Designing Intelligent Agents for Real-Time Experimental Control and Multi-Task Generalization

Thesis by
Enrique Amaya Perez

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2025
Defended June 13th, 2025

ACKNOWLEDGEMENTS

First, I would like to express my gratitude to Matt Thomson for his insightful guidance and continuous encouragement throughout the course of this research. His expertise and mentorship have been instrumental in shaping my academic development.

I am grateful to my thesis committee members, Paul Sternberg, David Van Valen, and Ueli Rutishauser, for their valuable feedback and support. I would also like to thank Professor Richard Murray for his early support and for giving me the opportunity to participate in exciting research in his lab during the SURF program in 2015, my very first research experience at Caltech.

From the Guttman Lab, I want to thank Mitch Guttman for the incredible opportunity to work in his lab for a full year before beginning my PhD. I'm especially grateful to Mario Blanco, who mentored me during that time. His generosity, thoughtful guidance, and genuine care for mentorship had a lasting impact on me. He also became a close friend who continued to support me throughout my PhD journey. I would also like to thank Jimmy Guo, who has remained a good friend since our time in the Guttman Lab and whose friendship and support have meant a lot over the years.

I would also like to thank a remarkable group of close friends and colleagues from the Mexican community at Caltech: Manuel Razo, Porfirio Quintero, Jorge Castillo, Andrés Ortiz, Alejandro Granados, Emanuel Flores, David Larios, and Jesús del Río. Each of them is an exceptionally talented scientist from whom I have learned a great deal. Beyond academic support, they have been a source of friendship, encouragement, and shared experience that I will always value. Their camaraderie and support have been a cornerstone of my journey.

Special thanks go to my colleagues and lab mates in the Thomson Lab. I am especially thankful to Shichen Liu, Shahriar Shadkhoo, Fan Yang, Dominik Schildknecht, and Guru Raghavan for their insightful conversations, collaborative spirit, and day-to-day support. Their perspectives, enthusiasm, and help with experiments made both the science and the lab experience far more enriching and enjoyable.

On a personal note, I owe my heartfelt appreciation to Carolina, my partner, for her unconditional love and unwavering support. I met her on the very first day of undergraduate studies in 2013, and she has stood by me ever since. It is no exaggeration to say that earning both of my degrees would not have been possible without her. Through good days and bad, she has always encouraged me to believe in myself and to keep going. Her love carried me through the most difficult moments, and her presence made many of my happiest PhD memories possible.

I would also like to thank my older brother, Tomás, who has always been an inspiration and protector. Despite the physical distance between us, I feel his support constantly, and I know he will always be there for me.

To my parents, I am endlessly grateful. Their sacrifices made it possible for my brother and me to pursue our dreams, and the love and affection they have always given us remain the greatest treasure in my life.

Finally, I would like to honor the memory of my grandparents, Mina and Ramiro. They passed away recently, but they were like a second set of loving parents. Their pride in our achievements and the love they gave us are things I will always carry with me.

ABSTRACT

Scientific discovery has traditionally relied on human-led iterative loops of observation, modeling, and intervention. This thesis explores the possibility of automating components of this loop using artificial intelligence (AI), particularly in systems characterized by non-equilibrium dynamics, high dimensionality, and emergent behaviors. Two foundational challenges are addressed: automating physical modeling and enabling adaptive interaction with dynamic experimental systems, and generalizing agent behavior across tasks and contexts without retraining.

To address the first challenge, we introduce a hierarchical AI framework for controlling active biomolecular matter, exemplified by microtubule–kinesin networks driven by light-activated motors. At the foundation are predictive models that learn the system's response to static light patterns, enabling inverse design by selecting inputs that yield desired structural outcomes. Building on this, dynamic models construct low-dimensional representations of the system's evolving state under time-varying stimuli, supporting forward simulation and real-time tracking. At the highest level, reinforcement learning agents—trained in simulation—discover and execute closed-loop control policies that achieve fine-grained manipulation objectives. These agents are deployed across $\sim$100 parallel experimental setups, demonstrating autonomous operation with robustness, scalability, and reliable transfer.

To address the second challenge, we investigate how generalist reinforcement learning agents can be constructed by leveraging the geometry of policy parameter space. We show that agents trained on distinct tasks self-organize into functionally segregated regions of weight space that encode both task identity and strategic variability. This insight enables the design of a hypernetwork—a network that generates the weights of other networks—that can interpolate smoothly between tasks and strategies via a single scalar input. Combined with a meta-controller, this architecture enables real-time modulation of agent behavior—ranging from conservative to risk-seeking—without retraining.

Together, these contributions demonstrate that intelligent systems can both design and control physical experiments in real time, and adapt cognitive strategies across tasks through principled representations in policy space. This work establishes a foundation for closed-loop scientific autonomy, programmable biomaterials, and generalist AI agents, converging at the intersection of machine learning, biophysics, and automation.

TABLE OF CONTENTS

Acknowledgements iii
Abstract v
Table of Contents vii
List of Illustrations ix
Chapter I: Introduction 1
1.1 Can We Automate the Scientific Process? 1
1.2 Two Foundational Challenges in Scientific Autonomy 3
A. Real-Time Interaction with Dynamical Systems 3
B. Generalization Across Tasks and Experimental Contexts 4
1.3 Organization of the thesis 4
Chapter II: Massively Parallel AI-Driven Closed-Loop Optical Control of Microtubule Networks 6
Abstract 7
2.1 Introduction 8
2.2 Results 9
Linear TraPhIC Predicts and Optimizes Microtubule Network Geometry in Static Light via Linear Latent Paths. 9
Flexible TraPhIC Predicts Microtubule Network Dynamics in Static and Dynamic Light via Latent Trajectories of Stimulus History 13
Deep Q-Learning Enables Active Control of Microtubule Networks for Target Object Capture. 18
Massively Parallel Deployment of AI Control Agents Demonstrates Scalable Manipulation of Active Matter Systems. 19
2.3 Discussion 21
2.4 Conclusion 22
2.5 Methods 23
Experimental Platform for High-Throughput Optical Control of Microtubule Networks 23
Microtubule-Kinesin Preparation and Chip Setup 23
Microscope Setup 24
Closed-loop Light Stimulation Software Pipeline 24
Deep Learning Models for Microtubule Network Prediction 27
Filament Simulation 27
L-PSTE on Simulation Data 29
G-PSTE on Experimental Data 34
Optimization of Experimental Conditions for Microtubule Bending 40
Formulation of the Bending Loss Function 40
Implementation of the Neural Network-Based Optimization Framework . . 41
Reinforcement Learning for Real-Time Object Capture 44
Simulation Environment for RL Agent Training 44

Reward Function Design 45
Training Algorithm for RL Agent 45
Chapter III: Building Generalist Agents by
Mapping the Geometry of Weight Space 48
Abstract 49
3.1 Introduction 50
3.2 Results 51
Population Training Reveals Task-Specific Differentiation of Neural Networks. . . . 51
Task Pressures Sculpt Distinct Functional Territories in Neural Weight Space . . . 54
Strategic Tradeoffs Map Onto the Internal Geometry of Functional Territories in Weight Space. 56
Learned Paths in Neural Policy Space Enable Generalist Agents with Tunable Task and Strategy. 58
3.3 Discussion 61
3.4 Conclusion 62
3.5 Methods 62
Agent Architecture, Training and Weight Extraction 62
Weight Projection, Visualization, and Fitness Modeling 64
Inter/Intra Task Clustering and Behavioral Phenotyping 65
Hypernetwork Modeling and Generalist Policy Construction 65
Bibliography 70

LIST OF ILLUSTRATIONS

Number *Page*

1.1 Conceptual analogy between reinforcement learning and the scientific method. . . . 1

2.1 Simulation-trained Linear TraPhIC accurately predicts microtubule network contraction and optimizes light inputs for experimentally validated network bending. 11

2.2 Flexible TraPhIC accurately predicts microtubule network trajectories under dynamic light and maintains performance on contraction under static patterns in real experiments. 15

2.3 Deep Q-Network agents transfer bead-capturing strategies from simulation to real microtubule network control. 17

2.4 Deployment of Deep Q-Network control across hundreds of microtubule network experiments demonstrates scalable RL for active microtubule systems. 20

3.1 Policy Evolution in Weight Space and Training Dynamics Across Atari Tasks 53

3.2 Structure of policy weight space across Atari games after training. 55

3.3 Reward-Risk Tradeoffs Organize Agent Variation in Policy Weight Space and Gameplay. 57

3.4 Geometric Traversal of Policy Space Enables Generalist Agent Design. 60

Chapter 1

INTRODUCTION

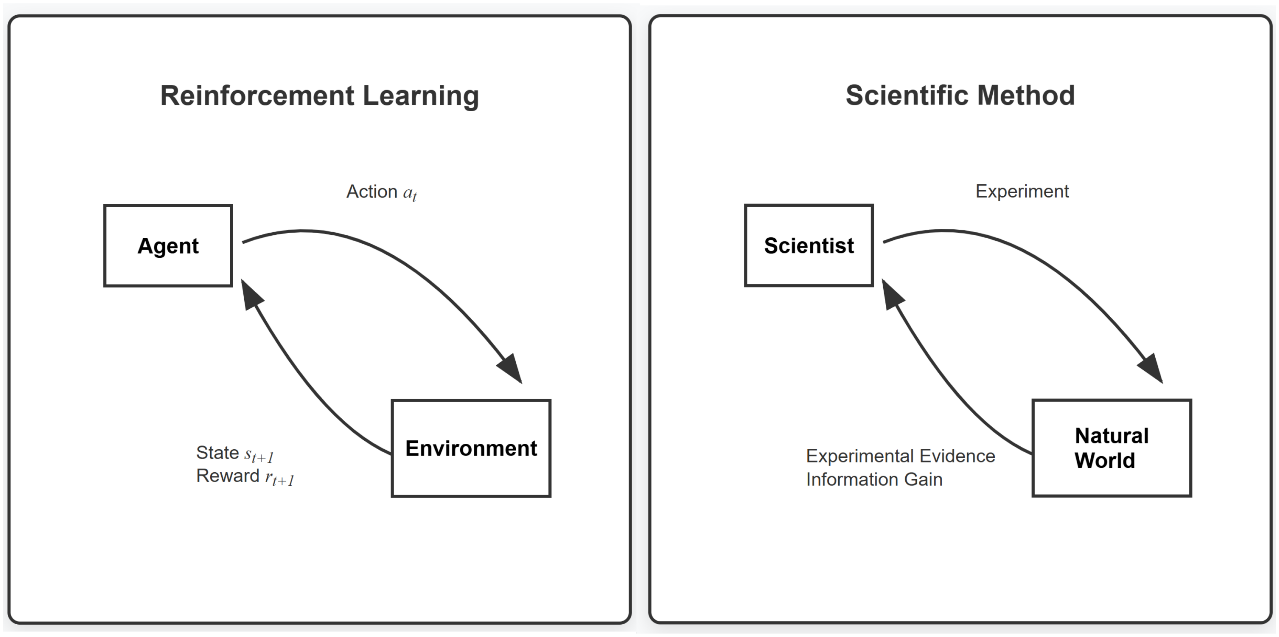

Figure 1.1: **Conceptual analogy between reinforcement learning and the scientific method.** The left panel illustrates the standard reinforcement learning (RL) loop, where an *agent* interacts with an *environment* by taking an action a_t. In response, the environment returns a new state s_{t+1} and a reward r_{t+1}, which the agent uses to update its policy and improve future behavior. The right panel depicts the analogous structure in the scientific method. Here, a *scientist* performs an *experiment* on the *natural world*, which yields *experimental evidence* and *information gain*. This feedback is used to refine hypotheses and inform future experiments. The figure highlights the shared core dynamic: an iterative cycle of intervention, observation, and adaptation aimed at improving predictive or explanatory models.

1.1 Can We Automate the Scientific Process?

Scientific discovery has long been understood as an iterative loop of observation, modeling, and intervention. Researchers observe phenomena in the natural world, extract patterns from data, formulate models to explain these patterns, and then design interventions—such as experiments or perturbations—to test hypotheses and uncover deeper causal structure. This cycle has been foundational across disciplines, from physics and chemistry to biology and neuroscience, and remains the central engine of scientific progress.

Traditionally, this loop has been entirely human-driven, requiring a combination of domain expertise, experimental skill, and conceptual abstraction. The process is often slow, expensive, and subject to cognitive limitations, especially in systems that are nonlinear, high-dimensional, or stochastic. As scientific questions become increasingly complex—often involving emergent behaviors, intricate feedback mechanisms, or parameter spaces that are too large to explore exhaustively—there is growing interest in whether aspects of this cycle can be delegated to machines.

Recent advances in artificial intelligence (AI) have introduced the possibility of automating components of the scientific process. Machine learning models have been applied to analyze experimental data (Reddy and Shojaee, 2025), infer physical laws (Cranmer et al., 2020), and accelerate simulation (Sanchez-Gonzalez et al., 2020; Thuerey et al., 2021). In materials science, robotic platforms now autonomously explore synthesis spaces using active learning (Burger et al., 2020; MacLeod et al., 2020). In synthetic biology, real-time feedback systems have been developed to modulate gene circuits using optogenetic inputs (Grosenick, Marshel, and Deisseroth, 2015; Lugagne, Sosa Carrillo, et al., 2017). These efforts suggest that the core loop of science—observe, model, intervene—can be at least partially operationalized by AI systems.

Yet most existing systems focus on static optimization or fixed pipelines, automating isolated tasks such as hyperparameter tuning in robotic chemists (Burger et al., 2020), candidate ranking in drug discovery (Zhavoronkov et al., 2019), or classification of microscopy images (Christiansen et al., 2018). These systems typically operate in an open-loop fashion, where model training and decision-making are decoupled from experimental feedback. For instance, active learning frameworks often rely on batch retraining between experimental rounds rather than updating models online (MacLeod et al., 2020; Häse, Roch, and Aspuru-Guzik, 2019); similarly, many optogenetic control systems use precomputed stimulation protocols or simple rule-based feedback rather than adaptively modulating input based on real-time cellular response (Grosenick, Marshel, and Deisseroth, 2015; Lugagne, Sosa Carrillo, et al., 2017). As a result, these approaches lack the closed-loop adaptivity required for genuine scientific reasoning. A true scientific agent would need to build internal models of the system it interacts with, refine them through continuous observation, and use them to guide interventions in real time under uncertainty—selecting actions not only for outcome optimization but also to test hypotheses, reduce ambiguity, and explore the structure of the system itself.

1.2 Two Foundational Challenges in Scientific Autonomy

For intelligent agents to truly contribute to science—not just as tools, but as autonomous participants helping steer the research—they need to go far beyond simply making inferences. Two such demands are especially critical. First, agents must be able to interact in real time with physical systems that are dynamic, noisy, and often only partially observed. Second, they must be capable of generalizing across tasks, adapting internal models and strategies to new goals, contexts, or experimental regimes without manual reprogramming. These challenges are not incidental—they reflect the essential character of scientific systems. Scientific environments rarely conform to fixed protocols or stationary distributions. Instead, they shift as questions evolve, measurements accumulate, and interventions perturb the system. The agent's role is not just to interpret data or optimize outcomes, but to operate under uncertainty, update its understanding on the fly, and choose actions that are informative, effective, and adaptable.

A. Real-Time Interaction with Dynamical Systems

Many experimental systems evolve on timescales where sensing, inference, and control must occur continuously and with minimal delay. For instance, adaptive optics in microscopy require real-time feedback to maintain focus in biological tissues (Häse, Roch, and Aspuru-Guzik, 2019), while brain-computer interfaces use streaming neural signals to drive motor output with sub-second latency (Booth, 2014). These domains demand not just prediction, but online control policies capable of adapting to system dynamics as they unfold. Real-time experimental agents must be able to filter noisy, partial observations, infer latent states, and intervene before the system drifts beyond recoverable regimes. Recent examples include adaptive electrophysiological stimulation systems that maintain neural responses under fluctuating noise (Pandarinath et al., 2017), and closed-loop platforms for stabilizing synthetic gene circuits (Milias-Argeitis et al., 2016). These applications illustrate the critical need for fast, adaptive control mechanisms that go beyond offline retraining or open-loop optimization.

B. Generalization Across Tasks and Experimental Contexts

Agents operating in scientific domains often face not a single task, but a continuum of related tasks that differ in system parameters, control goals, or environmental context. For example, in multi-robot manipulation, control policies that succeed on one object geometry may fail entirely on another without explicit retraining (James et al., 2022). Similarly, in automated experimentation, changes in chemical composition or temperature regimes often invalidate previously learned models (Coley et al., 2019).

Approaches based on meta-learning and unsupervised task inference have shown promise in addressing these issues (Zintgraf et al., 2019), but significant challenges remain in transferring structure across systems. Large-scale generalist models like Gato (Reed et al., 2022) and PaLM-E (Driess et al., 2023) highlight architectural pathways for integrating perceptual, language, and control modalities across domains, yet these models remain far from achieving robust generalization in settings where feedback, latency, and experimental stakes are high.

These limitations become even more pronounced when agents are expected to not only adapt, but do so in ways that are interpretable, efficient, and minimally disruptive to ongoing work. Meeting these demands requires abilities like composing behaviors from modular building blocks, identifying tasks on the fly, and adjusting exploration strategies based on uncertainty. Yet, these capabilities remain largely out of reach for current reinforcement learning systems, highlighting the gap between promising architectures and real-world generalization.

1.3 Organization of the thesis

This thesis is organized into two main parts, each addressing a foundational challenge in the development of intelligent systems for scientific discovery. The first part focuses on enabling real-time, adaptive control of dynamic experimental systems, specifically reconstituted microtubule–kinesin networks in which kinesin motors are activated by light-induced dimerization, allowing precise spatiotemporal regulation of activity. This control is implemented through a structured hierarchy of machine learning models designed to address increasingly complex tasks. The hierarchy begins with models that predict the system's response to static, time-invariant optical inputs and support inverse design, enabling the selection of input patterns that produce desired structural outcomes. It then advances to dynamic models that simulate system trajectories in response to arbitrary, time-varying light inputs by encoding behavior in a learned latent space. Finally, goal-directed, closed-loop control is achieved through reinforcement learning agents trained in simulation and deployed across 96 parallel experiments, autonomously discovering optical control policies to accomplish high-level transport objectives. This progression—from static prediction and inverse design, to dynamic modeling, to autonomous control—demonstrates how AI systems can be systematically developed to manipulate complex physical systems in an automated, scalable, and increasingly generalizable manner.

The second part focuses on the challenge of generalization in reinforcement learning. By analyzing the geometry of policy weight space, it is shown that agents trained on distinct tasks naturally form structured, segregated regions that encode both task identity and strategic variability. Building on this insight, a hypernetwork is introduced that enables smooth interpolation across these regions, generating agents capable of tunable behavior across multiple tasks without retraining. When combined with a meta-controller, this architecture allows for real-time modulation of strategy—ranging from conservative to risk-seeking—demonstrating flexibility and adaptability across 13 Atari games. Together, these two parts illustrate how AI can both autonomously control physical experiments and generalize cognitive strategies, laying groundwork for future systems that integrate automation, learning, and decision-making across physical and computational domains.

C h a p t e r 2

MASSIVELY PARALLEL AI-DRIVEN CLOSED-LOOP OPTICAL CONTROL OF MICROTUBULE NETWORKS

ABSTRACT

Intracellular transport relies on microtubules and kinesin motors, which operate effectively at the micron scale with a precision that exceeds current technological capabilities. These cytoskeletal systems exhibit complex, non-equilibrium dynamics that are difficult to model, predict, and externally control—limiting their potential for use in programmable materials and microscale manipulation. A central challenge has been developing scalable methods for real-time prediction and control of self-organizing microtubule networks in reconstituted systems. Here, we present a high-throughput optical platform and a family of neural network models that enable accurate prediction, optimization, and closed-loop control of microtubule network self-organization driven by light-activated kinesin motors. On the modeling side, our TraPhIC (Trajectory in Phase Space Inference via Compression) framework enables precise prediction and inverse design of light patterns by modeling experimental dynamics as trajectories in a learned latent space. For real-time control, we deploy ~100 trained Deep Q-learning agents in parallel, each using real-time feedback to control local microtubule network dynamics toward user-defined goals. Direct transfer of simulation-trained agents to physical experiments resulted in successful control of over 90% of microtubule networks, demonstrating the robustness and scalability of our approach. This integrated platform and modeling framework establishes a powerful paradigm for predictive and autonomous control of active matter systems, marking a significant step toward intelligent, self-organizing materials with programmable functionalities.

2.1 Introduction

Cells exert remarkable spatiotemporal control over their internal architecture through dynamic cytoskeletal networks composed of filaments and motor proteins. These adaptive structures enable complex functions like intracellular transport, division, and morphogenesis (Shelley, 2016). Inspired by this complexity, reconstituted microtubule-kinesin systems have emerged as platforms for engineering active materials, where molecular components consume energy to produce motion and organization (Stein et al., 2021; Suzuki et al., 2017). However, unlike cells, these systems lack endogenous regulation, making real-time external control challenging.

Recent experimental advances have enabled the regulation of activity across a range of engineered systems—including bacteria, colloids, and reconstituted cytoskeletal networks. While the specifics vary, these platforms commonly use optical methods to modulate activity with spatial and temporal precision (Ross et al., 2019; R. Zhang et al., 2021; Volpe et al., 2011; Buttinoni et al., 2012; Palacci et al., 2013). Light-responsive motors offer spatiotemporal control over microtubule dynamics through patterned illumination, suggesting a route to programmable self-organization (Ross et al., 2019). Yet, microtubule network responses remain unpredictable due to nonlinear interactions, emergent collective behaviors, and sensitivity to initial conditions, complicating the design of control strategies in reconstituted systems (Wu et al., 2017). This challenge is worsened by the current use of preprogrammed, open-loop illumination protocols that lack the adaptive feedback mechanisms found in living cells. Overcoming these limitations requires the development of predictive models, real-time feedback control, and scalable automation.

Recent advances in artificial intelligence (AI), particularly deep and reinforcement learning (RL), offer tools for real-time control of biological systems (Floreano and Mattiussi, 2008). In synthetic biology, deep learning has enabled optogenetic control of gene expression in thousands of cells (Lugagne, Blassick, and Dunlop, 2024), while RL frameworks have begun to manipulate neural dynamics in culture and *in vivo* (Pohlmeyer et al., 2014; Wülfing et al., 2019). Despite this progress, closed-loop control of self-organizing physical matter—where complex spatiotemporal patterns emerge from local interactions—remains largely unexplored (Falk et al., 2021).

We address this challenge by developing a general framework combining deep learning, RL, and high-throughput experimentation for closed-loop control of cytoskeletal active matter. We introduce two predictive deep learning models—Linear and Flexible variants of Trajectory in Phase Space Inference via Compression (TraPhIC)—that encode microtubule morphology as trajectories in a learned phase space, enabling prediction and inverse design under static and dynamic illumination. To achieve closed-loop control, we deploy deep Q-learning agents trained in simulation to dynamically guide microtubule network evolution via microscopy feedback. Unlike fixed-pattern approaches, our agents adapt actions based on continuous system observation. Finally, we scale this to ~100 parallel experiments, all autonomously guided by a shared pretrained agent.

This work establishes a new paradigm for adaptive control of active matter, combining optical precision with AI flexibility. By enabling real-time intervention in self-organizing systems, our approach opens pathways to programmable living materials, intelligent biomaterials, and control of non-equilibrium systems.

2.2 Results

Linear TraPhIC Predicts and Optimizes Microtubule Network Geometry in Static Light via Linear Latent Paths.

To model the temporal evolution of microtubule networks under constant light, we first developed the Linear Trajectory in Phase Space Inference via Compression (Linear TraPhIC) model. Trained on simulation data, Linear TraPhIC encodes the initial configuration of a microtubule network into a compact latent representation $z_0 = E(X_0)$, where $E(\cdot)$ is the state encoder. Future network states are predicted by translating this point along a straight-line trajectory in latent space using the linear phase-space propagator Φ_{lin}, which is a neural network that maps a scalar time shift Δt to a displacement vector:

$$\hat{X}_{\Delta t} = D(z_0 + \Phi_{\text{lin}}(\Delta t)) \tag{2.1}$$

where $D(\cdot)$ is the state decoder. This formulation enables temporally coherent and spatially accurate predictions at arbitrary future timepoints from a single input frame. By modeling time as a geometric shift in phase space, Linear TraPhIC compresses microtubule network contraction into a low-dimensional, interpretable latent path (Fig. 2.1A).

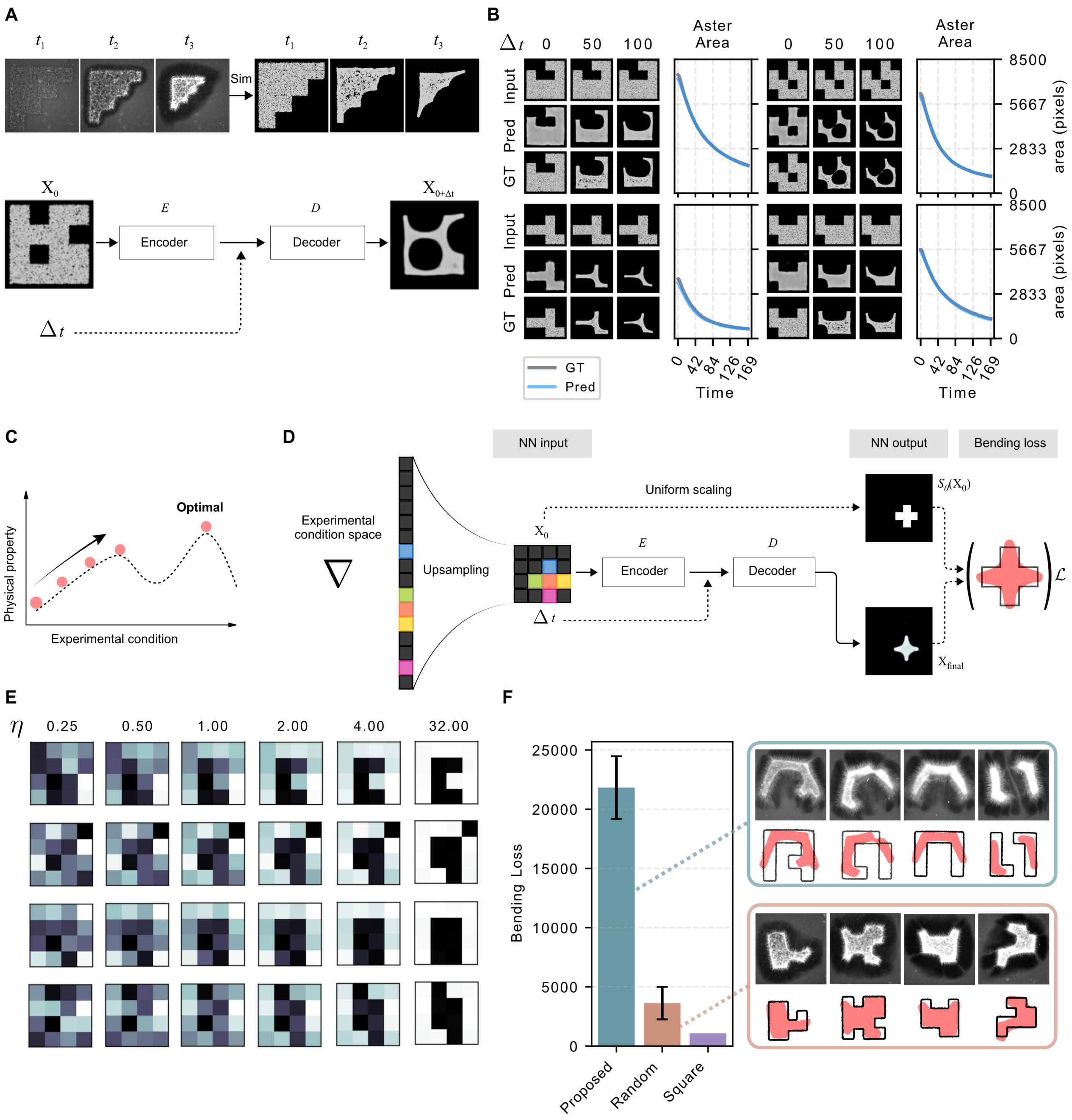

Figure 2.1: **Simulation-trained Linear TraPhIC accurately predicts microtubule network contraction and optimizes light inputs for experimentally validated network bending. A.** Top: Time-lapse microscopy images of microtubule contraction and matching simulation frames. Bottom: Linear TraPhIC schematic encoding initial state X_0, shifting by Δt in latent space, and decoding to $X_{0+\Delta t}$. **B.** Left: 3×3 grids comparing ground truth (GT) and predicted (Pred) images at $\Delta t = 0, 50, 100$ across experiments. Right: Plots of microtubule network area over time (GT: gray, Pred: blue). **C.** Schematic of optimization mapping experimental conditions to physical properties. **D.** Framework with a 4×4 light grid upsampled, input to a model predicting X_{final}, and compared to a scaled target shape via bending loss. **E.** Gradient ascent runs optimizing bending loss at varying learning rates (η). Rows show independent experiments; columns show progression. **F.** Left: Bar plot of bending loss across conditions (error bars: std. dev.). Right: Microscopy images (top) and segmented masks with target shapes (bottom) for different experiments.

We evaluated the predictive performance of Linear TraPhIC on test simulations not seen during training, using both qualitative and quantitative benchmarks. As a first check, we visualized predictions at time shifts $\Delta t = 0, 50, 100$ from a single initial frame X_0. These comparisons showed that the model accurately captured the spatial contraction dynamics of microtubule networks across diverse, nontrivial geometries (Fig. 2.1B, left). To quantify perceptual fidelity, we computed the Feature Similarity Index (FSIM) across input–target pairs $(X_t, \Delta t)$ from the test set. The model achieved a mean FSIM of 0.92 ± 0.03, indicating high structural correspondence with the ground truth images. Finally, to assess predictive accuracy using physically relevant metrics, we predicted full contraction dynamics by varying $\Delta t \in [0, 169]$ from each initial frame X_0, and compared the resulting microtubule network area dynamics to ground truth. Predicted area evolution closely followed the experimental measurements, with a mean absolute percentage error (MAPE) of $6.8 \pm 1.2\%$ across the test set (Fig. 2.1B, right).

We leveraged the differentiable structure of the Linear TraPhIC model to formulate an inverse design strategy for generating microtubule networks with targeted deformation profiles. We define a constrained experimental space consisting of a 4×4 light activation grid, which is transformed through upsampling to produce a spatial input compatible with the model (Fig. 2.1C–D). Using this representation, we defined a bending loss that quantifies deviation from uniform contraction:

$$\mathcal{L}_{\text{bend}} = \min_{\theta} \operatorname{MSE}(S_\theta(X_0), \hat{X}_{\Delta t}), \tag{2.2}$$

where $S_\theta(\cdot)$ applies isotropic scaling and translation, and $\hat{X}_{\Delta t}$ is the predicted final state from Linear TraPhIC. This loss reflects how much the final microtubule network deviates from a uniformly contracted version of its initial shape, thereby serving as a proxy for bending or asymmetry. Gradient-based optimization was performed over the 4×4 activation pattern space, propagating gradients through the full model to identify input conditions that maximized non-uniform microtubule network deformation.

Optimization consistently converged to distinct geometric motifs that induced pronounced bending (Fig. 2.1E). Experimental validation confirmed these predictions: optimized patterns produced significantly higher bending loss values (22500 ± 3500) than control patterns (4400 ± 520, $p < 0.001$), corresponding to a 5.1-fold increase in targeted deformation (Fig. 2.1F). These results demonstrate that Linear TraPhIC can serve not only as a predictive model but also as a differentiable engine for the inverse design of programmable cytoskeletal architectures.

Flexible TraPhIC Predicts Microtubule Network Dynamics in Static and Dynamic Light via Latent Trajectories of Stimulus History

Building on the latent phase space trajectory framework introduced in Linear TraPhIC, we developed the Flexible Trajectory in Phase Space Inference via Compression (Flexible TraPhIC) model to capture microtubule network dynamics under spatially and temporally varying light. While Linear TraPhIC relies on a fixed latent trajectory parameterized only by time, this assumption constrains it to static illumination settings. Flexible TraPhIC generalizes the approach by replacing the linear shift with a stimulus-dependent latent path, enabling accurate predictions of microtubule network evolution in response to dynamic illumination sequences.

The model constructs a latent trajectory from the history of external light inputs. At each time step t, we compute a temporally weighted light pattern, $\bar{\Lambda}_t$, by combining prior stimuli $\{\Lambda_1, \Lambda_2, \ldots, \Lambda_t\}$

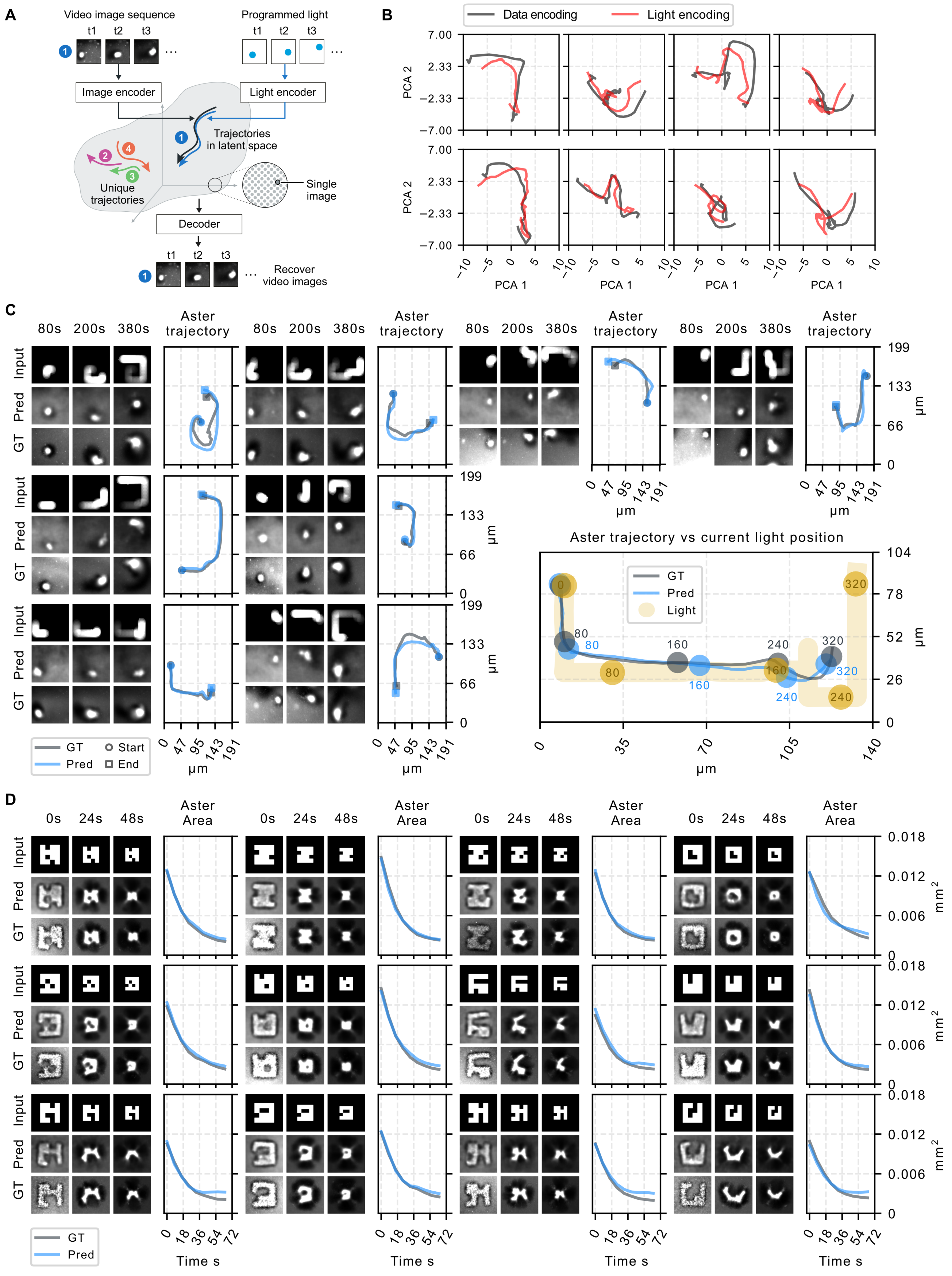
A
Video image sequence
t1 t2 t3
Programmed light
Image encoder
Light encoder
Trajectories in latent space
Unique trajectories
Single image
Decoder
Recover video images
B
Data encoding
Light encoding
PCA 1
PCA 2
C
80s 200s 380s
Aster trajectory
Input
Pred
GT
μm
Start
End
Aster trajectory vs current light position
Light
D
0s 24s 48s
Aster Area
Time s
mm²

Figure 2.2

Figure 2.2: **Flexible TraPhIC accurately predicts microtubule network trajectories under dynamic light and maintains performance on contraction under static patterns in real experiments. A.** Schematic of Flexible TraPhIC embedding microscopy video and light sequences into latent space, with a decoder reconstructing the video. **B.** PCA projection of latent trajectories from experimental data (dark curves) and light inputs (red curves). **C.** Left: 3×3 grids for dynamic light experiments. Top row: light sequence. Below: GT and Pred images at 80s, 200s, 380s. Right: GT and Pred trajectories with inset showing moving microtubule network centroid and light positions over time. **D.** Left: 3×3 grids for static light experiments. Top row: light pattern. Below: GT and Pred images at 0s, 24s, 48s. Right: Plots of microtubule network area over time (GT and Pred).

using an exponential decay that emphasizes recent inputs. This representation is passed through the light-conditioned phase-space propagator $\Phi_{\text{light}}(\cdot)$, a neural network trained to map light sequences to latent states that approximate those produced by the state encoder $E(X_t)$. The model is optimized to satisfy:

$$\Phi_{\text{light}}(\bar{\Lambda}_t) \longrightarrow E(X_t), \tag{2.3}$$

and the predicted configuration is generated by the decoder $D(\cdot)$ as:

$$\hat{X}_t = D\left(\Phi_{\text{light}}(\bar{\Lambda}_t)\right). \tag{2.4}$$

To assess consistency in the learned latent dynamics, we compared trajectories produced by the light input sequence to those derived from experimental image data. For each experiment—defined by a distinct dynamic light pattern—we extracted latent paths from both the encoder $E(X_t)$ and the propagator $\Phi_{\text{light}}(\bar{\Lambda}_t)$ across all timepoints. Projecting both into 2D space using Principal Component Analysis (PCA) revealed strong alignment across experiments (Pearson correlation: 0.92 for PC1, 0.89 for PC2; Fig. 2.2B), indicating that Flexible TraPhIC learns an internal representation that evolves coherently in response to external light stimuli.

To test whether this latent coherence translates into physically accurate predictions, we next evaluated performance of Flexible TraPhIC on experimental data of microtubule dynamics produced by dynamic light patterns. To begin, we examined representative predictions at 80s, 200s, and 380s across multiple experiments. The model faithfully reproduced both the displacement of microtubule networks and the evolving density of recruited material across a wide range of light stimulus sequences, resulting in close qualitative agreement with experimental observations (Fig. 2.2C, left). We quantified this visual correspondence using the Feature Similarity Index (FSIM), obtaining an average score of 0.853 across test frames, indicating strong perceptual alignment with ground truth. To check the model's physical accuracy, we tracked the microtubule network centroid over time. Predicted motion paths remained tightly aligned with those observed experimentally, capturing both directionality and trajectory shape (Fig. 2.2C, right). The median frame-to-frame displacement error between predicted and ground truth centroids was 8.9 µm, representing 5.2% of the average total path length (171 µm). Notably, the inset in Fig. 2.2C (bottom right) highlights not only this alignment but also reveals that both predicted and actual microtubule network motion exhibit a delayed, nonlinear response to the light stimulus trajectory — a complex behavior that Flexible TraPhIC captures with high fidelity.

To demonstrate that Flexible TraPhIC generalizes the capabilities of Linear TraPhIC, we evaluated its performance on a second dataset consisting of microtubule networks subjected to static light patterns (Fig. 2.2D). This static-input dataset serves as a benchmark for comparing Flexible and Linear TraPhIC on real experimental data. Flexible TraPhIC successfully captured the contraction dynamics of microtubule networks, preserving spatial features with high perceptual fidelity (FSIM = 0.90 ± 0.04, Fig. 2.2D, left). The predicted area dynamics closely matched ground truth experimental data, with a mean absolute percentage error (MAPE) of 7.1 ± 1.5% (Fig. 2.2D, right). These results confirm that Flexible TraPhIC maintains the predictive precision of the linear model while enabling accurate modeling under both static and dynamic illumination.

Together, these results demonstrate that Flexible TraPhIC generalizes the latent trajectory modeling framework to support prediction under both dynamic and static light conditions. By mapping light stimulation sequences to phase space representations, the model captures both motion and morphological changes in light-inducible microtubule networks, enabling accurate prediction of experimental dynamics in complex, time-varying environments.

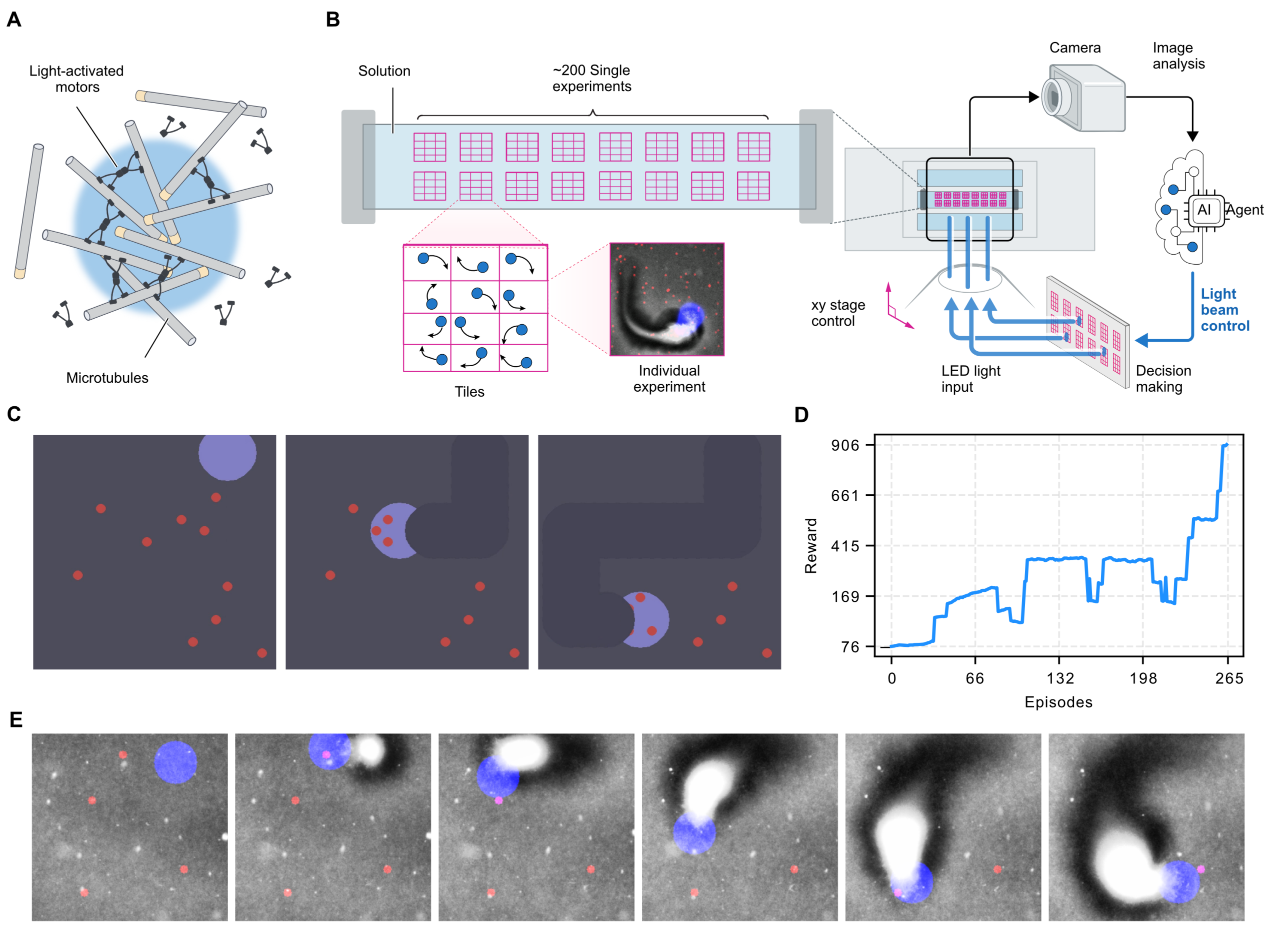

Figure 2.3: **Deep Q-Network agents transfer bead-capturing strategies from simulation to real microtubule network control. A.** Diagram of light-induced microtubule organization via kinesin dimerization (blue: light stimulus). **B.** High-throughput platform with a chip under a microscope, showing parallel experiments divided into tiles. **C.** Simulated environment with a light stimulus navigating to capture objects (red). **D.** Plot of reward progression over training episodes in the simulation. **E.** Sequential frames of a real microtubule network experiment. Blue: light stimulus; red: target objects.

Deep Q-Learning Enables Active Control of Microtubule Networks for Target Object Capture. Although Flexible TraPhIC represents a significant advance in our ability to predict microtubule responses across diverse light conditions, its passive modeling approach does not enable direct manipulation of these networks. To bridge this gap and shift from prediction to real-time optical control, we trained Deep Q-Learning agents to actively guide microtubule networks toward specific goals, such as capturing target objects.

We first trained agents in a custom Gym-based simulation environment that mimics the local dynamics of light-responsive microtubule systems (Fig. 2.3C). Within this environment, the agent learned to move a circular light stimulus to capture target objects while avoiding obstacles such as material depletion zones. Deep Q-Network (DQN) training enabled convergence to effective strategies, with simulated agents capturing all targets in over 92% of training episodes. Reward values improved from -76 at initialization to +906 after 265 episodes (Fig. 2.3D), reflecting the emergence of efficient and adaptive behavior.

To evaluate whether this simulated policy could generalize to real-world systems, we deployed a trained agent into the experimental control pipeline (Fig. 2.3A-B) and tested it on a single microtubule network. The agent received real-time microscopy input and output discrete movement actions to guide a 30 µm light stimulus every 8 seconds. In this trial, the microtubule network formed within 40 seconds of initial light activation and exhibited smooth, directed motion. The four predefined targets, located at varying distances up to 150 µm from the initial network centroid, were sequentially approached within 400 seconds, during which the agent made 50 discrete decisions. The network centroid traveled a total of 138 µm along in its path to the objects. This result demonstrates that a RL policy trained entirely in simulation can transfer to the physical domain (Fig. 2.3E).

Massively Parallel Deployment of AI Control Agents Demonstrates Scalable Manipulation of Active Matter Systems.

Having validated real-time control with a single agent, we next scaled our approach to demonstrate distributed autonomous manipulation of active matter across hundreds of simultaneous experiments. Leveraging our high-throughput imaging platform and closed-loop light control system, we deployed 96 trained Deep Q-Network (DQN) agents in parallel—one per tile—each independently controlling a localized light stimulus to guide the organization of a microtubule network (Fig. 2.4).

Using the same 30 µm circular light stimulus and 8-second decision interval as in the single-agent trial, each agent operated autonomously in response to real-time visual input. All agents shared a common pretrained policy and required no additional fine-tuning or inter-agent coordination. Over a 10-minute trial, individual agents completed 50–80 discrete actions, collectively constituting hundreds of simultaneous closed-loop control episodes.

To assess large-scale efficacy, we defined a successful trial as one in which the microtubule network maintains a single connected structure throughout its trajectory toward the targets, since object collection requires the material to remain localized. Across 96 experiments, 89 (92.7%) met this criterion. In the remaining trials, failures typically occurred when the agent moved the light stimulus too quickly, triggering the formation of a new microtubule network and leaving the original structure behind. Visual inspection of representative tiles (Fig. 2.4) revealed heterogeneity in agent behavior, with varied light trajectories that effectively directed cohesive microtubule networks toward their targets over extended distances.

These results demonstrate that RL policies trained in simulation can generalize across hundreds of independent physical systems, enabling massively parallel and autonomous control of dynamic biological materials. This approach establishes a scalable framework for intelligent manipulation of active matter, combining distributed decision-making with real-time feedback to drive self-organization at scale.

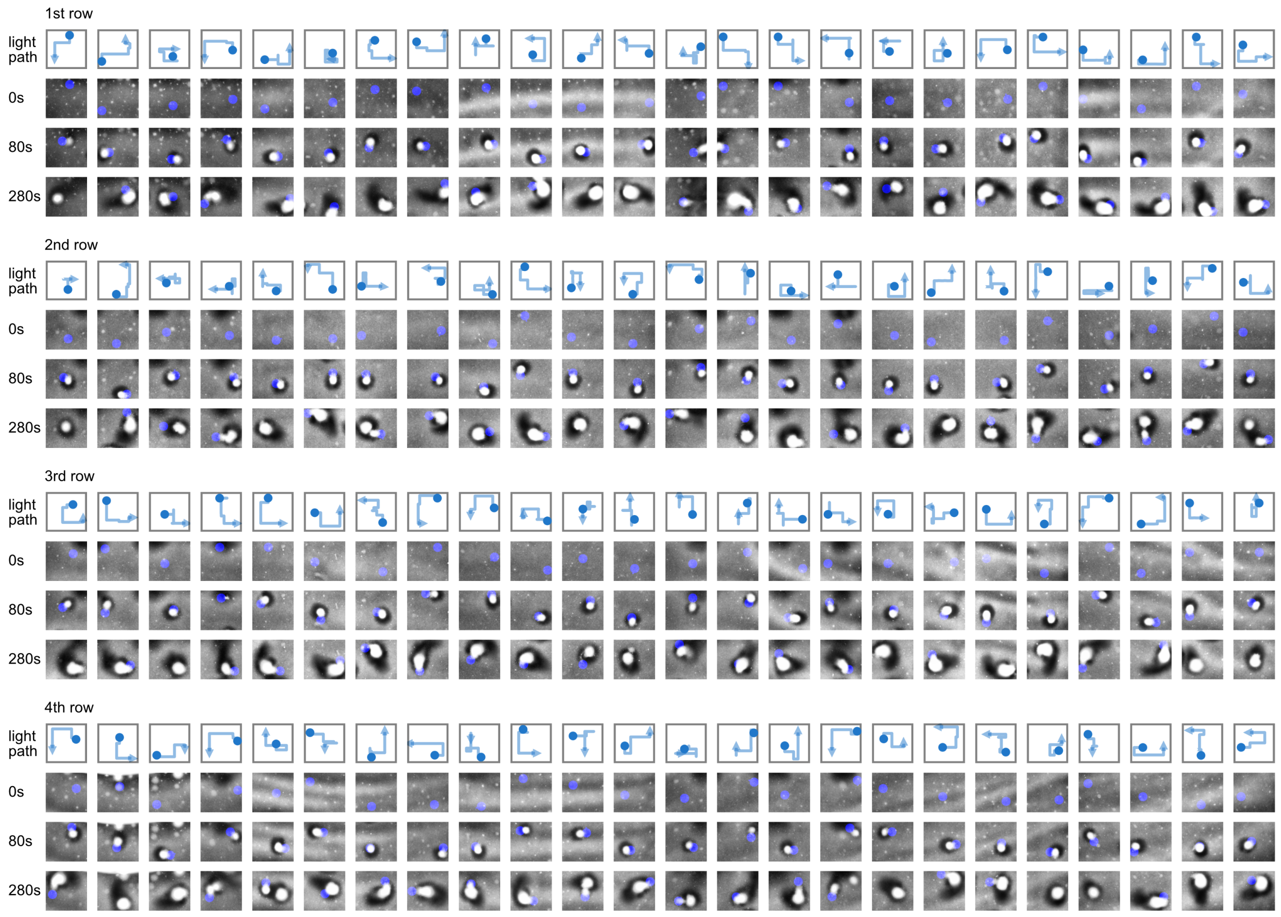

Figure 2.4: **Deployment of Deep Q-Network control across hundreds of microtubule network experiments demonstrates scalable RL for active microtubule systems.** Experimental data from the multi-agent pipeline. Bright regions show higher density of fluorescently labeled microtubules, while blue shows the light stimulus. Each subpanel shows one row of 24 contiguous experiments (eight rows per reaction). Within each subpanel, the top row shows the AI-controlled light path (starting at the circle marker), followed by three time points (0s, 80s, 280s) showing the evolution of microtubule networks.

2.3 Discussion

The ability to program active matter systems—both through principled design and real-time manipulation—remains a fundamental challenge in synthetic biology and materials science (Nguyen et al., 2018; Tang et al., 2021; A. P. Liu et al., 2022; Leech et al., 2025). Here, we demonstrate a comprehensive framework that addresses this challenge through two complementary approaches. First, we employed a data-driven predictive modeling to design experimental conditions that achieve targeted microtubule network morphologies. Second, we leverage RL approach for adaptive closed-loop control of network dynamics. These approaches enable both optimal design and autonomous guidance of light-responsive cytoskeletal assemblies *in vitro*.

Our predictive modeling framework establishes a low-dimensional, data-driven phase space that captures the essential dynamics of microtubule network evolution under light stimulation. By encoding dynamic morphology as trajectories in latent space, our TraPhIC models enable accurate prediction of future states under both static and dynamic illumination. This abstraction provides a flexible framework for modeling complex, high-dimensional biological systems without requiring explicit physical models (Qu et al., 2021; Ross et al., 2019). Crucially, the differentiable nature of these models enables inverse design—an approach we demonstrated through the optimization of light patterns that produce specific targeted deformations in microtubule networks. This contributes to the broader advancement of inverse design methodologies in materials and biological systems, representing a data-driven alternative to traditional trial-and-error strategies (Sherman et al., 2020; H. Liu et al., 2023).

For real-time control, our RL approach enables autonomous manipulation of microtubule networks toward specific goals. Simulation-trained Deep Q-Learning agents demonstrated remarkable transfer capabilities, successfully controlling real microtubule networks without requiring fine-tuning. The parallel deployment of nearly 100 agents highlights the scalability of this approach, transforming cytoskeletal networks into programmable elements capable of responsive, goal-directed behavior.

This dual approach, which uses prediction for design and RL for adaptive control, represents a significant improvement from traditional optical patterning techniques. Unlike conventional methods that rely on predefined or heuristic, open-loop illumination sequences (Ross et al., 2019; Chennakesavalu et al., 2024; Cai et al., 2025), our framework offers both optimal design and real-time adaptation. The predictive models enable principled pattern optimization before experiments begin, while RL provides responsive, feedback-driven control during execution. Together, they create an integrated platform for manipulating active matter systems with unprecedented automation and scalability.

Our approach opens several promising avenues for research. These include: (1) exploring alternative actuation mechanisms such as DNA-based switches, (2) adding different cytoskeletal components like actin or diverse motor proteins, (3) incorporating physical priors into our predictive models, (4) investigating model-based RL with TraPhIC as a world model, and (5) enabling inter-agent coordination for collective behavior across multiple networks.

2.4 Conclusion

We present a comprehensive platform for the prediction, optimization, and closed-loop control of microtubule networks using neural networks and RL. This work demonstrates that cytoskeletal assemblies can be both optimally designed and actively manipulated in real-time, transforming them from passive biomaterials into programmable, responsive systems. Through advances in both optimal experimental design and real-time control, powered by artificial intelligence and high-throughput experimentation, we establish a generalizable approach to programmable active matter. This convergence of machine learning, biophysics, and automation offers a powerful foundation for the development of intelligent microscale systems capable of self-organization, decision-making, and purposeful behavior.

2.5 Methods

Experimental Platform for High-Throughput Optical Control of Microtubule Networks

Microtubule-Kinesin Preparation and Chip Setup

Preparation of Microtubule-Kinesin Solutions We constructed two light-responsive *Drosophila melanogaster* K401 kinesin chimeras, K401-iLID and K401-micro (Addgene #122484, #122485). K401-iLID was engineered by fusing an iLID domain with a C-terminal His tag to K401, while K401-micro contained K401 inserted between an N-terminal His-MBP and micro domain; MBP ensured microdomain functionality during expression and was later cleaved using TEV protease. Constructs were expressed in *E. coli* BL21(DE3)pLysS cells and induced with 1 mM IPTG at 18°C for 16 hours. Cells were lysed and the supernatant was purified using Ni-NTA agarose affinity chromatography. Eluted proteins were dialyzed and stored in imidazole-based buffer with $MgCl_2$, DTT, MgATP, and sucrose, flash frozen in liquid nitrogen, and stored at −80°C. For microtubule polymerization, unlabeled and fluorescently labeled tubulin (20 mg/mL each) were thawed, mixed, and polymerized in a GMP-cpp–based buffer system at 37°C. After ultracentrifugation to remove aggregates, the supernatant containing microtubules was aliquoted and frozen. For reactions, K401-iLID, K401-micro, and polymerized microtubules were combined in a final buffer containing K-PIPES (pH 6.8), MgATP, DTT, glycerol, pluronic F-127, oxygen scavengers (pyranose oxidase, glucose, catalase, Trolox), and ATP-recycling enzymes. Final concentrations were 0.1 μM for each motor construct and 1.5–2.5 μM for tubulin. Reactions were prepared under red-filtered light (Kodak Wratten No. 25) to prevent unintended photoactivation, with experiments performed within 2 hours due to pH sensitivity.

Microfluidic Chip Design and Reaction Setup Glass slides and coverslips were cleaned via sequential sonication in 2% Hellmanex, ethanol, and 0.1 M KOH, with DI water rinses between steps, followed by HCl etching. Silanization was performed using a solution of ethanol, acetic acid, and silane agent, after which slides were baked. A 2% acrylamide layer was polymerized onto the surface using TEMED and ammonium persulfate, then slides were rinsed and dried.

Microtubules were immobilized onto flow-cell coverslips via surface adsorption using 0.01% poly-L-lysine treatment. Fluorescence imaging was conducted to assess microtubule length distribution. Images underwent normalization, thresholding, and morphological filtering to isolate individual filaments. Overlapping filaments were resolved via angular filtering. Microfluidic reactions were loaded into flow chambers immediately after mixing to preserve activity and pH, with all steps performed under dark-room conditions to minimize photoactivation of iLID domains.

Microscope Setup

Hardware Configuration Our custom imaging system is based on a widefield epifluorescence microscope (Nikon Ti-2) with two additional imaging modalities: pattern projection illumination and LED-gated transmitted light. For the pattern projection system, we employed a programmable Digital Light Processing (DLP) chip (EKB Technologies DLP LightCrafter™ E4500 MKII™ Fiber Couple) to project light patterns onto samples, using a fiber-coupled 470 nm LED (ThorLabs M470L3) as the light source. We modified an epi-illumination attachment (Nikon T-FL) with two entry ports to merge the projected pattern light beam with the standard fluorescence microscopy light beam using a dichroic mirror (Semrock BLP01-488R-25). The system's magnification was calibrated to ensure full illumination of the camera sensor (FliR BFLY-U3-23S6M-C) by the DLP chip. We developed a unified Python software framework that communicates with Micro-Manager to automate pattern projection and stage movement, integrating AI agents for real-time adaptive control. For the transmitted light, we replaced the standard white light brightfield source (Nikon T-DH) with an electronically time-gated 660 nm LED (ThorLabs M660L4-C5), allowing precise control over illumination. This modification aimed to minimize unwanted light-induced dimerization during brightfield imaging, improving experimental control and reproducibility.

Closed-loop Light Stimulation Software Pipeline

System Initialization A unified Python software framework was developed to automate the configuration and operation of the microscope, pattern projection system, and AI-driven light control. The system initializes by configuring key microscope parameters, including multi-channel imaging settings, exposure times, and multi-position acquisition with defined field-of-view (FOV) spacing. During this stage, configuration files are dynamically generated and loaded into Micro-Manager, ensuring seamless synchronization between the microscope, DLP projector, and AI-driven adaptive light control.

Image Acquisition and Tiling Imaging is performed at eight positions spaced 1500 µm apart along the X-axis. The system captures Cy5-labeled microtubules in the Cy5 channel (excitation: 650 nm, emission: 670 nm, exposure: 100 ms), while using 470 nm projection (exposure: 250 ms) as the active stimulus. At the core of the framework, each field of view (FOV) is divided into a 3×4 grid of independent tiles to enable localized control of light stimuli. After each image acquisition, the FOV is automatically divided into these tiles for individual processing.

AI Agent Control Each tile is modeled as an individual OpenAI Gym environment, where AI agents control the projected light stimulus with a discrete action space of four possible movements: up, down, left, and right. The agents move a circular light stimulus (30 µm diameter) in 10 µm steps per action. The system extracts observations from the acquired images, encoding the relative position of the light stimulus to the microtubule network to guide the agents' decisions.

Projection Assembly Once all agents determine their actions, the resulting light patterns from all tiles are merged into complete projections for each position, stacked into a 3D array, and loaded into memory for the projector to use. This cycle of imaging, observation extraction, action selection, and light projection ensures a continuous, real-time adaptive control mechanism for precise light modulation across all experimental positions.

The following pseudocode outlines the general algorithm for initializing the system and executing the main control loop:

Algorithm 1 General Pseudo-Code for AI-Driven Light Projection & Microscope Automation

```
procedure INITIALIZESYSTEM
    Set microscope_settings (channels, exposure_times, frame_count, interval)
    Set multiposition_acquisition (FOV_list, spacing_between_FOVs)
    Set grid_size (rows, columns)
    Set movement_rules (step_distance, light_radius)
    Load AI_model (pretrained neural_network)
    Initialize 3D_stack_multipos as empty array
    for each position in FOV_list do
        Divide FOV into grid_of_tiles (based on grid_size)
        for each tile in grid_of_tiles do
            Initialize independent_environment
                Set light_stimulus_position (random initial coordinates)
        end for
        3D_stack_multipos[position] ← CombineTiles(grid_of_tiles)
    end for
    SendToProjector(3D_stack_multipos) (Load initial projection set)
end procedure

procedure MAINLOOP
    while experiment_running do
        for each position in FOV_list do
            Move stage_to_position
            Project(3D_stack_multipos[position])
            WaitFor new_image from microscope
            if new_image_detected then
                for each tile in grid_of_tiles do
                    Extract tile_observation (relative_position_to_microtubule_net)
                    action ← AI_model.predict(tile_observation)
                    Update light_stimulus_position based on action
                end for
                3D_stack_multipos[position] ← CombineTiles(grid_of_tiles)
            end if
        end for
        SendToProjector(3D_stack_multipos)
    end while
end procedure
```

Deep Learning Models for Microtubule Network Prediction

Filament Simulation

Pattern Generation and Domain Confinement The domain geometry is defined by a 16-character binary string, representing a 4×4 grid where each digit (0 or 1) denotes an inactive or active cell, respectively. This grid is upscaled into a binary mask of size $4s \times 4s$ pixels, where s is a positive integer specifying the pixel side length of each cell. The mask is constructed by mapping each active cell (value 1) to an $s \times s$ pixel block assigned a value of 1, with inactive cells remaining 0. Contour extraction is then applied to the active regions of the binary mask to generate a polygonal geometry. This polygon delineates the spatial domain within which filaments are confined, and any filaments initially positioned outside this region are excluded from the simulation.

Initialization of Filament Network The filament network is initialized with N filaments, uniformly distributed across a rectangular domain spanning $[-W_x/2, W_x/2] \times [-W_y/2, W_y/2]$, where W_x and W_y are specified by the user. The initial filament count is $N = \text{round}(\rho W_x W_y/2)$, where ρ is the filament density. Filament lengths l_i are sampled from a normal distribution with mean $\langle l \rangle = \xi/\sqrt{\rho}$, where ξ is a scaling parameter, and standard deviation $\langle l \rangle/10$. Orientations ϕ_i are drawn uniformly from $[0, 2\pi)$. Each filament is represented by its center coordinates $\mathbf{c}_i = (c_{x,i}, c_{y,i})$, length l_i, and orientation ϕ_i, with endpoints computed as $\mathbf{p}_i^{\pm} = \mathbf{c}_i \pm \frac{l_i}{2}(\cos\phi_i, \sin\phi_i)$. After domain confinement, the filament count and positions are adjusted by removing those with centers outside the polygonal boundary.

Interaction Network and Laplacian Construction Filament interactions are determined by computing pairwise Euclidean distances d_{ij} between filament centers $\mathbf{c}_i$ and $\mathbf{c}_j$. Connectivity is determined by the adjusted distance $d_{ij} - \frac{l_i + l_j}{2}$: if negative (indicating overlap within the length-adjusted range), the filaments are connected (value 1); otherwise, they are not (value 0). This forms a sparse symmetric adjacency matrix A, with self-interactions excluded. The matrix A is scaled by an interaction probability $p_{\text{act}} \in [0, 1]$, modulating elastic coupling strength. The graph Laplacian

L is constructed as $L = D - A$, where D is a diagonal matrix with $D_{ii} = \sum_j A_{ij}$, the number of connections for filament i. The Laplacian governs elastic coupling of filament endpoints, with forces proportional to relative displacements.

Stochastic Time Evolution The system evolves over a total time T_{time} with a time step Δt, yielding round$(T_{\text{time}}/\Delta t)$ steps. Filament positions $\mathbf{c}_i$ and orientations ϕ_i are updated via an overdamped Langevin equation, incorporating elastic forces, torques, and thermal noise. Elastic coupling uses the minus-end positions $\mathbf{p}_i^-$, assuming interactions are localized to one end. These are collected as $\mathbf{P}^- = [\mathbf{p}_1^-, \mathbf{p}_2^-, \ldots, \mathbf{p}_N^-]^T$.

The net displacement of filament i's end relative to its neighbors' ends is:

$$\mathbf{r}_i = (L\mathbf{P}^-)_i = \sum_{j \in \mathcal{N}(i)} (\mathbf{p}_i^- - \mathbf{p}_j^-), \tag{2.5}$$

where $\mathcal{N}(i) = \{j \mid A_{ij} = 1\}$ is the neighbor set of filament i.

Elastic forces are given by Hooke's law with rest length l_{rest}:

$$\mathbf{F}_i = -k_i(t)\,(|\mathbf{r}_i| - l_{\text{rest}} D_{ii})\,\hat{\mathbf{r}}_i, \tag{2.6}$$

where $\hat{\mathbf{r}}_i = \mathbf{r}_i/|\mathbf{r}_i|$, $k_i(t) = a_i(t) K_{\text{Link}}$, and $a_i(t) \sim \text{Bernoulli}(\kappa_i(t))$ is a stochastic activation factor. The activation probability is:

$$\kappa_i(t) = \left(1 - e^{-t/\tau_{\text{link}}}\right) \left[1 + (p_{\text{act}} - 1) H(c_{x,i})\right], \tag{2.7}$$

with $\tau_{\text{link}} = T_{\text{time}}/10$ as the linking timescale and $H(\cdot)$ the Heaviside step function.

Torques driving orientational alignment are:

$$\tau_i = -\sin\left((L\phi)_i\right), \quad \text{where} \quad (L\phi)_i = \sum_{j \in \mathcal{N}(i)} (\phi_i - \phi_j). \tag{2.8}$$

Filament motion is subject to anisotropic drag due to their elongated shape. To model this, we define a diagonal drag tensor $\mathbf{\Gamma} = \text{diag}(\gamma_\perp, \gamma_\parallel)$ in the filament body frame, where $\text{diag}(a, b)$

denotes a diagonal matrix with entries a and b on the main diagonal. The drag coefficient along the filament's axis is $\gamma_{\|} = \gamma$ and perpendicular to it is $\gamma_{\perp} = \gamma \frac{l_i}{\pi w}$, with γ a baseline drag coefficient and w the filament diameter. Transforming this tensor into the lab frame via a similarity transformation using the rotation matrix $\mathbf{R}(\phi_i)\mathbf{\Gamma}\mathbf{R}(\phi_i)^{\top}$, the effective drag coefficients in the x and y directions for filament i are:

$$\gamma_{x,i} = \gamma\left(\cos^2\phi_i + \frac{l_i}{\pi w}\sin^2\phi_i\right), \quad \gamma_{y,i} = \gamma\left(\sin^2\phi_i + \frac{l_i}{\pi w}\cos^2\phi_i\right) \tag{2.9}$$

Translational velocities follow overdamped Langevin dynamics:

$$\mathbf{v}_i = \boldsymbol{\gamma}_i^{-1}\mathbf{F}_i + \boldsymbol{\eta}_i, \tag{2.10}$$

where $\boldsymbol{\gamma}_i = \text{diag}(\gamma_{x,i}, \gamma_{y,i})$, and $\boldsymbol{\eta}_i \sim \text{Norm}(0, T_{\text{temp}}\mathbf{I})$ is thermal noise, with $\mathbf{I}$ as the 2×2 identity matrix and T_{temp} as temperature. Angular velocity is:

$$\omega_i = \frac{\tau_i}{I_i} + \eta_{\phi,i}, \tag{2.11}$$

with $I_i = \gamma l_i/\mu$ (where μ is a constant) and $\eta_{\phi,i} \sim \text{Norm}(0, T_{\text{temp}})$ scaled by $2\pi/180$ to radians.

Updates use Euler integration:

$$\begin{cases} \mathbf{c}_i \leftarrow \mathbf{c}_i + \mathbf{v}_i\Delta t, \\ \phi_i \leftarrow \phi_i + \omega_i\Delta t. \end{cases} \tag{2.12}$$

L-PSTE on Simulation Data

Dataset Generation and Processing The Linear Phase Space Trajectory Encoder (L-PSTE) model was trained using data generated from our filament simulation framework described in Section 2.5, with simulation parameters summarized in Table 2.1. We started with all 2^{16} possible 16-character binary strings, each representing a 4×4 grid configuration where "1" indicates an active pixel and "0" an inactive one. From these, we kept only the 11,506 patterns that form a single connected region under 4-connectivity—that is, every active pixel must be adjacent to at least one other active pixel via a shared horizontal or vertical edge. After eliminating symmetric equivalents under the dihedral group D_4, we obtained 1,524 distinct patterns. Each of these patterns served as an initial geometry for the filament simulation, which generated one video per pattern, consisting of 170 grayscale frames of size 397×397 pixels. These frames captured the contraction dynamics of microtubule networks under static light condition over time. These frames were resized to 112×112 pixels to match the model's input dimensions, preserving the network structure while reducing computational requirements. The dataset was split at the video level into training (90%) and test (10%) sets, ensuring that all frames from a given simulation pattern remained together in either the training or test set.

Description Variable	**Code Variable**	**Value**	**Notes**
Pattern Generation and Domain Confinement			
s (pixel side length)	`px_cell_side`	7	Size of each cell in the binary mask
Initialization of Filament Network			
W_x (domain width)	`Lx`	40	x-extent of the rectangular domain
W_y (domain height)	`Ly`	40	y-extent of the rectangular domain
ρ (filament density)	`dens1`	50	Base density of filaments
ξ (length scaling)	`xi`	5	Scales mean filament length
Interaction Network and Laplacian Construction			
p_{act} (interaction probability)	`linkact`	1	Fully active elastic coupling
Stochastic Time Evolution			
T_{time} (total time)	`T_tot`	170	Total simulation duration
Δt (time step)	`dt`	0.1	Euler integration step size
l_{rest} (rest length)	`l_rest`	0.05	Equilibrium distance in Hooke's law
K_{Link} (spring constant)	`K_Link`	10	Base elastic coupling strength
γ (drag coefficient)	`gamma`	200	Baseline drag for filament motion
w (filament diameter)	`diam`	1	Diameter affecting drag anisotropy
T_{temp} (temperature)	`Temp`	0.1	Scales thermal noise

Table 2.1: Simulation parameters used for the filament-based microtubule network model.

Model Development and Training The L-PSTE model is designed to predict future states of microtubule network dynamics by encoding microtubule network states into a latent space and modeling their temporal evolution as linear paths in that space. The architecture is made of three submodules: a state encoder, a linear phase-space propagator, and a state decoder. Given an input frame $X_t^{(i)} \in \mathbb{R}^{112\times112\times1}$ at time t for the i-th simulation and a time difference Δt, the model predicts the frame at time $t + \Delta t$, denoted $X_{t+\Delta t}^{(i)}$.

The encoder E maps the input frame to a latent representation: $z_t^{(i)} = E(X_t^{(i)}) \in \mathbb{R}^{64}$. It consists of three convolutional layers with 32, 64, and 128 filters, respectively (3×3 kernels, ReLU activation), each followed by 2×2 max-pooling to reduce spatial dimensions, and a final dense layer with ReLU activation to produce the 64-dimensional latent vector. The linear phase-space propagator Φ_{lin} defines a linear path in the latent space by transforming the scalar time difference Δt into a direction vector: $\Delta t_\Phi = \Phi_{\text{lin}}(\Delta t) \in \mathbb{R}^{64}$, using three dense layers (32, 256, and 64 units, all with linear activation). The latent representation is then updated by moving along this path: $z_t^{(i)} + \Delta t_\Phi$, where Δt_Φ determines the distance proportional to Δt along the linear trajectory in the latent space. The decoder D reconstructs the predicted frame: $\hat{X}_{t+\Delta t}^{(i)} = D(z_t^{(i)} + \Delta t_\Phi) \in \mathbb{R}^{112\times112\times1}$. It starts with a dense layer expanding the 64-dimensional input to a 14×14×128 feature map, followed by four transposed convolutional layers (128, 64, 32, and 1 filters, 3×3 kernels, ReLU activation except for the final sigmoid layer). Between these layers, 2×2 upsampling layers progressively map back to the original spatial dimensions.

The training data was prepared by pairing input frames $X_t^{(i)}$ with target frames $X_{t+\Delta t}^{(i)}$. This pairing process is based on a stochastic mini-batch strategy, shown in Algorithm **??**, to efficiently capture temporal dynamics for different initial patterns without exhaustively pairing every possible t and Δt across all simulations m. The model was trained to minimize the binary cross-entropy loss between predicted and ground truth frames:

$$\mathcal{L}_{\mathrm{BCE}}(X,\hat{X}) \propto -\sum_{j,k}\left[X(j,k)\log\left(\hat{X}(j,k)\right) + (1-X(j,k))\log\left(1-\hat{X}(j,k)\right)\right] \tag{2.13}$$

Here, $X(j,k)$ and $\hat{X}(j,k) \in [0,1]$ represent the ground truth and predicted pixel values at position (j,k). Adam optimizer was used with learning rate set to 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$, and `amsgrad = False`, and training proceeded for 50 epochs, with each epoch sampling 50 random mini-batches of size 256.

Algorithm 2 Stochastic Mini-Batch Loss Calculation for L-PSTE

```
1: Arguments: $B$ (batch size), $T_{\mathrm{time}}$ (total time steps), $\Delta t_{\max}$ (max time difference), $m$ (number
   of simulations)
2: function StochasticBatchLoss($B, T_{\mathrm{time}}, \Delta t_{\max}, m$)        ▷ Loss over random mini-batch
3:     $\mathcal{L}_{\mathrm{batch}} \leftarrow 0$                                  ▷ Total batch loss
4:     for $b = 1$ to $B$ do                                                      ▷ Iterate over batch
5:         $i \sim \mathcal{U}\{1, m\}$                                           ▷ Uniformly random simulation index
6:         $t \sim \mathcal{U}\{1, T_{\mathrm{time}}\}$                           ▷ Uniformly random initial time
7:         $t_{\mathrm{target}} \sim \mathcal{U}\{t, \min(t + \Delta t_{\max}, T_{\mathrm{time}})\}$   ▷ Uniformly random target time
8:         $\Delta t \leftarrow t_{\mathrm{target}} - t$                          ▷ Time difference
9:         $X_t^{(i)} \leftarrow \mathrm{RetrieveImage}(i, t)$                    ▷ Input frame
10:        $X_{t+\Delta t}^{(i)} \leftarrow \mathrm{RetrieveImage}(i, t_{\mathrm{target}})$   ▷ Target frame
11:        $z_t^{(i)} \leftarrow E(X_t^{(i)})$                                    ▷ Latent encoding
12:        $\Delta t_{\Phi} \leftarrow \Phi_{\mathrm{lin}}(\Delta t)$             ▷ Latent time propagation
13:        $\hat{X}_{t+\Delta t}^{(i)} \leftarrow D(z_t^{(i)} + \Delta t_{\Phi})$ ▷ Predicted frame
14:        $\mathcal{L}_{\mathrm{batch}} \leftarrow \mathcal{L}_{\mathrm{batch}} + \mathcal{L}_{\mathrm{BCE}}(X_{t+\Delta t}^{(i)}, \hat{X}_{t+\Delta t}^{(i)})$   ▷ Accumulate BCE loss
15:    end for
16:    Return $\frac{1}{B}\mathcal{L}_{\mathrm{batch}}$                           ▷ Average batch loss
17: end function
```

Evaluation and Predictions We evaluated the L-PSTE model by its ability to predict the temporal evolution of microtubule networks, using both perceptual and quantitative comparisons. For each held-out simulation i, the initial frame $X_0^{(i)} \in \mathbb{R}^{112\times112\times1}$ at time $t = 0$ was passed through the state encoder to obtain a latent representation $z_0^{(i)} = E(X_0^{(i)})$. Predicted frames were then generated for all future time steps $\Delta t \in \{0, 1, \ldots, 169\}$ as:

$$\hat{X}_{t+\Delta t}^{(i)} = D(z_0^{(i)} + \Phi_{\text{lin}}(\Delta t)), \quad t = 0, \tag{2.14}$$

covering the full simulation trajectory of 170 steps. A subset of predicted frames at $\Delta t \in \{0, 50, 100\}$ was visually compared to the corresponding ground truth frames $X_{\Delta t}^{(i)}$, providing qualitative assessment of spatial contraction patterns over time.

To evaluate dynamics quantitatively, we computed the microtubule network area across all 170 time steps for both predicted and ground truth frames. Frames were binarized using Otsu's thresholding method, small objects (fewer than 50 pixels) were removed, and morphological operations (dilation followed by erosion using a disk of radius 2) were applied to refine the segmentation. The microtubule network area at each time step was then calculated as:

$$\mathcal{A}^{(i)}(\Delta t) = \sum_{j,k} \mathbb{I}(X_{\Delta t}^{(i)}(j,k) > \theta_{\text{otsu}}), \tag{2.15}$$

where $\mathbb{I}$ is the indicator function. This procedure yielded sequences of predicted areas $\{\mathcal{A}_{\text{pred}}^{(i)}(\Delta t)\}$ and ground truth areas $\{\mathcal{A}_{\text{true}}^{(i)}(\Delta t)\}$, which were plotted over time to assess temporal agreement between model predictions and true dynamics.

Perceptual accuracy was further quantified using the Feature Similarity Index (FSIM), which measures similarity based on phase congruency and gradient magnitude. FSIM scores were computed for held-out examples by randomly sampling pairs $(t, \Delta t)$ where $t \in \{0, \ldots, 169\}$ and $\Delta t \in \{0, \ldots, 170 - t\}$, consistent with the training procedure. FSIM values range from 0 to 1, with higher values indicating stronger perceptual similarity.

Finally, we assessed the accuracy of area dynamics using the Mean Absolute Percentage Error (MAPE). For each test simulation i, the predicted and ground truth areas were computed as above, and the MAPE was calculated as:

$$\mathrm{MAPE}^{(i)} = \frac{1}{T} \sum_{\Delta t=0}^{T-1} \left| \frac{\mathcal{A}_{\mathrm{pred}}^{(i)}(\Delta t) - \mathcal{A}_{\mathrm{true}}^{(i)}(\Delta t)}{\mathcal{A}_{\mathrm{true}}^{(i)}(\Delta t)} \right| \times 100, \tag{2.16}$$

where $T = 170$. The mean MAPE across all test simulations provided a quantitative measure of the model's ability to track the temporal evolution of microtubule network areas.

G-PSTE on Experimental Data

Dataset Generation and Processing The General Phase Space Trajectory Encoder (G-PSTE) was trained on experimental data collected from 96 parallel microtubule network experiments using our high-throughput microscopy platform. Each imaging cycle captured 8 positions, with 12 microtubule networks imaged per position using a 3×4 tiling scheme. Fluorescence images of Cy5-labeled microtubules were acquired at 16-bit depth and 2048×2048 resolution. Since light stimulation only affected a 1470×2048 subregion, each microscopy image was cropped accordingly and then divided into 12 tiles of size 490×512 pixels, each representing a distinct experiment.

Light stimulation data were extracted from the final projector input frame (800×1280 pixels). The effective stimulation region occupied a centered 800×1116 area, with 82 pixels of black padding on each side to meet projector input requirements. This stimulation region was divided into a 3×4 grid of 12 tiles, each measuring 267×279 pixels. These tiles defined the light patterns used to guide microtubule network organization and were extracted to align precisely with the corresponding microscopy recordings.

To accommodate experiments of varying duration, we identified the number of valid frames n_i for each experiment i by detecting periods with active light stimulation. Both microscopy and light stimulation tiles were resized to 112×112 pixels. Microscopy tiles were normalized independently using min-max scaling to the range $[0, 1]$ to ensure compatibility with neural network training. Light stimulation tiles were further processed to incorporate temporal context: given a light pattern $\Lambda_t^{(i)} \in \mathbb{R}^{112\times112}$ applied to tile i at time t, a temporally aggregated stimulation frame was computed using an exponential discount factor $\delta = 0.95$:

$$\bar{\Lambda}_t^{(i)} = \sum_{s=0}^{t} \Lambda_s^{(i)} \delta^{t-s}, \quad \text{for } t = 0, 1, \ldots, n_i - 1 \tag{2.17}$$

This transformation embedded memory of past stimuli while emphasizing more recent inputs. Each resulting frame $\bar{\Lambda}_t^{(i)}$ was then normalized by its maximum pixel value to ensure consistent dynamic range across time and experiments.

The 96 experiments were randomly split into training (84%) and test (16%) sets at the experiment level, with all frames from a given experiment assigned to the same partition to prevent data leakage.

Model Development and Training The G-PSTE model is designed to predict microtubule network configurations in response to temporally evolving sequences of spatial light inputs. For each experiment i, the evolution of the network is observed as a trajectory of microscopy images $\{X_1^{(i)}, X_2^{(i)}, \ldots, X_T^{(i)}\}$, where $X_t^{(i)} \in \mathbb{R}^{112\times112\times1}$. This trajectory is driven by a corresponding sequence of binary light input patterns $\{\Lambda_1^{(i)}, \Lambda_2^{(i)}, \ldots, \Lambda_T^{(i)}\}$, with $\Lambda_t^{(i)} \in \mathbb{R}^{112\times112}$. At each time point, a temporally aggregated input $\bar{\Lambda}_t^{(i)} \in \mathbb{R}^{112\times112}$ is computed using an exponential discounting scheme, embedding recent stimulation history into a single frame.

The model architecture consists of three submodules: a state encoder E, a light-driven latent space propagator Φ_{light}, and a state decoder D. The encoder maps each microscopy frame to a 64-dimensional latent vector, yielding a trajectory in latent space $\{z_1^{(i)}, z_2^{(i)}, \ldots, z_T^{(i)}\}$, where

$z_t^{(i)} = E(X_t^{(i)}) \in \mathbb{R}^{64}$. These latent vectors capture the internal state of the network across time in a compact representation. In parallel, the propagator maps each aggregated light input to a corresponding point in the same latent space, producing a light-induced trajectory $\{\psi_1^{(i)}, \psi_2^{(i)}, \ldots, \psi_T^{(i)}\}$, where $\psi_t^{(i)} = \Phi_{\text{light}}(\bar{\Lambda}_t^{(i)}) \in \mathbb{R}^{64}$. This trajectory approximates the system's latent dynamics induced purely by the light input history.

The decoder transforms the light-driven latent trajectory back into predicted microscopy frames $\{\hat{X}_1^{(i)}, \hat{X}_2^{(i)}, \ldots, \hat{X}_T^{(i)}\}$, where $\hat{X}_t^{(i)} = D(\psi_t^{(i)}) \in \mathbb{R}^{112\times112\times1}$. This decoding process enables the model to predict the observable evolution of the microtubule network directly from light input sequences, without requiring explicit access to prior network states.

The encoder consists of three convolutional blocks with 32, 64, and 128 filters (all using 3×3 kernels and ReLU activations), each followed by 2×2 max-pooling. The final feature map is flattened and passed through a dense layer with 64 units. The propagator consists of seven Conv2D layers: the first four with 64 filters, followed by two with 128 and one with 256 filters (each with 3×3 kernels and ReLU activations), with 2×2 max-pooling after each layer. The output is flattened and passed through a dense layer to produce a 64-dimensional vector. The decoder begins with a dense layer expanding the latent vector to a $14 \times 14 \times 128$ feature map, followed by four transposed convolutional layers with 128, 64, 32, and 1 filters (all using 3×3 kernels). ReLU activations are used throughout except in the final layer, which uses a sigmoid activation. 2×2 upsampling layers are applied between each transposed convolution to progressively restore the spatial resolution.

Training was performed in two phases using mini-batches of experiments. For a batch B of size $|B|$, the encoder and decoder were first trained together to reconstruct full microscopy trajectories.

The reconstruction loss $\mathcal{L}_{\text{recon}}$ was defined as the average binary cross-entropy:

$$\mathcal{L}_{\text{recon}} = \frac{1}{|B|} \sum_{i \in B} \sum_{t=1}^{T} \mathcal{L}_{\text{BCE}}(X_t^{(i)}, \hat{X}_t^{(i)}), \quad \text{with } \hat{X}_t^{(i)} = D(E(X_t^{(i)})) \tag{2.18}$$

After pretraining, the encoder and decoder were frozen, and the propagator was trained to align the light-driven latent trajectory $\{\psi_t^{(i)}\}$ with the encoder-derived trajectory $\{z_t^{(i)}\}$, using the batch-averaged mean squared error:

$$\mathcal{L}_{\text{path}} = \frac{1}{|B|} \sum_{i \in B} \sum_{t=1}^{T} \left\| z_t^{(i)} - \psi_t^{(i)} \right\|^2 \tag{2.19}$$

Both training phases used the Adam optimizer with a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$, and `amsgrad = False`. The autoencoder was trained for 50 epochs and the light propagator for 15 epochs, each with a batch size of 256.

Evaluation and Predictions We assessed the G-PSTE model's ability to predict microtubule network dynamics under dynamic light stimulation using qualitative visualizations and quantitative metrics. For each test experiment i in the held-out set (16% of 96 experiments), predictions were generated by processing the light stimulation sequence $\{\Lambda_t^{(i)}\}_{t=0}^{n_i-1}$ through the trained model to produce a trajectory of predicted microscopy frames

$$\hat{X}_t^{(i)} = D(\Phi_{\text{light}}(\bar{\Lambda}_t^{(i)})) \in \mathbb{R}^{112 \times 112 \times 1}, \tag{2.20}$$

where $\bar{\Lambda}_t^{(i)}$ is the temporally aggregated input computed as previously described, and n_i is the experiment-specific duration.

For qualitative assessment, we selected three representative time points ($t = 80\,\text{s},\ 200\,\text{s},\ 380\,\text{s}$) from each test experiment and visually compared the predicted frames $\hat{X}_t^{(i)}$ to the corresponding ground truth frames $X_t^{(i)}$, inspecting the spatial organization and morphology of moving microtubule networks.

To quantitatively evaluate the model's ability to capture the dynamics of moving microtubule networks, we tracked their centroid positions in both predicted and ground truth frames. Network centroids were localized using a two-step process. First, the center of mass was computed using image moments:

$$\bar{x}^{(i)}(t) = \frac{\sum_{j,k} j \cdot X_t^{(i)}(j,k)}{\sum_{j,k} X_t^{(i)}(j,k)}, \quad \bar{y}^{(i)}(t) = \frac{\sum_{j,k} k \cdot X_t^{(i)}(j,k)}{\sum_{j,k} X_t^{(i)}(j,k)}, \tag{2.21}$$

providing an initial estimate of the network centroid in pixel coordinates. This was refined by fitting a 2D Gaussian function to the intensity distribution around the estimated center, approximating the spatial extent of the moving network. The Gaussian model is defined as:

$$g(x,y;\Theta) = \text{offset} + \alpha \exp\left[-(a(x-x_0)^2 + 2b(x-x_0)(y-y_0) + c(y-y_0)^2)\right], \tag{2.22}$$

where $\Theta = (\text{offset}, \alpha, x_0, y_0, a, b, c)$ is the parameter vector:

- offset: Background intensity,
- α: Amplitude,
- (x_0, y_0): Centroid coordinates of the network,
- a, b, c: Shape coefficients, related to Gaussian widths σ_x, σ_y and rotation angle θ by:

$$a = \frac{\cos^2\theta}{2\sigma_x^2} + \frac{\sin^2\theta}{2\sigma_y^2}, \quad b = -\frac{\sin 2\theta}{4\sigma_x^2} + \frac{\sin 2\theta}{4\sigma_y^2}, \quad c = \frac{\sin^2\theta}{2\sigma_x^2} + \frac{\cos^2\theta}{2\sigma_y^2}. \tag{2.23}$$

The optimization problem for fitting this Gaussian to image data $X(x,y)$ (e.g., $X_t^{(i)}$ or $\hat{X}_t^{(i)}$) is formulated as:

$$\Theta^* = \arg\min_{\Theta} \sum_{j,k} [X(j,k) - g(j,k;\Theta)]^2, \tag{2.24}$$

This nonlinear least-squares problem is solved using the Levenberg-Marquardt algorithm, initialized with the center of mass $(\bar{x}^{(i)}(t), \bar{y}^{(i)}(t))$ and approximate values for α (maximum intensity minus minimum intensity) and offset (minimum intensity), with initial a, b, c estimated assuming isotropy ($\theta = 0, \sigma_x = \sigma_y$). The resulting (x_0^*, y_0^*) provides the refined centroid of the moving microtubule network.

Pixel coordinates were converted to physical units (μm) using calibration factors: 1.70625 μm/pixel (x-axis) and 1.78286 μm/pixel (y-axis), derived from the original 2048×2048 resolution (0.39 μm/pixel) and resizing to 112×112 (scaling factors 490/112 = 4.375 and 512/112 = 4.571).

The Euclidean displacement error (DE) was computed to assess positional accuracy of the network centroids across time:

$$\mathrm{DE}^{(i)}(t) = \sqrt{(x^{(i)}_{0,\mathrm{pred}}(t) - x^{(i)}_{0,\mathrm{true}}(t))^2 + (y^{(i)}_{0,\mathrm{pred}}(t) - y^{(i)}_{0,\mathrm{true}}(t))^2}, \tag{2.25}$$

and the median DE across all time steps and test experiments was calculated as:

$$\text{MEDIAN-DE} = \text{median}\left\{\mathrm{DE}^{(i)}(t) \mid i = 1, \ldots, M,\ t = 0, \ldots, n_i - 1\right\}, \tag{2.26}$$

where M is the number of test experiments. This metric quantifies the model's precision in predicting the movement of microtubule networks in response to light stimuli.

We also calculated the mean path length of the microtubule network trajectories to evaluate the extent of motion. For each experiment i, the path length $L^{(i)}$ was computed as the sum of Euclidean distances between consecutive centroid positions:

$$\mathrm{PathLen}(i) = \sum_{t=1}^{n_i-1} \sqrt{(x^{(i)}_0(t) - x^{(i)}_0(t-1))^2 + (y^{(i)}_0(t) - y^{(i)}_0(t-1))^2}, \tag{2.27}$$

and the mean path length across all test experiments was reported as $\frac{1}{M}\sum_{i=1}^{M} \mathrm{PathLen}(i)$.
This metric provides insight into the total displacement of the microtubule networks over time.

Perceptual similarity between predicted and ground truth frames was further quantified using the Feature Similarity Index (FSIM). For each test experiment i, FSIM scores were computed across all time steps:

$$\mathrm{FSIM}^{(i)}(t) = \mathrm{FSIM}(X^{(i)}_t, \hat{X}^{(i)}_t), \tag{2.28}$$

ranging from 0 to 1, with higher values indicating greater similarity in phase congruency and gradient magnitude. The mean FSIM across all experiments and time steps was reported as:

$$\text{MEAN-FSIM} = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{n_i}\sum_{t=0}^{n_i-1}\text{FSIM}^{(i)}(t), \tag{2.29}$$

providing a robust measure of perceptual fidelity.

Finally, to assess the alignment of latent space trajectories, we compared the encoder-derived latent vectors $z_t^{(i)} = E(X_t^{(i)})$ with the light-driven latent vectors $\psi_t^{(i)} = \Phi_{\text{light}}(\bar{\Lambda}_t^{(i)})$. Both sequences were projected into a 2D space using Principal Component Analysis (PCA), retaining the top two principal components. The trajectory divergence was quantified using the mean Euclidean distance in the reduced space:

$$\text{TD}^{(i)} = \frac{1}{n_i}\sum_{t=0}^{n_i-1}\left\|\text{PCA}(z_t^{(i)}) - \text{PCA}(\psi_t^{(i)})\right\|_2, \tag{2.30}$$

and averaged across test experiments:

$$\text{MEAN-TD} = \frac{1}{M}\sum_{i=1}^{M}\text{TD}^{(i)}. \tag{2.31}$$

This metric evaluates how well the light-driven propagator captures the latent dynamics of the microtubule network, with smaller values indicating tighter alignment (Fig. 2B). Together, these quantitative measures—MEDIAN-DE, MEAN-L, MEAN-FSIM, and MEAN-TD—provide a comprehensive assessment of G-PSTE's predictive accuracy and latent space consistency.

Optimization of Experimental Conditions for Microtubule Bending

Formulation of the Bending Loss Function

Loss Definition We define the bending loss function ($\mathcal{L}_{\text{bend}}$) as the minimum mean squared error (MSE) between the final observed state and a transformed version of the initial state:

$$\mathcal{L}_{\text{bend}} = \min_{\theta}\text{MSE}(S(\mathbf{X}_0, \theta), \mathbf{X}_{\text{final}}) \tag{2.32}$$

Here, $\mathbf{X}_0$ represents the initial configuration of the microtubule network, $\mathbf{X}_{\text{final}}$ is the observed final state, and $S(\mathbf{X}_0, \theta)$ denotes a geometric transformation parameterized by $\theta = (\lambda, T_x, T_y)$. The parameter θ consists of uniform scaling and translations, where λ controls the isotropic scaling of the network, and T_x and T_y define translations along the x- and y-axes, respectively.

Transformation Implementation To compute the bending loss in practice, we preprocess the microtubule network configurations by converting grayscale images of the initial and final states into binary masks using a predefined threshold. The initial state ($\mathbf{X}_0$) is then transformed using a spatial transformer network (STN) parameterized by θ. STNs provide a differentiable module that applies spatial transformations in a single forward pass, facilitating smooth optimization. We optimize θ using the Adam optimizer, which enhances convergence by combining adaptive learning rates with momentum-based updates. Gradient computation is handled via TensorFlow's automatic differentiation, enabling efficient backpropagation through the spatial transformer module. Constraints are imposed during optimization to restrict the transformation to isotropic scaling and translations, preventing rotation and non-uniform scaling. The optimization process terminates when the loss change falls below a convergence criterion of, indicating stagnation within a defined tolerance. At this point, the final values of θ represent the best-fit transformation parameters, quantifying the deviation from uniform contraction in the microtubule network.

Implementation of the Neural Network-Based Optimization Framework

Implementation Overview We implemented our optimization framework in TensorFlow, leveraging custom neural network architectures for both the affine transformation and the microtubule deformation prediction. The framework integrates three key components: an Affine Transformation Neural Network (ATNN), a Temporal Convolutional Autoencoder (L-PSTE), and the Spatial Transformer Network (STN).

Affine Transformation Neural Network (ATNN) The ATNN is a sequential model composed of two dense layers with 64 neurons and ReLU activation, followed by an output layer that produces the six parameters of the affine transformation matrix:

$$\theta = [a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}] \tag{2.33}$$

where a_{ij} represents the elements of the 2×3 affine transformation matrix. This network takes a 16-dimensional binary input vector and outputs the transformation parameters that map the initial state to the simulated final state.

ATNN Training Procedure The ATNN was trained on a dataset of pattern-transformation pairs where each transformation parameter θ was computed using the bending loss optimization method described in the previous section. Specifically, for each pattern in our dataset, we had already determined the optimal $\theta = [a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}]$ that minimizes $\mathcal{L}_{\text{bend}}$ between initial and final configurations. These optimal transformation parameters, along with their corresponding binary patterns, were compiled into a CSV file to create a supervised learning dataset. We split this dataset into training (90%) and testing (10%) sets, and trained the model using the Adam optimizer with a learning rate of 10^{-3} for 50 epochs with a batch size of 128. The model was trained to minimize the mean squared error between predicted and ground truth transformation parameters, with mean absolute error as an additional metric. This approach effectively transfers the knowledge from our optimization-based bending loss formulation into a neural network that can rapidly predict appropriate transformation parameters for new patterns.

Learned Physical Simulation with L-PSTE The L-PSTE serves as our physical simulator, predicting microtubule network configurations at arbitrary timepoints. This model processes both spatial information (microtubule configurations) and temporal information (simulation time steps) to generate physically plausible deformation trajectories.

Optimization Procedure For optimization, we employed a gradient-based approach using TensorFlow's automatic differentiation capabilities. The optimization procedure follows this algorithm:

Algorithm 3 Microtubule Network Bending Optimization

```
1: Initialize $n = 200$ random input vectors $\mathbf{v} \in \mathbb{R}^{16}$ with values in $[-1, 1]$
2: Set time step $dt = 150$ and learning rate $\eta = 10^2$
3: Initialize Adam optimizer with learning rate $\eta$
4: for step = 1 to max_steps do
5:     $\mathbf{v}_{bin} \leftarrow \sigma(\mathbf{v})$                                ▷ Apply sigmoid to get binary-like values
6:     $\theta \leftarrow \text{ATNN}(\mathbf{v}_{bin})$                              ▷ Get affine parameters
7:     $\mathbf{I} \leftarrow \text{Reshape}(\mathbf{v}_{bin})$ to $4 \times 4 \times 1$                  ▷ Create image
8:     $\mathbf{I} \leftarrow \text{Upsample}(\mathbf{I})$ to $112 \times 112 \times 1$
9:     $\mathbf{I}_{affine} \leftarrow \text{STN}(\mathbf{I}, \theta)$                         ▷ Apply predicted transformation
10:    $\mathbf{I}_{L-PSTE} \leftarrow \text{L-PSTE}(\mathbf{I}, dt)$                     ▷ Generate simulated final state
11:    $\mathcal{L} \leftarrow \text{MSE}(\mathbf{I}_{affine}, \mathbf{I}_{L-PSTE})$                  ▷ Compute loss
12:    $\nabla_{\mathbf{v}}\mathcal{L} \leftarrow \text{Gradient}(\mathcal{L}, \mathbf{v})$                    ▷ Compute gradients
13:    $\mathbf{v} \leftarrow \text{Adam}(\mathbf{v}, \nabla_{\mathbf{v}}\mathcal{L})$                       ▷ Update input vectors
14: end for
15: return optimized input vectors $\mathbf{v}$
```

This process iterates for a predetermined number of steps (max_steps = 5), with the loss monotonically decreasing as the optimization progresses. The resulting optimized input vectors and their corresponding affine transformations represent the best match between geometrically transformed initial states and physically simulated final states, thus quantifying the deviation from uniform contraction.

Reinforcement Learning for Real-Time Object Capture

Simulation Environment for RL Agent Training

We developed a Gym-based simulation environment called AM_GAME to train our reinforcement learning (RL) agent for active microtubule network manipulation and object capture. The environment was implemented using the OpenAI Gym framework and turtle graphics for visualization, providing a configurable platform for the agent to learn control strategies.

The simulation models a 2D arena of 20×20 units where an activation region (representing the active microtubule network) must capture multiple bead-like objects. The state space consists of a simplified representation encoding relative position of the activation region with respect to the closest uncaptured bead (4 binary values indicating whether the bead is above, right, below, or left of the activation region), proximity detection of depletion regions (4 binary values detecting nearby depleted areas in cardinal directions), wall detection (4 binary values indicating proximity to arena boundaries), and movement direction (4 binary values for up, right, down, left).

The action space is discrete with four possible actions corresponding to movements in cardinal directions (up, right, down, left). When the agent selects an action, the activation region moves in the corresponding direction, leaving behind a depletion region that represents areas where the microtubule network has already contracted and cannot be reactivated. Episodes terminate when the agent either collides with a wall, collides with a depletion region, captures all beads, or exceeds the maximum time steps (120 by default).

Reward Function Design

The reward function incentivizes efficient capture behavior while penalizing collisions. The agent receives +15 reward for each newly captured bead (beads within 40 units of the activation region), +3 reward when moving closer to the nearest uncaptured bead, and -2 penalty when moving away from the nearest uncaptured bead. Significant penalties of -100 are applied for colliding with walls or previously created depletion regions, and -50 for timeout (when the time counter reaches zero). Additionally, a bonus reward of (remaining time $\times$ 100) is granted for successfully capturing all beads.

This reward structure creates a challenging learning problem where the agent must develop strategies to efficiently navigate the arena while managing the creation of depletion regions that can block future movements. The temporal constraints and spatial limitations require the agent to balance exploration with exploitation, planning paths that optimize bead capture while avoiding self-imposed navigational constraints from depletion regions.

Training Algorithm for RL Agent

We implemented a Deep Q-Network (DQN) approach to train our agent within the AM_GAME environment. The DQN architecture employs experience replay and a target network to stabilize learning, enabling the agent to develop effective strategies for navigating the complex state space while maximizing cumulative rewards.

Algorithm 4 DQN for Active Microtubule Network Control

```
1: Initialize replay memory $\mathcal{D}$ to capacity $N$
2: Initialize action-value function $Q$ with random weights $\theta$
3: Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
4: Initialize exploration parameter $\epsilon = 1.0$
5: for episode = 1 to $M$ do
6:     Reset environment to obtain initial state $s_1$
7:     for $t = 1$ to $T_{max}$ do
8:         With probability $\epsilon$ select random action $a_t$
9:         Otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
10:        Execute action $a_t$, observe reward $r_t$ and next state $s_{t+1}$
11:        Store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathcal{D}$
12:        Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1}, d_j)$ from $\mathcal{D}$
13:        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
14:        Perform gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to $\theta$
15:        Update exploration parameter $\epsilon = \max(\epsilon \cdot \epsilon_{decay}, \epsilon_{min})$
16:        if $t \mod C == 0$ then
17:            Update target network parameters $\theta^- = \theta$
18:        end if
19:        if terminal state reached then
20:            Break
21:        end if
22:    end for
23: end for
```

Our DQN implementation for AM_GAME used a neural network architecture consisting of three hidden layers, each with 128 neurons and ReLU activation functions. The output layer maps to the four possible actions (corresponding to movement in cardinal directions) with a softmax activation function. The network was trained using the Adam optimizer with a learning rate of 2.5×10^{-4}. The experience replay buffer stored up to 2500 transitions, from which minibatches of size 500 were randomly sampled during training.

The hyperparameters were tuned specifically for the AM_GAME environment: discount factor $\gamma = 0.95$, initial exploration rate $\epsilon = 1.0$, minimum exploration rate $\epsilon_{min} = 0.01$, and exploration decay rate $\epsilon_{decay} = 0.995$. This configuration allowed the agent to initially explore the state space thoroughly before gradually transitioning to exploitation of learned strategies.

The training process consisted of 1000 episodes, with each episode terminated either when the agent reached a terminal state (collision with walls or depletion regions, capturing all beads) or after reaching the maximum number of steps (10000). The sum of rewards for each episode was recorded to track the agent's learning progress. The weights of the trained model were saved for later deployment and evaluation.

Chapter 3

BUILDING GENERALIST AGENTS BY MAPPING THE GEOMETRY OF WEIGHT SPACE

ABSTRACT

A central challenge in artificial intelligence (AI) is developing generalized agents capable of high performance across diverse tasks while dynamically adapting strategies based on task requirements. While reinforcement learning (RL) agents have achieved superhuman performance on specific tasks, the principles governing generalized agents remain elusive. Key questions include how tasks are encoded in model weights and how parameter geometry enables strategy diversification. Here, we train RL agents on 13 Atari games, revealing that task-specific agents occupy distinct, non-overlapping regions in weight space, with intra-task clusters encoding strategy diversification. By analyzing parameter distributions, we construct a generalized agent that dynamically adjusts parameters based on its visual input, achieving high performance across all tasks. This work provides fundamental insights into the relationship between model parameters, task encoding, and strategy diversification, offering a scalable strategy for building generalized AI agents.

3.1 Introduction

The pursuit of artificial intelligence (AI) systems capable of seamless adaptation across a wide array of tasks remains a cornerstone of modern research. While reinforcement learning (RL) has driven remarkable achievements in specialized domains, such as mastering complex games (Volodymyr Mnih, Kavukcuoglu, Silver, Graves, et al., 2013; Oh, Chockalingam, H. Lee, et al., 2016; Schaul et al., 2015; Volodymyr Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015), these successes are often confined to narrow contexts, with agents struggling to generalize beyond their training environments (Cobbe et al., 2019). Early strategies to overcome this limitation, including policy distillation (Rusu et al., 2015) and imitation learning (Ho and Ermon, 2016), sought to integrate multiple task-specific policies into a unified model. For example, policy distillation consolidates the behavior of several expert policies into a single student network by training it to replicate their outputs, as demonstrated in deep Q-network applications for Atari games. Similarly, imitation learning uses expert demonstrations to guide policy development, mitigating the need for exhaustive exploration. However, these efforts frequently encountered challenges, including catastrophic forgetting and degraded performance due to task interference.

Recent advances in multi-task learning have sought to overcome these barriers by training a single model across diverse environments simultaneously. Methods like IMPALA (Espeholt et al., 2018) and PopArt (Hessel et al., 2019) scale RL to multiple tasks by normalizing rewards and leveraging shared representations, achieving robust performance on benchmarks like Atari (Marc G Bellemare et al., 2013). While such methods expand task coverage, they often compromise on performance, as competing objectives within a shared parameter space can erode task-specific expertise (Parisotto, Ba, and Salakhutdinov, 2015). Meanwhile, sequence modeling architectures that integrate modalities like vision, control, and language have pushed the boundaries of generalization by leveraging vast datasets and computational scale. For instance, models like Gato (Reed et al., 2022) and Decision Transformer (Chen et al., 2021) leverage large datasets to perform diverse control tasks, such as Atari and robotics, using prompt conditioning or trajectory context for task disambiguation. However, these models struggle to adapt strategies within tasks without external input or fine-tuning.

In this study, we introduce a novel approach that harnesses the inherent geometry of neural parameter space to enable flexible generalization. By training independent RL agents on 13 Atari games within a consistent architecture, we observe that task-specific agents form distinct, non-overlapping clusters in weight space, with intra-cluster variations reflecting diverse strategic approaches. Leveraging this organization, we develop a generalized agent that dynamically modulates its parameters based on visual inputs, achieving robust performance across tasks without the need for joint optimization. Our method defines a continuous trajectory through normalized weight space, enabling smooth transitions between tasks and strategies. This framework not only advances the design of adaptable AI systems but also deepens our understanding of how neural parameters encode task identity and behavioral diversity.

3.2 Results

Population Training Reveals Task-Specific Differentiation of Neural Networks.

To examine how functional variation arises during training, we constructed large populations of RL agents, each trained independently on one of 13 Atari 2,600 games (M. G. Bellemare et al., 2013) using a shared convolutional policy architecture based on the DeepMind Atari DQN agent (Volodymyr Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015)(Fig. 3.1A). Agents received only raw pixel input and scalar reward; no architectural adaptation or multitask supervision was used. Each agent was trained for 1 million steps using proximal policy optimization (PPO), a widely used policy gradient method for reinforcement learning. All agents were initialized from the same distribution using orthogonal weight initialization and shared identical hyperparameters, including learning rate, batch size, and optimization schedule. In total, we trained approximately 15,000 agents (1,152 per game) under this protocol, and represented each by a high-dimensional vector formed by concatenating all trainable weights in the policy network.

Despite this standardized training protocol, we observed striking divergence in how agents evolved across games. For nine representative tasks, we sampled 100 checkpoints uniformly spaced in training time along the learning trajectories of 16 agents per game. At each checkpoint, we extracted the full policy network weights and projected them into a two-dimensional Uniform Manifold Approximation (UMAP) space (Fig. 3.1B). Each agent's trajectory began from a shared origin but soon branched into a distinct path, with different games producing consistently separate regions of weight space. Agents trained on the same game followed nearly identical routes, whereas those trained on different games diverged rapidly and did not overlap, forming task-specific branches in the learned landscape.

This differentiation was accompanied by reliable improvements in performance, with reward curves increasing steadily within each game (Fig. 3.1C). Each task induced distinct training dynamics across its agent population: some games, such as Galaxian and Pooyan, showed tightly clustered reward trajectories with consistent early gains and similar final performance, while others, like Qbert, began uniformly but diverged widely in final reward. Together, these results show that task structure not only guides the evolution of network weights, but also shapes how behavioral performance emerges and stabilizes over time.

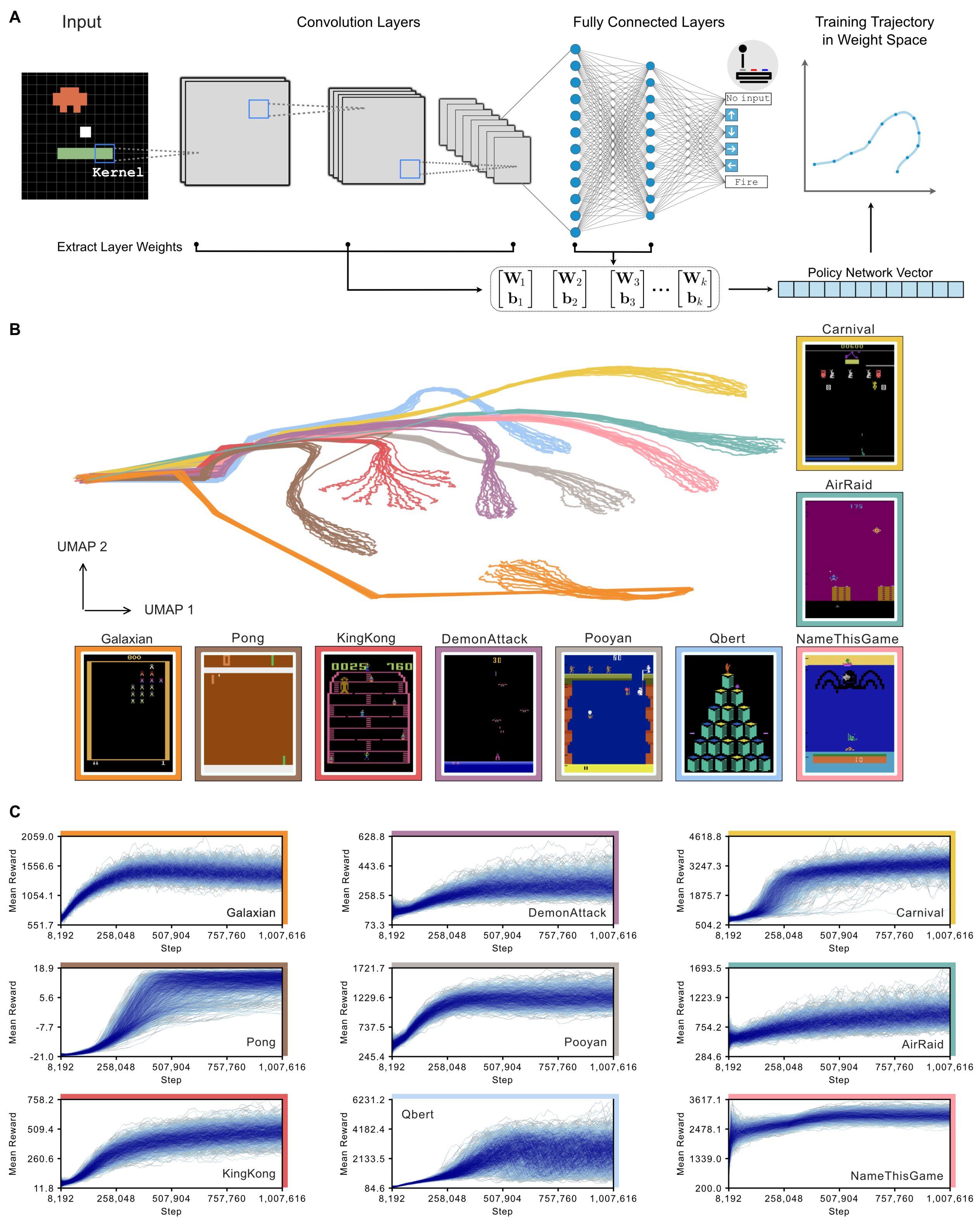

Figure 3.1: **Policy Evolution in Weight Space and Training Dynamics Across Atari Tasks.** **A.** Convolutional neural network (CNN) policy architecture used for all agents. Weights from all layers are concatenated into a high-dimensional Policy Network Vector. **B.** UMAP projection of Policy Network Vectors from 15 agents per game, shown for 9 of 13 total games. Each trajectory includes 100 checkpoints sampled uniformly over 1 million training steps. Colors indicate game identity and match the screenshots. **C.** Reward over training steps for 1152 agents per game. Each line represents a single agent; colors match panel (B).

Task Pressures Sculpt Distinct Functional Territories in Neural Weight Space

After training, the final policy network weights revealed persistent task-specific organization in high-dimensional weight space. To examine this structure, we compiled the final weight vectors from all 14,976 agents and visualized a fixed subset of 10,000 weights per agent, selected for high average contribution to top principal components. We plotted standardized magnitudes in a heatmap (Fig. 3.2A), with columns (weights) ordered by hierarchical clustering. This revealed structured, task-dependent patterns: agents grouped by game showed reproducible weight profiles despite independent initialization and training. Specific weight subsets were consistently modulated across tasks, shifting toward positive or negative values in a task-dependent manner. For example, weights in the ranges 0-2,000 and 8,000-10,000 formed repeated block-like structures that appeared across all games, but with polarity and magnitude systematically modulated by task. These results suggest the emergence of functional substructures, where shared representational components are repurposed across environments.

To visualize the geometric structure of trained policy networks, we projected all 14,976 full policy weight vectors into lower dimensions using principal component analysis (PCA) and UMAP. Both methods revealed clear task-level structure: agents trained on the same game formed dense, coherent clusters, while those from different games occupied well-separated regions of weight space (Fig. 3.2B-C). A corresponding fitness landscape projected into UMAP space (Fig. 3.2C) further revealed that reward was locally concentrated within these task-specific regions. To quantitatively validate this organization, we applied unsupervised Louvain community detection to a nearest-neighbor graph constructed in the high-dimensional PCA space. Without access to task labels, the resulting clusters aligned almost perfectly with the agents' training environments, achieving 99.95% clustering accuracy, an adjusted Rand index of 0.9988, and normalized mutual information of 0.9985. These results confirm that task identity is deeply encoded in the learned weights and is recoverable through purely geometric structure.

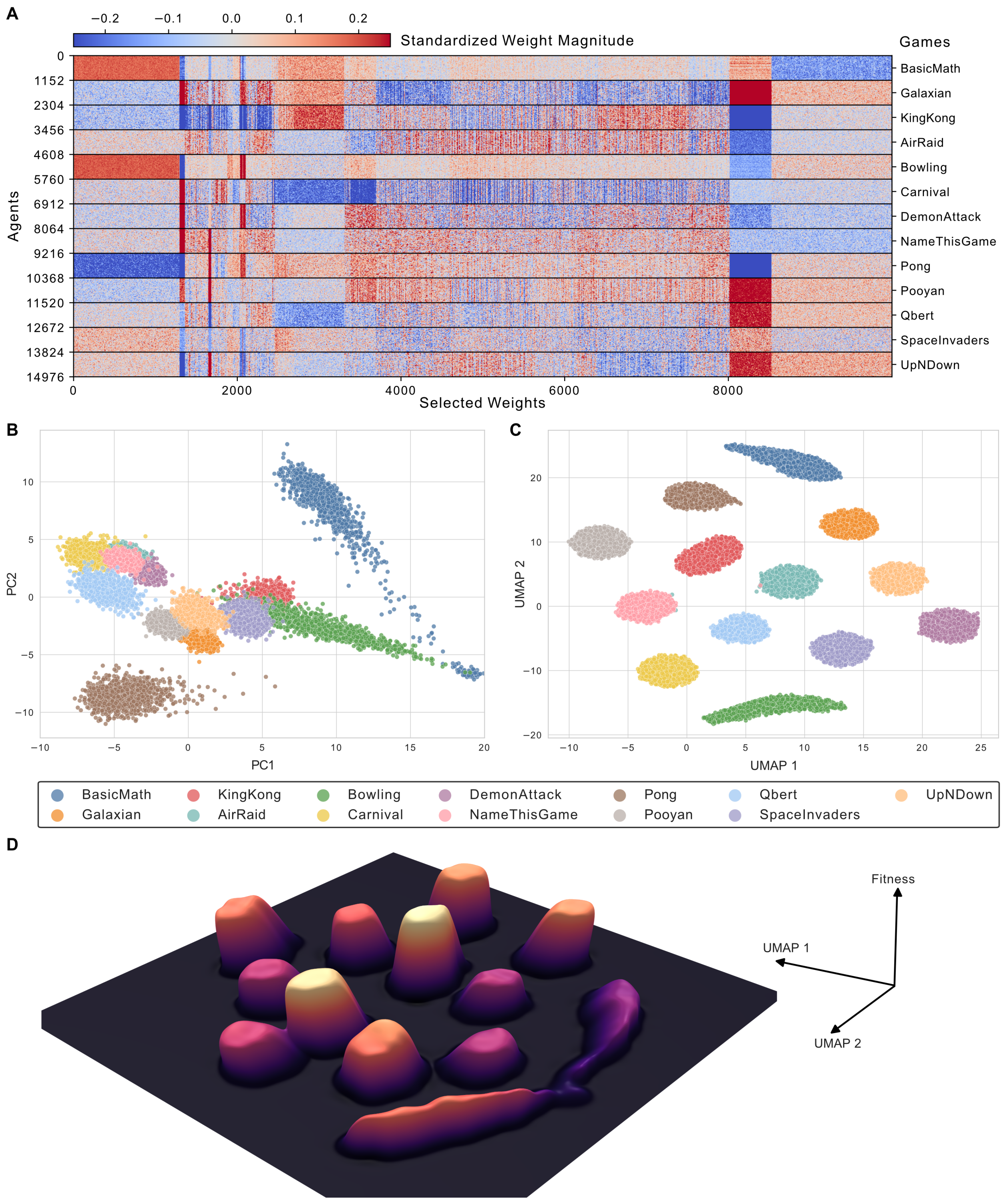

Figure 3.2: **Structure of policy weight space across Atari games after training. A.** Heatmap of standardized magnitudes for a subset of weights from 14,976 agents trained on 13 Atari games, each shown at 1 million training steps. Agents are grouped by game (rows), and weights (columns) are ordered via hierarchical clustering. **B.** Principal component analysis (PCA) of policy weight vectors, colored by game. **C.** UMAP projection of the same weight vectors, revealing clear task-specific clustering. **D.** Fitness landscape over UMAP space, where peak height corresponds to average agent performance in each region.

Strategic Tradeoffs Map Onto the Internal Geometry of Functional Territories in Weight Space.

While agents trained on the same task converged to compact regions of weight space, we observed meaningful behavioral diversity within each task-specific cluster. In particular, agent populations spontaneously explored different tradeoffs between reward maximization and risk, measured as the standard deviation of episodic returns. Plotting agents in behavior space (mean reward vs. risk) revealed continuous spectra spanning conservative to risk-seeking strategies (Fig. 3.3A, left). To identify representative behavioral phenotypes, we used unsupervised k-means clustering (k = 2), allowing phenotypes to emerge from the population without supervision. These tradeoffs emerged across games, indicating that reinforcement learning does not produce a single optimal behavior but a distribution of viable policies shaped by the task's structure.

To test whether this behavioral variation corresponded to structure in the learned parameters, we projected the same agents' policy weight vectors using PCA within each game. In all six examples shown, the two phenotypes gave rise to distinct distributions in the PCA-projected weight space within their game's territory (Fig. 3.3A, right), suggesting that internal geometry of each task cluster organizes strategic diversity. This risk-reward trade-off was also evident in actual gameplay. In Galaxian, for instance, conservative agents tended to remain on the left side of the screen and restricted their movement, which limited enemy engagement but also reduced exposure to incoming projectiles. In contrast, risk-seeking agents explored a broader portion of the playable area, positioning themselves more aggressively to clear enemies faster, at the cost of greater vulnerability (Fig. 3.3B-D). These differences in gameplay style emerged naturally from training, without supervision or explicit reward shaping. Together, these results show that learned weight space is not only segmented by task but also structured within tasks to reflect interpretable behavioral axes. Each functional territory supports a spectrum of strategies, with internal geometry encoding tradeoffs between competing behavioral modes.

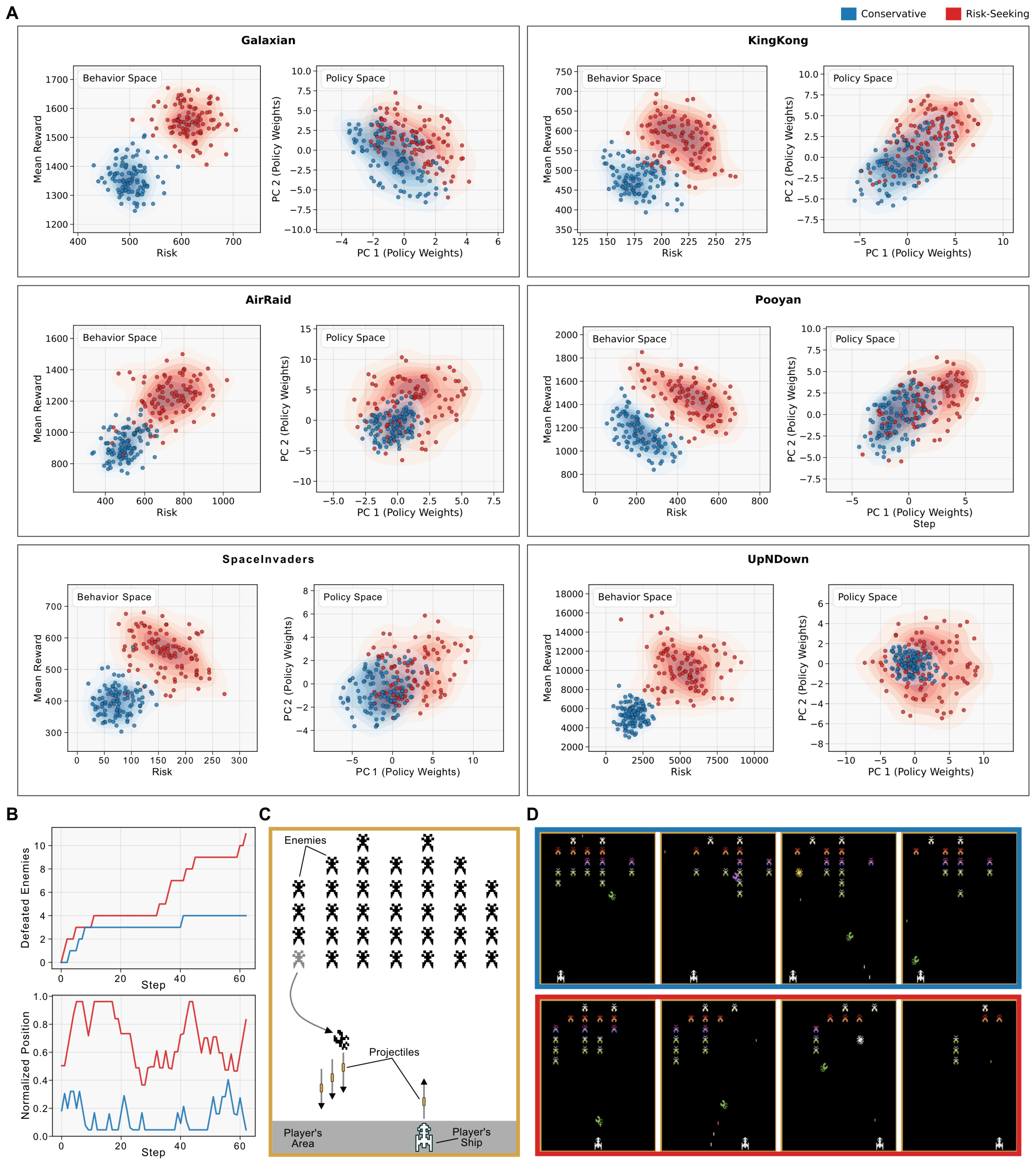

Figure 3.3: **Reward-Risk Tradeoffs Organize Agent Variation in Policy Weight Space and Gameplay. A.** Behavioral and policy space visualizations for six Atari games. Left: Mean reward versus risk, defined as the standard deviation of episode returns. Right: PCA of policy network weights. Agents are colored by phenotype: conservative (blue) or risk-seeking (red). **B.** Time series of in-game behavior for representative conservative and risk-seeking agents in Galaxian. Top: cumulative number of defeated enemies. Bottom: normalized horizontal position of the agent's ship, where 0 and 1 correspond to the leftmost and rightmost screen positions, respectively. **C.** Schematic of gameplay dynamics in Galaxian, illustrating interactions between player, enemies, and projectiles. **D.** Representative gameplay sequences for each phenotype in Galaxian. Top: conservative agent (blue border); bottom: risk-seeking agent (red border). Frames progress from left to right, illustrating temporal evolution and divergent strategies.

Learned Paths in Neural Policy Space Enable Generalist Agents with Tunable Task and Strategy.

To exploit the geometric structure of task-specific policy regions, we trained a hypernetwork to generate policies across the full spectrum of tasks and strategies. We ordered trained agents by combining inter-task similarity with intra-task risk preference variation, assigning each a scalar value $\theta \in [0, 1]$ linearly spaced by position. These $(\theta, \text{weights})$ pairs trained a hypernetwork that learned to map scalar inputs to complete policy network parameters (Fig. 3.4A). To evaluate the geometry of the learned path, we projected unit weight vectors into three-dimensional principal component space, L2-normalized them to lie on the unit sphere consistent with the cosine-based loss used during hypernetwork training, and applied the Robinson projection for two-dimensional visualization. The resulting trajectory formed a continuous path through distinct game-specific regions while remaining closely aligned with original trained agents (Fig. 3.4C).

Comparing hypernetwork traversal to linear weight interpolation revealed the importance of structured manifold learning. We generated 100 interpolated models using each method in Pong, reparameterized to a common variable $\alpha \in [0, 1]$ where $\alpha = 0$ and $\alpha = 1$ reproduced the anchor policies. While linear interpolation caused severe performance collapse with rewards approaching -21, hypernetwork-generated agents maintained near-optimal performance across the entire range (Fig. 3.4D). Evaluation on 1,300 unseen θ values spanning all 13 Atari games (100 per game) confirmed robust generalization. Rewards were normalized relative to expert performance, with most tasks reaching or exceeding expert-level performance and smooth variation within task segments. Brief performance dips at task boundaries indicated transient switching costs but rapid recovery (Fig. 3.4E).

To enable real-time behavioral modulation, we introduced a meta-agent that received both visual observations and scalar risk signals, outputting θ values for the hypernetwork to generate policy weights in a closed-loop design (Fig. 3.4B). Testing across three Atari games showed that the meta-agent successfully translated external preferences into behavioral variability. We measured empirical risk as the standard deviation of episodic returns, finding monotonic increases corresponding to increasing input risk signals. The shape of risk-response curves differed by game, reflecting environment-specific strategic tradeoffs and confirming dynamic policy adaptation based on both environmental context and external preferences (Fig. 3.4F).

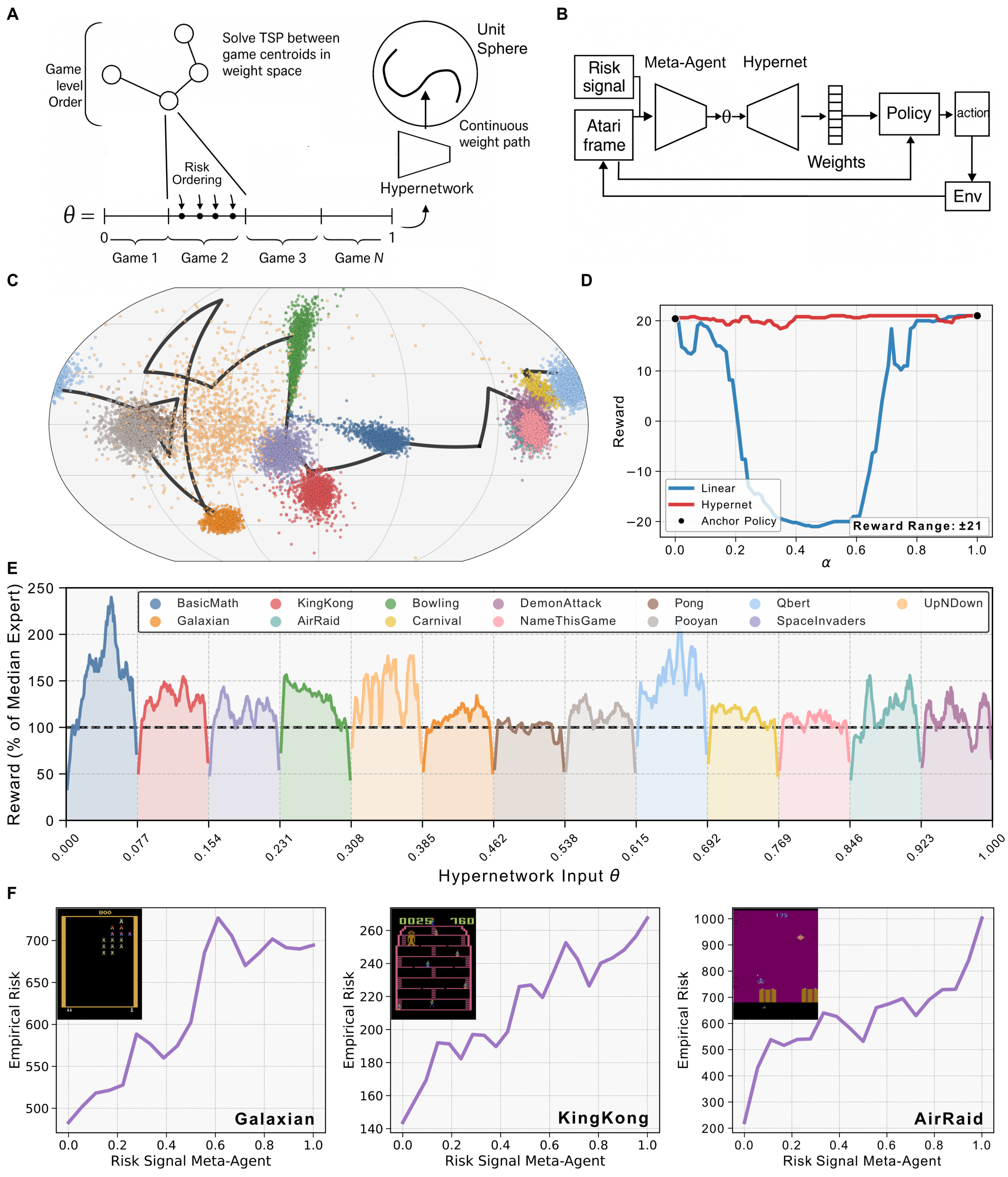

Figure 3.4: **Geometric Traversal of Policy Space Enables Generalist Agent Design. A.** Policy vectors are assigned $\theta \in [0,1]$ via game-level and risk-based ordering, normalized to the unit hypersphere. These $(\theta, \text{weights})$ pairs are used to train the hypernetwork. **B.** A meta-agent uses visual input and a risk signal to drive a hypernetwork, which generates weights for a fixed policy. Actions update the environment in a closed loop. **C.** Robinson projection of 3D PCA-reduced, normalized weight vectors. Colored points: trained agents; black curve: hypernetwork path. Color legend in panel E. **D.** Reward along interpolation paths between two anchor policies in Pong. Linear (blue) vs. hypernetwork (red); anchor values in black. Range: $[-21, 21]$. **E.** Normalized reward along hypernetwork path. Each θ segment corresponds to a different game. Colors and legend in-panel. **F.** Empirical risk (std of returns) vs. risk input for three games. Insets show meta-agent visual inputs.

3.3 Discussion

In this work, we propose a novel approach to multi-task reinforcement learning by training a single neural network with fixed weights to perform effectively across 13 Atari tasks. Unlike traditional RL methods that rely on task-specific policies or architectures to mitigate interference (Volodymyr Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015; Tassa et al., 2018; Schrittwieser et al., 2020; Espeholt et al., 2018; Hessel et al., 2019), our framework generalizes across tasks without the need for explicit disambiguation, external prompts, or context embeddings. This distinguishes it from both offline RL baselines (Gulcehre et al., 2020; Fu et al., 2020) and autoregressive models like Decision Transformer and Trajectory Transformer (Chen et al., 2021; Reid, Yamada, and S. S. Gu, 2022; Zheng, A. Zhang, and Grover, 2022; Janner, Q. Li, and Levine, 2021), which often encode task identity into weights or inputs. Here, we instead investigate whether a single, static policy can implicitly organize behavior across environments.

Our findings show that even under identical architectures and training protocols, RL agents develop distinct, task-specific representations in policy weight space. Each task induces a consistent geometric footprint, reflecting the structural demands of its environment. Within these task clusters, we observed further differentiation along interpretable behavioral axes—most notably, a continuous spectrum of strategies trading off reward and risk. This shows that neural policies not only adapt to tasks, but also self-organize to support a diverse set of viable behaviors within each environment.

These findings offer a blueprint for rational agent design. By training a hypernetwork to traverse the structured manifold of policy space, we built generalist agents that can fluidly modulate both task performance and strategy through a scalar input. This enables real-time control over agent behavior, such as shifting between cautious and aggressive modes, without retraining. Such a mechanism is particularly relevant for safety-critical or multi-objective domains, where agents must adaptively balance exploration, efficiency, and risk sensitivity based on context or external preferences.

Finally, the emergence of risk-reward tradeoffs in our agent populations offers compelling parallels with biological systems. Without explicit supervision, our agents naturally exhibited behavioral phenotypes reminiscent of those found in animal populations—suggesting that such diversity may arise as a fundamental outcome of optimization in uncertain environments. These structured, strategy-diverse populations provide a powerful *in silico* model for studying behavioral ecology, opening new avenues for using artificial agents to investigate the evolutionary and environmental dynamics of decision-making.

3.4 Conclusion

Our findings demonstrate that RL agents, even when identically structured and trained, self-organize into distinct, task-specific regions of neural weight space, each supporting a continuum of strategic behaviors. This geometry—emerging without supervision—encodes both task identity and interpretable tradeoffs, such as risk versus reward, offering a principled substrate for behavioral control. By learning to traverse these structured manifolds, we construct generalist agents capable of fluidly adapting task and strategy in real time via a single scalar input. These results establish that the internal geometry of learned policy spaces is not only descriptive but generative—supporting a new class of adaptive, controllable agents and offering a powerful framework for modeling the diversity and structure of intelligent behavior.

3.5 Methods

Agent Architecture, Training and Weight Extraction

We trained large populations of reinforcement learning agents to solve 13 Atari 2,600 games (M. G. Bellemare et al., 2013) using a standardized convolutional neural network policy. Each agent was trained using the Proximal Policy Optimization (PPO) algorithm, as implemented in Stable Baselines, with distributed execution over a Ray-based high-performance cluster. In total, we trained 14,976 agents—1,152 per game—each for one million environment steps.

The policy architecture followed the canonical DeepMind Atari DQN model (Volodymyr Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015), consisting of a convolutional feature extractor followed by fully connected layers for both the actor and critic heads. We adopted a four-frame input stacking protocol and vectorized each environment with a batch size of four. All training was performed on grayscale pixel inputs resized to 84×84.

All agents were trained independently from orthogonal weight initialization using identical hyperparameters: a learning rate of 3×10^{-4}, batch size of 64, rollout horizon of 2048 environment steps, and 10 optimization epochs per update. We used a reward discount factor of $\gamma = 0.99$, Generalized Advantage Estimation (GAE) with $\lambda = 0.95$, value function loss coefficient $c_1 = 0.5$, and gradient clipping with a maximum norm of 0.5. The surrogate objective was clipped using $\varepsilon = 0.2$, and entropy regularization was disabled throughout.

Training was distributed across 192 CPU workers and 8 GPUs. Each actor process instantiated an isolated vectorized environment and trained a single agent. To reach the full population of 1,152 agents per game, we launched six independent rounds of training per environment.

Policy updates were performed using the clipped surrogate PPO loss:

$$L^{\mathrm{CLIP}}(\phi) = \mathbb{E}_t \left[\min \left(r_t(\phi) \hat{A}_t,\ \mathrm{clip}\left(r_t(\phi), 1-\varepsilon, 1+\varepsilon\right) \hat{A}_t \right) \right], \tag{3.1}$$

where

$$r_t(\phi) = \frac{\pi_\phi(a_t \mid s_t)}{\pi_{\phi_{\mathrm{old}}}(a_t \mid s_t)} \tag{3.2}$$

is the probability ratio between new and old policies, and $\hat{A}_t$ is the estimated advantage. The clipping operation stabilizes learning by preventing large, potentially destabilizing policy shifts. It is defined as:

$$\mathrm{clip}(x, a, b) = \begin{cases} a & \text{if } x < a \\ x & \text{if } a \le x \le b \\ b & \text{if } x > b \end{cases} \tag{3.3}$$

In addition to the clipped policy loss, PPO includes a squared-error value function loss:

$$L^{\mathrm{VF}}(\phi) = \left(V_\phi(s_t) - V_t^{\mathrm{targ}} \right)^2, \tag{3.4}$$

where $V_\phi(s_t)$ is the predicted value of state s_t, and V_t^{targ} is the Monte Carlo return target. The final loss function used for optimization was the sum of the policy and value losses:

$$L(\phi) = \mathbb{E}_t \left[L^{\mathrm{CLIP}}(\phi) - c_1 L^{\mathrm{VF}}(\phi) \right]. \tag{3.5}$$

Each trained policy network was represented as a high-dimensional vector by flattening and concatenating the trainable parameters from the convolutional feature extractor and action head. The value head parameters were excluded since they are not part of the policy function. All vectors were stored alongside metadata identifying the agent, game environment, and training step.

This procedure yielded two primary datasets of policy weight vectors. The first is a final policy matrix $W^{\text{final}} \in \mathbb{R}^{N \times D}$, where $N = 14{,}976$ is the number of agents and $D = 1{,}687{,}206$ is the dimensionality of the flattened weights. Each row in this matrix corresponds to an agent's policy parameters at the end of training. The second is a trajectory matrix $W^{\text{traj}} \in \mathbb{R}^{M \times D}$, where $M = 20{,}800$ corresponds to a set of 16 agents per game across 13 games, each checkpointed at 100 distinct timepoints. These checkpoints were linearly spaced over the full course of 1 million training steps, providing uniform temporal resolution for capturing learning dynamics.

Weight Projection, Visualization, and Fitness Modeling

We applied dimensionality reduction to the policy weight vectors using Principal Component Analysis (PCA) and Uniform Manifold Approximation and Projection (UMAP). A single PCA model trained on W^{final} was reused across analyses. UMAP was applied to PCA-projected data using fixed parameters: 50 neighbors, minimum distance of 0.99, spread of 1.5, Euclidean metric, and a fixed random seed.

Learning Trajectories: Rows from W^{traj} were reduced to 50 PCA components, followed by 2D UMAP to visualize agent evolution over training time.

Weight Structure Visualization: To identify parameters with the highest variance across agents, we computed average absolute PCA loadings across the top five components and selected the top $k = 10{,}000$ features from W^{final}. This subset was z-scored across agents and hierarchically clustered by parameter using Ward's method. The resulting matrix was visualized as a heatmap, with agents grouped by game.

Task Geometry and Fitness: Vectors from W^{final} were reduced to 15 PCA dimensions, then embedded with UMAP. Each projected point was assigned a fitness value based on the average reward across evaluation episodes. Raw reward values were rescaled within each game using min-max normalization to map fitness scores into a common $[0, 1]$ range. A continuous fitness landscape was generated using radial basis function interpolation over the UMAP space, followed by Gaussian smoothing.

Hypernetwork Path: W^{final} and hypernetwork-generated vectors were L2-normalized, projected to 3D via PCA, renormalized to unit length, and visualized using the Robinson projection to reflect spherical structure (since the hypernetwork is trained with cosine loss on normalized policy vectors).

Inter/Intra Task Clustering and Behavioral Phenotyping

A k-nearest neighbors graph was constructed in the 15-dimensional PCA space of W^{final}, using Euclidean distance with $k = 25$. Louvain community detection was then applied to partition agents into clusters in an unsupervised manner. To assess clustering quality, ground-truth game labels were aligned to the discovered communities using the Hungarian algorithm to compute an optimal one-to-one assignment. Evaluation metrics included clustering accuracy (ACC), defined as the proportion of correctly aligned labels; adjusted Rand index (ARI), which measures pairwise label agreement corrected for chance; and normalized mutual information (NMI), which quantifies the shared information between predicted and true labels.

For behavioral characterization, each agent was represented by a two-dimensional feature vector comprising the mean and standard deviation of episodic rewards. Pairwise distance matrices were computed independently in the behavioral space and in the PCA-reduced policy space. These were averaged and embedded into two dimensions via multidimensional scaling (MDS). k-means clustering with $k = 2$ was performed on the embedded space. Cluster identities were assigned such that the group with higher reward variance was designated as the risk-seeking population.

Hypernetwork Modeling and Generalist Policy Construction

We trained a hypernetwork $\mathcal{H} : [0, 1] \to \mathbb{R}^D$ to generate L2-normalized policy weight vectors conditioned on a scalar input θ. The output dimensionality $D = 1{,}687{,}206$ matched the flattened convolutional and action-head parameters of the PPO policy.

Input Encoding: The input scalar $\theta \in [0, 1]$ was first embedded using a Fourier feature map $\gamma : [0, 1] \to \mathbb{R}^{2F}$ with $F = 64$ frequencies:

$$\gamma(\theta) = [\sin(2\pi f_1\theta), \ldots, \sin(2\pi f_F\theta), \cos(2\pi f_1\theta), \ldots, \cos(2\pi f_F\theta)], \tag{3.6}$$

where $f_i = i$ for $i = 1, \ldots, F$. This encoding was passed into a feedforward neural network $\mathcal{H}(\gamma(\theta))$ composed of four layers, each with hidden size 512 and ReLU activations, terminating in a linear output layer with D units.

Loss Function: The hypernetwork was trained to minimize the cosine distance between predicted and target policy vectors. Given a dataset of normalized policy weights $\{(\theta_i, \mathbf{w}_i)\}_{i=1}^{N}$, the training objective was:

$$\mathcal{L}_{\cos} = \frac{1}{N}\sum_{i=1}^{N} [1 - \cos(\mathcal{H}(\gamma(\theta_i)),\ \mathbf{w}_i)], \tag{3.7}$$

where $\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$. All target vectors $\mathbf{w}_i$ were pre-normalized to unit norm.

Trajectory Dataset Construction: We assign selected agents a scalar coordinate $\theta_i \in [0, 1]$ through a two-stage ordering process. We begin with a set of games G, where each game $g \in G$ is characterized by a game-averaged policy vector summarizing agent behaviors. These games are sequenced by solving a symmetric Traveling Salesman Problem (TSP), using cosine dissimilarity between policy vectors as the distance metric, which assigns each game g an index $i_g \in \{0, 1, \ldots, 12\}$ based on its position in the sequence. Within each game, we select 25 agents by binning them according to the standard deviation of their rewards, a measure of risk, and filtering to include only those with mean rewards above the 10th percentile to ensure high performance. These agents, denoted $\{\mathbf{w}_g^{(j)}\}_{j=1}^{25}$, are then ordered from lowest to highest risk, based on increasing standard deviation of rewards. Each agent is assigned a scalar $\theta_g^{(j)} \in [0, 1]$ using the formula $\theta_g^{(j)} = \frac{i_g}{|G|} + \frac{j}{25} \cdot \frac{1}{|G|}$, where $|G| = 13$ is the number of games, i_g is the game's TSP index, and $j \in \{1, 2, \ldots, 25\}$ is the agent's position in the ordered list. This formula partitions the $[0, 1]$ interval into 13 equal segments, one per game, with each game's 25 agents evenly spaced within their segment according to their risk order, ensuring a trajectory that reflects both inter-game similarity and intra-game risk progression.

Warped Input Space: To improve interpolation across game transitions, a differentiable piecewise transformation $\psi : [0,1] \to [0,1]$ was applied to all inputs. Let $\{b_0 = 0 < b_1 < \cdots < b_{|G|} = 1\}$ denote game-specific breakpoints. Within each interval $[b_k, b_{k+1}]$, the local transformation was:

$$\psi(\theta) = (1 - \sigma(k(\theta - m))) \cdot a\theta + \sigma(k(\theta - m)) \cdot (a(\theta - m) + (1 - w)), \tag{3.8}$$

where $a = 2w$, $m = \frac{b_k + b_{k+1}}{2}$, σ is the sigmoid function, w is the width parameter, and k controls the steepness of the transition.

Evaluation: The model's predictions were evaluated by reconstructing PPO agents from the generated vectors $\hat{\mathbf{w}} = \mathcal{H}(\gamma(\psi(\theta)))$. The predicted vectors were split into convolutional feature extractor and action-head weights, used to reconstruct policy networks, and evaluated via five-episode rollout in their respective target environments.

Two evaluation paradigms were employed. First, to assess interpolation fidelity, two PPO agents trained on *Pong* were selected as anchor policies $\mathbf{w}_1 = \mathcal{H}(\gamma(\psi(\theta_1)))$ and $\mathbf{w}_2 = \mathcal{H}(\gamma(\psi(\theta_2)))$. These policies were chosen from adjacent entries in the hypernetwork training set, ensuring they were contiguous under the learned task-space ordering. One hundred intermediate policies were generated by both linear interpolation:

$$\mathbf{w}_\alpha = (1 - \alpha) \cdot \mathbf{w}_1 + \alpha \cdot \mathbf{w}_2, \quad \alpha \in [0,1] \tag{3.9}$$

and hypernetwork traversal using values of θ linearly spaced between θ_1 and θ_2. To enable direct comparison across methods, the hypernetwork parameter was reparameterized via $\alpha = (\theta - \theta_1)/|\theta_2 - \theta_1|$. In both methods, $\alpha = 0$ and $\alpha = 1$ correspond to the two original anchors. The resulting models were evaluated in *Pong* to compare reward continuity and robustness across the trajectory.

Second, to probe the learned manifold across the full game spectrum, we partitioned the scalar latent domain $[0,1]$ into $|G|$ equal-length intervals, where $|G| = 13$ is the number of games used during hypernetwork training. The interval corresponding to game k was defined as $\left[\frac{k}{|G|}, \frac{k+1}{|G|}\right]$ for $k = 0, 1, \ldots, |G| - 1$. Within each interval, we sampled 100 evenly spaced values of θ in the subinterval $\left[\frac{k}{|G|} + \xi, \frac{k+1}{|G|} - \xi\right]$, with $\xi = 10^{-3}$ used to avoid boundary artifacts. Each sampled

θ was processed as $\hat{\mathbf{w}} = \mathcal{H}(\gamma(\psi(\theta)))$, decoded into policy networks, and evaluated in the target environment, producing reward trajectories as a function of θ, characterizing the task-conditioned policy manifold.

To enable closed-loop behavioral control, we trained a meta-agent to dynamically generate the scalar input $\theta \in [0, 1]$ for the hypernetwork, based on both visual observations and an externally specified *risk preference*, denoted $\rho_{\text{ext}} \in [0, 1]$. This is a user-defined control signal that encodes the desired level of risk sensitivity during policy generation. It is not learned from interaction data, nor inferred from rewards—rather, it expresses a target behavioral profile. The meta-agent receives as input a stack of four consecutive grayscale Atari frames, processed through a convolutional encoder identical to that used in the PPO policy network. The encoded visual features are passed into a multilayer perceptron (MLP) that outputs a softmax over the set of games, identifying the most likely current game.

Given the predicted game index $i_g \in \{0, \ldots, 12\}$, the meta-agent computes the base position in θ-space as $\frac{i_g}{|G|}$, with $|G| = 13$ denoting the number of games. The external risk preference ρ_{ext} is scaled to match the segment width for that game, resulting in:

$$\theta = \frac{i_g}{|G|} + \rho_{\text{ext}} \cdot \frac{1}{|G|}. \tag{3.10}$$

Because the trajectory dataset was constructed such that policies are ordered by increasing standard deviation of reward within each game segment—effectively encoding increasing behavioral risk—this formulation allows the meta-agent to modulate behavior in a risk-aware manner while remaining game-specific. The computed θ is passed into the hypernetwork, which produces the policy parameters. These are then loaded into a fixed architecture, enabling the agent to act in the environment using the same observation that generated θ, thus closing the loop from input perception to behavior.

We evaluated the system by sweeping ρ_{ext} across $[0, 1]$, generating 20 distinct policies per game, across games. Each policy was then rolled out in its respective game environment for 100 episodes. Empirical risk was quantified as the mean standard deviation of episodic returns across those episodes. This enabled a direct comparison between the intended risk profile ρ_{ext} and the realized behavioral variability for each game, highlighting the efficacy of the meta-agent's control.

BIBLIOGRAPHY

Bellemare, M. G. et al. (June 2013). “The Arcade Learning Environment: An Evaluation Platform for General Agents”. In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Bellemare, Marc G et al. (2013). “The arcade learning environment: An evaluation platform for general agents”. In: *Journal of artificial intelligence research* 47, pp. 253–279.

Booth, Martin J (2014). “Adaptive optical microscopy: the ongoing quest for a perfect image”. In: *Light: Science & Applications* 3.4, e165–e165.

Burger, Benjamin et al. (2020). “A mobile robotic chemist”. In: *Nature* 583.7815, pp. 237–241.

Buttinoni, Ivo et al. (2012). “Active Brownian motion tunable by light”. In: *Journal of Physics: Condensed Matter* 24.28, p. 284129.

Cai, Wenjie et al. (2025). “Reinforcement Learning for Active Matter”. In: *arXiv preprint arXiv:2503.23308*.

Chen, Lili et al. (2021). “Decision transformer: Reinforcement learning via sequence modeling”. In: *Advances in neural information processing systems* 34, pp. 15084–15097.

Chennakesavalu, Shriram et al. (2024). “Adaptive nonequilibrium design of actin-based metamaterials: Fundamental and practical limits of control”. In: *Proceedings of the National Academy of Sciences* 121.8, e2310238121.

Christiansen, Eric M et al. (2018). “In silico labeling: predicting fluorescent labels in unlabeled images”. In: *Cell* 173.3, pp. 792–803.

Cobbe, Karl et al. (2019). “Quantifying generalization in reinforcement learning”. In: *International conference on machine learning*. PMLR, pp. 1282–1289.

Coley, Connor W et al. (2019). “A robotic platform for flow synthesis of organic compounds informed by AI planning”. In: *Science* 365.6453, eaax1566.

Cranmer, Miles et al. (2020). “Discovering symbolic models from deep learning with inductive biases”. In: *Advances in neural information processing systems* 33, pp. 17429–17442.

Driess, Danny et al. (2023). “Palm-e: An embodied multimodal language model”. In.

Espeholt, Lasse et al. (2018). “Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures”. In: *International conference on machine learning*. PMLR, pp. 1407–1416.

Falk, Martin J et al. (2021). “Learning to control active matter”. In: *Physical Review Research* 3.3, p. 033291.

Floreano, Dario and Claudio Mattiussi (2008). *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press.

Fu, Justin et al. (2020). “D4rl: Datasets for deep data-driven reinforcement learning”. In: *arXiv preprint arXiv:2004.07219*.

Grosenick, Logan, James H Marshel, and Karl Deisseroth (2015). “Closed-loop and activity-guided optogenetic control”. In: *Neuron* 86.1, pp. 106–139.

Gulcehre, Caglar et al. (2020). “Rl unplugged: A suite of benchmarks for offline reinforcement learning”. In: *Advances in Neural Information Processing Systems* 33, pp. 7248–7259.

Häse, Florian, Loıc M Roch, and Alán Aspuru-Guzik (2019). “Next-generation experimentation with self-driving laboratories”. In: *Trends in Chemistry* 1.3, pp. 282–291.

Hessel, Matteo et al. (2019). “Multi-task deep reinforcement learning with popart”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01, pp. 3796–3803.

Ho, Jonathan and Stefano Ermon (2016). “Generative adversarial imitation learning”. In: *Advances in neural information processing systems* 29.

James, Stephen et al. (2022). “Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13739–13748.

Janner, Michael, Qiyang Li, and Sergey Levine (2021). “Offline reinforcement learning as one big sequence modeling problem”. In: *Advances in neural information processing systems* 34, pp. 1273–1286.

Leech, Gregor et al. (2025). “Programming scheduled self-assembly of circadian materials”. In: *Nature Communications* 16.1, p. 176.

Liu, Allen P et al. (2022). “The living interface between synthetic biology and biomaterial design”. In: *Nature materials* 21.4, pp. 390–397.

Liu, Han et al. (2023). “End-to-end differentiability and tensor processing unit computing to accelerate materials’ inverse design”. In: *npj Computational Materials* 9.1, p. 121.

Lugagne, Jean-Baptiste, Caroline M Blassick, and Mary J Dunlop (2024). “Deep model predictive control of gene expression in thousands of single cells”. In: *Nature Communications* 15.1, p. 2148.

Lugagne, Jean-Baptiste, Sebastián Sosa Carrillo, et al. (2017). “Balancing a genetic toggle switch by real-time feedback control and periodic forcing”. In: *Nature communications* 8.1, p. 1671.

MacLeod, Benjamin P et al. (2020). “Self-driving laboratory for accelerated discovery of thin-film materials”. In: *Science Advances* 6.20, eaaz8867.

Milias-Argeitis, Andreas et al. (2016). “Automated optogenetic feedback control for precise and robust regulation of gene expression and cell growth”. In: *Nature communications* 7.1, p. 12546.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, et al. (2013). “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, et al. (2015). “Human-level control through deep reinforcement learning”. In: *nature* 518.7540, pp. 529–533.

Nguyen, Peter Q et al. (2018). “Engineered living materials: prospects and challenges for using biological systems to direct the assembly of smart materials”. In: *Advanced Materials* 30.19, p. 1704847.

Oh, Junhyuk, Valliappa Chockalingam, Honglak Lee, et al. (2016). "Control of memory, active perception, and action in minecraft". In: *International conference on machine learning*. PMLR, pp. 2790–2799.

Palacci, Jeremie et al. (2013). "Living crystals of light-activated colloidal surfers". In: *Science* 339.6122, pp. 936–940.

Pandarinath, Chethan et al. (2017). "High performance communication by people with paralysis using an intracortical brain-computer interface". In: *elife* 6, e18554.

Parisotto, Emilio, Jimmy Lei Ba, and Ruslan Salakhutdinov (2015). "Actor-mimic: Deep multitask and transfer reinforcement learning". In: *arXiv preprint arXiv:1511.06342*.

Pohlmeyer, Eric A et al. (2014). "Using reinforcement learning to provide stable brain-machine interface control despite neural input reorganization". In: *PloS one* 9.1, e87253.

Qu, Zijie et al. (2021). "Persistent fluid flows defined by active matter boundaries". In: *Communications Physics* 4.1, p. 198.

Reddy, Chandan K and Parshin Shojaee (2025). "Towards scientific discovery with generative ai: Progress, opportunities, and challenges". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 27, pp. 28601–28609.

Reed, Scott et al. (2022). "A generalist agent". In: *arXiv preprint arXiv:2205.06175*.

Reid, Machel, Yutaro Yamada, and Shixiang Shane Gu (2022). "Can wikipedia help offline reinforcement learning?" In: *arXiv preprint arXiv:2201.12122*.

Ross, Tyler D et al. (2019). "Controlling organization and forces in active matter through optically defined boundaries". In: *Nature* 572.7768, pp. 224–229.

Rusu, Andrei A et al. (2015). "Policy distillation". In: *arXiv preprint arXiv:1511.06295*.

Sanchez-Gonzalez, Alvaro et al. (2020). "Learning to simulate complex physics with graph networks". In: *International conference on machine learning*. PMLR, pp. 8459–8468.

Schaul, Tom et al. (2015). "Universal value function approximators". In: *International conference on machine learning*. PMLR, pp. 1312–1320.

Schrittwieser, Julian et al. (2020). "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 588.7839, pp. 604–609.

Shelley, Michael J (2016). "The dynamics of microtubule/motor-protein assemblies in biology and physics". In: *Annual review of fluid mechanics* 48.1, pp. 487–506.

Sherman, Zachary M et al. (2020). "Inverse methods for design of soft materials". In: *The Journal of chemical physics* 152.14.

Stein, David B et al. (2021). "Swirling instability of the microtubule cytoskeleton". In: *Physical review letters* 126.2, p. 028103.

Suzuki, Kazuya et al. (2017). "Spatial confinement of active microtubule networks induces large-scale rotational cytoplasmic flow". In: *Proceedings of the National Academy of Sciences* 114.11, pp. 2922–2927.

Tang, Tzu-Chieh et al. (2021). “Materials design by synthetic biology”. In: *Nature Reviews Materials* 6.4, pp. 332–350.

Tassa, Yuval et al. (2018). “Deepmind control suite”. In: *arXiv preprint arXiv:1801.00690*.

Thuerey, Nils et al. (2021). “Physics-based deep learning”. In: *arXiv preprint arXiv:2109.05237*.

Volpe, Giovanni et al. (2011). “Microswimmers in patterned environments”. In: *Soft Matter* 7.19, pp. 8810–8815.

Wu, Kun-Ta et al. (2017). “Transition from turbulent to coherent flows in confined three-dimensional active fluids”. In: *Science* 355.6331, eaal1979.

Wülfing, Jan M et al. (2019). “Adaptive long-term control of biological neural networks with deep reinforcement learning”. In: *Neurocomputing* 342, pp. 66–74.

Zhang, Rui et al. (2021). “Spatiotemporal control of liquid crystal structure and dynamics through activity patterning”. In: *Nature materials* 20.6, pp. 875–882.

Zhavoronkov, Alex et al. (2019). “Deep learning enables rapid identification of potent DDR1 kinase inhibitors”. In: *Nature biotechnology* 37.9, pp. 1038–1040.

Zheng, Qinqing, Amy Zhang, and Aditya Grover (2022). “Online decision transformer”. In: *international conference on machine learning*. PMLR, pp. 27042–27059.

Zintgraf, Luisa et al. (2019). “Varibad: A very good method for bayes-adaptive deep rl via meta-learning”. In: *arXiv preprint arXiv:1910.08348*.