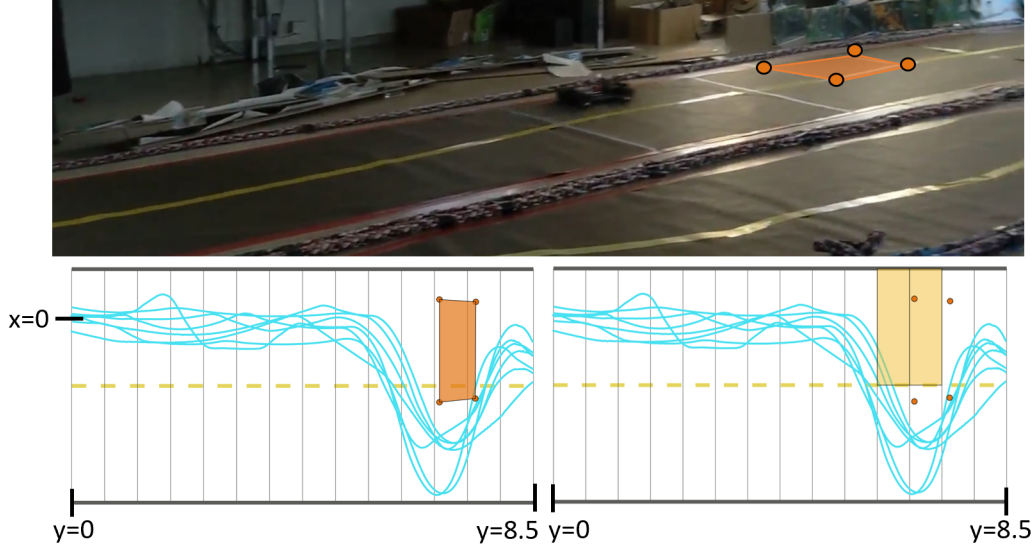


Figure 2.4: Constraint inference on a 1/10-scale car trajectory along a racetrack. The upper panel shows a picture of the car approaching the obstacle region, highlighted in orange. The bottom panels show the x and y components of the corresponding MDP space with 7 human demonstration trajectories plotted. The grid in these panels shows the hypothesized constraint set, where each constraint is a region in (x,y) space. In the bottom left panel, the ground truth constraint is highlighted. In the bottom right panel, the possible constraints identified as most likely after observing a single demonstration are shown. The left-hand constraint is the most likely for 3 demonstrations, while the neighboring region to the right is the most likely for the remaining 4 demonstrations. The orange circles mark the position of the true constraint, which is close to the inferred constraints.

responding trajectory samples to rank the most likely constraints. We compare the performance of our algorithm to the best hyperparameter settings for the approximate discrete state space approach proposed in [17]. (A direct comparison to the work of Malik et al. [27], which also performs MLCI on continuous systems, is not possible since this method does not utilize a constraint hypothesis set.) The results for each method are shown in Fig. 2.3. Our approach identifies the ground truth constraint as one of the most likely after observing only a few demonstrations for both C_1 and C_2 . The inference accuracy of our method is similar to the discrete state space method for C_1 , which lies in a region of the pendulum state space that is close to a stable equilibrium point. However, our method performs comparatively better at identifying C_2 , where the pendulum dynamics are unstable and the discrete state space approximation suffers accordingly. This difference highlights the advantage of using a continuous state space for performing constraint inference on continuous dynamical systems.

2.5 Constraint Inference on 1/10 Scale Car

To demonstrate the effectiveness of our method in a realistic setting, we performed constraint inference on demonstrations from a human radio-controlling a 1/10 scale Traxxas robotic car along a racetrack, attempting to stay in their lane and reach the goal $(\hat{x}, \hat{y})$ quickly (reward) while avoiding an obstacle region marked on the road (constraint). The position of the car was recorded with an Optitrack system, and the car’s speed and heading were inferred from the position data.

MDP formulation

We modeled the car system with idealized 4-dimensional unicycle dynamics, where the control inputs are turning velocity (u_1) and forward acceleration (u_2). These dynamics have the following form:

$$\begin{aligned} \dot{x} &= v \cos(\theta) & \dot{\theta} &= u_1 \\ \dot{y} &= v \sin(\theta) & \dot{v} &= u_2 \end{aligned} \quad (2.10)$$

The human is assumed to be optimizing the following reward function:

$$\begin{aligned} u^* &= \max_{u \in \mathcal{U}} - \int_0^T \left\| \begin{bmatrix} \alpha_1(x(t) - \hat{x}) \\ \alpha_2(y(t) - \hat{y}) \end{bmatrix} \right\|_1 dt \\ \text{s.t. } & s(t) \notin K \quad \forall t \in [0 : T] \end{aligned} \quad (2.11)$$

where the roadway is aligned with the y-axis so that the car starts at $(x(0) = 0, y(0) = 0)$, $\hat{x} = 0$ is the center of the right lane, and $\hat{y} = 8.5$ is the far end of the road segment. K is an obstacle in the roadway that prevents the human from being in the right lane between $y = 6.7$ and $y = 7.4$, but is unknown to our algorithm. T is a variable time horizon where the episode ends once the car crosses the $y = \hat{y}$ line.

To formulate the MDP model, we choose a discrete action space with an evenly spaced grid of 9 points between $(u_1 = -0.5, u_2 = -1)$ and $(u_1 = 0.5, u_2 = 1)$. We use a simulation time-step of $\delta t = 0.05$ s. These values, as well as the reward function parameters $\alpha_1 = 100$ and $\alpha_2 = 0.05$, and the human’s temperature parameter $\beta = 200$, were chosen to create a close match between the human’s behavior when not attempting to avoid an obstacle (i.e. ignoring the constraint K) and a deep RL policy trained with the unconstrained reward function. The constraint hypothesis set is 28 evenly spaced regions along the roadway in (x, y) space, 14 in each lane. The physical racetrack and a diagram of the corresponding MDP model are shown in Fig. 2.4. A policy was trained to perform the task of driving the car following the reward function in equation (2.11), with an additional reward bonus for reaching the goal and a penalty for leaving the track. 1000 trajectories were sampled from the resulting π_θ distribution. Trajectories that left the racetrack or did not reach the goal were discarded. The remaining samples allowed us to rank the most likely candidate constraints before observing any demonstrations.

Constraint inference

7 demonstration trajectories were collected. For each demonstration, the most likely constraint after observing the single demonstration was determined by removing constraints violated by the demonstration from the candidate set, then selecting the most likely remaining constraint using the pre-computed $Z_{s,c_i}/Z_{s,0}$ estimates. This calculation can be performed very quickly - all that is required is to determine which candidate constraints the demonstration violates. As shown in the bottom-right panel of Fig. 2.4, for 3 of the demonstrations, the candidate constraint that most closely matches the true constraint region is identified as the most likely constraint. For the remaining 4 demonstrations, a neighboring candidate constraint is selected, although it is still a close approximation to the ground truth constraint. This second region is selected because the human turns slightly earlier than is necessary to avoid the obstacle in these demonstrations.

2.6 Conclusion

This chapter presented a novel algorithm for maximum likelihood constraint inference that works on MDPs with continuous state spaces. For many continuous dynamical systems, these MDPs provide a much closer approximation to the true underlying dynamics than is possible with tabular spaces, which suffer from the curse of dimensionality. Our algorithm works with continuous state spaces by leveraging a Monte Carlo sampling approach to track expected constraint violation, which requires only an implicit representation of the maximum entropy trajectory distribution. For continuous state spaces, deep reinforcement learning yields such a representation. Although our method does rely on a discrete action space and discrete transition dynamics, we show in our experiments that accurate constraint inference is nevertheless possible for interesting continuous systems. By inferring constraints from a user-specified hypothesis set, we obtain an interpretable result (although this does come at the cost of restricting which constraints can be inferred). Our approach also allows for significant pre-computation, so that only a trivial filtering process is required at the time demonstrations are observed. The maximum likelihood framework we use is well-suited to noisy or sub-optimal demonstrators including non-expert humans, enables real-time constraint inference, and leads to interpretable and actionable inferences.

Chapter 3

Maximum likelihood constraint inference with Bayesian posteriors

3.1 Introduction

As we bring robots out of the factory and into unstructured human spaces, it becomes increasingly important for them to reason about and react to the behavior of other agents. While there are many things a robot might wish to infer about a demonstrator, such as their goal or even an entire reward function that they appear to be optimizing [15], constraint inference is an especially exciting topic because of its relevance to maintaining safety. For example, consider an autonomous vehicle that observes a driver in front of them unexpectedly swerving. Even if the autonomous vehicle can't sense the obstacle that caused the swerving behavior, it's reasonable to infer that there may be an obstacle and prepare to take evasive action (i.e., to form a new motion plan that respects the inferred constraint). Constraint inference formalizes the process of determining which kinds of constraints provide the best explanation for a particular demonstration.

Prior work has developed the notion of *maximum likelihood* constraint inference, which uses the principle of maximum entropy to identify the most likely constraints on a demonstrator [16], and in chapter 2, we saw how this procedure can be extended to continuous state spaces. However, neither of these approaches tackled the problem of estimating how likely a particular constraint is to be “real.” This is important because sub-optimal or noisy demonstrators may respect some constraints by chance, and this is especially likely to occur as we consider larger sets of possible constraints. Simply finding the most likely constraint doesn't tell us how confident we should be in the inference, which is crucial for making decisions based on the inferred constraint(s).

Ideally, we wish to obtain a posterior probability that a particular constraint (or set of multiple constraints) is the “true” constraint affecting the demonstrator, given some prior probability we place on the constraint being present. Unfortunately, calculating this probability exactly is computationally intractable, with a complexity on the order of $O(2^n)$ for n

possible constraints. In this work, we present an algorithm that, under certain conditions, can estimate the posterior probability with a complexity of $O(n^2)$. This improvement makes real-time estimation possible when the set of possible constraints is relatively small. Our algorithm represents a significant step towards practical constraint inference on live robotic platforms.

3.2 Background and terminology

We first quickly recap some relevant concepts from the previous chapter. Our approach relies on a system model formulated as a deterministic Markov Decision Process (MDP). We use the same definition of the MDP as in the previous chapter (see section 2.3 for more details). Briefly, the MDP is a tuple of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition kernel $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, and a reward function $R : \Xi \rightarrow \mathbb{R}$.

After observing a demonstrator’s behavior, we suspect that they may be avoiding a constraint we don’t know about when their trajectory is surprising, in the sense of accumulating significantly less reward than should be possible. We can formalize this intuition by defining a distribution of expected demonstrator behavior. Following [15], we adopt the maximum entropy likelihood distribution, under which the probability of a trajectory on a tabular MDP is defined as:

$$P(\xi) = \frac{1}{Z} e^{\beta R(\xi)} = \frac{1}{Z} q(\xi) \quad (3.1)$$

$$Z = \sum_{\xi' \in \Xi} e^{\beta R(\xi')} \quad (3.2)$$

where β is a temperature parameter that reflects how noisy (sub-optimal) the demonstrator is, $q(\xi)$ is the unnormalized trajectory probability, and Z is a normalizing constant. For continuous state-action spaces, this distribution can be generalized to a probability density:

$$p(\xi) = \frac{e^{\beta R(\xi)}}{\int_{\xi' \in \Xi} e^{\beta R(\xi')} d\xi'} = \frac{1}{Z} q(\xi) \quad (3.3)$$

To remain agnostic as to whether the state-action space is continuous or discrete, we define the maximum entropy distribution as $\pi(\xi) = \frac{1}{Z} q(\xi)$ for both cases.

Constraints invalidate any trajectory that would otherwise enter the constrained region of the state-action space, rendering its probability zero. Therefore, when we augment a deterministic MDP with a constraint c , we obtain a new maximum entropy distribution:

$$\pi_c(\xi) = \begin{cases} \frac{1}{Z_c} q(\xi), & (s_t, a_t) \notin c \ \forall t \in [0 : T] \\ 0, & \text{otherwise} \end{cases} \quad (3.4)$$

Where Z_c is a new, smaller normalizing constant that reflects the trajectory probability mass removed by imposing the constraint. Let Z_0 be the normalizing constant for the original,

unconstrained MDP. Assuming the demonstrator doesn't violate c , their behavior is more likely under the new distribution because $Z_c < Z_0$. In fact, given a uniform belief prior over each constraint in a constraint hypothesis set $\mathcal{C}$, the most likely constraint is the one that yields the smallest Z_c . Therefore, to perform MLCI, it is sufficient to calculate the ratio Z_{c_i}/Z_0 for each $c_i \in \mathcal{C}$ and identify the smallest value. If subsequent demonstrations violate new constraints, these are simply removed from the hypothesis set, and the smallest remaining Z_{c_i}/Z_0 indicates the likeliest constraint. As shown in chapter 2, if we let $\mathbb{1}_{c_i}(\xi)$ be the indicator function that trajectory ξ does not violate constraint c_i at any $t \in [0 : T]$, we have that

$$\frac{Z_{c_i}}{Z_0} = \mathbb{E}_{\xi \sim \pi_0}[\mathbb{1}_{c_i}(\xi)] \quad (3.5)$$

The expectation term on the right-hand side of equation (8.1) can be calculated using dynamic programming or a sampling-based method.

3.3 Estimating the Posterior Probability of a Constraint

Preliminaries

Consider a set of n possible constraints c_i for $i = 1, \dots, n$. Since in practice arbitrary combinations of these may affect the demonstrator, we define the complete hypothesis set of all possible *sets* of constraints, $\mathcal{H}$, as the power set of the individual constraints. After observing a demonstration, any constraints violated by it have probability zero and we can ignore them in our calculations. Therefore, without loss of generality, in the sequel we assume that $\mathcal{H}$ contains only constraints which are respected (not violated) by the demonstrator.

We wish to determine the probability that a particular constraint set $\hat{\mathcal{C}}$ is the true constraint set $\mathcal{C}$ affecting the demonstrator, given our observation of their behavior (the demonstration trajectory ξ). Using Bayes' rule, we have that:

$$\begin{aligned} P(\mathcal{C} = \hat{\mathcal{C}}|\xi) &= \frac{P(\xi|\mathcal{C} = \hat{\mathcal{C}})P(\mathcal{C} = \hat{\mathcal{C}})}{\sum_{\hat{\mathcal{C}}' \in \mathcal{H}} P(\xi|\mathcal{C} = \hat{\mathcal{C}}')P(\mathcal{C} = \hat{\mathcal{C}}')} \\ &= \frac{(1/Z_{\hat{\mathcal{C}}})q(\xi)P(\mathcal{C} = \hat{\mathcal{C}})}{\sum_{\hat{\mathcal{C}}' \in \mathcal{H}} (1/Z_{\hat{\mathcal{C}}'})q(\xi)P(\mathcal{C} = \hat{\mathcal{C}}')} \\ &= \frac{(Z_{\hat{\mathcal{C}}}/Z_0)^{-1}P(\mathcal{C} = \hat{\mathcal{C}})}{\sum_{\hat{\mathcal{C}}' \in \mathcal{H}} (Z_{\hat{\mathcal{C}}'}/Z_0)^{-1}P(\mathcal{C} = \hat{\mathcal{C}}')} \end{aligned} \quad (3.6)$$

where $P(\mathcal{C} = \hat{\mathcal{C}})$ is the prior probability we place on $\hat{\mathcal{C}}$ being the true constraint set.

Unfortunately, there are two significant challenges to computing the exact value of this expression. First, we need all of the partition constant ratios $(Z_{\hat{\mathcal{C}}}/Z_0)$. Using equation (8.1) allows us to compute these when $\hat{\mathcal{C}}$ consists of a single constraint, via sampling or

dynamic programming as described in [16] and chapter 2. Furthermore, it is possible to calculate the ratio for combinations of constraints by using this algorithm repeatedly. For example, if $\hat{C} = c_1 \cup c_2$, we can first calculate (Z_{c_1}/Z_0) as usual, then add the constraint c_1 to our dynamics model and run the constraint inference algorithm a second time to obtain $(Z_{\hat{C}}/Z_{c_1})$. Putting these together, we have $(Z_{\hat{C}}/Z_0) = (Z_{\hat{C}}/Z_{c_1})(Z_{c_1}/Z_0)$. However, following this method to calculate every ratio would require running constraint inference a very large number of times. Furthermore, even if we were able to calculate or estimate all of the $(Z_{\hat{C}}/Z_0)$ terms in the denominator, summing all 2^n terms scales very poorly in the number of constraints. Therefore, this calculation is intractable for all but extremely small constraint hypothesis sets.

Tractable Lower Bound for the Posterior Denominator

In this section, we tackle the two challenges described above that make an exact calculation of the posterior probability $P(\mathcal{C} = \hat{C}|\xi)$ intractable. Our goal is to obtain an estimate $\hat{S}$ of the denominator $S = \sum_{\hat{C}' \in \mathcal{H}} (Z_{\hat{C}'}/Z_0)^{-1} P(\mathcal{C} = \hat{C}')$. To estimate the $(Z_{\hat{C}}/Z_0)$ terms, consider that after running constraint inference with the nominal (unconstrained) dynamics model, we obtain the partition constant ratios (Z_{c_i}/Z_0) for every constraint set consisting of a single constraint c_i . For an arbitrary constraint set $\hat{C}$ that contains c_i , we must have that $Z_{\hat{C}} \leq Z_{c_i}$, because fewer trajectories sampled from the unconstrained policy will respect all of $\hat{C}$ than just c_i in expectation. Since this will be true for every $c_i \in \hat{C}$, we can obtain the tightest upper bound for $(Z_{\hat{C}}/Z_0)$ as follows:

$$(Z_{\hat{C}}/Z_0) \leq \arg \min_{c_i \in \hat{C}} (Z_{c_i}/Z_0) \quad (3.7)$$

Using equation (3.7) to approximate the partition constant ratios for every $\hat{C} \in \mathcal{H}$ gives us both an estimate and guaranteed lower bound for the value of S .

We still need to sum all 2^n of these terms, but this can be sped up substantially in the case where the prior $P(\mathcal{C} = \hat{C})$ depends only on the number of individual constraints in $\hat{C}$. This would occur, for example, if we place equal prior probability on each individual constraint, and assume that whether each constraint is present or absent is a random event independent from other constraints. We denote the number of individual constraints in $\hat{C}$ as $|\hat{C}|$, and the prior probability as $P(\mathcal{C} = \hat{C}) = K(|\hat{C}|)$.

Without loss of generality, assume that the individual constraints c_i are ordered so that $(Z_{c_1}/Z_0) \leq \dots \leq (Z_{c_n}/Z_0)$. With this convention, (Z_{c_1}/Z_0) is the best bound for every $(Z_{\hat{C}}/Z_0)$ where $c_1 \in \hat{C}$. The set $\{\hat{C} \in \mathcal{H} \text{ s.t. } c_1 \in \hat{C}\}$ is a power set of the remaining $n - 1$ individual constraints, since each of these can be either present or absent in $\hat{C}$. Of the constraint sets in this power set, $\binom{n-1}{|\hat{C}|-1}$ many will contain $|\hat{C}|$ individual constraints and therefore have a prior probability of $K(|\hat{C}|)$. Therefore, we estimate the sum of every term

in S corresponding to a constraint set that contains c_1 as

$$(Z_{c_1}/Z_0)^{-1} \sum_{k=0}^{n-1} \binom{n-1}{k} K(k+1) \quad (3.8)$$

Of the remaining 2^{n-1} terms in S that do not contain c_1 , (Z_{c_2}/Z_0) is now the best partition constant ratio estimator for those which contain c_2 . Therefore, we can repeat the process described above for $c_2, \dots, c_n$ to obtain the final estimate for S . The complete estimator can be written as

$$\begin{aligned} \hat{S} &= K(0) + \sum_{n'=1}^n (Z_{c_{n'}}/Z_0)^{-1} \sum_{k=0}^{n-n'} \binom{n-n'}{k} K(k+1) \\ &= K(0) + \sum_{n'=1}^n \hat{S}^{(n')} \end{aligned} \quad (3.9)$$

There are $\sum_{n'=1}^n n - n' + 1 = \sum_{i=1}^n i = n(n+1)/2$ terms in the estimator, resulting in a final complexity on the order of $O(n^2)$. Since $(Z_{c_{n'}}/Z_0)$ is an upper bound for the ratios it replaces, $\hat{S}$ is a lower bound for S .

Posterior Probability of a Single Constraint

In the simplest case, we wish to estimate the posterior probability $P(\mathcal{C} = c_1|\xi)$ where c_1 is the most likely individual constraint (following the convention in the previous section). This is similar to the setup in chapter 2, where we return the single likeliest constraint immediately following a demonstration by leveraging pre-computation of the (Z_{c_i}/Z_0) terms. The numerator of the posterior probability is exactly equal to $(Z_{c_1}/Z_0)^{-1}K(1)$, so using $\hat{S}$ from equation (3.9), our final estimate of $\hat{P}(\mathcal{C} = c_1|\xi) = \frac{1}{\hat{S}}(Z_{c_1}/Z_0)^{-1}K(1)$ is an upper bound on the true posterior probability.

Another useful quantity is the probability that the true constraint set contains c_1 . In this case, we wish to sum the posterior probability estimates for every constraint set that contains c_1 , resulting in the expression

$$\hat{P}(c_1 \in \mathcal{C}|\xi) = \frac{1}{\hat{S}} (Z_{c_1}/Z_0)^{-1} \sum_{k=1}^{n-1} \binom{n-1}{k} K(k) \quad (3.10)$$

where the numerator is exactly equal to the quantity calculated in equation (3.8), or equivalently the term $\hat{S}^{(1)}$ in equation (3.9). This can easily be obtained simply by saving the appropriate intermediate value while calculating $\hat{S}$, so there is no additional overhead. Unfortunately, since the numerator isn't exact in this case, $\hat{P}(c_1 \in \mathcal{C}|\xi)$ doesn't provide a guaranteed bound on the true value, but it may still be a useful estimate.

What if we're interested in a specific constraint c_i other than c_1 ? $\hat{P}(\mathcal{C} = c_i|\xi) = \frac{1}{\hat{S}}(Z_{c_1}/Z_0)^{-1}K(1)$ can be calculated in the same way as for c_1 , described above. To calculate $\hat{P}(c_i \in \mathcal{C}|\xi)$ requires a little more care, because the numerator sum

$$S_i = \sum_{\hat{\mathcal{C}} \text{ s.t. } c_i \in \hat{\mathcal{C}}} (Z_{\hat{\mathcal{C}}}/Z_0)^{-1}P(\mathcal{C} = \hat{\mathcal{C}}) \quad (3.11)$$

isn't estimated as part of equation (3.9). To handle this case, we note that of the 2^{n-1} terms containing c_1 , half of these will contain c_i and the other half will not. Furthermore, since we use (Z_{c_1}/Z_0) to estimate the partition constant ratio of all of these terms, each half has the identical estimated sum

$$\begin{aligned} & (Z_{c_1}/Z_0)^{-1} \sum_{k=0}^{n-2} \binom{n-2}{k} K(k+1) \\ &= \frac{1}{2} (Z_{c_1}/Z_0)^{-1} \sum_{k=0}^{n-1} \binom{n-1}{k} K(k+1) \\ &= \frac{1}{2} \hat{S}^{(1)} \end{aligned} \quad (3.12)$$

This holds for subsequent terms $c_2, \dots, c_{i-1}$ as well. Therefore, to estimate S_i in equation (3.11) while calculating $\hat{S}$, we can use the following:

$$\hat{S}_i = \hat{S}^{(i)} + \frac{1}{2} \sum_{n'=1}^{i-1} \hat{S}^{(n')} \quad (3.13)$$

where we stop once we reach $\hat{S}^{(i)}$ because after this term we have taken into account every constraint set containing c_i . Note that if we use this expression with c_1 it reduces to $\hat{S}^{(1)}$, as expected. Putting everything together, we have that

$$\hat{P}(c_i \in \mathcal{C}|\xi) = \frac{\hat{S}^{(i)} + \frac{1}{2} \sum_{n'=1}^{i-1} \hat{S}^{(n')}}{K(0) + \sum_{n'=1}^n \hat{S}^{(n')}} \quad (3.14)$$

which generalizes equation (3.10) to work for any single constraint c_i . In the worst case computing the numerator $\hat{S}_i$ only requires summing an additional n terms that were already calculated to obtain $\hat{S}$, so this procedure retains the same $O(n^2)$ complexity as before.

3.4 Posteriors for Sets of Multiple Constraints

Using the results from the previous section, after obtaining the (Z_{c_i}/Z_0) terms by running constraint inference, it is possible to both identify the most likely single constraint and estimate the probability that it was affecting the demonstrator. However, ideally we would

like to be able to search over the entire hypothesis space of constraint sets in an efficient manner. This is especially useful when the individual constraints represent a grid of the state space and we want to find constraint sets composed of multiple cells in the grid (for example, inferring the shape of a coffee spill on the ground that pedestrians are avoiding [16]). While we leave a thorough exploration of this goal to future work, here we present a preliminary greedy algorithm for selecting constraints to add to a running constraint set, and return its estimated posterior probability.

Greedy Algorithm for Multi-Constraint Sets

After running constraint inference, we can greedily add the most likely constraint c_1 to the overall estimate $\hat{C}$. Then, we can run constraint inference again, incorporating c_1 into the dynamics model. This allows us to calculate $(Z_{c_{1,i}}/Z_{c_1})$ for $i = 2, \dots, n$, where we define $c_{1,i} = c_1 \cup c_i$. If we order these so that $(Z_{c_{1,2}}/Z_{c_1}) \leq \dots \leq (Z_{c_{1,n}}/Z_{c_1})$, then $\hat{C} = c_{1,2}$ is the best constraint set of size 2 containing c_1 . This process can be repeated as many times as desired to obtain a greedy estimate of the most likely constraint set containing a particular number of constraints.

The new information obtained in these additional “rounds” of constraint inference can be used to improve our estimate of the posterior probability by extending our previous results. To see how, consider the case where we have selected c_1 as the most likely single constraint, and after adding it to the dynamics model, we have obtained $(Z_{c_{1,i}}/Z_{c_1})$ for $i = 2, \dots, n$. For the constraint sets that contain both c_1 and c_2 , since $(Z_{c_{1,2}}/Z_{c_0}) \leq (Z_{c_1}/Z_{c_0})$, we can refine our previous estimates of the partition constant ratios for these sets:

$$(Z_{c_1}/Z_0)^{-1} (Z_{c_{1,2}}/Z_{c_1})^{-1} \sum_{k=0}^{n-2} \binom{n-2}{k} K(k+2) \quad (3.15)$$

Putting these terms together for $c_3, \dots, c_n$, we have the following refined estimate for $\hat{S}^{(1)}$:

$$\begin{aligned} \hat{S}^{(1)'} &= \left(\frac{Z_{c_1}}{Z_0}\right)^{-1} \left[K(1) + \sum_{n'=2}^n \left(\frac{Z_{c_{1,n'}}}{Z_{c_1}}\right)^{-1} \sum_{k=0}^{n-n'} \binom{n-n'}{k} K(k+2) \right] \\ &= \left(\frac{Z_{c_1}}{Z_0}\right)^{-1} K(1) + \sum_{n'=2}^n \hat{S}^{(1 \cup n')} \end{aligned} \quad (3.16)$$

$\hat{S}^{(1)'}$ is a new, tighter upper bound for the sum of the terms corresponding to constraint sets containing c_1 . It can be used to refine $\hat{S}$ by simply replacing the $\hat{S}^{(1)}$ term in equation (3.9). It also further improves the estimate that $c_1 \in \mathcal{C}$ by replacing the numerator of equation (3.11) with $\hat{S}^{(1)'}$. After the second round of constraint inference, in addition to returning the identified constraint set $\hat{C} = c_{1,2}$, we can therefore estimate the probability that $\mathcal{C} = c_1$, $\mathcal{C} = c_{1,2}$, $c_1 \in \mathcal{C}$, $c_{1,2} \in \mathcal{C}$, etc.

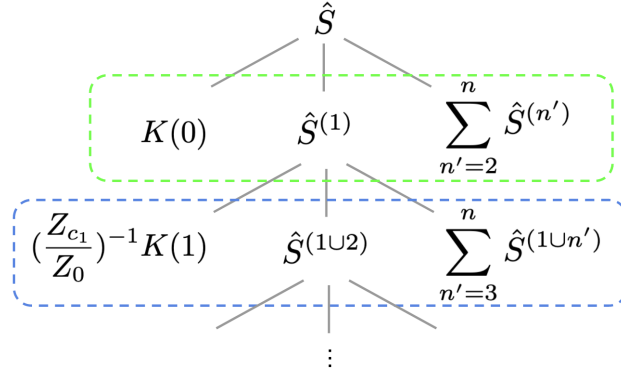


Figure 3.1: As we greedily add the best constraint to $\hat{C}$, we can refine the posterior probability estimate of $\hat{C}$ by updating the denominator $\hat{S}$. This figure shows the first two rounds of this process. We assume that c_1 is the likeliest single constraint, and c_2 is the next likeliest constraint after assuming that c_1 is indeed a true constraint on the demonstrator. In the first round, we sum the three terms in the upper green box to obtain the first estimate for $\hat{S}$ (compare with equation (3.9)). In the second round, we further refine this by replacing $\hat{S}^{(1)}$ with the sum of the terms in the lower blue box (compare with equation (3.16)). This can be repeated for as many rounds as desired.

Additional constraints can be iteratively added to the greedy constraint set $\hat{C}$ following a similar procedure. Although explicitly writing the expressions for the updates to $\hat{S}$ for subsequent constraints quickly becomes cumbersome, these can be extrapolated from equation (3.16). The process of updating $\hat{S}$ for the first two rounds of adding a constraint to $\hat{C}$ is illustrated in Fig. 3.1.

The posterior probability estimates also provide a principled condition for when to stop adding additional constraints to $\hat{C}$. If the posterior probability we estimate for $\mathcal{C} = \hat{C}$ decreases after adding a new constraint, then running additional rounds of constraint inference may improve our estimate of $\hat{S}$ but is unlikely to find a better candidate for $\hat{C}$. The posterior probability captures both how well the identified constraint set explains the demonstrator's behavior (the $Z_{\hat{C}}/Z_0$ term) as well as the lower prior probability we place on a greater number of constraints being active simultaneously. This contrasts with prior work that used only the quality of the explanation as the stopping criterion and neglected the prior [16].

3.5 Conclusion

In this chapter, we proposed an efficient way to estimate the probability that a hypothesized constraint was affecting a demonstrator. This is a useful extension to previous maximum likelihood approaches to constraint inference, which can find the most likely constraints but do not give any insights into how confident an agent should be in its findings. However, this

work makes some significant assumptions, including only handling a single demonstration and a limited set of possible priors over constraints. Future work might aim to relax some of these assumptions in this chapter as well as working towards an iterative algorithm for reasoning about sets of multiple constraints. Rather than taking a greedy approach, future work should focus on explicitly negotiating the trade-off between refining the estimate of the posterior denominator $\hat{S}$ with searching for the most likely constraint set.

Chapter 4

Linking vision and motion for self-supervised object-centric perception

This chapter is adapted from the previously published paper “Linking vision and motion for self-supervised object-centric perception,” presented at the 2023 Vision-centric Autonomous Driving workshop at CVPR, and included with permission from my coauthors Zak Murez, Vijay Badrinarayanan, Jamie Shotton, Alex Kendall, Claire Tomlin, and Christopher P. Burgess.

4.1 Introduction

People and robots both operate in environments where objects—things that tend to behave as single coherent entities—are a natural organizing principle for perception. Objects play a central role in human vision. We group features into objects [31], describe our surroundings in terms of them, and seek out semantic labels for ones we’re unfamiliar with [32]. When using visual representations for downstream tasks such as robotics, object-centric models are therefore desirable if only because they are easier for people to interpret than end-to-end models, which is important for verifying safety and engendering trust in vision-enabled systems. More than this, however, object-centric representations enable a diverse and powerful set of tools for reasoning about the world, such as models for physics understanding [33], multi-agent prediction and planning [34], and causal reasoning [35]. Representations that support these kinds of models are likely to be crucial for autonomous driving, where choosing the best action requires reasoning about many interacting agents as well as physical factors.

Traditionally, object-centric representations have been obtained by training supervised object detection models and extracting object properties such as position and velocity from their predictions. This approach has two major drawbacks. First, it requires an annotated dataset that matches the desired detections, which is expensive to obtain at scale and may

introduce unwanted bias. To make the system work with new kinds of objects or in visually novel situations, new annotations have to be collected. Not having enough of the right kind of labeled data can hamper generalization, which promises to be one of the key advantages of object-centric models [36].

Second, creating object representations from the predictions of supervised vision models introduces a decoupling between the perception and downstream decision-making components of the system. For example, should a person riding a bike be treated as one object or two? What about two people on a tandem bike? Is it important that they’re waving at a pedestrian on the sidewalk? What if they’re making a hand signal for turning? The right answer to these questions depends on how the information will be used to make decisions. Ideally, success at using perception for action should feed back into improving perception, leveraging the power of end-to-end learning to find better object-centric features than what can be designed by hand.

These considerations motivate *self-supervised* object-centric perception, where the model learns to encode images into a latent space that divides up relevant information about the scene into a number of discrete “slots.” The contents of these slots can then be decoded into a self-supervised target such as reconstructing the original RGB input, as well as for downstream tasks. Various methods have been reported for encouraging models to encode information about different objects in separate slots, such as by using an encoder that has slots compete for attention over pixels [37] or incentivizing slot disentanglement with a variational autoencoding loss [38]. However, these methods have struggled to obtain good results with complex real-world data. Recently, the algorithm SAVi++ has shown segmentation on the Waymo Open dataset of real-world driving videos [39]. However, these results required additional depth supervision, as well as initializing the slots with bounding boxes around the desired objects to be tracked for best performance.

In this work we present a model which uses only RGB video and information about camera movement to obtain promising self-supervised segmentation results on real-world driving videos. Camera movement can be readily obtained in an autonomous vehicle setting (for example from simultaneous localization and mapping (SLAM) or wheel odometry), requiring no specialized sensors such as lidar. This makes it a particularly economical form of auxiliary input. Our model learns to segment many vehicles and pedestrians from the background and additionally shows consistent tracking of objects over time and across multiple cameras.

4.2 Related work

Object-centric representation learning in the self-supervised setting has received increased attention in recent years. A survey of learning and using these representations can be found in [36]. Techniques that operate on a single image work well with very simple synthetic scenes [37], [40], but for more complex scenes the correct segmentation is ill-defined. Sequences of multiple images over time and/or space provide richer information about 3D shape and allow decomposition of more complex scenes [38], [39], [41]. However, most approaches have

still been demonstrated only on synthetic datasets, with segmentation of real-world images remaining an open challenge.

The recent SAVi++ model [42] has shown promising instance segmentation and tracking of vehicles in the real-world Waymo Open dataset. However, this model requires ground-truth depth images in addition to RGB image input, and strong performance additionally requires that bounding boxes for the objects to be tracked to be provided at test-time.

Another recent model, STEVE, does not use additional supervision and presented results on a dataset of real videos of aquariums and traffic from a fixed overhead camera [43]. However, the fixed background in these videos simplifies the task. Additionally, the freeway environment in the traffic videos has less variety in background textures and shapes than driving through different environments including busy city streets.

The Object Scene Representation Transformer (OSRT) has a similar architecture to our proposed model, but uses novel viewpoint synthesis, rather than autoencoding, as the training task [41]. Although it was originally only demonstrated on synthetic datasets, recent work has used OSRT as the vision backbone in a real-world robotic manipulation task, albeit in a simple environment consisting of a blank table with a small number of solid-colored shapes [44].

The closest work to ours is SIMONe [38], which uses variational autoencoding and independent per-slot reconstruction to learn an object-centric decomposition of image sequences. To adapt SIMONe for complex real-world driving videos, we:

1. Provide the ground-truth transform matrix for each input image instead of learning self-supervised “frame latents,”
2. Change the reconstruction decoder from a unimodal normal distribution to a normal mixture distribution, as described in section 4.3.

Additionally, we experiment with using multiple camera viewpoints and adding prediction of the future ego-vehicle pose as an auxiliary training task. This task can be viewed as behavioral cloning, which is a common technique in end-to-end autonomous driving [45]. Previous work has demonstrated the potential of using an object-centric representation for predictions involving multiple dynamic objects [33].

4.3 Approach

For our model architecture, we build on the view-supervised variant of SIMONe [38]. The goal of the model is to decompose the scene into a set of K *object slots* that encode information about each object in the scene. These slots are obtained as follows: first, the input X (a sequence of F images, optionally from multiple camera viewpoints), is processed in parallel by a standard convolutional neural network (CNN), resulting in a set of feature patches. The patches are concatenated with positional embeddings that encode the position of each patch within its source image, as well as the time and viewpoint transform matrix associated

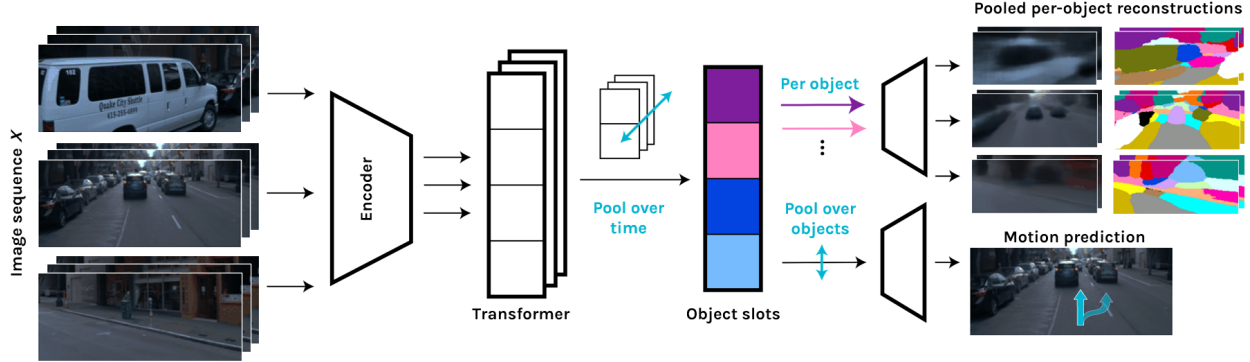


Figure 4.1: An overview of the proposed model architecture. From left to right, a sequence of images (optionally with multiple camera viewpoints such as front-left, front-forward, and front-right) is encoded into a set of feature patches with a CNN and decomposed into a set of K slots via a transformer. In the top right, the slots are independently decoded into predicted RGB value distributions for each pixel. In the bottom right, the slots are pooled and used for auxiliary tasks such as predicting the future trajectory of the ego vehicle. The training losses described in section 4.3 encourage the model to store information about different objects in each slot.

with the source image. They then serve as input tokens to a decoder-only transformer. The output tokens are averaged across the image dimension. In the original SIMONE model this dimension corresponds to images from a single camera captured over several timepoints, but in this work we use three different camera viewpoints, and the average operation is over both time and viewpoint. Finally, each token is decoded via an MLP into m -dimensional vectors z_k and σ_k that contain, respectively, the predicted mean and variance of the latent information in a single slot. If the number of input tokens is different from the desired number of object slots, an optional spatial pooling operation across the token features is added halfway through the transformer layers.

Training and losses

In order to encourage the model to store information about different objects in different slots, during training we apply three complementary losses. The first is the KL-divergence between each slot latent vector and a unit normal distribution, summed over all slots:

$$\mathcal{L}_{KL}(X) = \sum_k D_{KL}(q_k(X) || p(\cdot)) \quad (4.1)$$

where X are the input frames, $q_k(X) = \mathcal{N}(z_k(X), \sigma_k(X)\mathbb{I})$ is an m -dimensional normal distribution with mean and variance predicted by the model for slot k , and the prior distribution $p(\cdot)$ is a unit spherical normal distribution. Intuitively, this loss encourages the

model to avoid representing the same object using multiple slots, as doing so would incur a larger penalty than using a single slot to represent the object and letting the others remain closer to the unit normal distribution. This loss also encourages disentanglement between the features represented in each dimension of the latent vector.

The second loss is based on the model’s ability to perform an object-wise reconstruction task. First, object latent vectors $o_k \sim q_k(X)$ are sampled from the latent distribution for each slot k . Then, each o_k is independently decoded into pixel-wise predictions using a spatial broadcast decoder [46]. To keep the computational and memory requirements of the model manageable, only a random sample of N pixels is decoded from each input sequence during training. The per-slot predictions for each pixel parameterize a distribution over possible RGB values, $p(x^{(n)}|o_k)$, and a logit $\hat{\alpha}_k$ that (after normalization across slots) represents the likelihood that slot k represents the pixel. To obtain the final prediction for each pixel, we take a weighted mixture of each slot’s predictions:

$$p(x^{(n)}|o_1, \dots, o_K) = \frac{1}{K} \sum_k \bar{\alpha}_k [p(x^{(n)}|o_k)] \quad (4.2)$$

$$\bar{\alpha}_k = \frac{\exp(\hat{\alpha}_k)}{\sum_{k'} \exp(\hat{\alpha}_{k'})} \quad (4.3)$$

where the per-slot distributions for pixel $x^{(n)}$ are weighted by the softmaxed $\hat{\alpha}_k$ values. The distribution of $p(x^{(n)}|o_k)$ is discussed in more detail later in this section.

The final reconstruction loss is the log probability of the ground truth RGB values for each pixel under the mixture distribution:

$$\mathcal{L}_{recon}(X) = \frac{-1}{N} \sum_n \log p(x^{(n)}|o_1, \dots, o_K) \quad (4.4)$$

$$o_k \sim q_k(X) \quad (4.5)$$

Since the object latent vectors from each slot are decoded independently, the model is forced to use only information encoded in a single slot at a time when predicting the RGB values for each pixel. Therefore, intuitively this loss encourages the model to store all of the information it needs to predict the color of pixel n in a single slot.

Optionally, the learned slot representations can also be used for auxiliary tasks. In this work, inspired by the close coupling between objects that matter in the autonomous driving context and representations that are useful for predicting good driving actions, we experiment with predicting the future waypoints of the ego vehicle as the auxiliary task. After the image-wise pooling step, the slot tokens are passed through two transformer decoder layers, averaged, and decoded into a series of predicted offsets, $\hat{s}$, in the xy -plane of the ego reference frame via a single-layer MLP. We then apply the following task loss:

$$\mathcal{L}_{task}(s) = \sum_{t'} \|s_{t'} - \hat{s}_{t'}\|_1 \quad (4.6)$$

where the summation is over each future time point. We use 16 future waypoints in the ego reference frame at a frequency of 10 Hz, starting $t'_0 = 0.1s$ after the final image frame.

The final training loss is similar to the negative-ELBO loss in variational autoencoding [47] with the addition of the auxiliary task loss:

$$\mathcal{L}_{total}(X, s) = \mathcal{L}_{recon}(X) + \omega_{task}\mathcal{L}_{task}(s) + \beta\mathcal{L}_{KL}(X) \quad (4.7)$$

where the β and ω_{task} hyperparameters balance the different loss terms.

Additional Model outputs

In addition to parameterizing the weighted mixture pixel distribution, the $\bar{\alpha}_k$ weights serve as per-slot alpha masks that make it easy to see which pixels are best represented by each slot. Taking the argmax of this across slots results in a predicted segmentation of the scene. This segmentation can aid in model debugging and interpretation. For example, failing to track a particular vehicle with a mask indicates that the model is not disentangling that object’s features from other features in the scene, and therefore is not independently representing its dynamics.

The object slots or latent vectors may also be decoded into other outputs besides image reconstruction or waypoint prediction. Other potential downstream tasks could include video prediction [33], production system models [48], or action-conditioned world models [49]. Which auxiliary tasks improve performance synergistically in end-to-end robot learning is an exciting open question.

Slot decoding distribution

The original SIMONE model uses a unimodal normal distribution for the predicted pixel RGB values $p(x^{(n)}|o_k)$. (Note that throughout this section we refer to an RGB tuple as a single mode, but in reality the R, G, and B channels are treated independently.) We find that this distribution causes the model to over-rely on differences in color when decomposing the scene into object slots. This leads to failure cases such as separately segmenting the body and windshield of vehicles, and failing to pick out objects with similar colors to the background. We speculate that this is due to the fact that around the border of two differently-colored regions of a single object, the model may be uncertain about which color to assign a given pixel. To express this uncertainty with a unimodal color distribution, the model is forced to assign the different colored regions to different slots, and use the per-slot $\hat{\alpha}_k$ weights to give likelihoods to each color.

For our architecture, we replace the unimodal distribution in SIMONE with a multi-headed normal distribution to alleviate this issue. Qualitatively, we find that this leads to segmentations that better reflect the motion of objects. For each pixel and each slot k , the decoder outputs H modes with a predicted mean RGB tuple $\hat{\mu}_k^{(h)}$ and $\alpha_k^{(h)}$ logit which determines the weight of each mode. (Note that there is additionally a separate “global” $\hat{\alpha}_k$

logit which controls the k th slot’s contribution to the total mixture distribution, as shown in equation (4.2). The per-slot distribution is therefore

$$p(x^{(n)}|o_k) \sim \frac{1}{H} \sum_h \frac{\exp(\alpha_k^{(h)})}{\sum_{h'} \exp(\alpha_k^{(h')})} \mathcal{N}(\hat{\mu}_k^{(h)}, \sigma_x) \quad (4.8)$$

where the variance of the normal distribution, σ_x , is a hyperparameter. When $H = 1$, this simplifies into the decoding distribution in SIMONe:

$$p(x^{(n)}|o_k) = \mathcal{N}(\hat{\mu}_k, \sigma_x) \quad (4.9)$$

In our experiments we use $H = 3$ and $\sigma_x = 0.08$.

4.4 Results

We evaluate our approach on the Waymo Open Perception dataset [50]. We train our model on sequences of 8 frames at 5 Hz, with input images cropped to remove the sky and resized to a resolution of 96 x 224. The validation dataset includes approximately 2000 frames with semantic and instance segmentation labels that we leverage to evaluate our model’s ability to segment and track objects. Of these frames, we sample 208 sequences, which like the training data are 8 frames at 5 Hz, and use the front-left, front-forward, and front-right camera viewpoints for a total of 24 frames per sequence.

Qualitative results

We find that our model emergently learns to assign the same mask to the same parts of the scene across time and different camera viewpoints, such that the movement of the masks across time tends to track both dynamic vehicles and pedestrians and static background features in the scene. An example of the learned masks for all three camera viewpoints is shown in figure 4.2. Although the learned masks do not always provide a good shape match for the tracked objects, they do tend to follow them consistently across multiple time-steps and camera viewpoints. Examples of tracking vehicles and background features are shown in figures 4.3 and 4.4, respectively.

Because we provide no supervision for which parts of the scene the object slots should attend to, the masks provide an exhaustive partition of the scene. We note that many elements of the background in this complex real-world dataset don’t lend themselves to clear object-centric interpretations, so the correct partition is ambiguous.

We additionally find that the waypoint prediction auxiliary task learns to predict qualitatively reasonable trajectories, as shown in figure 4.5. This suggests that the object-centric representation is compatible with learning to predict appropriate driving actions.

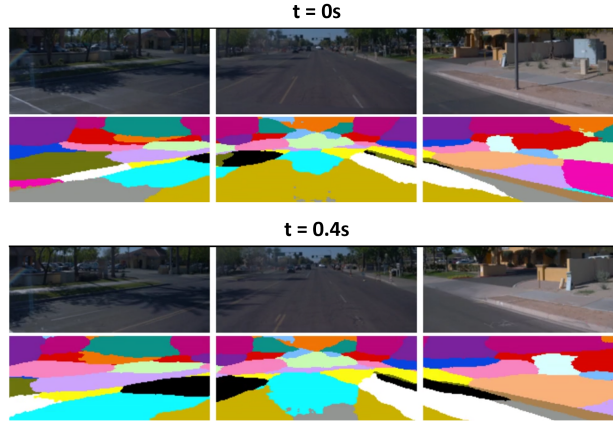


Figure 4.2: Our model partitions the scene between objects in a way that is consistent across time and multiple camera viewpoints, despite being given no explicit fusion information besides the camera transformation matrix for each input image. For example, in this scene note the tree shadows on the pavement in the left camera image and the orange wall on the right. The segmentation masks in this figure are derived from the slot with the largest predicted weight for each pixel.



Figure 4.3: Our model is able to track many vehicles and pedestrians in held-out validation runs from the real-world Waymo Open dataset. We use Hungarian matching to determine best-fit predicted masks for each ground-truth object as described in section 4.4. In this figure, the alpha-value of the colored mask pixels is determined by the weight of the mask at each pixel.



Figure 4.4: Due to its partitioning of the scene, our model tracks background features as well as dynamic objects. Here, three of the masks from a turning sequence are shown.



Figure 4.5: An example of the ground-truth future waypoints (green) and waypoints predicted by our model (orange). The predictions are for 1.6s into the future after the last frame in the input sequence.

Quantitative results

We compare our model’s performance with the prior state of the art models [39] in table 4.1. The SAVi++ models receive additional depth supervision, and the conditional version of SAVi++ is additionally initialized with bounding boxes around the ground-truth objects in the first frame of the sequence. The SAVi (RGB) model, like ours, does not receive depth or bounding box supervision.

We report sequence Adjusted Rand Index for Foreground objects (ARI-F) and centroid tracking. ARI-F is calculated for all instance labels in the sequence, which for the Waymo Open dataset include vehicles and pedestrians, but ignores background objects such as buildings or signs that do not typically exhibit independent motion. To measure tracking performance, we use the Center-of-Mass (CoM) distance metric proposed by [39]. The centroids of ground-truth instances are calculated via the geometric mean of their masks in each frame, and for the predicted slots via the weighted geometric mean of each slot’s α mask. Ground-truth instances that make up more than 0.05% of the total per-camera pixels are matched with the closest predicted centroid via Hungarian matching. We report the distance normalized by the length of the frame diagonal and averaged across frames and instances.

While our model does not achieve as good center-of-mass tracking as SAVi++, it approaches its performance without depth supervision. Our model also significantly outperforms vanilla SAVi while receiving only camera pose as an additional input.

Furthermore, our results suggest that learning to predict the future pose of the ego vehicle is a synergistic task that does not harm the object-centric scene decomposition as measured by center-of-mass tracking performance. When training data is collected from expert drivers, future waypoint prediction (also known as behavioral cloning) is a leading method for picking driving actions in an autonomous driving context [45]. Our results suggest that the object-

centric representation learned by our model may be usable directly as a vision backbone for AVs.

Table 4.1: Object segmentation performance metrics on the Waymo Open dataset. SAVi, SAVi++, and depth-augmented SIMONe metrics are those reported in [42] and may differ slightly in data-loading methodology. We report the standard error across 3 random seeds.

Model	Privileged Information	ARI-F $\uparrow$	CoM (%) $\downarrow$
SAVi (RGB)	-	-	21.5 ± 1.8
SAVi++	Bounding boxes and depth	-	4.4 ± 0.2
SAVi++ (unconditioned)	Depth	-	6.9 ± 0.5
SIMONe	Depth	-	7.4 ± 0.2
Ours (no mixture)	Camera pose	0.193 ± 0.00	10.0 ± 0.3
Ours (no viewpoint)	None	0.237 ± 0.00	9.8 ± 0.3
Ours (no waypoint pred.)	Camera pose	0.257 ± 0.02	9.9 ± 0.7
Ours	Camera pose	0.253 ± 0.01	9.6 ± 0.4

Ablations

In table 4.1 we also report metrics from ablations of our model. In the no mixture condition, the number of modes for the pixel reconstruction distribution is $H=1$. In the no viewpoint condition, the transformer patch positional embeddings and decoder pixel coordinate queries refer only to the index of the camera (left, front, or right) and position in the sequence rather than containing the camera transformation matrix. Note that this is not equivalent to the original SIMONe model because we do not infer separate “frame latents” to represent the camera viewpoint. In the no waypoint prediction condition, we do not add the waypoint prediction auxiliary task. Hyperparameters are kept constant across models, except for the waypoint prediction condition which uses a slightly higher β ($5e-7$ vs. $4.5e-7$) to ensure that the KL-divergence of the object latent distribution is similar to the other models.

We find that both removing the viewpoint supervision and the additional mixture modes in the decoding distribution significantly harm the ARI-F of the model, and there is additionally a trend towards slightly higher CoM in these conditions. The waypoint prediction auxiliary task has little effect on the segmentation metrics, suggesting that the object-centric representation can be used in end-to-end learning with useful downstream tasks.

Limitations

While the fusion and tracking properties of our model suggest that it is learning to capture the movement of objects in the scene, the quality of the masks still lags behind what is possible with supervised instance segmentation [51], [52] and what has been demonstrated

with self-supervised learning on synthetic datasets [38]. Two failure modes exhibited by our model are “over-segmentation,” where more than one mask is used to represent the same object, and mask re-use, where the same slot represents multiple independent parts of the scene (as can be seen in figure 4.2). We found that simply increasing the number of slots available to the model does not resolve mask re-use and may lead to increased tessellation-type failures, an issue that has been reported in other object-centric models [41], [53]. While in the self-supervised setting the correct partitioning of the scene is ambiguous, both of these limitations could be concerning for the use of object-centric representations in downstream driving tasks such as planning and scene understanding, and therefore merit further investigation.

4.5 Conclusion

Self-supervised object-centric representation learning approaches have recently shown strong performance on synthetic datasets with clearly defined objects, but continue to struggle with complex real-world data with complicated textures and ambiguous objects. In this chapter we have presented results suggesting that by using camera pose as additional input, it is possible to obtain reasonable dynamic, object-centric representations in the context of RGB driving video. Since pose estimation, unlike 3D depth sensors, is a ubiquitous feature of autonomous vehicles, we believe that our approach is a promising avenue towards scalable and practical object-centric representation learning in the context of autonomous driving. Furthermore, our results suggest that predicting the future pose of the ego vehicle is a synergistic task that doesn’t hinder the quality of the learned representations. This is particularly exciting for end-to-end driving models because it opens up the possibility of driving performance and representation learning improving together in a virtuous cycle while retaining key advantages of object-centric representations, such as interpretability.

We suspect further improvements to the object segmentation quality are still possible, for example via careful scaling up of model size and a data augmentation strategy (both of which were important for the performance of SAVi++ [39]). We also note that the size of the Waymo Open perception dataset, containing 480,000 frames across the three forward-facing cameras, may not be sufficiently large relative to its complexity for ideal representation learning. The recent Object Scene Representation Transformer model, in contrast, was trained on 10M frames of a synthetic dataset [41].

Finally, we note that the KL-divergence loss in our model encourages learning disentangled object latent features [47]. Examining these features in more detail is an exciting direction for future work.

Part II

Interpreting robot representations

Chapter 5

Representations in a deep end-to-end driving model predict human brain activity in an active driving task

This chapter is adapted from material prepared with my co-authors Tianjiao Zhang, Christopher Strong, Jingqi Li, Jack Gallant, and Claire Tomlin, and included here with their permission.

5.1 Introduction

Fully autonomous vehicles now share the roads with human drivers in several major cities. In contrast with most previous commercially successful applications of robotics, such as robotized warehouses [54], drone medication delivery in rural areas [55], and infrastructure inspection [56], autonomous driving systems perform the same tasks as humans do in diverse and complex environments involving dynamic interactions with other agents. Ensuring that the algorithms used in autonomous driving are effective and safe to very low margins of error poses a significant challenge. Furthermore, most autonomous driving systems use deep neural networks (DNNs). DNNs are notoriously difficult to analyze and interpret [57], which makes verification of these desirable properties an additional problem. However, there are also exciting opportunities: for the first time, human and artificial intelligences (AIs) are performing the same complex sensorimotor and social tasks in a shared environment. This raises many interesting questions at the intersection of artificial intelligence and human neuroscience: to what extent do the human brain and DNNs rely on similar representations when integrating diverse sensory information to produce task-appropriate behavior? Do the human brain and DNN share any functional organization for solving complex, closed-loop tasks? Do they make similar kinds of inferences about each other when engaged in a shared task environment?

In this work, we provide some initial insights into these questions by comparing two sets

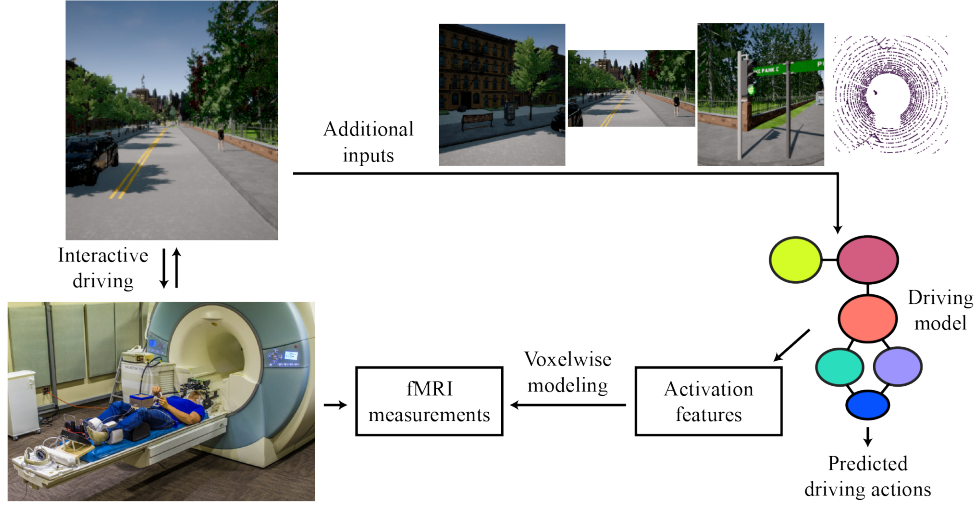


Figure 5.1: **Using deep neural network activations to model brain activity during a naturalistic simulated driving task.** Human subjects drove through a large virtual city in the CARLA simulator and performed a taxi-driver task while brain activity was recorded with functional magnetic resonance imaging (fMRI). The recorded experiment state was used to generate the image and LiDAR sensor inputs that were then passed to Learning from All Vehicles (LAV), a deep neural network driving model. Activations in the LAV model in response to these inputs were regressed onto brain activity to produce a predictive encoding model of brain activity during the driving task. This model can be used to investigate and quantify the alignment of the neural network driving model to the representations the human brain employs during driving.

of features: brain activity from human drivers recorded with functional magnetic resonance imaging (fMRI), and activations from the Learning from All Vehicles (LAV) driving DNN [45] when given similar inputs to those seen by the human drivers. To perform this comparison, we regress the LAV activations onto the brain activity to build a voxelwise encoding model [58], [59], then test this model’s predictive power on held-out brain data. We find that the LAV voxelwise encoding model explains a substantial amount of variance in human brain activity. Furthermore, features from the modules in the LAV DNN explain variance in distinct functional regions in the brain. The known functional properties of these regions align well with the corresponding modules. Our results are an exciting first step towards investigating the cognitive and representational basis for human and AI driving by leveraging each to better understand the other.

5.2 Related work

Human brain activity during driving

The cognitive processes underlying human driving behavior have been studied in many experiments and with a variety of brain imaging modalities, including fMRI, electroencephalography (EEG), functional near-infrared spectroscopy (fNIRS), and magnetoencephalography (MEG) [60]. Of these modalities, fMRI provides the highest spatial resolution and enables the localization of brain activity [61], and has been used to localize various aspects of driving to functional brain regions. Both [62] and [63] have used fMRI to examine brain activity from subjects performing simulated driving tasks. [64] pooled these and seven other fMRI studies of driving to find characteristic patterns of brain activity for different driving tasks, such as turning or planning to overtake, and related these patterns to qualitative, non-predictive models of human driving. Unfortunately, these studies only used contrast-based fMRI analysis methods to localize driving-related activity to certain brain regions, and did not investigate hypotheses about specific representations and computations in the brain during driving. Also, these studies provide no insights into AI driving algorithms. [65] followed a similar methodology to our work but used simple vehicle-following models instead of full driving models.

DNNs as encoding models of brain activity

Encoding models are a powerful method for the analysis of brain activity. In this approach, the time series of brain activity is modelled as a function of stimulus and task feature time series. Encoding models have been extensively used to study how the brain represents visual inputs. Features used to fit encoding models of brain activity in these studies range from simple transformations of the visual input, such as Gabor wavelets [66], to deep neural networks trained on vision tasks [67]–[69]. Because previous studies using DNN-based encoding models have focused on visual processing, they used data from tasks in which subjects observed stimuli passively. Here we aim to apply the encoding model approach to data from an active sensorimotor task in a naturalistic environment. While vision is an important part of this task, active driving also involves planning, motor control, and social interaction with other agents.

5.3 Voxelwise modeling with deep neural networks for driving

To compare deep neural network activations with brain activity, we first collected a dataset of driving behavior and brain activity from human subjects. We then used the recorded behavior from this dataset to generate a time series of appropriate inputs to the LAV driving network. Next, we gave these inputs to the LAV driving network, and extracted features

representing the activity of the LAV network. Finally, we built voxelwise models of brain activity by regressing the LAV driving network features against the brain activity.

Measuring brain activity with fMRI

We used functional magnetic resonance imaging (fMRI) to record brain activity from five subjects performing a taxi-driver task in a simulator. We used Unreal Engine 4 and the CARLA plugin to implement a driving simulator that contains a large 2×3 km virtual city populated by dynamic pedestrians and vehicles. Prior to recording, subjects learned the layout of the world. During recording, subjects controlled a virtual car with an MR-compatible steering wheel and pedals (illustrated in Figure 5.1), performing the taxi-driver task, in which they were cued to navigate to particular locations. Blood-oxygenation-level dependent (BOLD) [70] activity from the brain was recorded by the MRI scanner as subjects performed the task. Data were collected in 11-minute runs (110 minutes for subject 3 and 180 minutes for all other subjects) at a temporal resolution of 2.0045 seconds and a voxel (3D pixel) size of $2.24 \times 2.24 \times 3.5$ mm³ (matrix size = 100×100 voxels, 30 axial slices). Please refer to Appendix C in section 8.5 for more details on data collection and preprocessing. The experimental procedures were approved by the Institutional Review Board at the University of California, Berkeley, and written informed consent was obtained from all participants.

Measuring deep neural network activity

DNN driving model

For this work, we chose to examine a pre-trained driving deep neural network developed by Chen and Krähenbühl [45], Learning from All Vehicles (LAV). LAV was developed for the sensors track of the CARLA Leaderboard challenge (version 1.0), where teams compete to submit algorithms that can complete a sequence of driving scenarios while obeying traffic rules and avoiding collisions. LAV achieved first place in the 2021 edition of the challenge, with a route completion rate of 94% [45]. It uses an imitation learning-based training objective to mimic “expert” driving behavior provided by the built-in CARLA AI driver. The CARLA AI driver uses ground-truth simulator state to generate appropriate driving actions, and LAV attempts to imitate its behavior with only RGB images and LiDAR as inputs. Although the simulator used in our human subject experiments has some slight differences from the simulator that LAV was trained on (CARLA version 0.9.10), LAV produced reasonable inferences with our data (example outputs shown in Figure 5.3a).

The LAV DNN has an end-to-end architecture that is split into different distinct modules that each perform a specific sub-task for driving (encouraged via module-specific objective functions during training). We used this architecture to conceptually group the measured DNN activations by module. The modules are as follows:

1. **Semantic segmentation:** This module predicts the semantic class of visual features in RGB images from the vehicle’s cameras. It receives three images as input, one from

a camera facing forwards and two at yaw offsets of ± 60 degrees. It categorizes each pixel as one of 5 semantic classes: roads, road markings, vehicles, pedestrians, and other. The architecture is an ERFNet [71], a convolutional neural network specialized for segmentation.

2. **Bird’s-eye-view (BEV) perception:** This module predicts local scene features, including the layout of the road and the locations of other vehicles, in a BEV (top-down) reference frame fixed on the ego vehicle. It receives 3D LiDAR points as input, where points that lie within the field of view of the RGB cameras are “painted” with their semantic class label from the semantic segmentation module. The architecture is split into two learned components: a point pillar net and a perception backbone. The point pillar splits painted LiDAR points into a 320×320 grid of the environment surrounding the ego vehicle in the BEV frame, and returns a feature vector for each bin of the grid. (Since the point pillar network operates on an unordered and variable-length set of LiDAR points, we only include the final grid-based feature vector in the extracted features.) The perception backbone then passes these grid features through convolutional layers to produce a feature map, which is decoded into a semantic segmentation of the local road layout as well as the position and orientation of other vehicles.
3. **Planning:** This module generates a trajectory plan for the ego vehicle. It takes as input the BEV feature map from the previous module, cropped and centered on the ego vehicle. This feature map is first passed through a ResNet-18 [72], which performs driving plan-agnostic processing. Then, for each of 6 possible high-level commands (turn left / right, go straight through an intersection, stay in the current lane, and lane change left / right), a gated recurrent unit (GRU) network predicts the future position in 0.25 s intervals (up to a planning horizon of 2.5 s). Finally, a second GRU network is used to refine the intermediate set of trajectories into a final plan conditioned on a target waypoint indicating the desired route. Since our human subjects do not follow a fixed route, we define the waypoints for the planning module as the driver’s position 40 m ahead along their future trajectory.
4. **Trajectory prediction:** This module generates predicted trajectories for each nearby detected vehicle. The inputs and architecture are the same as in the planning module, except that the BEV feature map is cropped and centered around the detected vehicle, and there is no target waypoint (which is unknown for the other vehicles) and therefore no final trajectory-refinement GRU. Since the true high-level command is also unknown for the detected vehicles, this module also uses a single linear layer to predict the probability of each possible command, resulting in a probability distribution over the trajectories.
5. **Hazard detection:** This module predicts whether there is a hazard present that necessitates braking. It receives four RGB images as input (the same inputs as the semantic segmentation module, plus an additional front-facing image at a higher zoom

level). The architecture is a ResNet-18. During training, the final-layer feature vector is additionally passed into a separate network which outputs a semantic segmentation, but this network is not part of inference and not included in our extracted features.

6. **Control:** This module takes the outputs of the planning, prediction, and hazard detection modules, and determines the final control action that should be applied. Specifically, if the hazard detection model predicts that braking needs to occur or if the planned trajectory for the ego vehicle intersects the predicted trajectory of another vehicle, the control module will brake to stop the car. Otherwise, it computes acceleration and steering commands via PID control to follow the planned trajectory. Unlike the other modules, the controller is not a DNN and has no learned parameters.

Measuring DNN activity

To measure DNN activity, we used recorded simulator state data from the human subjects' driving sessions to generate RGB images and 3D LiDAR scans at 15 frames per second (FPS). There are four cameras, two pointing forwards (one with a smaller field of view and longer focal length) and two pointing at an offset angle of 60 degrees to the left and right of the forward cameras. Because LAV does not have any form of memory other than selecting the current target waypoint from a pre-defined route, we used the recorded human data as the input at every frame, and ignored LAV's final control outputs without disrupting the operation of the LAV network. Doing so enabled us to measure the activation of the LAV network in response to the same simulator state encountered by the subjects and align the activity of the LAV DNN and the human brain activity for our models.

Across all modules, the LAV DNN contains on the order of 10^8 measurable values that vary with the input data. This large number makes it computationally intractable to store the activations from all units and regress them against brain activity. To overcome this computational challenge and reduce the dimensionality of the LAV network activations, for each module except the controller, we used a sparse random projection (SRP) of k components per module [73], using the implementation in Scikit-learn to generate the random projection matrix [74]. SRPs approximately preserve the distance between points with high probability and dramatically speed up linear regression [75]. We provide a proof in Appendix A in section 8.3 that solving ridge regression with features projected with an SRP results in an approximate solution to the original ridge regression problem with high probability, where choosing k large enough results in low approximation error and high success probability. We used the same k for each module because it controls for the number of regression features when comparing encoding model performances across modules. In this work, we found $k = 20,000$ to produce stable regression results across multiple random projections.

For the controller module, we used the planned trajectory and brake probability for a total of 121 features, and therefore did not apply any dimensionality reduction. The planned trajectory is in a BEV reference frame.

Voxelwise modeling

We applied the voxelwise modeling (VM; [58], [76], [77]) framework to the features extracted from LAV to determine whether a DNN and the human brain rely on similar representations while driving, and whether the DNN and the human brain share any functional organization for active driving. In VM, the time series of activity in each brain voxel is modeled as a linear combination of the time series of all features. Models are evaluated by predicting brain activity on a held-out dataset. High prediction performance by a set of features, or feature space, suggests that the brain represents information as parameterized by that feature space [58]. VM has enabled many new insights about brain activity, such as in experiments where subjects listen to narrative stories [77], [78] or watch movies [76], [79].

In this work, we treat each module of the LAV network as a separate feature space. To capture the contributions of each module to the final predictive performance of the encoding model, we used banded ridge regression [59], [80], which imposes a different ridge regularization parameter on each feature space. After regression, the overall model performance was quantified by computing the R^2 (explained variance) between predicted and actual activity in each voxel on a held-out test set, and individual model performances for each of the feature spaces were determined by partitioning the R^2 between them [80], [81]. Intuitively, each of the feature spaces can be considered as a hypothesis of how information may be represented in (some part of) the brain; the split R^2 for each feature space provides a test for its corresponding hypothesis.

The BOLD signal captured by fMRI does not respond to stimuli instantaneously but rather over a period of several seconds (the hemodynamic response function). We modeled this by implementing a finite impulse response (FIR) filter with four delays of 2 s for each LAV feature, capturing up to 8 s in the past. The best-performing feature weights, regularization parameters, and FIR filter shape were empirically selected by cross-validating over 10,000 random samples.

5.4 Results and discussion

Our encoding model explains variance across much of the brain, including much of the visual and motor cortices, as illustrated in Fig.5.3. The model also explains brain activity in parts of the prefrontal cortex. In contrast, passive vision-only experiments only engage visual perceptual areas, and encoding models for these experiments are limited to explaining variance in these areas [4], [69], [82].

To determine whether the driving DNN shares any functional organizational principles with the brain, we used the split R^2 scores to identify the best-performing driving DNN module for every voxel (Figure 5.3b). Quantitatively, we find that the spatial organization of the best-performing modules for each voxel is non-random in all five subjects (see Appendix C in section 8.5 for more details). Qualitatively, We find that earlier LAV modules, such as the semantic segmentation module, best explain brain activity in visual areas, whereas later

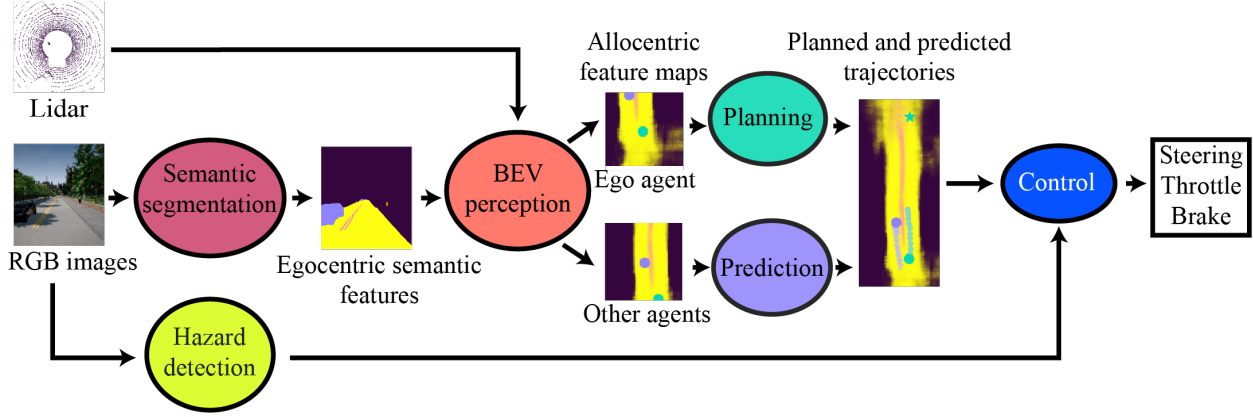


Figure 5.2: **Features from a driving DNN can be split into separate functional modules.** The architecture of the Learning from All Vehicles (LAV) driving DNN is shown. Each colored oval represents a different module, while inputs and outputs are shown with rectangles.

modules, such as the planning module, best explain brain activity in sensorimotor and higher-level cognitive brain regions. Overall, this result suggests that the representations learned by the driving DNN may be similar to those used by the human brain. This similarity is despite significant differences in perceptual inputs (human drivers have no LiDAR-like perception modalities, and our subjects were able to see only the image from a single camera instead of the four cameras used by LAV), training data (humans learn to drive primarily through closed-loop control rather than imitation learning), and physical substrate (real neurons behave very differently from artificial neural network units) between the two systems. In the following sections, we describe the predictive performance of each driving network module in more detail.

Semantic segmentation and hazard detection

The semantic segmentation and hazard detection modules, which both process RGB images from the vehicle cameras, explain brain activity almost exclusively in the posterior visual cortex. This pattern of explained variance is qualitatively similar to previous encoding model studies that used features derived from convolutional neural networks (CNNs) trained on vision tasks such as image recognition [68], [82]. The hazard detection module specifically explains more variance in inferior visual cortex, corresponding to the upper visual field. This bias may be due to the fact that, when approaching a vehicle, participants are likely to shift their gaze towards the ground. Because this downward shift moves the stimulus image upwards in the visual field, it is likely correlated with a consistent signal in the upper visual field. Another possible explanation is that traffic lights are located in the upper right part of the screen, and the hazard detection module is responsible for detecting red traffic lights

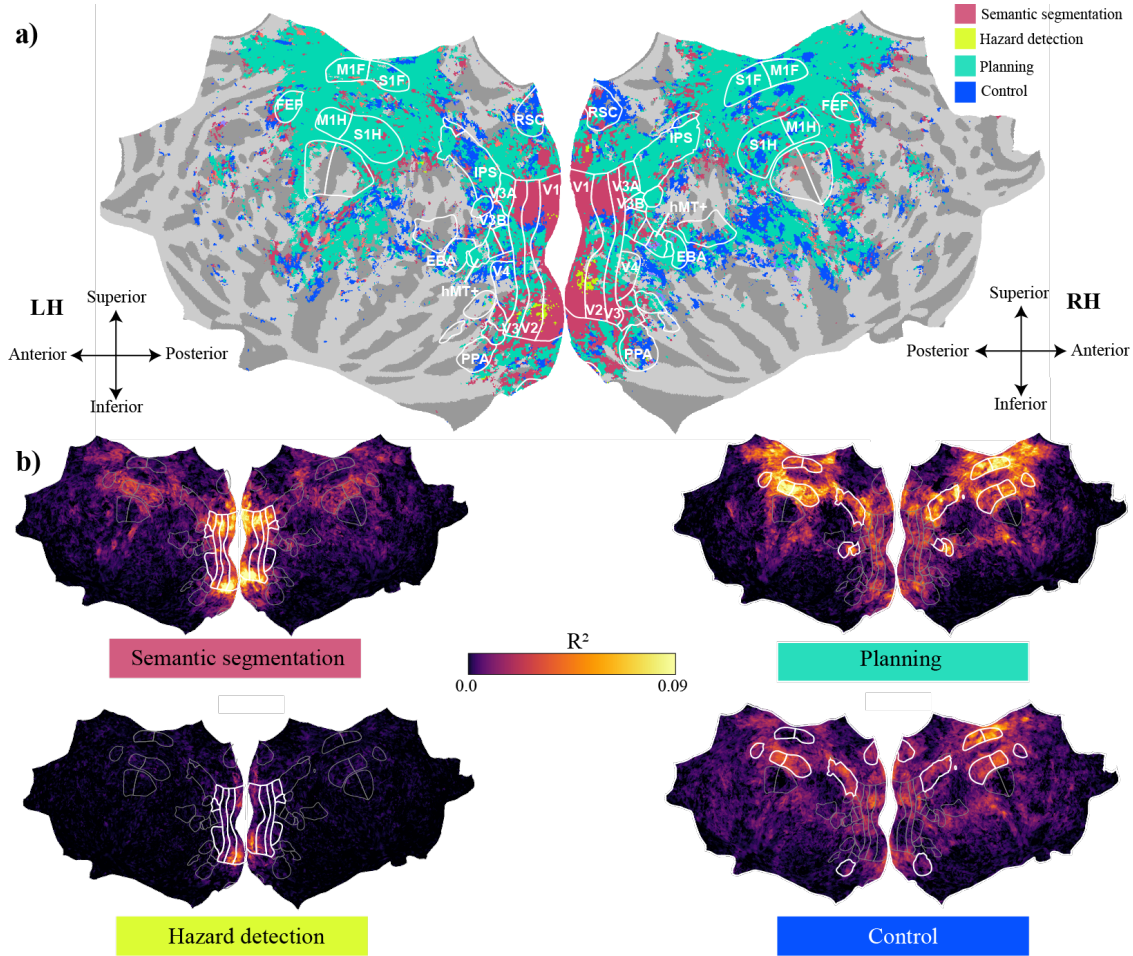


Figure 5.3: **Features from different modules in a deep neural network driving model predict activity in distinct functional brain networks.** Model performance was quantified as the explained variance (R^2) in brain activity on a held-out dataset. a): A group-level map across all subjects of the best performing LAV modules across the flattened cortical surface. Voxels with a performance p-value of < 0.01 in at least 3 subjects are shown in the color corresponding to the module with the highest explained variance. (Module colors match those in Fig. 5.2.) Features from the semantic segmentation and hazard detection networks, which process RGB images, best explain variance in visual areas V1-V4. The planning module best explains variance in the sensorimotor areas and the intraparietal sulcus (IPS). The control module has a similar pattern of explained variance to the planning module but additionally provides a good fit to the retrosplenial cortex (RSC) and perahippocampal place area (PPA). b): Group-level encoding model performance for individual modules across all subjects. The consistent patterns of brain activity explained by each of these modules suggests that humans and the driving network share an analogous organization of cognitive information in the active driving task.

that would require braking.

Together, these results suggest that the visual components of a DNN in an active driving task learn similar representations to human visual processing during driving.

BEV perception

The BEV perception module explains brain activity in punctate locations around the sensory and motor cortex and the anterior IPS (Figure 5.4a). (Because of the punctate nature of these regions and anatomical variability across subjects, they do not project to the same locations on the group-level map in Figure 5.3b). The performance of this module in explaining somatosensory activity is likely due to correlations between the layout of the local environment (e.g. an upcoming turn) and the driver’s control actions. Because the IPS is implicated in coordinate transformations [83], and the BEV perception module operates in a top-down reference frame instead of an egocentric one, it is likely that these two processes are correlated, enabling the BEV perception module to predict IPS activity.

Planning

The planning module best explains activity in voxels in the somatosensory and motor cortices, the intraparietal sulcus (IPS), and anterior visual areas such as the extrastriate body area (EBA). Well-explained regions of the somatosensory and motor cortices include the primary motor (M1H/F) and primary sensory (S1H/F) cortex for hand and foot, as well as supplementary hand and foot motor areas and the supplementary eye fields. These regions are directly responsible for planning and producing motor actions to physically control the simulated vehicle. The IPS is involved in perceptual-motor coordination [83], [84]. In addition to these known functional regions, the planning module also explains some activity in the lateral prefrontal cortex.

The planning module processes local features around the subject’s vehicle to output a planned trajectory based on these features. Since the driver produces control actions to follow their intended trajectory, it is intuitive that the planning and control modules provide the best fit to motor and supplementary motor areas — no LAV module contains explicit information about steering and acceleration outputs, and the features from the other modules are not as well-correlated with motor control as the planned trajectory. The IPS is known to be responsible for coordinate transformations [83]; since like the BEV perception module the planning module operates in a bird’s-eye-view reference frame, it may make use of representations similar to those found in the IPS. However, we note that overall activity in the IPS is better explained by the planning module than the BEV perception module. It is likely that the IPS simultaneously supports multiple processes for active driving, and features in our model related to coordinate frame transforms and perceptual-motor integration may both be associated with activity in the IPS. Finally, the explained variance in high-level visual areas is likely due to the correlations between certain semantic visual features (e.g.

the presence of an intersection or other vehicles) and the local BEV features and appropriate driving plan.

Trajectory prediction

The trajectory prediction module explains variance in multiple anterior visual regions, including the human middle temporal cortex (hMT+), extrastriate body area (EBA), occipital face area (OFA), and fusiform face area (FFA) (Fig.5.4b). (Similar to the well-predicted regions by the BEV perception module, some of these regions, particularly EBA, do not project well to a group-level space). The EBA is canonically implicated in the perception of body parts [85] and biological motion [86], the FFA in the perception of faces [87], and hMT+ in the perception of optic flow [88]. Note that while the FFA is canonically a face perception area, some evidence suggests that it may also function as a general expertise area for object perception [89], [90]. Because vehicles are an object category of particular importance during driving, the FFA (and associated regions for the perception of other agents) may be recruited in this capacity while driving [91].

Because this module is downstream of other modules that receive visual inputs, explained activity in high-level vision areas may indicate alignment between the intermediate representations of other agents in the LAV architecture and the representations of human drivers. However, even in its best-performing regions, the trajectory prediction module explains less variance than the planning module, even though the planning module does not explicitly represent information about other drivers. This suggests that the representations used by the trajectory planning module may not be well-aligned with those used by the human brain. Reasoning about the behavior of other agents is a problem that has proven especially challenging in autonomous driving [92]. Our results suggest that the strategy of the LAV algorithm, which projects all agents into a common representation and predicts trajectories for the ego vehicle and other vehicles in the same way, may not be a good match for how humans represent and reason about other drivers.

Control

The control module best explains activity in a similar network of brain regions as the planning module, namely the somatosensory and motor cortices, the IPS, and some anterior visual regions. This similarity is likely because the trajectory plan that the control module attempts to follow is produced by the planning module and is therefore highly correlated with some of the features in the planning module. In the anterior visual cortex, the controller module slightly outperforms the planning module in the retrosplenial cortex (RSC) and the parahippocampal place area (PPA), two visual scene perception regions. This is likely because while the control and planning module features are both correlated with the contents of the local scene, the trajectory plan and braking probability features from the control module are more succinct than the high-dimensional representation in the planning

module, and might therefore provide a stronger correlation with certain aspects of visual scene perception encoded in the RSC and PPA.

Limitations and future work

In our experimental paradigm, the human subject controls the vehicle and therefore the stimulus (rendered RGB images from the driving simulator). This interactivity produces a naturalistic stimulus distribution, whereas previous work on comparing visual representations between the brain and CNNs [4], [67] have all used tightly controlled stimuli. While having an interactive task allowed us to gain insights about an active sensorimotor task, it also results in strong correlations between task state, brain activity, and driving DNN activations. These strong correlations allow for brain activity to be predicted from driving DNN activations even if algorithms and representations are not aligned between the DNN and the brain. For example, if the subject always uses the brake pedal to stop their car when another vehicle is directly in front, then activity in the feet motor areas will be predictable from a representation that encodes the presence of another vehicle. We mitigate this issue by using banded ridge regression, which partitions the total explained variance across models [59], [80] to identify the best-performing feature space out of many correlated feature spaces that could independently explain similar amounts of variance.

These results do not tell us about the performance of our features relative to other possible feature spaces that we have not included in the regression. Since our encoding model, based on features from the LAV driving DNN, is only one hypothesis about the representations the brain may use for driving, in future work we plan to compare encoding models based on other algorithms for driving, such as those studied in [65] or deep learning models that incorporate emerging trends in deep learning for autonomous driving such as world modeling and uncertainty modeling [93]. Building encoding models using features from alternative models will enable us to explore which of these best reflect the algorithms and representations used by the human brain for driving.

5.5 Conclusion

In this work, we have presented an in-depth comparison of the cognitive representations for driving in humans and in a deep end-to-end neural network for autonomous driving. By directly comparing the representations in a deep learning driving algorithm with humans for the first time, we evaluate the alignment between the human brain and the components of a driving network. Our results highlight a striking similarity in the way that driving-relevant representations are organized between humans and the deep neural network. However, we also identified aspects of the network that might be less aligned with humans, including the representation of other vehicles on the road. While driving algorithms do not necessarily need to mimic humans to perform well, humans are still more capable than autonomous vehicles in most environments. Thus, better alignment with human representations may

improve autonomous vehicle performance. Furthermore, better alignment with human representations may enable autonomous vehicles to make better inferences about other (human) drivers, an important component of safe and socially acceptable driving. In turn, deep neural networks can provide novel insights about how the brain represents information for solving complex tasks in dynamic environments. Our approach provides a promising framework for understanding how both humans and AI agents solve dynamic, active tasks, both in driving as well as in other domains.

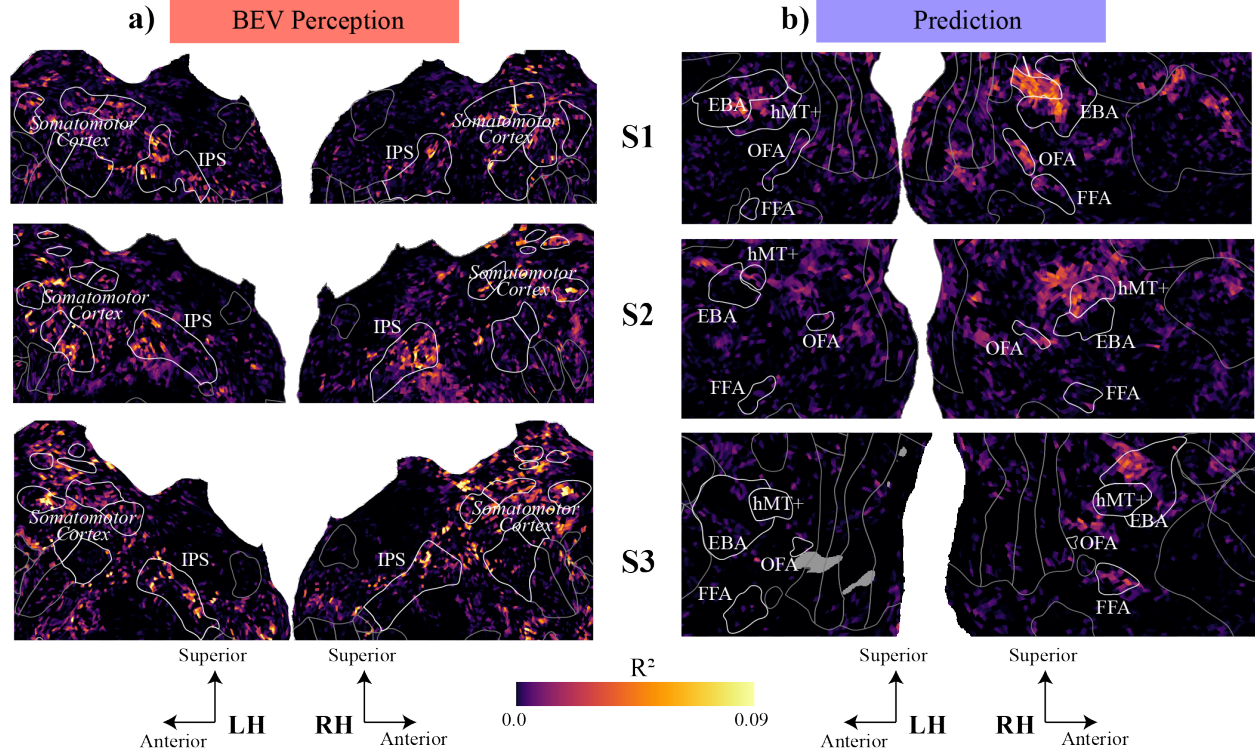


Figure 5.4: **Features from modules implicated in social navigation explain variance in known functional regions in the brain that perform similar tasks.** In the LAV architecture, the bird’s-eye-view (BEV) perception module (left) predicts the layout of the road and other vehicles around the driver from a top-down viewpoint, while the prediction module (right) predicts the trajectories of other nearby vehicles. Unlike the other LAV modules, these are not directly connected to either sensory inputs or control outputs of the human driver, and their features are thus more abstract sets of driving-relevant information relevant to deciding how to navigate around other vehicles. a): In all five subjects, the BEV perception features explain variance in anterior intraparietal sulcus (IPS), which is implicated in coordinate transformations. They also explain variance in punctate somatomotor cortex locations. b): The prediction module explains variance in human middle temporal cortex (hMT+)/extrastriate body area (EBA) complex, and in the occipital face area (OFA) and fusiform face area (FFA). The FFA and OFA are canonically implicated in the perception of faces, while the EBA/hMT complex is canonically implicated in the perception of body parts and biological motion. These results suggest that the LAV modules are specialized in a similar manner to the functional brain regions mediating driving.

Chapter 6

Mechanistic interpretability for steering vision-language-action models

This chapter is adapted from the paper “Mechanistic interpretability for steering vision-language-action models,” accepted for presentation at the 2025 Conference on Robot Learning, and included with permission from my co-authors Bear Häon, Ian Chuang, and Claire Tomlin, and included here with their permission.

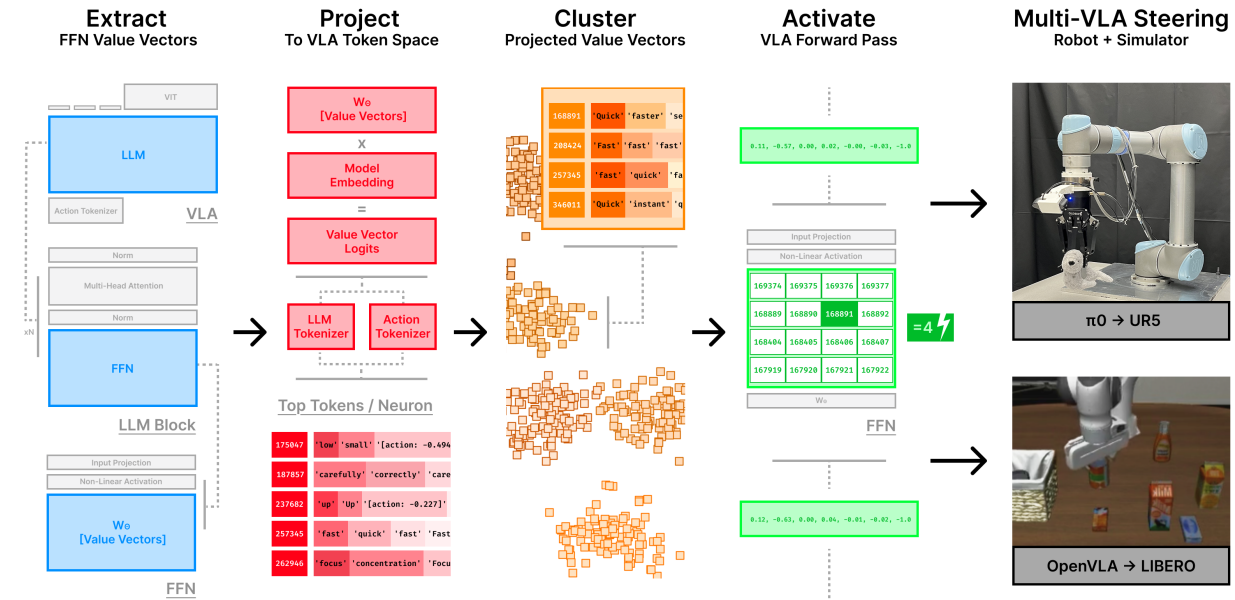


Figure 6.1: We present a framework for steering Vision-Language-Action (VLA) models. We extract feed-forward network (FFN) vectors, project them to the VLA token space, cluster them by semantic alignment, and inject activations at inference time to modulate behavior. Our experiments demonstrate interpretable zero-shot control in both simulation (OPENVLA in LIBERO) and on a physical robot (π_0 on a UR5).

6.1 Introduction

Large language-conditioned models represent a paradigm shift for robotics. By leveraging Internet-scale natural language and image data, these models promise to yield generalist robot policies that can follow directions, exhibit common-sense reasoning about their environments, and quickly adapt to novel tasks and scenarios.

One leading type of frontier robotics model is the Vision-Language-Action (VLA) model, where a pre-trained Vision-Language Model (VLM) is trained to produce robot actions conditioned on a task description and image observations. While most VLAs are designed to be fine-tuned for specific tasks, future models may even be able to perform a variety of tasks without the need for fine-tuning. Indeed, the π_0 model [94] can often complete zero-shot evaluation tasks on the DROID platform [95] and Gemini Robotics demonstrates zero-shot control via code generation [96]. This promise of generality and ease of deployment signals a large shift in how we deploy robot systems. In contrast to rigorous evaluation and certification on individual tasks in factories or warehouses, robots will be placed in novel environments and expected to behave well “out-of-the-box”. This raises important questions about safety, robustness, and transparency. How and when can we trust that VLA policies will behave safely? When they fail, how can we diagnose and fix the underlying problems?

A similar set of questions and challenges is being faced by Large Language Model (LLM) researchers. While they have by no means been solved, the field of *mechanistic interpretability* has yielded a number of exciting insights about the inner workings of LLMs. For example, researchers have discovered the functional role of specific attention heads [97] and identified semantically interpretable concepts in model activations [98], [99]. There is also evidence that mechanistic interpretability can be useful for diagnosing problems with models in practice [100]. These insights lead us to ask: can mechanistic interpretability also help with understanding and shaping the behavior of frontier models in robotics?

In this work, we present a series of experiments that together allow us to provide a resounding affirmative answer. We focus our efforts on an interpretability method that analyzes the semantic meanings of vectors in the output weight matrix of the transformer block’s feed-forward layer (FFN) [101]. Although this technique has less power to disentangle concepts that might be spread across multiple neurons in the model than alternatives such as sparse autoencoders [98], it allows us to interpret and steer the model without any additional training data. This is a key advantage in robotics where training data is expensive and difficult to acquire, and where available open datasets are small compared to what is available in the natural language domain. Our chosen method leads us to several surprising conclusions about VLAs, including:

1. Despite being trained to produce only robot actions, the internal activity of the VLA model is largely semantic, with less than 25% of FFN neurons being rewired for action prediction and a large fraction of the remainder representing clear, semantically interpretable concepts.
2. Internal model concepts such as “slow” are causally linked to robot actions that express

these concepts (i.e. moving the end effector more slowly), despite the model being trained without any feedback on the meaning or consequences of different action tokens.

3. Targeted interventions on model concepts allow VLA behavior to be modified at inference time out-of-the-box, without any fine-tuning data or environment interaction required to select the intervention.

These contributions represent the first attempt to use mechanistic interpretability techniques developed for LLMs with large robotics models. Our exciting initial results speak to the great promise of adapting these techniques to the robotics context, where there is an especially strong need for AI models that are safe and transparent.

6.2 Related Work

Mechanistic Interpretability and Robotics. The size and complexity of deep learning systems make it challenging for humans to understand how they arrive at a particular output from a given input. To help address this problem, a variety of methods have been proposed to help expose and interpret the internal representations and computations in models. The most powerful form of interpretability work doesn’t simply yield qualitative insights, but elucidates the causal mechanisms that determine the output of the model - this is often referred to as “mechanistic interpretability”. Despite promising developments in mechanistic interpretability in the vision [102] and natural language [8], [98], [103] settings, there has been relatively little work investigating DNN interpretability for robotics. In [104], they aim to interpret a deep learned policy by associating its factors of variation with logic rules, but the system they investigate is small in scale. In [105], they use techniques inspired by vision interpretability work to improve a robotic image-conditioned policy. Finally, there has been recent interest in interpretability for reinforcement learning policies, as surveyed in [106].

6.3 Interpreting VLAs

Deep neural networks build their predictions layer by layer, with features in one layer supporting further computations and representations in the next. According to the “linear representation hypothesis”, intermediate network layers often represent different features or concepts as directions in the latent embedding space [107]. This means that if, for example, a model represents the concept of “fast”, if we can identify the corresponding direction in its latent space, we can understand when and how the model uses this concept to produce its final output. Serendipitously, it turns out that for many large models, semantically meaningful directions align with individual neurons. This fact has been used to help interpret both vision [102] and language [101], [108] models. Recently, there has been excitement around using sparse autoencoders (SAEs) to uncover even more meaningful directions [98].

However, training SAEs require large amounts of data, and the results vary depending on the exact dataset used. Both of these factors are significant barriers in the robotics context, where open-source datasets are limited and are unlikely to cover all of the semantic concepts that the model learned and might retain from VLM pretraining. Therefore, in our work, we focus on single-neuron features.

Specifically, we follow the approach outlined in [101] and focus on the feedforward layer of each transformer block:

$$\text{FFN}(x) = f_{\theta}(x)W_{\theta} \quad (6.1)$$

where $x \in \mathbb{R}^n$ is the input to the layer, $f_{\theta}(x) \in \mathbb{R}^m$ is an input-dependent set of activations, and $W_{\theta} \in \mathbb{R}^{m \times n}$ is an input-independent parameter matrix. Using $w_{\theta}^{(i)} \in \mathbb{R}^n$ to refer to the i th row of W_{θ} , equation 6.1 can be rewritten as follows:

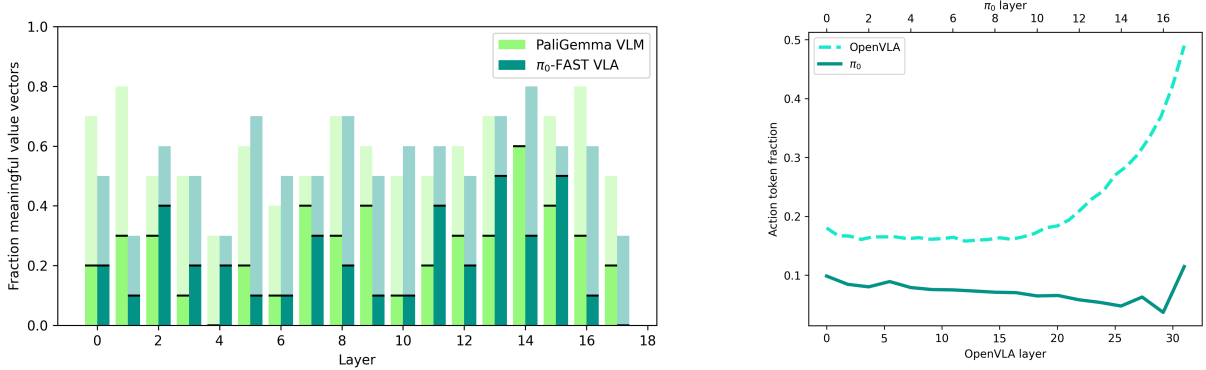
$$\text{FFN}(x) = \sum_i [f_{\theta}(x)]_i w_{\theta}^{(i)} \quad (6.2)$$

This allows us to interpret the FFN as a weighted sum over the $w_{\theta}^{(i)}$ vectors, which we refer to as *value vectors*. Since these vectors are independent of the inputs, they can be thought of as basis functions for the output of the FFN. Furthermore, as they belong to the same linear space as the final output of the transformer, they can be interpreted as probability distributions over possible output tokens. Therefore, we can assign semantic meanings to value vectors based on the set of tokens to which they assign the highest probability.

How does VLA Training Affect Concepts Learned During VLM Pre-training?

During VLA training, a pre-trained VLM is fine-tuned to imitate the control actions of expert robot trajectories conditioned on a task description and image and/or state observation, using a dataset such as Open X-Embodiment [109]. The most common strategy for handling the actions, followed by VLAs such as RT-2 [110], OPENVLA [111], and π_0 -FAST [95], is to denote a small set of rarely used VLM tokens as “action tokens” to be decoded into control outputs. The final output of the model during VLA training is exclusively action tokens, which raises the question - does the VLA still use natural language-like representations to choose appropriate control actions? If so, how do language and action interact in the trained model? Our chosen interpretability technique allows us some insight into these questions:

VLA Value Vectors Retain Semantic Meanings. First, we aimed to quantify whether the VLM value vectors represent semantically interpretable concepts, and if so, whether this is preserved after VLA training. We focused on π_0 and its base pretrained VLM, PaliGemma-3B [112], because its larger vocabulary size, compared to OPENVLA’s Llama2-7B base VLM [113], makes individual tokens easier to interpret. We randomly selected 10 value vectors from each layer of the VLM and VLA models and examined whether at least 4 of the top 30 tokens followed a common pattern. The results are shown in Figure 6.2a. Value vectors throughout both the VLM and VLA models exhibit identifiable patterns, many of



(a) **Meaningful patterns in top value vector tokens.** VLA training does not substantially change the proportion of FFN value vectors which have interpretable patterns (top lighter bars) and semantically meaningful patterns (bold bottom bars) in their top tokens.

(b) **Action tokens are incorporated into every layer of the VLA.** However, they make up the largest proportion of final layer value vectors.

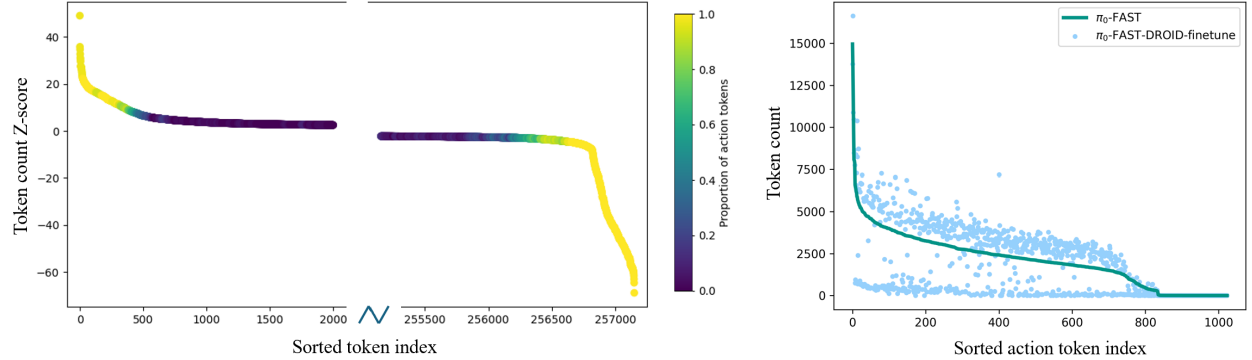
Figure 6.2: VLA training incorporates action tokens into FFN value vectors but retains semantic meanings from VLM pre-training.

them semantic, at similar rates. Therefore, VLA training does not cause FFN outputs to lose meaningful organization of concepts, despite producing only non-semantic tokens at the final output. We show qualitative examples of value vectors in Appendix B in section 8.7.

Action Tokens Appear in All VLA Layers. We show the proportion of top-100 tokens which are action tokens for value vectors in each layer of the model (Figure 6.2b). For both OPENVLA and π_0 , action tokens are most common at the final layer, but make up at least a few percent of every layer. This suggests that there is not a hard transition from “thinking about the task” in earlier layers to “thinking about control” in later layers. Instead, it appears VLA models learn to reason about and refine their control action predictions continuously, starting from their earliest computations.

How Task-Specific Fine-Tuning Affects VLA Concepts

After VLA pre-training, models generally need to be fine-tuned for a specific robot setup and set of tasks to realize good performance. To understand what happens to the value vectors during fine-tuning, we compared the base π_0 -FAST model with a checkpoint fine-tuned on the DROID dataset [114]. We compared top tokens in the value vectors using a two-proportion z-test statistic to quantify the significance of an increase (or decrease) in the number of occurrences of each token between the two models. Our results are shown in Figure 6.3a and 6.3b. We find that action tokens account almost exclusively for the most significant differences between the two sets of value vectors, and that this shift occurs because the fine-tuned model develops a more uneven, specialized distribution across action tokens compared to the relatively flat distribution of the pre-trained model. There is also a



(a) **Task fine-tuning mainly affects action tokens.** The most up-weighted and down-weighted tokens between the π_0 -FAST and π_0 -FAST-DROID-finetune models are action tokens.

(b) **Fine-tuning induces a more specialized (less general) distribution of action tokens across value vectors.**

Figure 6.3: Task fine-tuning mainly affects action tokens in FFN value vectors.

modest boost to some of the most common tokens in the DROID task instructions: these tokens appear 1.2 times more frequently in the fine-tuned model value vectors than in the pre-trained ones (see Appendix B in section 8.7 for more details). Overall, these results suggest that the main effect of fine-tuning is to re-wire the model to more easily produce action tokens utilized in the fine-tuning tasks and neglect the others, with only a modest effect on semantic tokens.

Summary

The above results suggest that VLAs reason about control actions by mixing semantic concepts from pre-training with action tokens at all layers. However, these results do not prove a direct causal link between specific semantic concepts and robot behavior. If such a link exists, we expect to be able to *intervene* on semantic concepts to change behavior. In the following section, we examine whether the semantic concepts we uncovered can support such interventions.

6.4 Steering VLAs

We formalize our method for *interpretable activation-level steering* of VLAs (Figure 6.1). This technique modifies selected neurons within the FFN submodules of transformer blocks inside the VLA’s base VLM, steering robot behavior in real time - without fine-tuning or environment reward signal.

Let $x \in \mathbb{R}^n$ be the residual input to a transformer FFN. As shown in Equation (6.2), the FFN output is a sum of fixed value vectors $w_\theta^{(i)} \in \mathbb{R}^n$ weighted by input-dependent activations $f_\theta(x) \in \mathbb{R}^m$. Motivated by evidence that individual FFN neurons encode semantic features

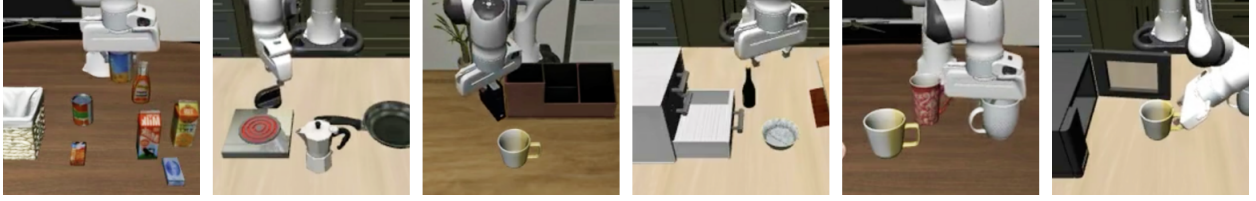


Figure 6.4: **Sample tasks from LIBERO-Long.** Six representative long-horizon tasks - involving sequential manipulation goals such as object placement, containment, and appliance interaction.

[98], [108], we override a subset $\mathcal{S} \subseteq \{1, \dots, m\}$ of activations using a fixed scalar $\alpha \in \mathbb{R}$. Each $\mathcal{S}$ corresponds to an interpretable neuron cluster aligned with a control concept - e.g., *fast*, *up*, *careful* - identified by grouping neurons with similar token projections (Figure 6.1, steps 1–3), either manually or via kNN over semantic embeddings.

$$\tilde{f}_{\theta}^{(i)}(x) = \begin{cases} \alpha & \text{if } i \in \mathcal{S} \\ [f_{\theta}(x)]_i & \text{otherwise} \end{cases} \quad (6.3)$$

$$\text{FFN}_{\text{steered}}(x) = \sum_{i=1}^m \tilde{f}_{\theta}^{(i)}(x) \cdot w_{\theta}^{(i)} \quad (6.4)$$

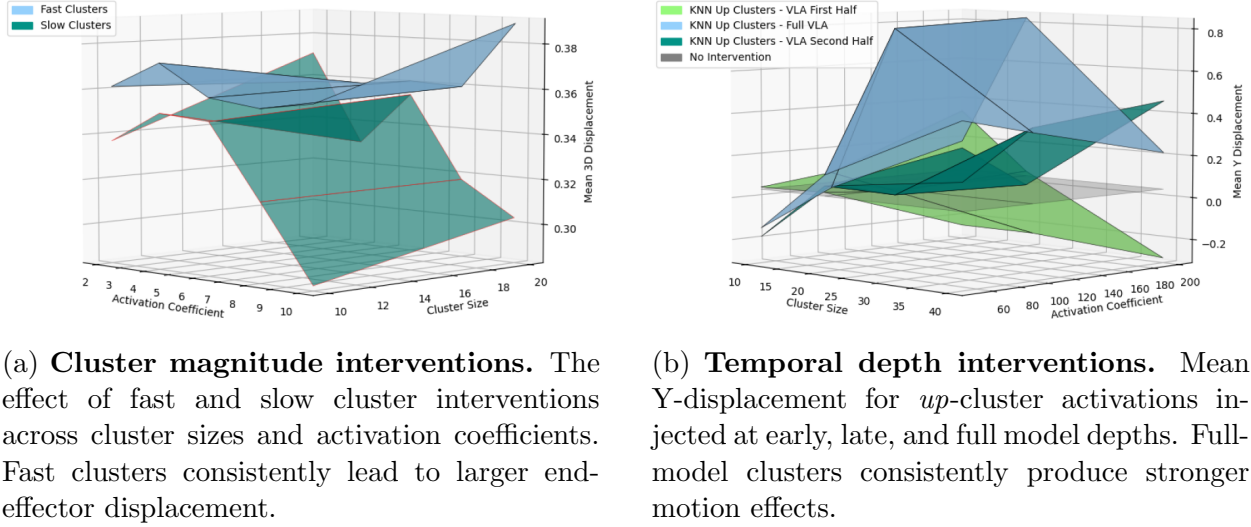
This induces a residual shift $\Delta x = \text{FFN}_{\text{steered}}(x) - \text{FFN}(x)$ that propagates through the transformer and modulates the final VLA action token distribution.

Implementation. In OPENVLA (PyTorch), we apply a forward hook on the FFN’s `down_proj` to overwrite neuron activations. In π_0 (JAX), the neuron indices $\mathcal{S}$ and activation coefficient α are passed into a refactored FFN. Both realize the same operator $\mathcal{I}_{\mathcal{S}}^{\alpha}(f_{\theta}(x))$ for real-time robot control.

Simulation Experiments

We evaluate our VLA steering method within the suite of ten long-horizon manipulation tasks making up the LIBERO-Long benchmark [115], visualized in Figure 6.4. Rollouts are seeded deterministically for reproducibility, enabling precise comparisons between baseline and intervention trajectories. All simulation experiments use the 7B-parameter OPENVLA model, with the fine-tuned LIBERO-Long checkpoint, implemented in PyTorch and executed on a NVIDIA H100 GPU.

Cluster Magnitude Interventions. To evaluate whether interpretable semantic structure in VLA value vectors can steer physical behavior, we construct *fast* and *slow* aligned clusters by manually selecting value vectors whose top-projected tokens reflect motion magnitude semantics. These clusters are injected into the model’s residual stream with a fixed coefficient, sweeping over cluster sizes $[10, 20]$ and activation strengths $[2, 4, 6, 8, 10]$. Each configuration is evaluated over 10 long-horizon tasks, using 10 rollouts per task. The fast



(a) **Cluster magnitude interventions.** The effect of fast and slow cluster interventions across cluster sizes and activation coefficients. Fast clusters consistently lead to larger end-effector displacement.

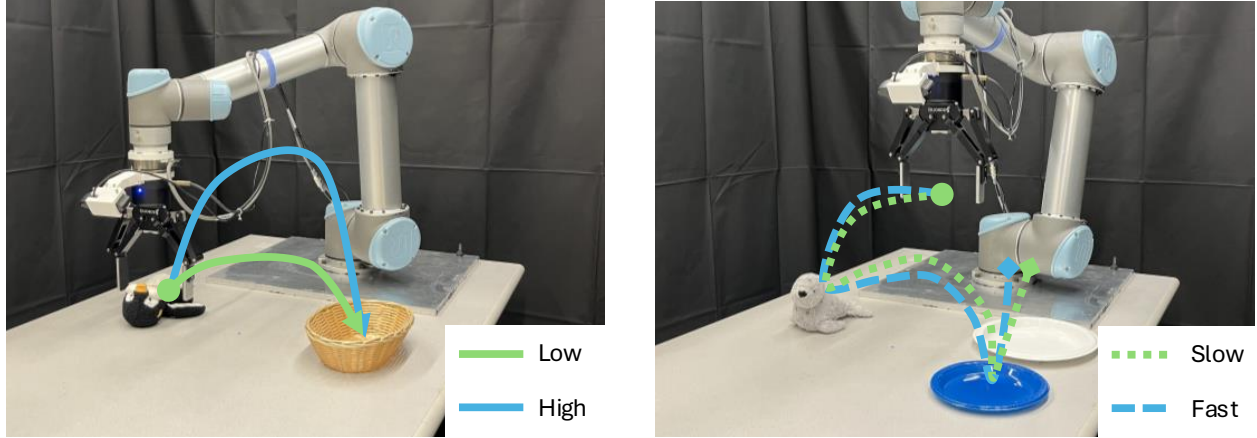
(b) **Temporal depth interventions.** Mean Y-displacement for *up*-cluster activations injected at early, late, and full model depths. Full-model clusters consistently produce stronger motion effects.

Figure 6.5: **Simulation results:** Steering OPENVLA with value vector interventions.

clusters consistently induce larger end-effector displacements across tasks, with an average improvement of 10.9% over slow clusters and maximum gains of 28.4% in some configurations. All 10 matched comparisons yield statistically significant differences ($p < 0.05$, paired t-test), and effect sizes ($d = 0.01 - 0.23$) suggest consistent directional influence. These results demonstrate that directional behavior can be modulated by activating semantically interpretable value vectors alone (Figure 6.5a).

Temporal Depth Interventions. We examine how the position of activated value vectors within the VLA affects steering performance. Using kNN clustering over token-projected value vector embeddings, we extract *up*-themed clusters and restrict their injection to early, late, or all layers. Each configuration spans three cluster sizes and three activation coefficients, with all other variables held constant. Our VLA steering method with full-model interventions produced the largest Y-displacements of the end-effector ($\mu = 0.316$), outperforming late-layer clusters ($\mu = 0.123$) and early-layer clusters ($\mu = 0.026$) in nearly every setting. The largest contrast emerges between full and early activation depths ($p = 0.052$, $d = -0.805$), with weaker but consistent differences between full and late ($p = 0.16$, $d = -0.547$). These findings, summarized in Figure 6.5b, suggest that distributed composition across layers amplifies the behavioral effect of value vector steering, while early-layer only modifications exert less control influence.

***Up* is All You Need?** We evaluate our VLA steering method on task performance across the LIBERO-Long benchmark with a single, hand-curated *up*-themed cluster of size 10 and coefficient 4. Across 50 rollouts per task, it raised the average task success rate from 52.2% to 55.0%, improving outcomes in 7 out of 10 tasks with minor declines in the others. To understand this effect, we analyzed the 239 no-intervention failure episodes and found that 97 involved the robot becoming motionless - taking no further actions despite an incomplete task state. Of those motionless cases, 51 were successfully completed under



(a) **Low/high transport.** The robot picks up a toy penguin and places it into a basket, with variations in the robot’s trajectory height during data collection.

(b) **Slow/fast transport.** The robot picks up a toy seal and places it onto a plate, with variations in the robot’s movement speed during data collection.

Figure 6.6: **Physical robot experiments:** Steering π_0 on a UR5.

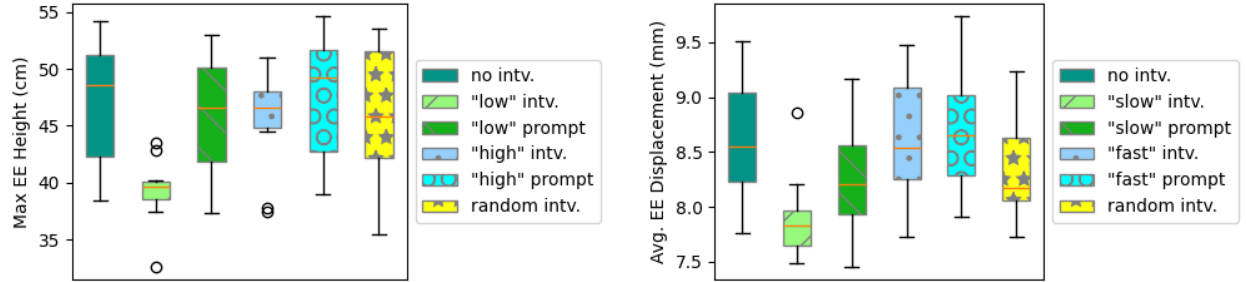
the up-themed intervention. This suggests that semantically aligned value vectors can act as high-level behavioral nudges, reinitiating stalled execution and guiding the model toward goal completion.

Robot Experiments

We also evaluate our VLA steering method using the 3B-parameter π_0 -FAST VLA on a real robot. Specifically, we test the ability to steer binary control opposites - *slow vs. fast* and *low vs. high* - in two pick-and-place tasks with a UR5 robot arm. We use a JAX implementation of π_0 -FAST running on a NVIDIA A4500 GPU. Details on our custom robot setup are provided in Appendix D in section 8.9.

Fine-Tuning. Unlike benchmarks such as DROID, which use a standardized robot platform and include fine-tuned π_0 -FAST checkpoints, our robot setup is not represented in the π_0 -FAST training data and cannot reasonably perform tasks zero-shot. To enable meaningful evaluation, we fine-tune with LoRA [116] on small datasets collected from our robot. Although fine-tuning on a limited dataset may hinder generalization and steering capability, it enables controlled experimentation: we manage the full data collection process and can introduce variations and behaviors in the robot data that we wish to steer during evaluation. Crucially, we do not explicitly label these variations in the VLA prompt during training, allowing us to test whether the VLA can reproduce these specific behaviors through steering by associating them with high-level semantic concepts.

Robot Tasks. We evaluate steering on two tasks. The first task, *Low/High Transport*, involves the robot picking up a toy penguin and placing it in a basket and includes 75



(a) **Low/high transport.** Box plots for maximum end-effector height (cm) showing distribution across 10 rollouts for each steering intervention (intv.) and baseline.

(b) **Slow/fast transport.** Box plots for average end-effector displacement (mm) between each successive action showing distribution across 10 rollouts for each steering intervention (intv.) and baseline.

Figure 6.7: **Physical robot experiments:** Binary control opposites.

episodes with unlabeled variations in how high the penguin is lifted during transport. The second task, *Slow/Fast Transport*, involves the robot picking up a toy seal and placing it on a specified plate and includes 120 episodes with variations in the speed of the robot's motion. Visualizations of both tasks are shown in Figures 6.6a and 6.6b, and data collection and fine-tuning details are provided in Appendix D in section 8.9.

For both tasks, we evaluate steering interventions where we hand-select and upweight a cluster of semantically meaningful vectors. We create a *low* and *high* themed cluster for *Low/High Transport* and *slow* and *fast* themed cluster for *Slow/Fast Transport*. We also compare these interventions against several baselines: (1) running the model without any intervention, (2) adding qualitative descriptors to the prompt (e.g. appending *low* to the beginning of the prompt), and (3) applying steering with randomly upweighted vectors. These comparisons evaluate the model's default behavior without intervention, whether prompt modification alone can influence behavior, and whether selecting steering vectors based on semantic concepts are more effective than simply selecting random vectors. Specific details on each steering intervention and baseline is in Appendix D in section 8.9.

Task Evaluation. For *Low/High Transport*, we perform 10 independent rollouts for each steering variant and baseline and analyze the maximum height of the end-effector relative to the robot's base over the course of the task. We then record the maximum height across 10 rollouts for each variant in Figure 6.7a. In *Slow/Fast Transport*, we focus on measuring the robot's speed. To do this in a consistent manner we run inference for all steering variants and baselines simultaneously, but only execute the baseline predicted action with no intervention. We measure average end-effector displacement between each step along the trajectory across 10 rollouts for each variant and plot results in Figure 6.7b. More detailed analysis of individual trajectories for both tasks are in Appendix D in section 8.9.

Robot Results. Our results suggest that steering interventions can meaningfully influ-

ence the actions of π_0 -FAST. In the *Low/High Transport* task, the *low* intervention generated the lowest overall trajectories. Similarly, the *slow* intervention in *Slow/Fast Transport* resulted in the slowest overall movements. However, while *low* and *slow* interventions clearly altered behavior, their opposites (*high* and *fast*) resembled the baseline without intervention. This may be because the model already “considers” the baseline trajectory to be fast and high. Nevertheless, the minimal difference between using a random intervention compared to no intervention suggests that choosing semantically meaningful vectors can more effectively guide the model towards desired behavior. Finally, we found that steering interventions were more impactful than simply adding target descriptive words to the prompt. In both tasks, modifying the prompt had weaker impact than our steering intervention.

6.5 Discussion and Conclusion

This work shows that VLAs can be steered via interpretable internal components. By activating sparse, semantically meaningful neurons in FFN layers - aligned with concepts like “up” and “slow” - we modulate robot behavior at inference time. Experiments demonstrate that our method generalizes across models and tasks, influencing both simulated and real-world behavior, including long-horizon LIBERO tasks and binary control opposites on a UR5.

From a mechanistic perspective, this work provides causal evidence that VLAs retain structured, compositional semantics even after fine-tuning. That activating a small set of semantically aligned neurons can shift motion behavior suggests control-relevant features remain accessible and manipulable. These effects emerge without explicit behavioral supervision, indicating that VLAs internalize semantic concepts during pretraining in a way that supports compositional reasoning across layers.

For robotics practitioners, this method offers a simple, interpretable tool for post-deployment behavior shaping. It requires no retraining, runs on existing policies, and enables fast testing and adjustment. Our experiments show that it can modulate some aspects of robot behavior - such as end-effector speed or height - and suggest that it may be more effective than adding similar prompts.

Looking ahead, activation steering opens a new interpretable control interface for guiding VLA behavior. As embodied foundation models become more capable, steering techniques could play an important role in inspecting, auditing, and intervening in how models reason and act. We see this approach - using just one tool from the mechanistic interpretability toolbox - as a step toward a broader class of transparent, steerable, and semantically grounded control tools for robotics.

6.6 Limitations

This work introduces one of the first interpretable control knobs for foundation models in robotics, enabling behavior to be modulated through semantically grounded activations without retraining. Establishing such control-layer transparency is a step toward safe, adaptive embodied AI. While the results are promising, several challenges remain - both in reliably mapping internal structure to behavior and scaling this method to broader settings. Addressing these presents concrete opportunities to advance the robustness, generality, and practical utility of activation-based steering.

Semantic Ambiguity and Representational Drift. Our clustering method is based on token-level semantic similarity rather than direct behavioral outcomes. For example, approaches like kNN over token-projected value vectors assume that local similarity implies shared semantics. In practice, clusters can conflate distinct behaviors - e.g., “slow and careful” versus “slow and stuck” - leading to inconsistent effects. Furthermore, we found that the meaning of value vectors can shift across models, tasks, and time. A vector aligned with “up” in one context may behave differently in another. Understanding and mitigating this representational drift will be critical for developing steering interventions that transfer reliably across domains and deployment scenarios.

Fine-Tuning and Steerability. We do not yet fully understand how VLA fine-tuning affects steerability. While some semantic directions appear to retain causal influence after adaptation, it’s possible that fine-tuning alters internal representations in ways that make VLM-pretrained concepts harder to access or less aligned with behavior. Investigating how steerable directions evolve with continued training is an ideal direction for future work.

Evaluation Scope and Generalization. Our evaluations, while spanning both simulation and hardware, are limited to pick-and-place tasks with a robot arm. Extending to mobile and multi-limbed platforms, as well as unstructured environments, will be important for validating control stability, generalization under physical variation, and alignment with human intent across real-world settings.

Part III

Conclusion and end matter

Chapter 7

Final thoughts and future directions

Chapter 1 of this dissertation laid out an ambitious vision for strong mechanistic understanding of learned robot representations in the pursuit of safer and more trustworthy autonomous systems. The research described in the subsequent chapters approaches this vision from several different angles, covering different kinds of representations, different robotic applications, and different methods for improving interpretability. These works represent only some initial small steps towards achieving the goals I have outlined. There is much more to be done!

Going forwards, there are a number of directions that seem especially exciting for future work. On the side of design, the self-supervised disentangled representation learning method described in chapter 4 should be extended to handle evolving scenarios over time, partial observability, and representation learning that is more tightly linked with a closed-loop robot control policy. Learning the representation and policy simultaneously would allow for prioritizing visual features that are more relevant to the task at hand while still emphasizing disentanglement, compositionality of different objects or agents, and interpretability.

On the side of interpretation, there are many more large language model (LLM) mechanistic interpretability techniques that might be fruitfully applied to vision-language-action models in robotics than the initial one that I explored in chapter 6. For example, it would be interesting to examine methods such as knowledge representations [117], representation engineering [118], or circuit tracing [119]. Despite their similarity to LLMs, VLAs have unique features, such as outputting control actions or plans instead of natural language, that may require adaptations to existing interpretability techniques. However, these unique features might also inspire new approaches that could be useful for improving our understanding not only of VLAs, but of LLM-based models more generally.

Finally, a crucial piece of the story that has not been addressed in this dissertation is whether the insights we gain from interpretability improve understanding and safety in practice. Recent work has questioned whether existing interpretability methods translate into more accurate inferences about data and model behavior [120]. Future work should take care to ground novel methods in how, and under what circumstances, they might actually improve understanding.

The path toward capable AI we understand, whether we follow the route of design or that of interpretation, is likely to be difficult. However, if better interpretability can indeed improve safety of autonomous systems, the need for this work is clear and urgent: we are currently giving these systems more and more influence over our lives even as their complexity increasingly outstrips our ability to ensure their safety. But this isn't inevitable. We must aim for AI systems that are not just highly capable most of the time, but safe all of the time.

Bibliography

- [1] D. H. Hubel and T. N. Wiesel, “Receptive fields of single neurones in the cat’s striate cortex,” *The Journal of physiology*, vol. 148, no. 3, p. 574, 1959.
- [2] L. G. Roberts, “Machine perception of three-dimensional solids,” Ph.D. dissertation, Massachusetts Institute of Technology, 1961.
- [3] D. Amodei, *The urgency of interpretability*, 2024. [Online]. Available: <https://www.darioamodei.com/post/the-urgency-of-interpretability>.
- [4] D. L. Yamins, H. Hong, C. F. Cadieu, E. A. Solomon, D. Seibert, and J. J. DiCarlo, “Performance-optimized hierarchical models predict neural responses in higher visual cortex,” *Proceedings of the national academy of sciences*, vol. 111, no. 23, pp. 8619–8624, 2014.
- [5] K. P. Wabersich, A. J. Taylor, J. J. Choi, *et al.*, “Data-driven safety filters: Hamilton-Jacobi reachability, control barrier functions, and predictive methods for uncertain systems,” *IEEE Control Systems Magazine*, vol. 43, no. 5, pp. 137–177, 2023.
- [6] J. Li, D. Lee, J. Lee, K. S. Dong, S. Sojoudi, and C. Tomlin, “Certifiable reachability learning using a new Lipschitz continuous value function,” *IEEE Robotics and Automation Letters*, 2025.
- [7] S. Dean, A. Taylor, R. Cosner, B. Recht, and A. Ames, “Guaranteeing safety of learned perception modules via measurement-robust control barrier functions,” in *Conference on Robot Learning*, PMLR, 2021, pp. 654–670.
- [8] J. Lindsey, W. Gurnee, E. Ameisen, *et al.*, “On the biology of a large language model,” *Transformer Circuits Thread*, 2025. [Online]. Available: <https://transformer-circuits.pub/2025/attribution-graphs/biology.html>.
- [9] E. J. Michaud, A. Gleave, and S. Russell, *Understanding learned reward functions*, 2020. arXiv: 2012.05862 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2012.05862>.
- [10] S. Ettinger, S. Cheng, B. Caine, *et al.*, *Large scale interactive motion forecasting for autonomous driving: The Waymo Open Motion Dataset*, 2021. arXiv: 2104.10133 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2104.10133>.

- [11] M. Huh, B. Cheung, T. Wang, and P. Isola, *The platonic representation hypothesis*, 2024. arXiv: 2405.07987 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2405.07987>.
- [12] A. J. Kell, D. L. Yamins, E. N. Shook, S. V. Norman-Haignere, and J. H. McDermott, “A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy,” *Neuron*, vol. 98, no. 3, pp. 630–644, 2018.
- [13] R. Greenblatt, C. Denison, B. Wright, *et al.*, *Alignment faking in large language models*, 2024. arXiv: 2412.14093 [cs.AI]. [Online]. Available: <https://arxiv.org/abs/2412.14093>.
- [14] T. Korbak, M. Balesni, E. Barnes, *et al.*, *Chain of thought monitorability: A new and fragile opportunity for ai safety*, 2025. arXiv: 2507.11473 [cs.AI]. [Online]. Available: <https://arxiv.org/abs/2507.11473>.
- [15] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, “Maximum entropy inverse reinforcement learning,” *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, p. 6, 2008.
- [16] D. R. Scobee and S. S. Sastry, “Maximum likelihood constraint inference for inverse reinforcement learning,” in *International Conference on Learning Representations*, 2019.
- [17] K. C. Stocking, D. L. McPherson, R. P. Matthew, and C. J. Tomlin, *Discretizing dynamics for maximum likelihood constraint inference*, 2021. arXiv: 2109.04874 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2109.04874>.
- [18] G. Gergely, H. Bekkering, and I. Király, “Rational imitation in preverbal infants,” *Nature*, vol. 415, no. 6873, pp. 755–755, 2002.
- [19] F. Warneken and M. Tomasello, “Helping and cooperation at 14 months of age,” *Infancy*, vol. 11, no. 3, pp. 271–294, 2007.
- [20] F. Warneken and M. Tomasello, “Altruistic helping in human infants and young chimpanzees,” *Science*, vol. 311, no. 5765, pp. 1301–1303, 2006.
- [21] M. Vazquez-Chanlatte, S. Jha, A. Tiwari, M. K. Ho, and S. Seshia, “Learning task specifications from demonstrations,” *Advances in Neural Information Processing Systems*, vol. 31, pp. 5367–5377, 2018.
- [22] G. Chou, N. Ozay, and D. Berenson, “Learning constraints from locally-optimal demonstrations under cost function uncertainty,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3682–3690, Apr. 2020.
- [23] L. Pais, K. Umezawa, Y. Nakamura, and A. Billard, “Learning robot skills through motion segmentation and constraints extraction,” *HRI Workshop on Collaborative Manipulation*, p. 5, 2013.

- [24] C. Li and D. Berenson, “Learning object orientation constraints and guiding constraints for narrow passages from one demonstration,” in *2016 International Symposium on Experimental Robotics*, ser. Springer Proceedings in Advanced Robotics, Cham, 2017, pp. 197–210.
- [25] H. Lin, M. Howard, and S. Vijayakumar, “Learning null space projections,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 2613–2619.
- [26] N. Mehr, R. Horowitz, and A. D. Dragan, “Inferring and assisting with constraints in shared autonomy,” in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016, pp. 6689–6696.
- [27] S. Malik, U. Anwar, A. Aghasi, and A. Ahmed, “Inverse constrained reinforcement learning,” in *International Conference on Machine Learning*, PMLR, 2021, pp. 7390–7399.
- [28] C. K. Liu and D. Negrut, “The role of physics-based simulators in robotics,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 35–58, May 3, 2021.
- [29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1707.06347>.
- [30] The garage contributors, *Garage: A toolkit for reproducible reinforcement learning research*, 2019. [Online]. Available: <https://github.com/rlworkgroup/garage>.
- [31] J. M. Wolfe, K. R. Kluender, D. M. Levi, *et al.*, *Sensation & perception*, Fourth. Sinauer Sunderland, MA, 2015.
- [32] A. Gopnik and A. N. Meltzoff, *Words, thoughts, and theories*. MIT Press, 1997.
- [33] Z. Wu, N. Dvornik, K. Greff, T. Kipf, and A. Garg, “Slotformer: Unsupervised visual dynamics simulation with object-centric models,” in *The Eleventh International Conference on Learning Representations*, 2023.
- [34] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal n-body collision avoidance,” in *Robotics Research: The 14th International Symposium ISRR*, Springer, 2011, pp. 3–19.
- [35] M. J. Vowels, N. C. Camgoz, and R. Bowden, “D’ya like dags? A survey on structure learning and causal discovery,” *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–36, 2022.
- [36] K. Greff, S. van Steenkiste, and J. Schmidhuber, *On the binding problem in artificial neural networks*, 2020. arXiv: 2012.05208 [cs.NE]. [Online]. Available: <https://arxiv.org/abs/2012.05208>.

- [37] F. Locatello, D. Weissenborn, T. Unterthiner, *et al.*, “Object-centric learning with slot attention,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 525–11 538, 2020.
- [38] R. Kabra, D. Zoran, G. Erdogan, *et al.*, “SIMONe: View-invariant, temporally-abstracted object representations via unsupervised video decomposition,” in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 20 146–20 159.
- [39] T. Kipf, G. F. Elsayed, A. Mahendran, *et al.*, “Conditional object-centric learning from video,” in *International Conference on Learning Representations*, 2021.
- [40] K. Greff, R. L. Kaufman, R. Kabra, *et al.*, “Multi-object representation learning with iterative variational inference,” in *International Conference on Machine Learning*, PMLR, 2019, pp. 2424–2433.
- [41] M. S. Sajjadi, D. Duckworth, A. Mahendran, *et al.*, “Object scene representation transformer,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 9512–9524, 2022.
- [42] G. Elsayed, A. Mahendran, S. van Steenkiste, K. Greff, M. C. Mozer, and T. Kipf, “SAVi++: Towards end-to-end object-centric learning from real-world videos,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 28 940–28 954, 2022.
- [43] G. Singh, Y.-F. Wu, and S. Ahn, “Simple unsupervised object-centric learning for complex and naturalistic videos,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 18 181–18 196, 2022.
- [44] D. Driess, F. Xia, M. S. Sajjadi, *et al.*, “Palm-e: An embodied multimodal language model,” in *International Conference on Machine Learning*, PMLR, 2023, pp. 8469–8488.
- [45] D. Chen and P. Krähenbühl, “Learning from all vehicles,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 17 222–17 231.
- [46] N. Watters, L. Matthey, C. P. Burgess, and A. Lerchner, *Spatial broadcast decoder: A simple architecture for learning disentangled representations in VAEs*, 2019. arXiv: 1901.07017 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1901.07017>.
- [47] I. Higgins, L. Matthey, A. Pal, *et al.*, “Beta-VAE: Learning basic visual concepts with a constrained variational framework,” in *International conference on learning representations*, 2017.
- [48] A. Goyal, A. Didolkar, N. R. Ke, *et al.*, “Neural production systems,” in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 25 673–25 687.
- [49] Y. Sun, S. Ma, R. Madaan, R. Bonatti, F. Huang, and A. Kapoor, “SMART: Self-supervised multi-task pretraining with control transformers,” in *The Eleventh International Conference on Learning Representations*, 2022.

- [50] P. Sun, H. Kretzschmar, X. Dotiwalla, *et al.*, “Scalability in perception for autonomous driving: Waymo Open Dataset,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.
- [51] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.
- [52] A. Kirillov, E. Mintun, N. Ravi, *et al.*, “Segment anything,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 4015–4026.
- [53] L. Karazija, I. Laina, and C. Rupprecht, “Clevrtex: A texture-rich benchmark for unsupervised multi-object segmentation,” in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.
- [54] K. Azadeh, R. De Koster, and D. Roy, “Robotized and automated warehouse systems: Review and recent developments,” *Transportation Science*, vol. 53, no. 4, pp. 917–945, 2019.
- [55] J. Demuyakor, “Ghana go digital agenda: The impact of zipline drone technology on digital emergency health delivery in ghana,” *Shanlax International Journal of Arts, Science and Humanities*, vol. 8, no. 1, pp. 242–253, 2020.
- [56] D. Lattanzi and G. Miller, “Review of robotic infrastructure inspection systems,” *Journal of Infrastructure Systems*, vol. 23, no. 3, 2017.
- [57] Y. Zhang, P. Tiño, A. Leonardis, and K. Tang, “A survey on neural network interpretability,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, no. 5, pp. 726–742, 2021.
- [58] T. Naselaris, K. N. Kay, S. Nishimoto, and J. L. Gallant, “Encoding and decoding in fMRI,” *Neuroimage*, vol. 56, no. 2, pp. 400–410, 2011.
- [59] A. O. Nunez-Elizalde, A. G. Huth, and J. L. Gallant, “Voxelwise encoding models with non-spherical multivariate normal priors,” *Neuroimage*, vol. 197, pp. 482–492, 2019.
- [60] M. Haghani, M. C. Bliemer, B. Farooq, *et al.*, “Applications of brain imaging methods in driving behaviour research,” *Accident Analysis & Prevention*, vol. 154, p. 106 093, 2021.
- [61] J. Gross, “Magnetoencephalography in cognitive neuroscience: A primer,” *Neuron*, vol. 104, no. 2, pp. 189–204, 2019.
- [62] T. A. Schweizer, K. Kan, Y. Hung, F. Tam, G. Naglie, and S. J. Graham, “Brain activity during driving with distraction: An immersive fMRI study,” *Frontiers in human neuroscience*, vol. 7, p. 53, 2013.
- [63] H. J. Spiers and E. A. Maguire, “Neural substrates of driving behaviour,” *Neuroimage*, vol. 36, no. 1, pp. 245–255, 2007.

- [64] J. Navarro, E. Reynaud, and F. Osiurak, “Neuroergonomics of car driving: A critical meta-analysis of neuroimaging data on the human brain behind the wheel,” *Neuroscience & Biobehavioral Reviews*, vol. 95, pp. 464–479, 2018.
- [65] C. Strong, K. Stocking, J. Li, T. Zhang, J. Gallant, and C. Tomlin, “A framework for evaluating human driver models using neuroimaging,” in *6th Annual Learning for Dynamics & Control Conference*, PMLR, 2024, pp. 1565–1578.
- [66] M. Carandini, J. B. Demb, V. Mante, *et al.*, “Do we know what the early visual system does?” *Journal of Neuroscience*, vol. 25, no. 46, pp. 10 577–10 597, 2005.
- [67] P. Agrawal, D. Stansbury, J. Malik, and J. L. Gallant, *Pixels to voxels: Modeling visual representation in the human brain*, 2014. arXiv: 1407.5104 [q-bio.NC]. [Online]. Available: <https://arxiv.org/abs/1407.5104>.
- [68] U. Güçlü and M. A. Van Gerven, “Deep neural networks reveal a gradient in the complexity of neural representations across the ventral stream,” *Journal of Neuroscience*, vol. 35, no. 27, pp. 10 005–10 014, 2015.
- [69] Y. Takagi and S. Nishimoto, “High-resolution image reconstruction with latent diffusion models from human brain activity,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 14 453–14 463.
- [70] S. Ogawa, T.-M. Lee, A. R. Kay, and D. W. Tank, “Brain magnetic resonance imaging with contrast dependent on blood oxygenation.,” *proceedings of the National Academy of Sciences*, vol. 87, no. 24, pp. 9868–9872, 1990.
- [71] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, “ERFNet: Efficient residual factorized convnet for real-time semantic segmentation,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2017.
- [72] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [73] P. Li, T. J. Hastie, and K. W. Church, “Very sparse random projections,” in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 287–296.
- [74] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [75] D. P. Woodruff *et al.*, “Sketching as a tool for numerical linear algebra,” *Foundations and Trends® in Theoretical Computer Science*, vol. 10, no. 1–2, pp. 1–157, 2014.
- [76] A. G. Huth, S. Nishimoto, A. T. Vu, and J. L. Gallant, “A continuous semantic space describes the representation of thousands of object and action categories across the human brain,” *Neuron*, vol. 76, no. 6, pp. 1210–1224, 2012.

- [77] A. G. Huth, W. A. De Heer, T. L. Griffiths, F. E. Theunissen, and J. L. Gallant, “Natural speech reveals the semantic maps that tile human cerebral cortex,” *Nature*, vol. 532, no. 7600, pp. 453–458, 2016.
- [78] F. Deniz, A. O. Nunez-Elizalde, A. G. Huth, and J. L. Gallant, “The representation of semantic information across human cerebral cortex during listening versus reading is invariant to stimulus modality,” *Journal of Neuroscience*, vol. 39, no. 39, pp. 7722–7736, 2019.
- [79] S. Nishimoto, A. T. Vu, T. Naselaris, Y. Benjamini, B. Yu, and J. L. Gallant, “Reconstructing visual experiences from brain activity evoked by natural movies,” *Current biology*, vol. 21, no. 19, pp. 1641–1646, 2011.
- [80] T. Dupré la Tour, M. Eickenberg, A. O. Nunez-Elizalde, and J. L. Gallant, “Feature-space selection with banded ridge regression,” *NeuroImage*, vol. 264, p. 119728, 2022.
- [81] J. W. Pratt, “Dividing the indivisible: Using simple symmetry to partition variance explained,” in *Proceedings of the second international Tampere conference in statistics, 1987*, Department of Mathematical Sciences, University of Tampere, 1987, pp. 245–260.
- [82] T. D. la Tour, M. Lu, M. Eickenberg, and J. L. Gallant, “A finer mapping of convolutional neural network layers to the visual cortex,” in *SVRHM 2021 Workshop@ NeurIPS*, 2021.
- [83] C. Grefkes, A. Ritzl, K. Zilles, and G. R. Fink, “Human medial intraparietal cortex subserves visuomotor coordinate transformation,” *Neuroimage*, vol. 23, no. 4, pp. 1494–1506, 2004.
- [84] C. Grefkes and G. R. Fink, “The functional organization of the intraparietal sulcus in humans and monkeys,” *Journal of anatomy*, vol. 207, no. 1, pp. 3–17, 2005.
- [85] P. E. Downing, Y. Jiang, M. Shuman, and N. Kanwisher, “A cortical area selective for visual processing of the human body,” *Science*, vol. 293, no. 5539, pp. 2470–2473, 2001.
- [86] S. V. Astafiev, C. M. Stanley, G. L. Shulman, and M. Corbetta, “Extrastriate body area in human occipital cortex responds to the performance of motor actions,” *Nature neuroscience*, vol. 7, no. 5, pp. 542–548, 2004.
- [87] N. Kanwisher and G. Yovel, “The fusiform face area: A cortical region specialized for the perception of faces,” *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 361, no. 1476, pp. 2109–2128, 2006.
- [88] M. C. Morrone, M. Tosetti, D. Montanaro, A. Fiorentini, G. Cioni, and D. Burr, “A cortical area that responds specifically to optic flow, revealed by fMRI,” *Nature neuroscience*, vol. 3, no. 12, pp. 1322–1328, 2000.

- [89] I. Gauthier, P. Skudlarski, J. C. Gore, and A. W. Anderson, “Expertise for cars and birds recruits brain areas involved in face recognition,” *Nature neuroscience*, vol. 3, no. 2, pp. 191–197, 2000.
- [90] M. J. Tarr and I. Gauthier, “FFA: A flexible fusiform area for subordinate-level visual processing automatized by expertise,” *Nature neuroscience*, vol. 3, no. 8, pp. 764–769, 2000.
- [91] D. A. Ross, B. J. Tamber-Rosenau, T. J. Palmeri, J. Zhang, Y. Xu, and I. Gauthier, “High-resolution functional magnetic resonance imaging reveals configural processing of cars in right anterior fusiform face area of car experts,” *Journal of cognitive neuroscience*, vol. 30, no. 7, pp. 973–984, 2018.
- [92] L. Peters, A. Bajcsy, C.-Y. Chiu, *et al.*, “Contingency games for multi-agent interaction,” *IEEE Robotics and Automation Letters*, 2024.
- [93] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, “End-to-end autonomous driving: Challenges and frontiers,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [94] K. Black, N. Brown, D. Driess, *et al.*, π_0 : A vision-language-action flow model for general robot control, 2024. arXiv: 2410.24164 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2410.24164>.
- [95] K. Pertsch, K. Stachowicz, B. Ichter, *et al.*, FAST: Efficient action tokenization for vision-language-action models, 2025. arXiv: 2501.09747 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2501.09747>.
- [96] S. Abeyruwan, J. Ainslie, J.-B. Alayrac, *et al.*, Gemini robotics: Bringing AI into the physical world, 2025. arXiv: 2503.20020 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2503.20020>.
- [97] C. Olsson, N. Elhage, N. Nanda, *et al.*, “In-context learning and induction heads,” *Transformer Circuits Thread*, 2022. [Online]. Available: <https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html>.
- [98] T. Bricken, A. Templeton, J. Batson, *et al.*, “Towards monosemanticity: Decomposing language models with dictionary learning,” *Transformer Circuits Thread*, 2023. [Online]. Available: <https://transformer-circuits.pub/2023/monosemantic-features/index.html>.
- [99] H. Cunningham, A. Ewart, L. Riggs, R. Huben, and L. Sharkey, Sparse autoencoders find highly interpretable features in language models, 2023. arXiv: 2309.08600 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2309.08600>.
- [100] S. Marks, J. Treutlein, T. Bricken, *et al.*, Auditing language models for hidden objectives, 2025. arXiv: 2503.10965 [cs.AI]. [Online]. Available: <https://arxiv.org/abs/2503.10965>.

- [101] M. Geva, A. Caciularu, K. R. Wang, and Y. Goldberg, *Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space*, 2022. arXiv: 2203.14680 [cs.CL]. [Online]. Available: <https://arxiv.org/abs/2203.14680>.
- [102] C. Olah, A. Mordvintsev, and L. Schubert, “Feature visualization,” *Distill*, 2017. [Online]. Available: <https://distill.pub/2017/feature-visualization>.
- [103] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, “Locating and editing factual associations in GPT,” *Advances in neural information processing systems*, vol. 35, pp. 17 359–17 372, 2022.
- [104] T.-H. Wang, W. Xiao, T. Seyde, R. Hasani, and D. Rus, “Measuring interpretability of neural policies of robots with disentangled representation,” in *Conference on Robot Learning*, PMLR, 2023, pp. 602–641.
- [105] S. Pohland and C. Tomlin, “Understanding the dependence of perception model competency on regions in an image,” in *World Conference on Explainable Artificial Intelligence*, Springer, 2024, pp. 130–154.
- [106] C. Glanois, P. Weng, M. Zimmer, *et al.*, “A survey on interpretable reinforcement learning,” *Machine Learning*, vol. 113, no. 8, pp. 5847–5890, 2024.
- [107] K. Park, Y. J. Choe, and V. Veitch, “The linear representation hypothesis and the geometry of large language models,” in *International Conference on Machine Learning*, PMLR, 2024, pp. 39 643–39 666.
- [108] M. Geva, R. Schuster, J. Berant, and O. Levy, *Transformer feed-forward layers are key-value memories*, 2021. arXiv: 2012.14913 [cs.CL]. [Online]. Available: <https://arxiv.org/abs/2012.14913>.
- [109] A. O’Neill, A. Rehman, A. Gupta, *et al.*, *Open X-Embodiment: Robotic learning datasets and RT-X models*, 2024. arXiv: 2310.08864 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2310.08864>.
- [110] A. Brohan, N. Brown, J. Carbajal, *et al.*, *RT-2: Vision-language-action models transfer web knowledge to robotic control*, 2023. arXiv: 2307.15818 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2307.15818>.
- [111] M. J. Kim, K. Pertsch, S. Karamcheti, *et al.*, “OpenVLA: An open-source vision-language-action model,” in *8th Annual Conference on Robot Learning*, 2024.
- [112] L. Beyer, A. Steiner, A. S. Pinto, *et al.*, *PaliGemma: A versatile 3B VLM for transfer*, 2024. arXiv: 2407.07726 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2407.07726>.
- [113] H. Touvron, L. Martin, K. Stone, *et al.*, *Llama 2: Open foundation and fine-tuned chat models*, 2023. arXiv: 2307.09288 [cs.CL]. [Online]. Available: <https://arxiv.org/abs/2307.09288>.
- [114] A. Khazatsky, K. Pertsch, S. Nair, *et al.*, “Droid: A large-scale in-the-wild robot manipulation dataset,” in *Robotics: Science and Systems*, 2024.

- [115] B. Liu, Y. Zhu, C. Gao, *et al.*, *Libero: Benchmarking knowledge transfer for lifelong robot learning*, 2023.
- [116] E. J. Hu, P. Wallis, Z. Allen-Zhu, *et al.*, “Lora: Low-rank adaptation of large language models,” in *International Conference on Learning Representations*, 2022.
- [117] E. Hernandez, B. Z. Li, and J. Andreas, “Inspecting and editing knowledge representations in language models,” in *First Conference on Language Modeling*, 2024.
- [118] A. Zou, L. Phan, S. Chen, *et al.*, *Representation engineering: A top-down approach to AI transparency*, 2025. arXiv: 2310.01405 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2310.01405>.
- [119] E. Ameisen, J. Lindsey, A. Pearce, *et al.*, “Circuit tracing: Revealing computational graphs in language models,” *Transformer Circuits Thread*, 2025. [Online]. Available: <https://transformer-circuits.pub/2025/attribution-graphs/methods.html>.
- [120] H. Kaur, M. R. Conrad, D. Rule, C. Lampe, and E. Gilbert, “Interpretability gone bad: The role of bounded rationality in how practitioners understand machine learning,” *Proceedings of the ACM on Human-Computer Interaction*, vol. 8, no. CSCW1, pp. 1–34, 2024.
- [121] D. Achlioptas, “Database-friendly random projections: Johnson-Lindenstrauss with binary coins,” *Journal of computer and System Sciences*, vol. 66, no. 4, pp. 671–687, 2003.
- [122] Y. Lu, P. Dhillon, D. P. Foster, and L. Ungar, “Faster ridge regression via the subsampled randomized hadamard transform,” *Advances in neural information processing systems*, vol. 26, 2013.
- [123] Y. C. Eldar and G. Kutyniok, *Compressed sensing: theory and applications*. Cambridge university press, 2012.
- [124] F. Bach, “Sharp analysis of low-rank kernel matrix approximations,” in *Conference on learning theory*, PMLR, 2013, pp. 185–209.
- [125] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [126] J. D. Power, B. M. Silver, M. R. Silverman, E. L. Ajodan, D. J. Bos, and R. M. Jones, “Customized head molds reduce motion during resting state fMRI scans,” *NeuroImage*, vol. 189, pp. 141–149, 2019.
- [127] M. Jenkinson and S. Smith, “The role of registration in functional magnetic resonance imaging,” *Medical Image Registration. CRC Press, New York*, pp. 183–198, 2001.
- [128] M. Jenkinson, P. Bannister, M. Brady, and S. Smith, “Improved optimization for the robust and accurate linear registration and motion correction of brain images,” *Neuroimage*, vol. 17, no. 2, pp. 825–841, 2002.

- [129] J. S. Gao, A. G. Huth, M. D. Lescroart, and J. L. Gallant, “Pycortex: An interactive surface visualizer for fmri,” *Frontiers in neuroinformatics*, vol. 9, p. 23, 2015.
- [130] Y. Behzadi, K. Restom, J. Liau, and T. T. Liu, “A component based noise correction method (compcor) for bold and perfusion based fMRI,” *Neuroimage*, vol. 37, no. 1, pp. 90–101, 2007.
- [131] K. A. Hansen, K. N. Kay, and J. L. Gallant, “Topographic organization in and near human visual area V4,” *Journal of Neuroscience*, vol. 27, no. 44, pp. 11 896–11 911, 2007.
- [132] N. Shazeer, *GLU variants improve transformer*, 2020. arXiv: 2002.05202 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2002.05202>.

Chapter 8

Appendices

8.1 Chapter 2 Appendix: Proofs of theorems

For simplicity, the proofs here assume a continuous state-action space, but similar results hold for the discrete setting. Recall that $q(\xi) = e^{\beta R(\xi)}$. Furthermore, define the normalized trajectory distributions as $p_0(\xi) = \frac{1}{Z_0}q(\xi)$ for the nominal unconstrained case and $p_c(\xi) = \frac{1}{Z_c}q(\xi)\mathbb{1}_c(\xi)$ for the constrained case.

Proof of Theorem 1:

$$\begin{aligned}\frac{Z_{c_i}}{Z_0} &= \frac{1}{Z_0} \int q(\xi) \mathbb{1}_{c_i}(\xi) d\xi \\ &= \int p_0(\xi) \mathbb{1}_{c_i}(\xi) d\xi \\ &= \mathbb{E}_{\xi \sim p_0}[\mathbb{1}_{c_i}(\xi)]\end{aligned}\tag{8.1}$$

Note that the expression for Z_{c_i} comes from removing trajectories that violate c_i from the integral, so that the constrained probability distribution p_{c_i} integrates to 1.

Proof of Theorem 2:

First, define $q_d(\xi) = e^{\beta_d R(\xi)}$, $q_s(\xi) = e^{\beta_s R(\xi)}$, and $p_{s,c}(\xi) = \frac{1}{Z_{s,c}} q_s(\xi) \mathbf{1}_c(\xi)$. We have:

$$\begin{aligned}
\frac{Z_{d,c}}{Z_{s,c}} &= \frac{1}{Z_{s,c}} \int q_d(\xi) \mathbf{1}_c(\xi) d\xi \\
&= \frac{1}{Z_{s,c}} \int \frac{q_d(\xi)}{p_{s,c}(\xi)} p_{s,c}(\xi) \mathbf{1}_c(\xi) d\xi \\
&= \frac{1}{Z_{s,c}} \int \frac{e^{\beta_d R(\xi)}}{e^{\beta_s R(\xi)} / Z_{s,c}} p_{s,c}(\xi) \mathbf{1}_c(\xi) d\xi \\
&= \int e^{(\beta_d - \beta_s) R(\xi)} p_{s,c}(\xi) d\xi \\
&= \mathbb{E}_{\xi \sim p_{s,c}} [e^{(\beta_d - \beta_s) R(\xi)}]
\end{aligned} \tag{8.2}$$

The constraint avoidance indicator $\mathbf{1}_c(\xi)$ disappears because it is redundant with $p_{s,c}(\xi)$.

8.2 Chapter 4 Appendix: Additional architecture details

Hyperparameters for our model are shown in table 8.1. The metrics in table 4.1 are reported after 2e5 training steps. We found that minor improvements in segmentation quality continued after this point, and qualitative figures in the paper are from a model trained for up to 6e5 steps. The transformer used for decoding in the waypoint prediction task used the same hyperparameters as the main model transformer, except with 2 layers instead of 6.

Table 8.1: Model hyperparameters used in our experiments.

Parameter	Value
$\mathcal{L}_{KL}$ weight, β (with waypoint pred. task)	5e-7
β (without waypoint pred. task)	4.5e-7
$\mathcal{L}_{task}$ weight, ω_{task}	1e-4
Pixel likelihood scale, σ_x	0.08
Number of object slots	21
Object latent dim.	32
Transformer num. layers	6
Transformer num. heads	4
Transformer feature dim.	512
Transformer feedforward dim.	1024
Reconstruction MLP num. layers	3
Reconstruction MLP hidden dim.	1536
Image sequence length	8
Image size (H, W)	(96, 224)
Pixels decoded per training sequence, N	2016
Batch size	8
Learning rate	1e-4

8.3 Chapter 5 Appendix A: Risk analysis of ridge regression under sparse random projection

In this section, we characterize how a sparse random projection of the data matrix X affects the regression error in ridge regression. Our analysis is based on combining the random projection matrix analysis in [121] and the ridge regression analysis technique in [122].

Let $X \in \mathbb{R}^{n \times d}$ be the data matrix containing n i.i.d. samples from a d -dimensional independent random variable x such that $d \gg n$. We denote by $\mathcal{X}$ the set of these sampled vectors (rows of X). $Y \in \mathbb{R}^{n \times 1}$ is the real valued response vector. $\mu \in \mathbb{R}^{n \times 1}$ is the Homoscedastic noise vector with zero mean and common variance σ^2 . Let $\beta \in \mathbb{R}^d$ be the parameter vector, and we have $Y = X\beta + \mu$. We infer the parameter vector $\hat{\beta}_\lambda \in \mathbb{R}^{d \times 1}$ by solving the ridge regression optimization problem:

$$\hat{\beta}_\lambda = \arg \min_{\beta \in \mathbb{R}^{d \times 1}} \frac{1}{n} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \quad (8.3)$$

where $\hat{\beta}_\lambda = (X^\top X + n\lambda I_d)^{-1} X^\top Y$. Since inverting $(X^\top X + n\lambda I_d) \in \mathbb{R}^{d \times d}$ is expensive, alternatively we can consider the dual formulation of the ridge regression by first defining its kernel matrix $\mathbf{K} = XX^\top$:

$$\hat{\alpha}_\lambda = \arg \min_{\alpha \in \mathbb{R}^{n \times 1}} \frac{1}{n} \|Y - \mathbf{K}\alpha\|_2^2 + \lambda \alpha^\top \mathbf{K}\alpha \quad (8.4)$$

The optimal solution is $\hat{\alpha}_\lambda = (\mathbf{K} + n\lambda I_n)^{-1} Y$ (where inverting $(\mathbf{K} + n\lambda I_n) \in \mathbb{R}^{n \times 1}$ is a cheaper operation for $d \gg n$). $\hat{\beta}_\lambda$ can then be recovered using $\hat{\beta}_\lambda = X^\top \hat{\alpha}_\lambda$.

Let $\beta > 0$ and $0 < \epsilon < \frac{3}{2}$. Pick $k \geq \frac{4+2\beta}{\epsilon^2/2-\epsilon^3/3} \log(n)$. Define the projection matrix $\Theta \in \mathbb{R}^{k \times d}$ as in [121].

$$\Theta_{ij} = \sqrt{\frac{3}{k}} \times \begin{cases} +1 & \text{with probability } 1/6 \\ 0 & \text{with probability } 2/3 \\ -1 & \text{with probability } 1/6 \end{cases} \quad (8.5)$$

Theorem 1.1 in [121] shows that, with probability at least $1 - n^{-\frac{1}{\beta}}$,

$$(1 - \epsilon) \|u - v\|_2^2 \leq \|\Theta u - \Theta v\|_2^2 \leq (1 + \epsilon) \|u - v\|_2^2, \quad \forall u, v \in \mathcal{X} \quad (8.6)$$

This implies that, with probability at least $1 - n^{-\frac{1}{\beta}}$, the singular values of Θ can be bounded as follows [123, Ch.5]:

$$(1 - \epsilon) \leq \sigma_{\min}(\Theta) \leq \sigma_{\max}(\Theta) \leq (1 + \epsilon) \quad (8.7)$$

Furthermore, we can characterize the projected kernel matrix $(X\Theta^\top)(\Theta X^\top)$ with the following result.

Lemma 1. Let $\beta > 0$ and $0 < \epsilon < \frac{3}{2}$. Pick $k \geq \frac{4+2\beta}{\epsilon^2/2-\epsilon^3/3} \log(n)$. Define $\Theta \in \mathbb{R}^{k \times d}$ as in (8.5), then with probability at least $1 - n^{-\frac{1}{\beta}}$,

$$\|\Theta^\top \Theta - I_d\|_2 \leq \epsilon \quad (8.8)$$

Proof. From Theorem 1.1 in [121] and the singular value bounds on the random projection matrix in Chapter 5 of [123], we have with probability at least $1 - n^{-\frac{1}{\beta}}$,

$$\|\Theta^\top \Theta - I_d\|_2 \leq \sigma_{\max}(\Theta) - 1 \leq \epsilon \quad (8.9)$$

□

Define $\mathbf{K} := XX^\top$, and $\mathbf{K}_\Theta := (X\Theta^\top)\Theta X^\top$.

Corollary 1. Assuming the same conditions as in Lemma 1, we have, with probability at least $1 - n^{-\frac{1}{\beta}}$,

$$(1 - \epsilon)\mathbf{K} \preceq \mathbf{K}_\Theta \preceq (1 + \epsilon)\mathbf{K} \quad (8.10)$$

Let $Z = \mathbb{E}_\mu(Y) = X\beta$. Under the fixed design setting [122], the risk of a prediction $\hat{Y}$ is $\frac{1}{n} \mathbb{E}_\mu \|\hat{Y} - Z\|_2^2$. For any positive semidefinite matrix $M \in \mathbb{R}^{n \times n}$, let $\hat{Y}_M = X\hat{\beta}_\lambda(M)$ be the prediction obtained by solving the ridge regression problem (8.4) using M in place of $\mathbf{K}$. It turns out that the risk of this prediction can be expressed using the following function R of M [122]:

$$R(M) := \frac{\sigma^2}{n} \text{Tr}(M^2(M + n\lambda I_n)^{-2}) + n\lambda^2 Z^\top (M + n\lambda I_n)^{-2} Z \quad (8.11)$$

Note that the expectation in the risk function is only over the random noise μ .

Similar to Theorem 1 in [122], we have the following result:

Proposition 1. Let $\beta > 0$ and $0 < \epsilon < \frac{3}{2}$. Pick $k \geq \frac{4+2\beta}{\epsilon^2(1/2-\epsilon/3)} \log(n)$. Consider a random projection matrix $\Theta \in \mathbb{R}^{d \times k}$ defined as in (8.5). With probability at least $1 - n^{-\frac{1}{\beta}}$,

$$R(\mathbf{K}_\Theta) \leq (1 - \epsilon)^{-2} R(\mathbf{K}) \quad (8.12)$$

Proof. For the symmetric positive semi-definite matrix $\mathbf{K}_\Theta$, from [124], we have that the function

$$B(\mathbf{K}_\Theta) := n\lambda^2 Z^\top (\mathbf{K}_\Theta + n\lambda I_n)^{-2} Z \quad (8.13)$$

is non-increasing in $\mathbf{K}_\Theta$, and the function

$$V(\mathbf{K}_\Theta) := \frac{\sigma^2}{n} \text{Tr}(\mathbf{K}_\Theta^2 (\mathbf{K}_\Theta + n\lambda I_n)^{-2}) \quad (8.14)$$

is non-decreasing in $\mathbf{K}_\Theta$. Note that $R(\mathbf{K}) = B(\mathbf{K}) + V(\mathbf{K})$. By Corollary 1, we have

$$\begin{aligned} R(\mathbf{K}_\Theta) &= V(\mathbf{K}_\Theta) + B(\mathbf{K}_\Theta) \\ &\leq V((1 + \epsilon)\mathbf{K}) + B((1 - \epsilon)\mathbf{K}) \\ &\leq (1 + \epsilon)^2 V(\mathbf{K}) + (1 - \epsilon)^{-2} B(\mathbf{K}) \\ &\leq (1 - \epsilon)^{-2} R(\mathbf{K}) \end{aligned} \quad (8.15)$$

□

Proposition 1 shows that the risk of ridge regression under a sparse random projection can be upper bounded as a function of ϵ with high probability (at least $1 - n^{-\frac{1}{\beta}}$). However, there are still two differences between the methodology in our work and the above analysis. First, we use modified probabilities for the entries of Θ that enable faster projections at the cost of a small additional approximation error, as described in [73], which is important for efficiently extracting features from a large number of samples. Second, we use banded ridge regression (which is equivalent to multiple-kernel ridge regression [80]) instead of regular ridge regression. We leave an extension of our analysis to multiple-kernel ridge regression to future work.

8.4 Chapter 5 Appendix B: Additional figures

In this section, we first present an additional experiment that compares the explainable variance using the encoding model derived from LAV features with one derived from features from a standard image classification CNN (see Figure 8.1). For this experiment, we used an AlexNet CNN architecture [125]. AlexNet received the same RGB inputs as the LAV DNN, but no lidar inputs or destination waypoint. The four RGB images at each frame were used as separate inputs to AlexNet, and then the activations corresponding to each image were concatenated before performing dimensionality reduction with a sparse random projection with $K = 20000$ components. We find that LAV explains more variance across the brain than AlexNet, supporting the hypothesis that the additional control-relevant features in LAV are genuinely useful for explaining brain activity in the active driving task above and beyond what is possible with passive vision features alone.

Furthermore, we present additional plots that highlight the inter-subject variance across all 5 subjects for best-module visualization (Figure 8.2) and each individual module (Figure 8.3). While there are notable differences between individual subjects, there are clear consistent trends in which brain areas are best explained by each LAV module.

8.5 Chapter 5 Appendix C: More details on methodology

Scanning procedure

BOLD activity were acquired on a 3T Siemens Trio with a 32-channel head coil, located at [redacted], with a T2*-weighted gradient-echo EPI sequence customized with a water-excitation radiofrequency pulse to prevent contamination from fat signal (TR = 2.0045 s, echo time = 35 ms, flip angle = 74°, voxel size = $2.24 \times 2.24 \times 3.5$ mm³, field of view = 224×224 mm², matrix size = 100×100 , and 30 axial slices to cover the entire cortex).

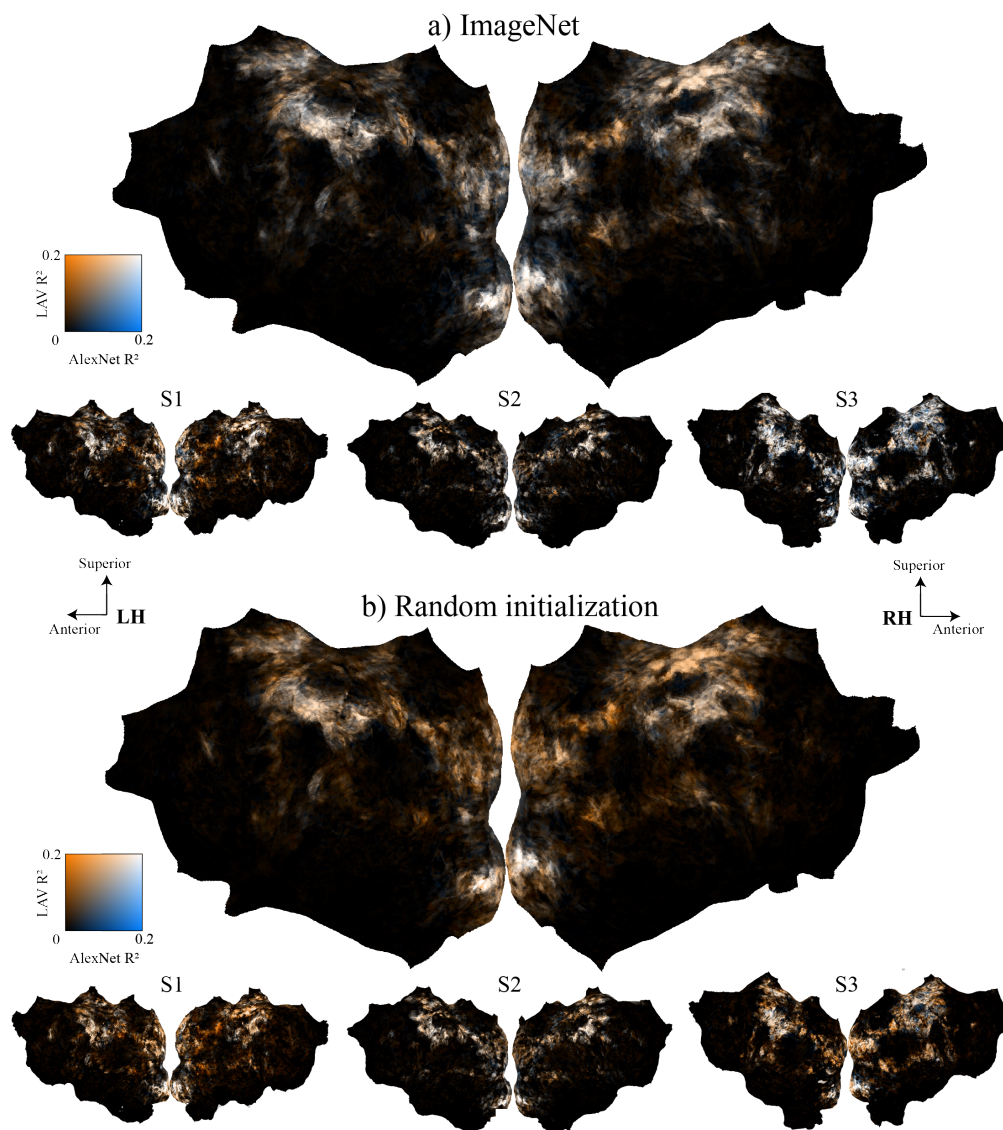


Figure 8.1: **Features from the LAV DNN explain more variance across the brain than an image classification CNN.** In each subplot, voxels are colored based on the explained variance (R^2) of the model derived from LAV DNN activations (orange) and that of a model derived from the activations of an AlexNet CNN. a): Comparison between LAV and an AlexNet CNN trained on ImageNet (using the best weights in the `torchvision` library) at the group level for subjects S1, S2, and S3 (top) and for individual subjects (below). LAV explains more variance across the brain, especially in high-level vision areas. b): Comparison between LAV and an untrained AlexNet CNN with random weight initializations at the group level (top) and for individual subjects (below). LAV explains more variance across the brain.

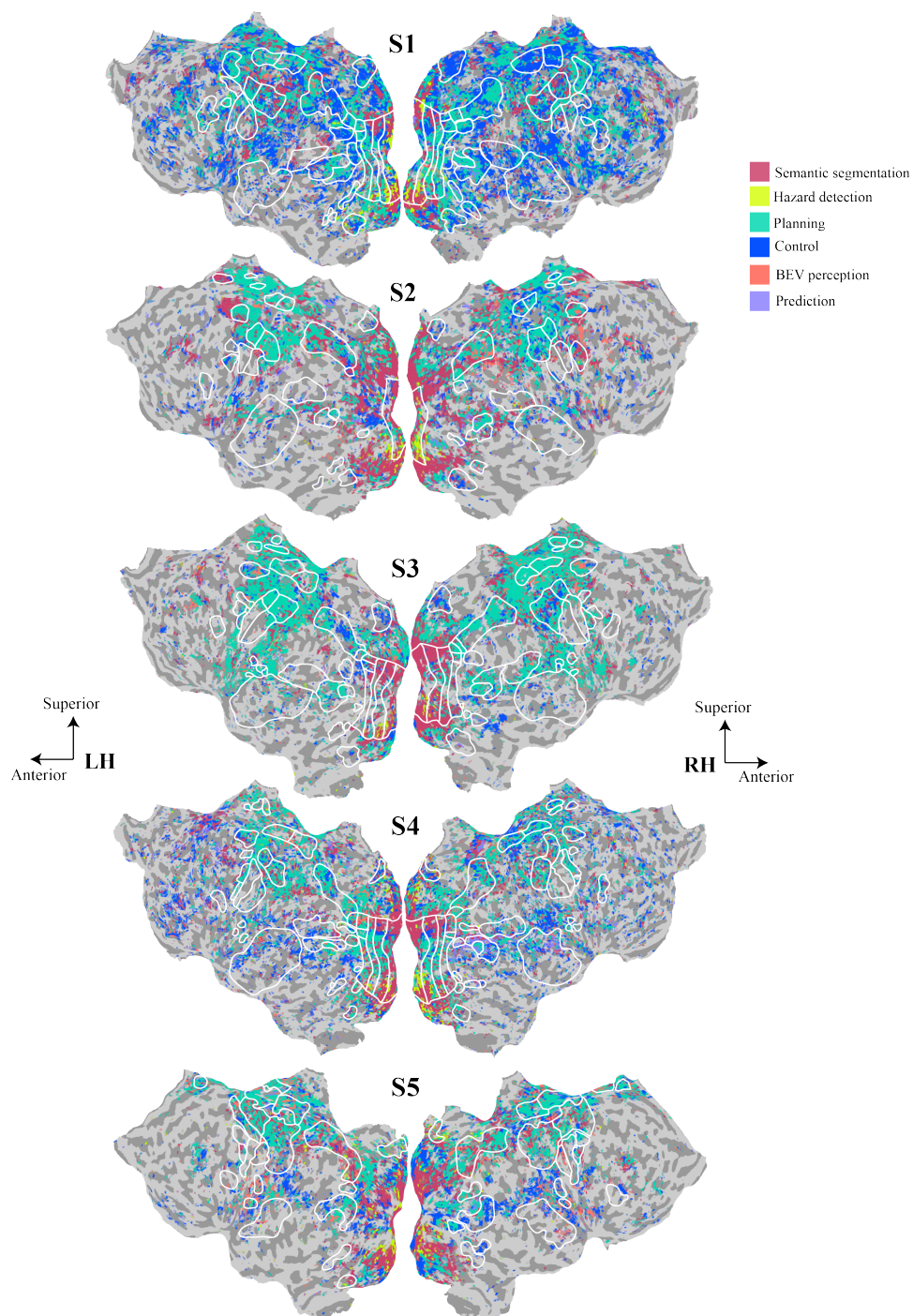


Figure 8.2: Subject-level figures corresponding to the group-level figures in Fig.5.3b.

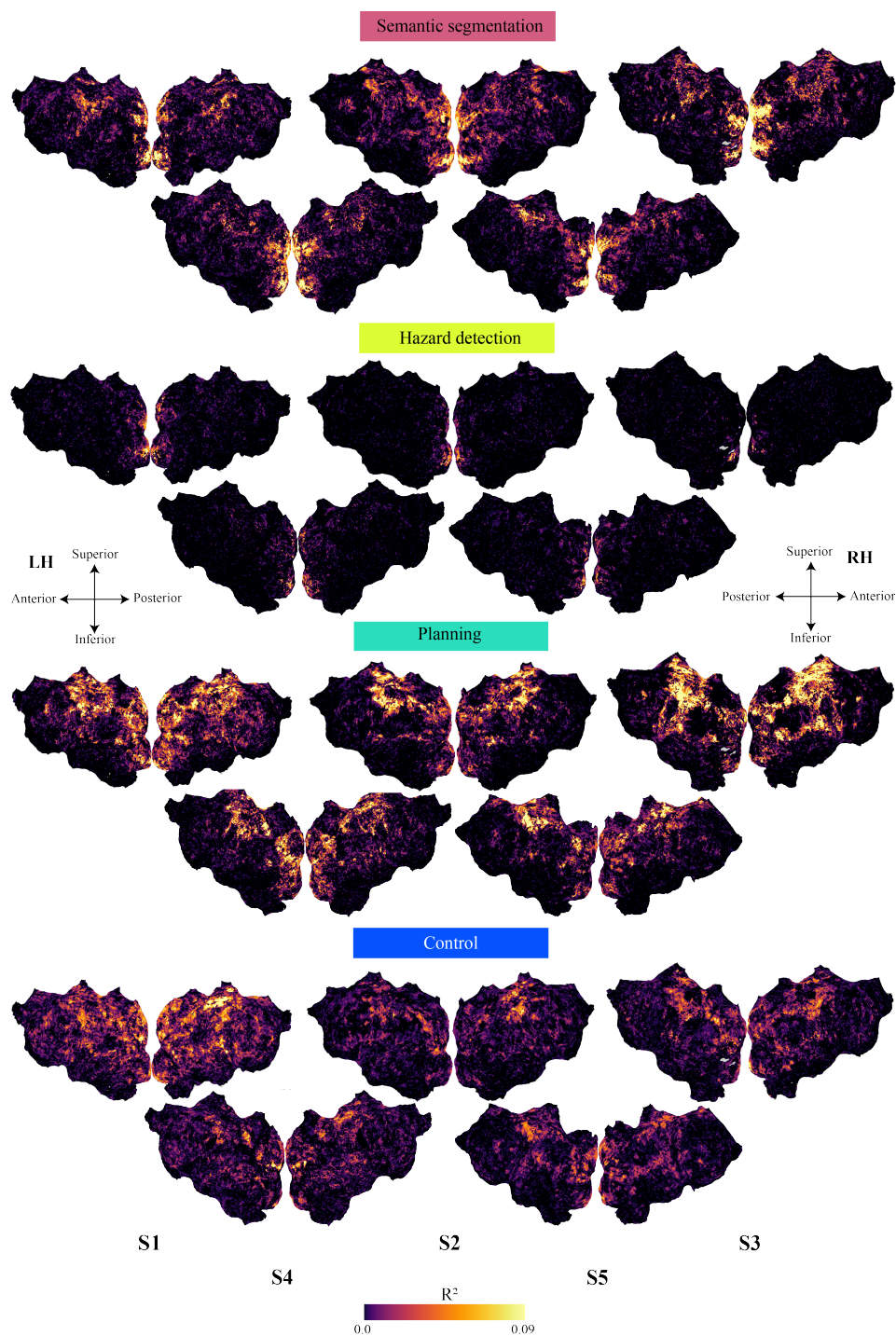


Figure 8.3: Subject-level figures corresponding to the group-level plot in Fig.5.3c.

Custom personalized headcases (caseforge, [126]) were used to stabilize the head and to reduce motion artifacts. Anatomical data were also collected to reconstruct the cortical surface (three-dimensional T1-weighted MP-RAGE sequence, $1 \times 1 \times 1 \text{ mm}^3$ voxel size and $256 \times 212 \times 256 \text{ mm}^3$ field of view).

Data were collected across multiple scanning sessions. Each session consisted of six 11-minute functional runs. Pilot experiments showed that good voxelwise models could be fit with about two hours of data, so at least two hours of data were recorded from each participant (3 hours each from subjects 1 and 2 and 2 hours from subject 3). Respiration and heart rate were recorded using a BIOPAC MP150 system (BIOPAC Systems, Inc.). Gaze location were recorded using an Avotec dark-pupil IR eyetracker at 60 Hz.

fMRI data preprocessing

Each functional run was first motion-corrected using the FMRIB Linear Image Registration Tool from FSL 5.0 [127], [128]. Next, functional images were unwarped by applying FUGUE from FSL to fieldmaps collected between functional runs. All volumes in the run were then averaged across time to obtain a high-quality template volume. Data across multiple sessions and runs were then registered to the template volume from the first session. Pycortex [129] was then used to register the functional data to the anatomical surface. Alignment was checked manually and adjusted as necessary to improve accuracy. Low-frequency voxel response drift was identified using COMPCOR [130] and removed from the signal. Finally, physiological signals from respiration and heartbeats were regressed out. Voxel activity in each 11-minute run was z-scored separately; that is, within each run, the mean response for each voxel was subtracted and the remaining response was scaled to have unit variance. To remove confounds from the eyetracking calibration sequence and detrending artifacts, the first 35 and last 5 TRs were then discarded from each run.

Localizers for known functional regions

Separately from the functional data collection, five sets of localizer data were collected from each subject. These included a retinotopic localizer used to delineate V1, V2, V3, V4, V3A, V3B, and V7 [131], a MT localizer to delineate the human middle temporal complex (hMT+), a visual category localizer to delineate the fusiform face area (FFA), the extrastriate body area (EBA), the parahippocampal place area (PPA), the occipital place area (OPA), and the retrosplenial cortex (RSC), a motor localizer to delineate the primary motor and somatosensory areas for hands (M1H, S1H), feet (M1F, S1F), and mouth (M1M, S1M), and also the intraparietal sulcus (IPS), frontal eye fields (FEF), the frontal operculum (FO), the superior ventral premotor speech area (sPMv), and Broca's area.

Statistical tests

Permutation tests were used to establish statistical significance for per-voxel encoding model performances. Model predictions were permuted 1,000 times to establish a null distribution for model performance. Significant voxels were then selected at the Benjamini-Hochberg FDR-corrected $p < 0.01$ threshold.

The Moran’s I statistic was used to quantify the spatial autocorrelation of the best-performing LAV DNN modules across the brain. The weight matrix was set so that the i, j th index was 1 for voxels adjacent to or diagonal from each other in 3D space, and 0 otherwise. The best performing modules at each voxel were permuted 1,000 times to establish a null distribution for the significance of the statistic. A threshold of $p < 0.01$ was used to determine significance.

Computational resources

There are two computationally demanding components of our data processing pipeline: 1) feature extraction from the LAV DNN and 2) regression of extracted LAV features against fMRI measurements. We report computational resources and requirements for each component separately.

Feature extraction: We run inference with the LAV DNN and multiply the extracted features by a sparse random projection matrix on the GPU before copying the reduced-dimension features to CPU for saving to disk. This requires about 8.5 GB of VRAM and 15GB of CPU RAM, and runs at >6 fps on a single NVIDIA RTX 4090 GPU. Across all 5 subjects, feature extraction required approximately 42 hours in total on a single NVIDIA RTX 4090 GPU. During development, we experimented with other dimensionality reduction techniques that required other rounds of feature extraction, although we typically did this only for a single subject. We estimate that this brought the total time for feature extraction throughout our experiments to around 10x, or 400 hours on a single GPU.

Banded ridge regression: For banded ridge regression, we use the Himalaya Python package [80]. The most computationally intensive step in the regression is using cross-validation to find the optimal regularization hyperparameters λ for each feature space and voxel - we sample 10,000 hyperparameters before selecting the best-performing ones for the final regression result. This takes around 24 hours using 20GB of VRAM and 45GB of CPU RAM for a single subject on a single NVIDIA RTX A6000 GPU, or 5 days across all subjects. Because the hyperparameter selection is batched, different batch parameters could reduce the amount of required VRAM at the cost of a longer regression time. Since we ran a number of regressions during the development of experiments, we estimate that the total time for regressions was around 40 GPU-days on NVIDIA RTX A6000 GPUs.

8.6 Chapter 6 Appendix A: More details on steering interventions

Let $x \in \mathbb{R}^n$ be the residual input to a transformer FFN. As shown in Equation 6.2, the FFN output is a sum of fixed value vectors $w_\theta^{(i)} \in \mathbb{R}^n$ weighted by input-dependent activations $f_\theta(x) \in \mathbb{R}^m$. Motivated by evidence that individual FFN neurons encode semantic features [98], [108], we override a subset $\mathcal{S} \subseteq \{1, \dots, m\}$ of activations using a fixed scalar $\alpha \in \mathbb{R}$. Each $\mathcal{S}$ corresponds to an interpretable neuron cluster aligned with a control concept - e.g., *fast*, *up*, *careful* - identified by grouping neurons with similar token projections (Figure 6.1, steps 1–3), either manually or via kNN over semantic embeddings.

$$\tilde{f}_\theta^{(i)}(x) = \begin{cases} \alpha & \text{if } i \in \mathcal{S} \\ [f_\theta(x)]_i & \text{otherwise} \end{cases} \quad (8.16)$$

$$\text{FFN}_{\text{steered}}(x) = \sum_{i=1}^m \tilde{f}_\theta^{(i)}(x) \cdot w_\theta^{(i)} \quad (8.17)$$

This induces a residual shift $\Delta x = \text{FFN}_{\text{steered}}(x) - \text{FFN}(x)$ that propagates through the transformer and modulates the final VLA action token distribution.

Implementation

In both π_0 and OPENVLA, the FFN layers use a GEGLU activation function [132] which has the form $[f_\theta(x)]_i = [\text{GELU}(W_1 x)]_i \cdot [W_2 x]_i$, where $W_1, W_2 \in \mathbb{R}^{m \times n}$ are additional parameter matrices. To implement the steering intervention, for π_0 (JAX), the neuron indices $\mathcal{S}$ and activation coefficient α are passed into modified FFN code that implements equation 8.16. For OPENVLA (PyTorch), we apply a forward hook on the FFN’s `gate_proj` to overwrite activations, which is equivalent to the following:

$$\tilde{f}_\theta^{(i)}(x) = \begin{cases} \text{GELU}(\alpha) \cdot [W_2 x]_i & \text{if } i \in \mathcal{S} \\ [f_\theta(x)]_i & \text{otherwise} \end{cases} \quad (8.18)$$

This modified intervention was more computationally efficient with PyTorch’s forward hook functionality, and we found that it performed similarly to the intervention defined in equation 8.16 during our exploratory experiments.

8.7 Chapter 6 Appendix B: Interpretability Experiments

Patterns in Value Vectors

Following [101], we classified a value vector as meaningful if 4 of its top 30 predicted tokens followed a common pattern. Furthermore, we classified patterns as “non-semantic” if they could be explained by a shallow syntax-level rule, such as words starting with the same 3 letters, and “semantic” otherwise. If multiple different patterns were found, we selected the one containing the highest-ranked token for determining the classification of the vector. Some examples of vectors from each category are show in table 8.2a (PaliGemma VLM) and 8.2b (π_0 VLA). Both models have 18 layers.

Layer	Pattern type	Pattern de- scription	Top 10 Tokens
3	None	N/A	strick, campa, laun, increa, accla, reluct, embra, exem, desir, fath
6	Non-semantic	Camel case variable names	<bos>, EndGlobalSection, Clik, expandindo, SourceChecksum, NSCoder, EditorBrowsable, UnusedPrivate, nahilalakip, ScopeManager
6	Semantic	Image-sourcing websites	pixabay, gettyimages, shutterstock, unsplash, uefa, Ferdin, plak, makro, Hæ, Lombar
14	Semantic	Mice and keyboards	mouse, keyboard, keys, KEY, keyboard, Mouse, keys, Mouse, mouse, clavier

(a) Example value vectors and their top tokens from the PaliGemma VLM.

Layer	Pattern type	Pattern de- scription	Top 10 Tokens
1	Non-semantic	Action tokens	<Ac0701>, <Ac0689>, <Ac0706>, <Ac0695>, <Ac0714>, <Ac0719>, <Ac0684>, <Ac0688>, <Ac0305>, <Ac0334>
14	None	N/A	maneu, shenan, affor, impra, increa, encomp, fortn, accla, philanth, volunte
15	Semantic	Strong	tough, strength,tough, toughness, toughest, tougher, strong,strength, strongest,strong
18	Semantic	Given names	javier, jorge, alberto, felipe, Áng, ricardo, eduardo, roberto, sergio, fernando

(b) Example value vectors and their top tokens from the π_0 VLA.

DROID Fine-tuning

To understand whether the contents of the task instructions in the DROID training dataset affected the value vectors of the model checkpoint fine-tuned on DROID data, we examined whether common instruction tokens became more common in the value vectors after fine-tuning. Specifically, we first tokenized all task instructions (including multiple alternative instructions for the same demonstration where available) using the PaliGemma tokenizer. We then took the top 200 most frequent instruction tokens and examined the number of times they appeared in the top-100 tokens for each value vector in the π_0 -FAST and π_0 -FAST-DROID-finetune model checkpoints. We assign a z-score to each token based on a two-proportion z-test and plot the results in Figure 8.4. A positive z-score indicates the token is more common in the value vectors of π_0 -FAST-DROID-finetune compared to π_0 -FAST, while a negative score indicates that it is more rare in π_0 -FAST-DROID-finetune. We find that the majority of the top instruction tokens are indeed more common in the π_0 -FAST-DROID-finetune value vectors, with a mean z-score of 1.0 and a mean count ratio of 1.2. While some tokens do have negative z-scores, we hypothesize that these tokens may be even more common in the base π_0 training data instructions, which unfortunately we do not have access to. Overall, these results suggest that semantic instruction inputs during VLA training may contribute to these tokens being retained in the value vectors, despite the VLA model being trained to only output action tokens.

8.8 Chapter 6 Appendix C: Simulation Experiments

KNN for Temporal Depth Interventions

1. Semantic Embedding Construction. To assign meaning to each value vector, we project it into the model’s output token space using the language modeling head, as described in Section 6.3. For each value vector $w_\theta^{(i)}$, we compute token logits and identify the top-5 tokens it most strongly activates. We then construct a semantic embedding by computing the softmax-weighted average of those token embeddings:

$$\mathbf{e}_{\text{sem}}^{(i)} = \sum_{j \in t_i} \text{softmax}(l_j^{(i)}) \cdot W_j$$

where t_i are the top token indices for value vector $w_\theta^{(i)}$, $l_j^{(i)}$ is the logit that $w_\theta^{(i)}$ assigns to token j , and W_j is the output embedding vector corresponding to token j . This yields a normalized, interpretable embedding for each value vector.

2. Depth Partitioning. The full OPENVLA model contains 352,255 value vectors across all transformer layers. To analyze temporal structure, we partition them: the first half (176,128 vectors) represents *early-layer* vectors, while the second half represents *late-layer* vectors.

3. kNN Clustering. We apply cosine-based k-nearest neighbor (kNN) clustering to these semantic embeddings using the cuML GPU-accelerated library. Each value vector is

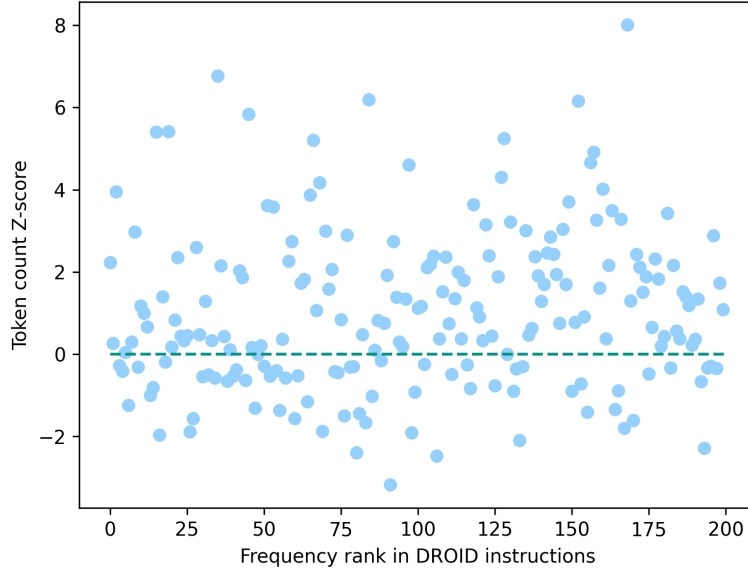


Figure 8.4: **Common DROID instruction tokens become more common in the value vectors of a model fine-tuned on the DROID dataset.** The majority of the top-200 instruction tokens are more common in the value vectors of the π_0 -FAST-DROID-finetune checkpoint, corresponding to a positive z-score (lying above the teal dashed line).

assigned a cluster based on its k nearest neighbors ($k \in \{10, 20, 40\}$). For each cluster, we compute a semantic centroid by averaging its members’ embeddings.

4. Concept-Aligned Cluster Selection. To find clusters aligned with a target concept (e.g., “up”), we tokenize the concept word/phrase using the model’s tokenizer, embed it using the language modeling head, and compare it to all cluster centroids using cosine similarity. The cluster with the highest similarity is selected for activation.

5. Temporal Partitioning for Layer-Specific Interventions. To isolate temporal effects, the above clustering pipeline was run independently on the *early-layer* and *late-layer* vector subsets. This ensured that the selected clusters were sourced exclusively from the intended depth region. For full-model interventions, clustering was performed on the full set of value vectors.

8.9 Chapter 6 Appendix D: Hardware Experiments

Data Collection Details

During data collection and evaluation, objects for both tasks are placed within a consistent area with slight variations in position. In both tasks, the robot begins from the same initial

Vector	Top 5 Tokens
153784	up, [action: 0.428], Up, oks, cker
263055	up, Up, Up, up, pto
276423	up, coming, Up, up, forth
225616	up, left, Up, up, Up
274043	up, Up, Up, up, UP
285310	up, Up, Up, up, UP
287766	up, Up, Up, up, UP
287383	up, Up, Up, up, UP
263704	Up, up, Up, up, Down
323343	up, Up, Up, up, UP

Figure 8.5: Top-5 token projections for the most semantically aligned “**up**” clusters from the full model using kNN clustering with $k = 10$.

pose with a small random deviation in position. During data collection, the end-effector is controlled in Cartesian space using a joystick. Data is recorded at 10 Hz, including images from the wrist-mounted and scene cameras, arm joint positions, gripper position, and the corresponding target joint and gripper commands at each timestep. Example images of both cameras are shown in Figure 8.7.

For *Low/High Transport*, we collect 75 episodes in total—25 each where the robot transports the toy penguin at a low, medium, or high height. Each episode is labeled with the prompt, “place penguin in basket”. For *Slow/Fast Transport*, we collect 20 episodes each with a slow, medium, or fast speed and placing the seal on either a blue or white plate, totaling 120 episodes. Each episode is labeled either with the prompt “place seal on blue plate” or “place seal on white plate” depending on which plate color the seal is placed on.

Finetuning π_0 -FAST

We use LoRA for fine-tuning π_0 -FAST and adopt the default training configuration provided by the official π_0 -FAST codebase¹¹. Fine-tuning is performed separately for each task, resulting in two distinct models—one for the *Low/High Transport* task and one for the *Slow/Fast Transport* task. For both tasks, the model input consists of the UR5’s six joint positions, the gripper position, and both scene and wrist camera images. The model outputs an action chunk as a sequence of joint and gripper positions. We fine-tune each model for 5000 steps using a batch size of 32 and an action chunk size of 10. We use preset normalization values provided by π_0 -FAST specifically for the UR5e robot arm.

¹¹<https://github.com/Physical-Intelligence/openpi>

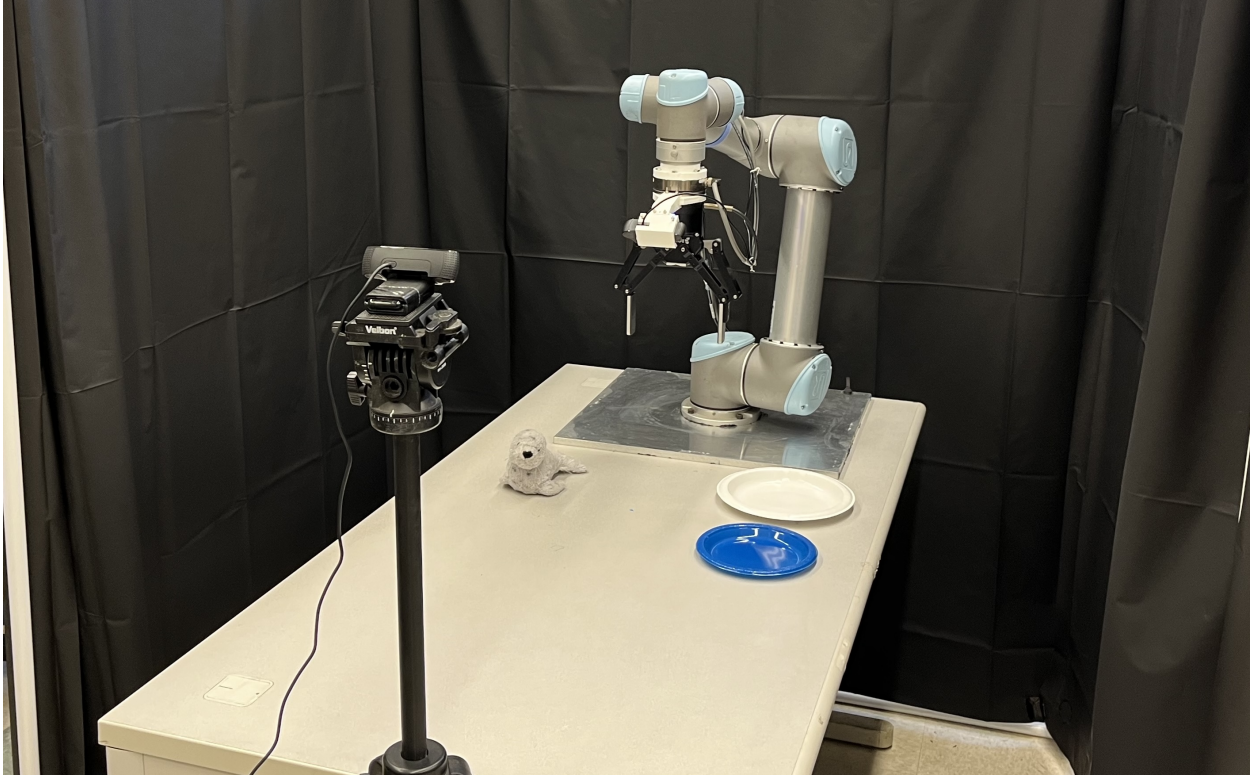


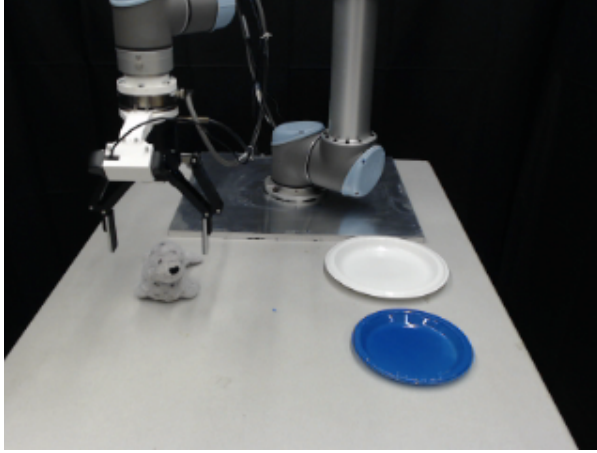
Figure 8.6: **Robot Setup:** Our hardware experiments use a UR5 robot arm equipped with a Robotiq 2F-140 gripper. The setup includes two cameras: a static scene camera overlooking the workspace and a wrist-mounted camera facing the gripper.

Steering Intervention and Baselines

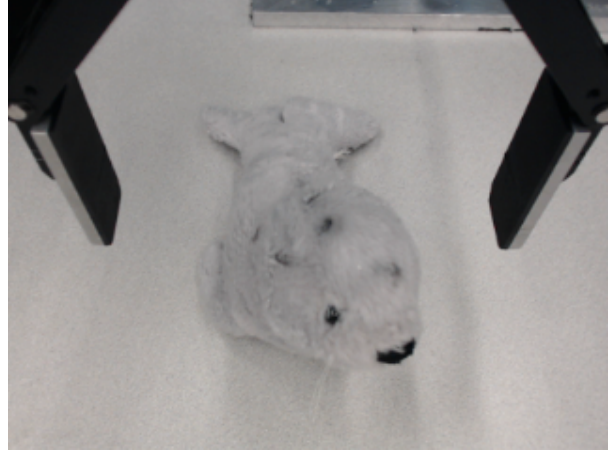
Steering Intervention: We hand-select and upweight a cluster of semantically meaningful value vectors by a coefficient of 10. Since each value vector can be interpreted as a probability distribution over output tokens, we examine the top-10 tokens with the highest probabilities for each vector. We then identify the top six value vectors containing the highest frequency of task-relevant keywords. For *Low/High Transport* we search for the keyword “low” for the low intervention and “high” for the high intervention. For *Slow/Fast Transport* we search for keywords “slow” and “safe” for the slow intervention and “fast” and “risk” for the fast intervention. We provide the value vectors used in the hardware experiments and their top-10 tokens in Table 8.3.

Random Intervention Baseline: we randomly select a cluster of six value vectors and upweight them by a coefficient of 10. We keep these vectors the same for the whole evaluation.

No Intervention Baseline: we run inference on π_0 -FAST without any modification and use the same prompt as during fine-tuning: “place penguin in basket” for *Low/High*



(a) Scene Camera



(b) Wrist Camera

Figure 8.7: Example of raw images that are input into π_0 -FAST.

Transport and “place seal on blue plate” for *Slow/Fast Transport*.

Prompt Modification Baseline: We prepend steering-related keywords to the original prompts used during finetuning. For *Low/High Transport*, the prompt “place penguin in basket” is modified to “low place penguin in basket” or “high place penguin in basket” for the low and high interventions, respectively. For *Slow/Fast Transport*, the prompt “place seal on blue plate” is modified to “slow safe place seal on blue plate” or “fast risk place seal on blue plate” for the slow and fast interventions, respectively.

Steering Intervention Trajectories

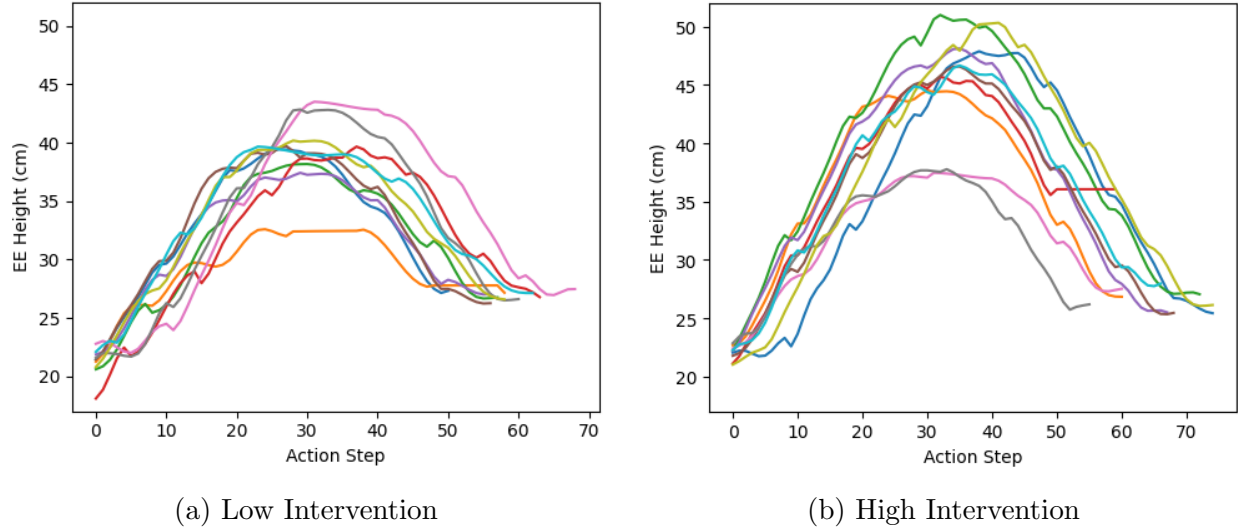


Figure 8.8: **Low/High Transport:** End-effector height from 10 trajectories with low intervention (a) vs. high intervention (b). We plot the segment of the trajectory from when the robot picks up the penguin to when it releases it into the basket. Low interventions have a lower maximum height on average.

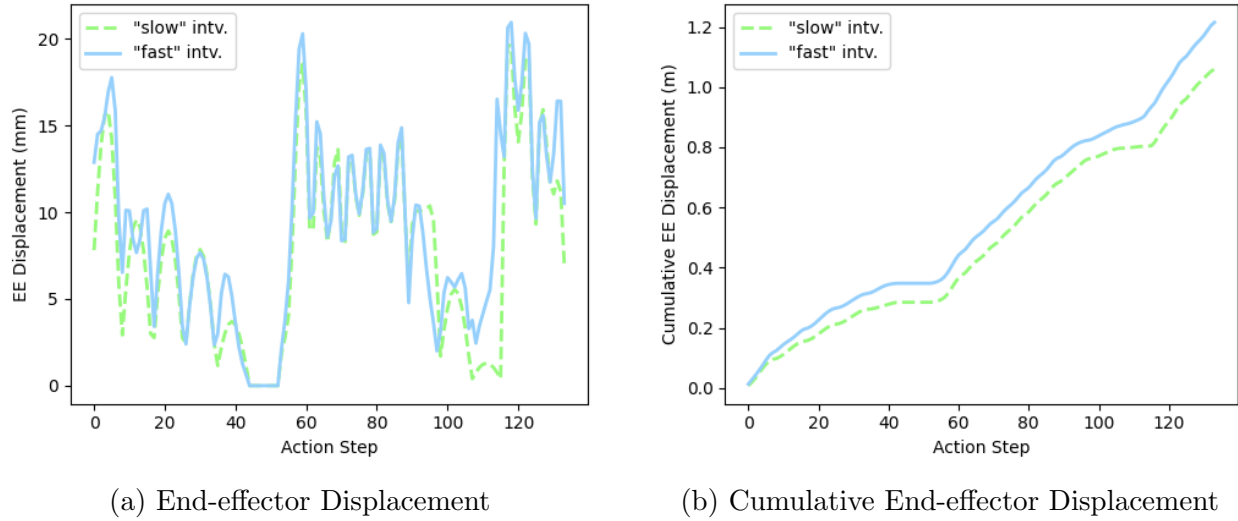


Figure 8.9: **Slow/Fast Transport:** End-effector displacement (a) and cumulative end-effector displacement (b) over a single trajectory with slow intervention vs. high intervention. While it may be hard to compare displacement at each action step, the cumulative displacement shows how the slow intervention reduces displacement.

Vector	Top 10 Tokens
5674	low, low, Low, Low, lower, LOW, lower, lowest, LOW, lows
253282	low, low, Low, Low, LOW, LOW, Lower, lower, Lower, lowest
263305	low, high, low, high, High, High, LOW, Low, Low, LOW
230595	low, medium, LOW, high, low, MEDIUM, Low, Low, Medium, medium
256528	low, Low, fillType, low, Low, StoreMessageInfo, PerformLayout, LOW, lows, WithIOException
246672	low, lows, low, Darío, cushi, Valentín, ecru, LOW, LOW, chaub

(a) Low intervention for *Low/High Transport*.

Vector	Top 10 Tokens
260217	high, high, High, High, HIGH, HIGH, low, low, Low, LOW
265014	high, high, High, High, HIGH, HIGH, hight, shenan, intersper, 高
266729	high, high, High, High, HIGH, HIGH, highs, 高, middle, hight
267454	high, High, High, high, HIGH, hight, HIGH, Odys, hig, quitted
270442	high, high, High, High, HIGH, HIGH, hight, higher, inappro, unwarran
275165	High, high, high, High, HIGH, HIGH, yüksek, 高, hoog, hight

(b) High intervention for *Low/High Transport*.

Vector	Top 10 Tokens
242252	safe, safe, safely, Safe, Safe, SAFE, SAFE, safety, safer, <bos>
104785	çük, SLOW, slow, Slow, slow, Slow, maksi, lacable, jät, tempo
243496	safe, safe, SAFE,.ArgumentParser, Safe, Safe, <bos>, Confira, memoized, Ainda
73159	soeur, slow, slowest, calm, Slow, slow, SLOW, calmer, hairc, atguigu
230490	safe, safety, safe, Safe, Safe, Safety, Safety, safety, safely, SAFETY
234025	slow, slowly, slow, <bos>, <Ac1403>, <Ac1327>, Slow, Slow, <Ac1340>, <Ac1450>

(c) Slow intervention for *Slow/Fast Transport*.

Vector	Top 10 Tokens
239787	risk, risk, <bos>, Risk, Risk, RISK, risks, Risks, RISK, riesgo
240184	fast, fast, FAST, Fast, Fast, FAST, quickly, tolerably, hentai, Quickly
259344	risk, risk, Risk, Risk, risks, Risks, riesgo, RISK, RISK, risky
262123	fast, fast, Fast, Fast, faster, FAST, fastest, quick, quicker, FAST
271027	fast, fast, fastest, faster, Fast, Fast, FAST, Faster, faster, FAST
273133	fast, fast, Fast, Fast, FAST, FAST, <Ac1446>, faster, <Ac0390>, veloce

(d) Fast intervention for *Slow/Fast Transport*.Table 8.3: Value vectors and their top-10 highest probability output tokens for steering interventions in *Low/High Transport* and *Slow/Fast Transport*.