SAFE PLANNING UNDER UNCERTAINTY
USING SURROGATE MODELS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Robert J. Moss
May 2025

Robert J. Moss

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mykel J. Kochenderfer) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Clark Barrett)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Jef Caers)

Approved for the Stanford University Committee on Graduate Studies

Abstract

To make safe decisions in real-world environments, algorithms must account for the inherent uncertainty in perception systems and agent dynamics, resulting in high-dimensional problems. The use of surrogate models to replace hand-crafted planning heuristics and avoid running the computationally expensive true system has shown promise in enabling large-scale, safe planning. This thesis introduces five main contributions to address the challenges of safe planning under uncertainty and validation. To improve planning efficiency over beliefs in partially observable Markov decision processes (POMDPs), we introduce *batched belief-state MDPs*, which abstract belief-state planning using parallelizable batches of the underlying models. This abstraction requires models that can be easily parallelized; therefore, we can learn surrogate transition and observation models from data and propose the *inversion variational autoencoder* ($\mathcal{I}$-VAE) to sample from the posterior belief given partial observations. To replace planning heuristics and enable long-horizon planning, we introduce *BetaZero*, a policy iteration algorithm that combines offline learning with online belief-state planning. Extending BetaZero to safety-critical problems, we propose *ConstrainedZero*, which solves chance-constrained POMDPs by optimizing the balance between utility and a target level of safety. Finally, given a learned safe policy, we develop a *Bayesian safety validation* method to estimate the failure probability of a black-box system using probabilistic surrogate models. We apply these algorithms to real-world problems, including aircraft collision avoidance, autonomous aircraft runway detection, safe carbon capture and storage, critical mineral exploration, robot navigation, and aerial wildfire suppression.

Acknowledgments

Without a doubt, I would not be here at Stanford without my advisor Mykel Kochenderfer. Going back to 2012 when we first met at MIT Lincoln Laboratory, you captured an excitement in me that has allowed me to flourish as a researcher. Your style of advising and constant care for the well-being of your students is incredibly inspiring. Whether I am having a stressful week, productive week, or thrilled to show you *''one more thing,''* you have given me the space and mentorship to be a completely open and honest student. I would also like to recognize my CS co-advisor, Clark Barrett, my unofficial Earth and Planetary Sciences advisor, Jef Caers, and my defense committee, Mert Pilanci and Tapan Mukerji, for their encouragement, thoughtfulness, and constant reminder of what my primary goals as an objective researcher are. I am also grateful for Dorsa Sadigh for being on my master's thesis committee and providing valuable feedback that led me to be a successful PhD student.

This work is an accumulation of years of discussion with my peers. I am indebted to the following brilliant minds I have had the pleasure of collaborating with: Anthony Corso, Ritchie Lee, Arec Jamgochian, Sydney Katz, Markus Zechner, Johannes Fischer, David Zhen Yin, Kyla Guru, and Mariia Kozlova. I am incredibly lucky to be a part of the *Stanford Intelligent Systems Laboratory* (SISL) and want to thank the following for their useful discussions and ideas: Dylan Asmar, Harrison Delecki, Anka Reuel, John Mern, Bernard Lange, Josh Ott, Chelsea Sidrane, Liam Kruse, Marc Schlichting, Romeo Valentine, Grace Kim, Amelia Hardy, Houjun Liu, Lisa Einstein, Duncan Eddy, Kiana Jafari, Mansur Arief, Edward Balaban, Tomar Arnon, Jayesh Gupta, Mark Koren, Shushman Choudhury, Jean-Raymond Betterton, Ellen Xu, Alex Kufos, and Zach Sunberg. I also want to shoutout Ashwim Kanhere for his continual support and willingness to review my papers.

I have also had the pleasure to work with James G. O'Brien as an undergraduate researcher while at Wentworth and I'm grateful for the inspiration and trust he provided me. I owe an enormous debt to everyone I have interacted with while at MIT Lincoln Laboratory

over the years—they each played a role in shaping me into a better scientist, engineer, analyst, and friend. This includes Ted Londner, Michael Owen, Wes Olson, Rod Cole, Gregg Shoults, Ian Jessen, Adam Panken, Tomas Elder, Cindy McLain, Luis Alvarez, Tan Trinh, Robert Klaus, Justin MacKay, Jared Wikle, Larry Capuder, Emilie Cowen, Randy Wicken, Tom Teller, Matt Edwards, Adam Gjersvik, Sam Wu, Anshu Das, Jack Lepird, Charles Leeper, and Dan Griffith. Particular appreciation goes to Jeff Bezanson for co-creating Julia and showing me its divinity. Also a special thanks to Neal Suchy from the Federal Aviation Administration for his admiration and his faith in my work.

Research sponsorship provided most of the funding for this dissertation. I would like to thank Maxime Gariel and Arthur Dubois for their willingness to let me explore ideas relating to my thesis while working at Xwing, and Rob Timpe and Alex Bridi for the development of the runway detection system that I study in this work. I would like to thank OMV and, in particular, Torsten Clemens for their research support and always being excited about the directions we take our work. I am grateful for the support from Ideon to investigate an exciting application of POMDPs to muon-based intrusion discovery, and I am particularly grateful for Patrick Belliveau, Nigel Phillips, Doug Schouten, and Max Howarth. Stanford research centers have also been incredibly supportive during my PhD, and I would like to thank the Mineral-X Initiative, the Stanford Center for AI Safety, the Stanford Institute for Human-Centered AI (HAI), and the Stanford Center for Earth Resources Forecasting (SCERF) as part of the Doerr School of Sustainability.

To my friends and family, I am extremely grateful that you have given me the opportunity to pursue a PhD. Mom and Dad, thank you for always supporting every decision I make in this uncertain world and thank you Travis, Emily, and Jake for always telling me how proud you are to call me your brother. To my close friends back in Rockport, MA, you all have seen me throughout each of my phases of life and continue to remind me to be a better person.

Most importantly, I need to thank my wife, Eva, for being my foundation. A PhD is never something you do alone, and no one knows best like the spouse of a student. Eva, you endured this stressful adventure with me, and I cannot express how thankful I am that you have given me the space to be selfish in my studies. I am excited to share the rest of my life with you and our dog, Tzvi, and I look forward to the next generation of Mosses.

Contents

PART III SAFE PLANNING

List of Tables

List of Figures

List of Algorithms

PART I:

BACKGROUND

Chapter 1

Introduction

The right question is intellectually superior to finding the right answer.

Edward O. Wilson

Our ability to make informed decisions depends on how well we can reason about the environment and understand the consequences of our actions. When decisions involve high-dimensional information or complex dynamics, we increasingly rely on computational tools to augment our understanding or automate this reasoning entirely by using faster measurements, higher fidelity environment models, or more accurate predictions. Predictive models trained to perform classification or regression play an important role in decision making systems that must act based on partial or high-dimensional information. Machine learning models are often used in the context of single-step decisions, as it has been shown that they can learn patterns from unstructured or noisy data [1, 2]. While single-step decisions may be useful in settings such as recommendation systems [3] or content moderation [4], many complex real-world problems require making a *sequence* of decisions over time, where each action affects the immediate outcome and shapes future decisions—indicating the necessity to reason for both short-term utility and long-term consequences.

Sequential decision making, i.e., *planning*, involves reasoning about actions over time to achieve long-term objectives [5]. The dependency on long time horizons introduces complexity to the decision making problem. Long-horizon problems—such as robotic exploration [6], autonomous driving [7], and geological drilling operations [8]—require

search-based algorithms [10] (figure 1.1) or dynamic programming [11] to effectively reason about potential future outcomes. In these settings, the planning process must operate over a large or continuous set of actions, where enumerating all possibilities is computationally infeasible. Moreover, real-world problems often have high-dimensional state spaces that describe their environments. This combined complexity motivates the need for planning algorithms that can scale and generalize to large decision spaces.

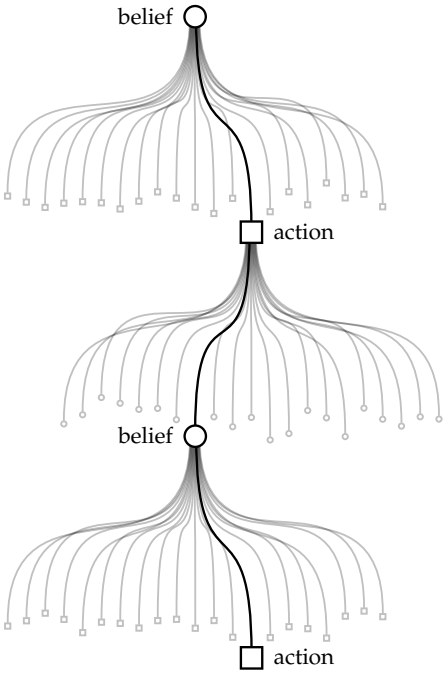
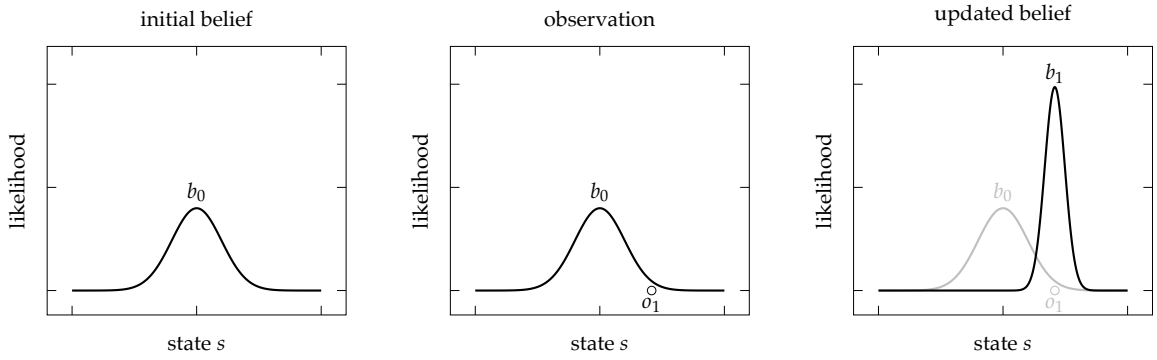


Figure 1.1: Sequential search tree (inspired by Kohs *et al.* [9]).

In addition to the challenges of high-dimensional state and action spaces, real-world planning problems need to operate *under uncertainty*—whether due to stochastic system dynamics or noisy observations. These planning problems are often modeled as *Markov decision processes* (MDPs) to handle uncertainty in state transition dynamics, and *partially observable MDPs* (POMDPs) to handle state uncertainty [5]. In such problems, the outcome of an action is not deterministic, and only partial information about the current state is available. For example, in autonomous driving, we cannot directly observe the true x-y position of the vehicle. Instead, we use noisy measurements from sensors such as GPS or camera feeds to localize and form a *belief* about our position—encoding state uncertainty as shown in figure 1.2—and use probabilistic models to capture how beliefs change over time. Without modeling belief dynamics, planning algorithms cannot reliably assess risk to ensure safe behavior.

Real-world problems often require balancing expected performance with the need to maintain safety. *Safe planning under uncertainty* refers to the design of policies that not only account for uncertainties, but also avoid catastrophic failures. For example, aircraft collision avoidance systems [12] must balance operational suitability by minimizing guidance alerts given to pilots, while also maximizing safety by attempting to avoid midair collisions. Therefore, safe planning algorithms require reasoning about the full distribution of possible outcomes and identifying actions that achieve objectives while also satisfying safety constraints. Incorporating such constraints often requires defining risk bounds, constraining



(a) Initial belief at time $t = 0$. (b) Noisy observation. (c) Updated belief at time $t = 1$.

Figure 1.2: POMDPs encode state uncertainty using *beliefs* (i.e., distributions over states).

policies to stay within a target level of safety, or model approximations that reduce the dimensionality of the problem to make it tractable.

While exact safe planning under uncertainty is the goal, it is often computationally intractable in complex, high-dimensional domains. In practice, simulating all possible outcomes, evaluating risk bounds, or computing belief updates over long horizons can be prohibitively expensive. To address this limitation, *surrogate models* [13, 14] have been increasingly used to approximate costly components of the planning process—such as state-transition dynamics, safety evaluations, or belief updaters—enabling faster and more scalable decision making. These surrogate models serve as efficient low-dimensional stand-ins for more expensive simulators or domain-specific heuristics.

Once a policy has been deemed safe through planning to satisfy safety constraints, a crucial next step is to perform *safety validation* [15]: validating that the policy behaves safely across the full range of conditions it may encounter during deployment in the real world. This is particularly important in systems where rare events may exhibit failure modes not seen during planning. Safety validation has three primary tasks: find failures, find the *most-likely* failure, and estimate the failure probability. Validating the safety of a policy over a large or continuous operating domain can be computationally expensive, especially when using high-fidelity simulators. Along with increased planning efficiency, surrogate models offer a benefit during validation by approximating where the system under test may fail—enabling efficient sampling of predicted failures. Before system deployment, safety validation algorithms can be used as one piece of the full validation process, accelerating the traditional validation loop of simulation and real-world testing.

Research question. This thesis attempts to address the challenges of safe planning under uncertainty and safety validation through the use of surrogate models and scalable safety-aware algorithms. Therefore, our primary research question is the following:

> *How can low-dimensional surrogate models be used to enable high-dimensional planning and validation for safety-critical systems?*

1.1 Contributions

We outline our five main contributions relating to safe planning under uncertainty and efficient safety validation using surrogate models.

Batched Belief-State Planning

To scale planning problems to be easily parallelizable, we introduce the *batched planning* problem formulation for MDPs and its extension to POMDPs. We prove that planning in the batched setting preserves the optimal policy, while also speeding up the lookahead process.

Inversion Surrogates for Belief Updating

Methods for parallelized belief updating are required in the batched planning setting. Therefore, we introduce the *inversion variational autoencoder* ($\mathcal{I}$-VAE), which is a deep conditional generative model that acts as a surrogate to efficiently sample from the posterior belief. We show that fine-tuning our conditional model to maximize mutual information provides benefits in both planning and estimation. We demonstrate our approach on a common toy problem of masked hand-written digits using the MNIST dataset [16] and a novel muon-based intrusion discovery POMDP. We develop theoretically optimal one-step information-gathering heuristic policies that perform close to optimal in practice.

Planning and Learning in Belief Space

To replace hand-crafted heuristics and to generalize POMDP planning algorithms to long-horizon problems, we introduce the *BetaZero* policy iteration algorithm. We use insights from recent success in game-playing agents [17] that combine online Monte Carlo tree search (MCTS) planning with learned offline neural network surrogates that replace value function and action selection heuristics. We address several challenges relating to stochastic

belief-state transitions, expensive belief updates given a limit tree search budget, and how to represent non-parametric belief distributions as inputs to the neural network. We demonstrate the generalizability of our method across various benchmark POMDPs—including robot localization and navigation, robot exploration, and critical mineral exploration—and compare to state-of-the-art online algorithms. We show that BetaZero scales well given available compute and can learn approximately optimal value functions and action selection policies. We provide insights into how BetaZero replaces expert heuristics by using learned offline experience to reduce online planning horizons.

Safe Planning using Failure Surrogates

As a natural extension to BetaZero for chance-constrained POMDPs, we introduce the *ConstrainedZero* policy iteration algorithm. In the safety-critical setting, we extend the neural network model to predict the failure probability given a belief. We then introduce Δ-*MCTS* as a method for safety-aware online planning using the failure probability surrogates to guide the search. We demonstrate our method on a safety-critical version of robot localization and navigation, an aircraft collision avoidance system, the problem of safe CO_2 storage, and an airborne wildfire resource allocation problem. We show that by separating safety constraints from the primary objective, we can achieve a target level of safety without directly optimizing the balance between rewards and costs.

Safety Validation using Probabilistic Surrogates

We investigate the use of probabilistic surrogate models (i.e., Gaussian processes) for black-box safety validation. We introduce the *Bayesian safety validation* method that iteratively selects new design points to satisfy the objectives of finding failures, finding likely failures, and ultimately estimating the failure probability over the operating domain of a system. Our method introduces three new acquisition functions, each with their own safety validation objective. We demonstrate the method on three test problems, a stochastic POMDP, and a neural network-based runway detection system—treating each system as a black box. Results from our experiments indicate that Bayesian safety validation finds likely failures (allowing developers to triage appropriately) and can efficiently estimate the failure probability.

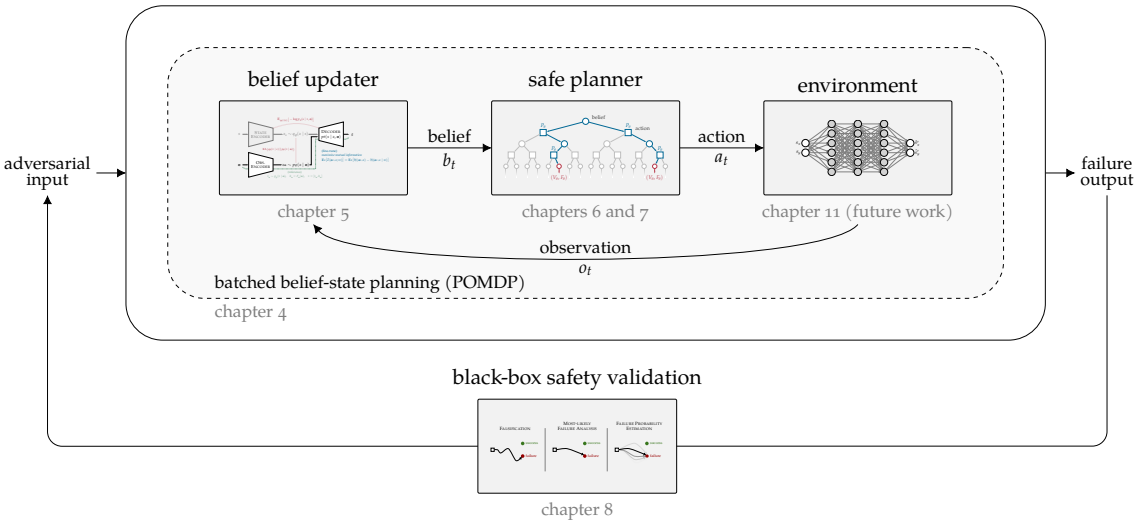


Figure 1.3: Overview of our thesis contributions relating to safe planning and validation.

1.2 Overview

Figure 1.3 provides an overview of how our contributions are connected in the safe planning and validation process. The structure of the remaining thesis is organized as follows:

- The remainder of Part I provides background on the problems and foundations of the technical methods used in this thesis. In chapter 2, we provide details about the safety-critical problem domains we study. In chapter 3, we review technical background on topics including Markov decision processes (MDPs), partially observable MDPs (POMDPs), belief-state MDPs, belief updating methods, offline and online planning algorithms, surrogate modeling, and safety validation.

- Part II explores how to efficiently scale POMDP planning algorithms to high-dimensional problems. In chapter 4, we introduce the batched planning framework and prove that it preserves the optimal policy. In chapter 5, we introduce a surrogate model for belief updating, allowing us to use the batched planning framework efficiently. In chapter 6, we introduce a scalable policy iteration algorithm for long-horizon POMDPs that learns from offline experience to reduce online planning horizons.

- Part III explores methods for safe planning at scale and surrogates for safety validation. In chapter 7, we introduce a chance-constrained POMDP policy iteration algorithm

that learns a failure surrogate through experience to inform safe online planning. In chapter 8, we introduce a black-box method to efficiently estimate the failure probability using probabilistic surrogate models.

- Part IV concludes the thesis and provides future research directions. In chapter 9, we provide a summary of this thesis. In chapter 10, we reiterate the contributions of this thesis, provide explanations for the open-source contributions, and detail our publications. Finally, in chapter 11, we examine the limitations, possible next steps, and extensions to the research provided in this thesis.

Chapter 2

Problem Domains

But oversimplifying the real world is a bad idea—and unfortunately, that's exactly what the Primitive Mind likes to do.

Tim Urban

In this thesis, we consider several diverse, safety-critical, real-world problems. Broadly, the domains we study focus on the fields of aviation, robotics, and geological sustainability.

2.1 Aviation: Safety-Critical Systems

A culture of safety is ingrained in the field of aviation [18]. Lessons regarding safety are often shared among competing aviation companies, with a collective objective of increasing safety throughout the field. Stakeholders from industry, government, and academia come together to discuss safety concerns and develop solutions for the broader community [19]. In our work, we study three different safety-critical aviation systems.

2.1.1 Aircraft Collision Avoidance

Aircraft collision avoidance systems (CAS) are decision support tools designed to alert pilots when an immediate collision is imminent and provide recommended guidance to mitigate midair collisions [12]. In our work, we apply our methods to a simplified CAS problem, modeled after ACAS X [20]: the successor to the *traffic alert and collision avoidance*

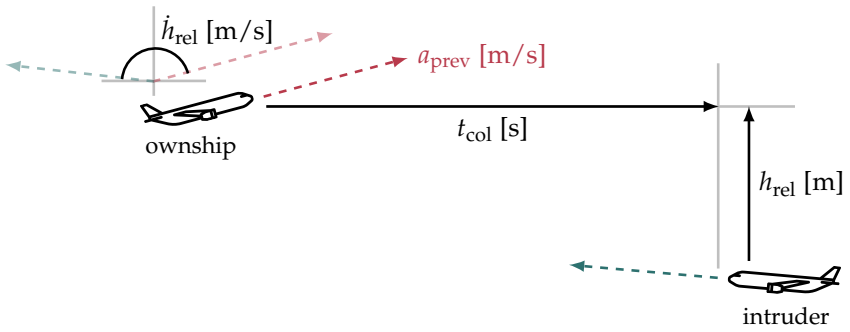


Figure 2.1: State space of the aircraft collision avoidance system.

system (TCAS) [21], mandated worldwide on all large aircraft [22]. Shown in figure 2.1 (adapted from Kochenderfer *et al.* [5]), the CAS state space consists of the relative altitude h_{rel} in meters, relative vertical rate $\dot{h}_{\text{rel}}$ in meters per second, the previous action a_{prev} in meters per second, and the time to collision t_{col} in seconds. The *ownship* is our own aircraft making the decisions and the *intruder* is a non-cooperative intruding aircraft. The objective is to provide climb or descend maneuver guidance by issuing 5 m/s, -5 m/s, or 0 m/s relative vertical rate change actions.

2.1.2 Autonomous Runway Detection

To support the development of safe autonomous aircraft, subsystems such as camera-based runway detectors help localize the autonomous aircraft on an approach to land [23]. Companies such as Airbus (through their A^3 subdivision) [24] and Xwing (now part of Joby Aviation) [25] have been developing vision-based runway detection systems that rely on complex neural networks to detect and track the edges of the runway. The runway information is used to augment GPS/INS localization information during landing. Validating the runway detection system is an important step not only for system safety before deployment, but to help in the certification process of machine learning components for autonomous flight [25]. The state space for runway detection systems consists of all possible RGB images of runways the system will encounter. Thus, validating in simulation over a parametric space (e.g., glide slope angle, distance to runway, time of day, weather, etc.) is important to study to reduce the dimensionality of the near-infinite possible runway images. It is important that high-fidelity simulators generate the runway images to ensure that the system sees representative examples of images it would encounter during deployment.

2.1.3 Airborne Wildfire Suppression

Wildfire incident commanders are tasked with processing continuous streams of information to enable effective decision making when allocating resources to mitigate the spread of and suppress wildfires [26]. Recently, the development of decision support tools to provide recommendations to incident commanders has been studied to help offload planning where to place resources to reduce the over-saturation of information [27]. Airborne wildfire suppression systems are tools designed to recommend approximately optimal placements of aerial suppression resources by planning over the uncertainty in the wildfire spread. Such tools need to consider the potential catastrophic damage to land and minimize the risk of the wildfire spreading to populated areas. The state space consists of all currently burning areas, the remaining fuel levels (as a function of the terrain), elevation, and the angle and speed of the current winds. The suppression agent can make aerial resource placements to suppress the wildfire with some probability of complete suppression and the environment transitions based on models of the wildfire spread [28]. The goal is to quickly suppress the wildfire while simultaneously protecting nearby populated areas.

2.2 Robotics: Safety-Critical Systems

Robots have been used as autonomous systems in cases deemed too risky for humans [29]. Our work studies robotics systems dealing with localization, navigation, and exploration.

2.2.1 Robot Localization and Navigation

When robots operate in real-world environments, they require sensor measurements to observe the world around them. Using these measurements, robots can localize their position to enable better performance in their primary objective, e.g., navigate to a goal. A common benchmark used to study sequential decision making algorithms is the *light dark* problem [30], consisting of a simplified 1D environment where a robot receives noisy observations in the *dark* region, and more accurate observations in the *light* region. The task of the robot is to first localize its position in the light region, and then navigate to a goal. Considered an information gathering problem, this problem deals with planning over long horizons to get to the light region to localize, delaying the primary goal.

2.2.2 Robot Exploration

Robots have also been used to explore unknown or dangerous environments, such as the lunar surface [6]. Another common benchmark for sequential decision making is the *rock sample* problem introduced by Smith *et al.* [31]. In rock sample, an autonomous agent operates in an $n \times n$ grid and is tasked with collecting information about the k rocks in the environment. Certain rocks are marked *good*, while others are marked *bad*. The objective is to collect information about all of the good rocks, while avoiding any penalties if observing the bad rocks. Rock sample is a scalable problem in n and k, allowing researchers to study more challenging, high-dimensional problems.

2.3 Geological Sustainability: Safety-Critical Systems

Geological sustainability plays a crucial role in mitigating the impacts of climate change through systems like carbon capture and storage [32], geothermal energy production [33], and critical mineral exploration and discovery [8]. Many geological systems evolve over long time horizons and present significant challenges due to limited subsurface observability and high uncertainty. In these safety-critical scenarios, it is often challenging to balance information gathering with preserving safety of infrastructure or human life.

2.3.1 Critical Mineral Exploration

As we move away from reliance on fossil fuels, the need for rare Earth metals—such as copper and lithium [34]—becomes increasingly important. Recent research has proposed the use of decision support tools to automate the selection of drilling locations in critical mineral exploration and discovery problems [8]. In the critical mineral exploration problem, an agent selects drilling locations to better understand the uncertain subsurface through bore hole measurements. Using prior models developed by geologists, the agent can collect information about the subsurface and use this information to make a final `mine` or `abandon` decision regarding the economic value of the project. Safety, in this case, can be considered either as minimizing the risk of induced seismicity [35] or constraints on the risk of exceeding the project's budget. In more recent work, detailed in chapter 5, the use of passive cosmic-ray muon tomography [36] has been developed as an alternative means to collect subsurface information about an ore body or intrusion.

2.3.2 Safe Carbon Capture and Storage

To help offset CO_2 emissions, carbon capture and storage (CCS), or *carbon sequestration*, is a method to capture CO_2 from the atmosphere and store it in underground porous rock formations [32]. Wang *et al.* [37] proposed a formal CCS problem formulation, enabling direct application and study of sequential decision making algorithms. In the CCS problem, an agent makes decisions on where to drill to inject CO_2 and builds knowledge of the CO_2 plume migration through information gathering actions. Safely storing CO_2 is important as improper understanding of the subsurface may lead to concentrated CO_2 leakage back out into the atmosphere [38]. We apply our methods to a 2D problem formulation of safe CCS developed by Corso *et al.* [39].

2.4 Broader Applications

Across all domains presented in this chapter, a common theme is the need to make sequential decisions under uncertainty with only partial information. The POMDP framework provides a principled way to model these problems, allowing autonomous systems or decision support tools to explicitly reason about uncertainty and risk. The safety-critical domains studied in this thesis were chosen to reflect a range of problem sizes, time horizons, and failure modes. Although we focus on aviation, robotics, and geological systems, many other safety-critical applications—including healthcare [40], autonomous vehicles [41], disaster response [42], and financial planning [43]—share these characteristics. Extending this work to such domains requires scalable planning under uncertainty, efficient belief representation, and domain-specific safety constraints—challenges our contributions are designed to address.

Chapter 3

Preliminaries

The present d-separates the future from the past.

Stefano Ermon

In this chapter, we provide background on concepts used in this thesis: sequential decision making frameworks, planning algorithms, surrogate modeling, and safety validation.

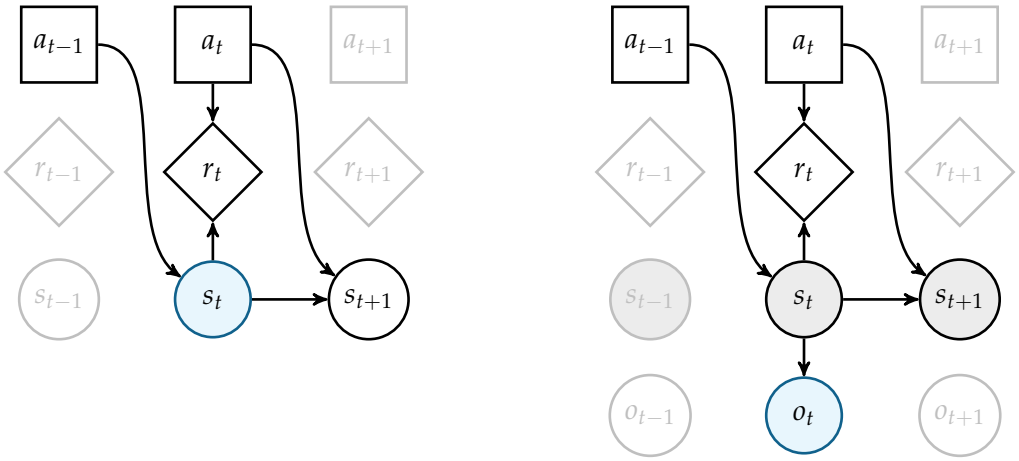
3.1 Markov Decision Processes

The *Markov decision process* (MDP) [5, 11], shown in figure 3.1a, is a sequential decision making framework used to model problems where the state-transition dynamics are uncertain. An MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, where $\mathcal{S}$ is the space of all states $s \in \mathcal{S}$ (discrete or continuous), the action space is denoted $\mathcal{A}$ where $a \in \mathcal{A}$ (discrete or continuous), the state-transition function is denoted $s' \sim T(\cdot \mid s, a)$ for a next state s', the reward function is denoted $r = R(s, a)$ or $r = R(s, a, s')$, and the discount factor $\gamma \in [0, 1)$ controls the problem's degree of myopia, where the discount factor could be $\gamma = 1$ for undiscounted problems (noting that the focus of this thesis is on *discounted MDPs*).

To solve an MDP, planning algorithms (section 3.3) construct a policy π that maps states to actions: $a = \pi(s)$ or $a \sim \pi(\cdot \mid s)$. The objective of planning algorithms is to find a policy that maximizes the expected discounted sum of rewards (referred to as the *value*):

$$\underset{\pi}{\text{maximize}} \;\; V^\pi(s_0) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 \sim \mathcal{S}_0,\, a_t \sim \pi(\cdot \mid s_t) \right] \tag{3.1}$$

where the initial state s_0 is sampled from the initial state distribution $\mathcal{S}_0$ (also denoted μ_0).



(a) The MDP problem formulation.

(b) The POMDP problem formulation.

Figure 3.1: Dynamic decision networks for the MDP and POMDP problem formulations.

3.2 Partially Observable Markov Decision Processes

The *partially observable Markov decision process* (POMDP) [5, 44, 45] is a model for sequential decision making problems where the true state is unobservable, modeling the real world. Defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$, the POMDP is an extension to the MDP framework used in reinforcement learning and planning with the addition of an observation space $\mathcal{O}$ (where $o \in \mathcal{O}$) and observation model $O(o \mid a, s')$. Given a current state $s \in \mathcal{S}$ and taking an action $a \in \mathcal{A}$, the agent transitions to a new state s' using the transition model $s' \sim T(\cdot \mid s, a)$. Without access to the true state, an observation is received $o \sim O(\cdot \mid a, s')$ and used to update the belief b over the possible next states s' to get the posterior:

$$b'(s') \propto O(o \mid a, s') \int T(s' \mid s, a) b(s) \, \mathrm{d}s \tag{3.2}$$

where the belief is a distribution over states—capturing state uncertainty. Examples of the different types of belief representations and updating methods are detailed in section 3.2.2.

A stochastic POMDP policy $\pi(a \mid b)$ is defined as the distribution over actions given the current belief b. After taking an action $a \sim \pi(\cdot \mid b)$, the agent receives a reward r from the environment according to the reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ or $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ using the next state. Note that the policy may be deterministic, i.e., $a = \pi(b)$.

3.2.1 Belief-State MDPs

In *belief-state MDPs*, a POMDP is converted to an MDP by treating the belief as a state [5, 45]. The reward function then becomes a weighting of the state-based reward function:

$$R_b(b, a) = \int b(s)R(s, a)\,\mathrm{d}s \tag{3.3}$$

The belief-state MDP (BMDP) shares the same action space as the underlying POMDP and operates over a belief space $\mathcal{B}$ that is the simplex over the state space $\mathcal{S}$. The belief-state MDP defines a new belief-state transition function $b' \sim T_b(\cdot \mid b, a)$ as:

$$s \sim b(\cdot) \qquad s' \sim T(\cdot \mid s, a) \qquad o \sim O(\cdot \mid a, s') \qquad b' \leftarrow \textsc{Update}(b, a, o) \tag{3.4}$$

where the belief update can be done using methods from section 3.2.2. Therefore, the belief-state MDP is defined by the tuple $\langle \mathcal{B}, \mathcal{A}, T_b, R_b, \gamma \rangle$ with the finite-horizon discount factor $\gamma \in [0, 1)$ that controls the effect that future rewards have on the current action.

The objective to solve BMDPs is to find a policy π that maximizes the *value function*:

$$\underset{\pi}{\text{maximize}} \;\; V^\pi(b_0) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_b(b_t, a_t) \;\middle|\; b_t \sim T_b(\cdot \mid s_t, a_t),\; a_t \sim \pi(\cdot \mid b_t) \right] \tag{3.5}$$

given an initial belief b_0, the transitioned belief b_t, and an action a_t from the policy π. Different belief updating methods are used depending on the problem structure (i.e., discrete or continuous state and observation spaces and potentially nonlinear transition dynamics).

3.2.2 Belief Updating

Beliefs can be represented in different ways, namely parametric or non-parametric representations [5]. For parametric beliefs, the belief distribution is represented by a set of parameters for a fixed distribution family (e.g., categorical distribution where probability mass is assigned to each discrete state or multivariate Gaussian distributions for continuous problems). For non-parametric beliefs, the belief distribution is represented by a collection of states (often called *state particles*), which are points sampled from the posterior state space. Depending on the belief representation, different belief updating algorithms can be used.

Discrete state filtering. If the state and observation spaces are discrete, then we can perform exact belief updates [5]. This means we can arrive at an analytical solution to the belief updating without approximations. For discrete states and observations with relatively small space sizes $|\mathcal{S}|$ and $|\mathcal{O}|$, we can use the *discrete state filter* (also called *recursive Bayesian estimation*) to infer the unknown belief distribution. The discrete state filter updates the belief recursively over time, using the Markov assumption which says that the current state s_t is conditionally independent of all past states and actions given the previous state s_{t-1} and action a_{t-1}. Due to the independence assumption, if an agent with belief b takes an action a and receives an observation o, then the new belief b' becomes:

$$\begin{aligned}
b'(s') &= P(s' \mid b, a, o) \\
&\propto P(o \mid b, a, s')P(s' \mid b, a) && \text{(Bayes' rule)} \\
&\propto O(o \mid a, s')P(s' \mid b, a) && \text{(observation definition)} \\
&\propto O(o \mid a, s') \sum_s P(s' \mid b, a, s)P(s \mid b, a) && \text{(law of total probability)} \\
&\propto O(o \mid a, s') \sum_s T(s' \mid s, a)b(s) && \text{(state transition model)}
\end{aligned}$$

We can call this the *unnormalized* belief $\hat{b}'$. The exact belief update is then normalized so that beliefs sum to one:

$$b'(s') = \frac{\hat{b}'(s')}{\sum_{s''} \hat{b}'(s'')} = \frac{O(o \mid a, s') \sum_s T(s' \mid s, a)b(s)}{\sum_{s''} O(o \mid a, s'') \sum_s T(s'' \mid s, a)b(s)} \tag{3.6}$$

This ensures that $b'(s') \geq 0$ for all $s' \in \mathcal{S}$ and that $\sum_{s'} b'(s') = 1$, making b' a valid probability distribution (i.e, a valid probability mass function in the discrete case).

Kalman filtering. So far, we have shown how to update beliefs for discrete POMDPs (discrete states and observations). In continuous state and observation problems, updating the belief involves integrating instead of summing:

$$b'(s') \propto O(o \mid a, s') \int T(s' \mid s, a)b(s) \, \mathrm{d}s \tag{3.7}$$

Integrating over the state space may be intractable; therefore, methods have been developed to address this challenge. If certain linear-Gaussian assumptions are met, then we can update the belief using a *Kalman filter* [5, 46, 47]. In a standard Kalman filter, we can compute the

Algorithm 3.1: Kalman filter belief updater.

function KALMANFILTER(b, a, o)
  $(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p) \leftarrow$ KALMANPREDICT(b, a) ▷ linear-Gaussian prediction
  $b' \leftarrow$ KALMANUPDATE($b, o, \boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p$) ▷ Gaussian update
  return b'

exact belief update under the following assumptions. We assume the transition model T and observation model O to be linear-Gaussian and assume the belief b to be Gaussian, then:

$$T(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) = \mathcal{N}(\mathbf{s}' \mid \mathbf{T}_s\mathbf{s} + \mathbf{T}_a\mathbf{a},\ \boldsymbol{\Sigma}_s) \quad \text{(linear-Gaussian transition)}$$

$$O(\mathbf{o} \mid \mathbf{s}') = \mathcal{N}(\mathbf{o} \mid \mathbf{O}_s\mathbf{s}',\ \boldsymbol{\Sigma}_o) \quad \text{(linear-Gaussian observation)}$$

$$b(\mathbf{s}) = \mathcal{N}(\mathbf{s} \mid \boldsymbol{\mu}_b,\ \boldsymbol{\Sigma}_b) \quad \text{(Gaussian belief)}$$

where $\boldsymbol{\mu}_b$ and $\boldsymbol{\Sigma}_b$ are the mean vector and covariance matrix that define the Gaussian belief, the state transition covariance is $\boldsymbol{\Sigma}_s$, and the observation covariance is $\boldsymbol{\Sigma}_o$. As shown in algorithm 3.1, the first step is prediction. The prediction step uses the transition dynamics to get a predicted distribution that is parameterized by the following mean and covariance:

$$\boldsymbol{\mu}_p = \mathbf{T}_s\boldsymbol{\mu}_b + \mathbf{T}_a\mathbf{a} \quad \text{(predicted mean)}$$

$$\boldsymbol{\Sigma}_p = \mathbf{T}_s\boldsymbol{\Sigma}_b\mathbf{T}_s^\top + \boldsymbol{\Sigma}_s \quad \text{(predicted covariance)}$$

The update step then uses the prediction to update the belief:

$$\mathbf{K} = \frac{\boldsymbol{\Sigma}_p\mathbf{O}_s^\top}{\mathbf{O}_s\boldsymbol{\Sigma}_p\mathbf{O}_s^\top + \boldsymbol{\Sigma}_o} \quad \text{(Kalman gain)}$$

$$\boldsymbol{\mu}_b = \boldsymbol{\mu}_p + \mathbf{K}(\mathbf{o} - \mathbf{O}_s\boldsymbol{\mu}_p) \quad \text{(updated mean)}$$

$$\boldsymbol{\Sigma}_b = (\mathbf{I} - \mathbf{K}\mathbf{O}_s)\boldsymbol{\Sigma}_p \quad \text{(updated covariance)}$$

Here we note that the states, observations, and actions may be vectors $\mathbf{s}$, $\mathbf{o}$, and $\mathbf{a}$, respectively.

Nonlinear Kalman filtering. To address problems with nonlinear dynamics and Gaussian noise, the *extended Kalman filter* (EKF) [48] was developed. The EKF method requires differentiable functions $\mathbf{f}_T(\mathbf{s}, \mathbf{a})$ and $\mathbf{f}_O(\mathbf{a}, \mathbf{s}')$, where the Gaussian transition function is defined as $T(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) = \mathcal{N}(\mathbf{s}' \mid \mathbf{f}_T(\mathbf{s}, \mathbf{a}), \boldsymbol{\Sigma}_s)$ and the Gaussian observation function is

defined as $O(\mathbf{o} \mid \mathbf{a}, \mathbf{s}') = \mathcal{N}(\mathbf{o} \mid \mathbf{f}_O(\mathbf{a}, \mathbf{s}'), \mathbf{\Sigma}_o)$ [5]. The key idea is that the EKF uses a local linear approximation of the nonlinear dynamics. Another extension to the standard Kalman filter, namely the *unscented Kalman filter* (UKF) [49], can be applied in cases where the dynamics functions $\mathbf{f}_T$ and $\mathbf{f}_O$ are non-differentiable. The UKF uses weighted sigma point samples to approximate the continuous Gaussian belief and passes the sigma points through an unscented transform.* Sigma points are then updated by the UKF and used to reconstruct the updated $\boldsymbol{\mu}_b$ and $\mathbf{\Sigma}_b$ (instead of updating the nonlinear Gaussian directly).

Particle filtering. In problems with large discrete state and observation spaces or continuous problems not well approximated by linear-Gaussian dynamics, other approximations are required to estimate the belief update. One such method is the *particle filter* [51], which can represent a broad range of non-parametric distributions [52]. In a particle filter, the belief is represented as a collection of states where each state in the approximated belief is called a *state particle*. Shown in algorithm 3.2, the particle filter first transitions all state particles through the stochastic transition function $s_i' \sim T(\cdot \mid s_i, a)$ for $s_i \in \mathbf{s}$, where $\mathbf{s}$ is a collection of states approximating the belief, i.e., $b \approx \mathbf{s}$. The particle filter then weighs each next state based on how likely the input observation o is given the next states $s_i' \in \mathbf{s}'$ and current action a. Then the particles are resampled based on the particle weights, and the new approximated belief (i.e., particle set) is returned.

Algorithm 3.2: Particle filter belief updater.

function PARTICLEFILTER($\mathbf{s}, a, o$)
  $\mathbf{s}' \sim T(\cdot \mid \mathbf{s}, a)$ ▷ propagate past particles $\mathbf{s}$ through the transition model
  $\mathbf{w} \leftarrow O(o \mid \mathbf{s}', a)$ ▷ weight next states based on the observation model
  particles $\sim$ SetCategorical $\left(\mathbf{s}', \frac{\mathbf{w}}{\sum_i w_i}\right)$ ▷ sample new particles with normalized weights
  return particles

In chapter 5, we learn belief updating surrogates to address problems where the observation likelihood model $O(o \mid a, s')$ may be unknown or hard to compute and where beliefs may not be well approximated directly by Gaussian distributions.

*The *unscented transform* was arbitrarily named by Jeffrey Uhlmann, inspired by a colleague's unscented deodorant to prevent the name ''Uhlmann filter'' from sticking (it's derivative-free—how clean!) [50].

3.3 Planning Algorithms

Separating the *problem structure* (i.e., POMDP definition) from the *solution method* (i.e., planning algorithm) is beneficial as separate research can focus on solution methods given a variety of challenging problems. This section will outline several different MDP and POMDP solution methods that each have the goal of computing an (approximately) optimal policy π that maximizes the expected discounted sum of rewards.

3.3.1 Offline Planning

In *offline planning* algorithms, a policy is constructed offline (i.e., pre-computed) that maps states to actions. The offline policy can be represented as a lookup table that returns an action for every state. The offline policy can also be represented as a neural network surrogate, as shown by Mnih *et al.* [53] in the development of deep Q-learning networks (DQNs). Although the differences are subtle, this work focuses on *planning algorithms* and not *reinforcement learning algorithms* (such as Q-learning [54]).

Planning vs. reinforcement learning. In MDP and POMDP *planning*, models of the transitions, rewards, and observations (for POMDPs) are known. In contrast, in the model-based reinforcement learning (RL) domain (or the partially observable reinforcement learning (PORL) domain), these models are learned along with a policy or value function [55, 56]. A key difference between these settings is that RL algorithms reset the agent and learn through experience, while planning algorithms must consider future trajectories from *any state*. When RL problems have deterministic state transitions, they can be cast as planning problems by replaying the full state trajectory along a path, which may be prohibitively expensive for long-horizon problems. Both settings are closely related and pose interesting research challenges. Specifically, sequential planning over given models in high-dimensional, long-horizon POMDPs remains challenging [57]. The content of this thesis attempts to address some of these challenges.

Policy Iteration

In discrete MDPs with relatively small state and action spaces, *policy iteration* [11, 55] can be used to solve for an optimal policy. In policy iteration, the optimal policy π^* is computed by iterating the following two steps [58]. Starting with any policy π_0, policy iteration performs:

1. **Policy evaluation**: Given the current policy, compute the value function.
2. **Policy improvement**: Using the value function, compute an improved policy.

During *policy evaluation*, the current policy at iteration k uses dynamic programming to compute the value function for n steps (where $t \in [1, \ldots, n]$):

$$V_t^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s' \mid s, \pi(s)) V_{t-1}^{\pi}(s') \quad \text{for all } s \in \mathcal{S} \tag{3.8}$$

Then during *policy improvement*, the value function returned from policy evaluation is used to compute the improved policy, again using the Bellman equation:

$$\pi_{k+1}(s) = \arg\max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s'} T(s' \mid s, a) V^{\pi_k}(s') \right) \tag{3.9}$$

The resulting policy is often stored in a lookup table [58] so it can easily be queried during runtime usage (i.e., queries to the pre-computed offline policy online). In chapter 6, we will use the policy iteration paradigm combined with learning a neural network surrogate of the value function and an action selection policy for high-dimensional POMDPs.

Value Iteration

Another offline method to solve for the optimal policy in discrete MDPs is called *value iteration* [11]. Similar to policy iteration, value iteration uses dynamic programming and the Bellman equation to compute the optimal *value* (or *utility*) V iteratively [58]:

$$V^*(s) = \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s'} T(s' \mid s, a) V^*(s') \right) \tag{3.10}$$

Once the value function has converged, namely $|V_{k-1}(s) - V_k(s)| \leq \epsilon$ for all $s \in \mathcal{S}$, the optimal policy π^* can then be extracted using the final value function:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s'} T(s' \mid s, a) V^*(s') \right) \tag{3.11}$$

Similar to policy iteration, the policy output from value iteration can be stored in an efficient lookup table to be queried during online evaluation.

Local approximation value iteration

For large or continuous state spaces, approximate dynamic programming can be useful in finding approximately optimal polices for the substructure of the problem [58]. For a finite set of n discretized states, a function $\beta(s)$ is used to weigh ''nearby'' states, where $\sum_{i=1}^{n} \beta_i(s) = 1$. The value $\lambda_i \in \boldsymbol{\lambda}$ for each state s_i is used to approximate the value of an arbitrary state as $V(s) = \sum_{i=1}^{n} \lambda_i \beta_i(s) = \boldsymbol{\lambda}^\top \beta(s)$. Given a discretization of n states, the values are initialized to zero, namely $\boldsymbol{\lambda} = \mathbf{0}$, and then the following is iterated until convergence:

$$u_i = \max_{a \in \mathcal{A}} \left(R(s_i, a) + \gamma \sum_{s' \in s_{1:n}} T(s' \mid s_i, a) \boldsymbol{\lambda}^\top \beta(s') \right) \text{ for } i \in [1, \ldots, n] \tag{3.12}$$

$$\boldsymbol{\lambda} = \mathbf{u} \tag{3.13}$$

The final policy is extracted the same way as equation (3.11) except now using the neighboring approximation $\beta(s)$ (also known as a *kernel* function). To approximately solve POMDPs, *local approximation value iteration* (LAVI), as presented here, can be used by discretizing the belief space and treating the problem as a belief-state MDP—as is done in chapter 6.

3.3.2 Online Planning

In contrast to offline methods, *online planning* algorithms compute the approximately optimal action *during runtime* (i.e., online). Online methods are often used in problems with large or continuous state, action, and observation spaces. Instead of explicitly constructing a policy over all states or (discretized) beliefs, online planning algorithms estimate the next best action through a planning procedure, often a best-first tree search.

Monte Carlo Tree Search (MCTS)

Monte Carlo tree search [10, 59] is an online, recursive, best-first tree search algorithm to determine the approximately optimal action to take from a given root state of an MDP. Extensions to MCTS have been applied to POMDPs through several algorithms: *partially observable Monte Carlo planning* (POMCP) treats the state nodes as histories h of action-observation trajectories [60], *POMCP with observation widening* (POMCPOW) constructs weighted particle sets at the observation nodes and extends POMCP to fully continuous domains [61], and *particle filter trees* (PFT) and *information PFT* (IPFT) treat the POMDP as

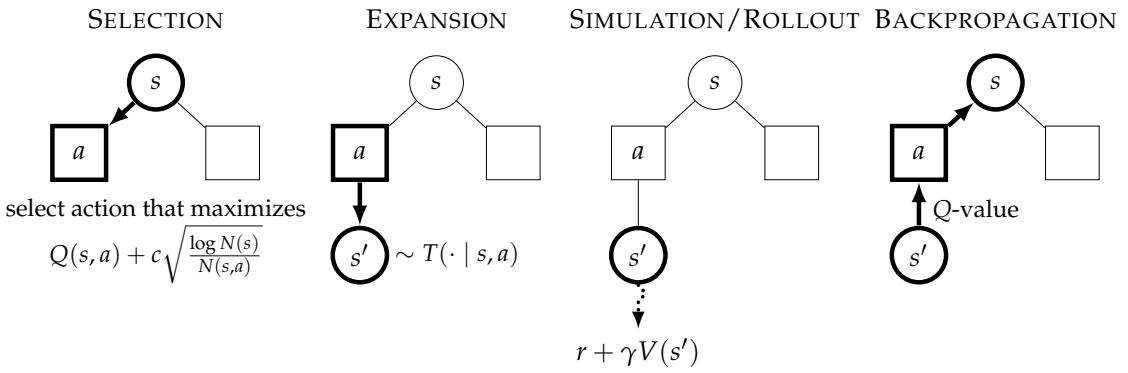


Figure 3.2: The four stages of Monte Carlo tree search.

a belief-state MDP and plan directly over the belief-state nodes using a particle filter [61, 62]. All variants of MCTS execute the following four steps shown in figure 3.2. Because MCTS can be applied to MDPs and belief-state MDPs, this section uses s to represent the state, the history h, and the belief state b, and will refer to them as ''the state''.

1. **Selection.** During *selection*, an action is selected from the children of a state node based on criteria that balances exploration and exploitation. The *upper-confidence tree* algorithm (UCT) [63] is a common criterion that selects an action that maximizes the upper-confidence bound $Q(s,a) + c\sqrt{\log N(s)/N(s,a)}$ where $Q(s,a)$ is the Q-value estimate for state-action pair (s,a) with a visit count of $N(s,a)$, the total visit count of $N(s) = \sum_a N(s,a)$ for the children $a \in A(s)$, and an exploration constant $c > 0$. Rosin [64] introduced the *UCT with predictor* algorithm (PUCT), modified by Silver *et al.* [65], where a predictor $P(s,a)$ guides the exploration towards promising branches and selects an action according to the following:

$$\underset{a \in A(s)}{\arg\max}\ Q(s,a) + c\left(P(s,a)\frac{\sqrt{N(s)}}{1 + N(s,a)}\right) \tag{3.14}$$

2. **Expansion.** In the *expansion* step, the selected action is taken in simulation and the transition model $T(s' \mid s,a)$ is sampled to determine the next state s'. When the transitions are deterministic, then the child node is always a single state. If the transition dynamics are stochastic, techniques to balance the branching factor, such as progressive widening [66] and state abstraction refinement [67], have been proposed.

3. **Rollout/Simulation.** In the *rollout* step, also called the *simulation* step due to recursively simulating the MCTS tree expansion, the value is estimated through the execution of a rollout policy until termination or using heuristics to approximate the value function from the given state s'. Expensive rollouts done by AlphaGo [68] were replaced with a value network lookup in AlphaGo Zero and AlphaZero [17, 65].

4. **Backpropagation.** Finally, during the *backpropagation* step, the Q-value estimate from the rollout is propagated up the path in the tree as a running average.

Root node action selection. After repeating the four steps of MCTS, the final action is selected from the children $a \in A(s)$ of the root state s and executed in the environment. One way to select the best root node action, referred to as the *robust child* [59, 69], selects the action with the highest visit count as $\arg\max_a N(s,a)$. Sampling from the normalized counts, exponentiated by an exploratory temperature $\tau > 0$, is also common [65]. Another method uses the highest estimated Q-value as $\arg\max_a Q(s,a)$. Both criteria have been shown to have problem-based trade-offs [59].

Double progressive widening. To handle stochastic state transitions and large or continuous state and action spaces, double progressive widening (DPW) balances between sampling new nodes to expand or selecting from existing nodes already in the tree [66]. Two hyperparameters $\alpha \in [0,1]$ and $k \geq 0$ control the branching factor. If the number of actions tried from state s is less than $kN(s)^\alpha$, then a new action is sampled from the action space and added as a child of node s. Likewise, if the number of expanded states from node (s,a) is less than $kN(s,a)^\alpha$, then a new state is sampled from the transition function $s' \sim T(\cdot \mid s,a)$ and added as a child. If the state widening condition is not met, then a next state is sampled from the existing children. Typically, different parameters (α_a, k_a) and (α_s, k_s) are used.

3.4 Surrogate Modeling

A *surrogate model* is an approximate function that acts as a ''stand-in'' for unknown, complex, or more expensive functions or simulators [13]. Concretely, a surrogate model is a function $\hat{f}(\mathbf{x})$ that approximates a true function $f(\mathbf{x})$ for some input domain $\mathbf{x} \in \mathcal{X}$. Two commonly used surrogate models studied in this work are Gaussian processes (probabilistic surrogate models) [70] and neural networks (which can handle high-dimensional data) [71].

3.4.1 Gaussian Processes as Probabilistic Surrogate Models

Probabilistic surrogate models are particularly useful in Bayesian optimization when we want to maximize or minimize some expensive, black-box function $f(\mathbf{x})$.

Bayesian optimization. The basic optimization problem [72] is to maximize (or minimize) a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ subject to $\mathbf{x}$ lying in the design space $\mathcal{X} \subseteq \mathbb{R}^n$:

$$\begin{aligned} \underset{\mathbf{x}}{\text{maximize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \mathcal{X} \end{aligned} \tag{3.15}$$

Bayesian optimization is a black-box approach to globally optimize the objective f without requiring any information about internals of the function, e.g., no requirement on gradient information [73, 74]. The main idea is to iteratively fit a *probabilistic surrogate model*—such as a Gaussian process [70]—to evaluation points of the true objective function and then propose new design points to evaluate based on the information and uncertainty quantified in the surrogate. Bayesian optimization is especially useful when f is computationally expensive to evaluate and the surrogate $\hat{f}$ is fast to evaluate in comparison [73]. Figure 3.3 illustrates a Bayesian optimization example where the next sampled design point $\mathbf{x}'$ (shown as a green triangle) maximizes the *upper-confidence bound* (UCB) acquisition function [72]:

$$\mathbf{x}' = \underset{\mathbf{x} \in \mathcal{X}}{\arg\max}\, \hat{f}(\mathbf{x}) + \lambda \hat{\sigma}(\mathbf{x}) \tag{3.16}$$

where $\hat{f}$ is the mean of the surrogate model, $\hat{\sigma}$ is the standard deviation, and $\lambda \geq 0$ controls the trade-off between exploration (based on the uncertainty) and exploitation (based on

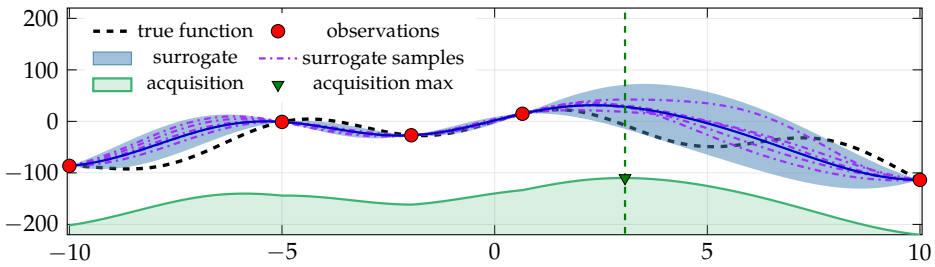


Figure 3.3: Example problem using GP Bayesian optimization with UCB exploration.

the mean). Using a probabilistic approach when fitting the surrogate model allows us to use uncertainty in the underlying objective when acquiring subsequent samples.

Gaussian processes. One method for constructing a probabilistic surrogate model is to use a *Gaussian process* (GP) [70]. Given true observations from the objective function, a GP is defined as a distribution over possible underlying functions that describe the observations [72] (illustrated in figure 3.3 as purple dashed lines showing five functions sampled from the GP). Given the set of n inputs $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and n true observations $\mathbf{y} = [y_1, \ldots, y_n]$ where $y_i = f(\mathbf{x}_i)$, a Gaussian process is parameterized by a mean function $\mathbf{m}(\mathbf{X})$, generally set to the zero-mean function $\mathbf{m}(\mathbf{X})_i = m(\mathbf{x}_i) = 0$ if no prior information is given, and a kernel function $\mathbf{K}(\mathbf{X}, \mathbf{X})$ that captures the correlations between data points as covariances. The output of the kernel function is an $n \times n$ matrix where the element $\mathbf{K}(\mathbf{X}, \mathbf{X})_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. The kernel $k(\mathbf{x}_i, \mathbf{x}_j)$ may be selected based on spatial information about the relationship of neighboring data points in design space (e.g., if the relationship is smooth, then one can use a squared exponential kernel [72]). In chapter 8, we use the isotropic Matérn 1/2 kernel [75] with length scale $\ell = \exp(-1/10)$ and signal standard deviation $s_\sigma = \exp(-1/10)$:

$$k(\mathbf{x}_i, \mathbf{x}_j) = s_\sigma^2 \exp\Big(-|\mathbf{x}_i - \mathbf{x}_j|/\ell\Big) \tag{3.17}$$

The choice of kernel and its parameters can be separately optimized depending on the problem and information about the behavior of the system (see Williams *et al.* [70]), where the kernel was chosen for the work in chapter 8 based on the characteristic that the Matérn kernel can capture more variation in neighboring values [70]. Using the mean function and kernel parameterization and conditioning on the true observations $\mathbf{y}$, the GP produces samples for new points $\mathbf{X}'$ of the function it is trying to estimate as:

$$\hat{\mathbf{y}} \mid \mathbf{y} \sim \mathcal{N}\big(\boldsymbol{\mu}(\mathbf{X}, \mathbf{X}', \mathbf{y}), \boldsymbol{\Sigma}(\mathbf{X}, \mathbf{X}')\big) \tag{3.18}$$

$$\boldsymbol{\mu}(\mathbf{X}, \mathbf{X}', \mathbf{y}) = \mathbf{m}(\mathbf{X}') + \mathbf{K}(\mathbf{X}', \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\big(\mathbf{y} - \mathbf{m}(\mathbf{X})\big) \tag{3.19}$$

$$\boldsymbol{\Sigma}(\mathbf{X}, \mathbf{X}') = \mathbf{K}(\mathbf{X}', \mathbf{X}') - \mathbf{K}(\mathbf{X}', \mathbf{X})\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\mathbf{K}(\mathbf{X}, \mathbf{X}') \tag{3.20}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the mean and covariance functions. Across the domain $\mathbf{X}'$, the estimates $\hat{\mathbf{y}}$ can now be used as surrogates for the true function $f(\mathbf{x}')$ for $\mathbf{x}' \in \mathbf{X}'$.

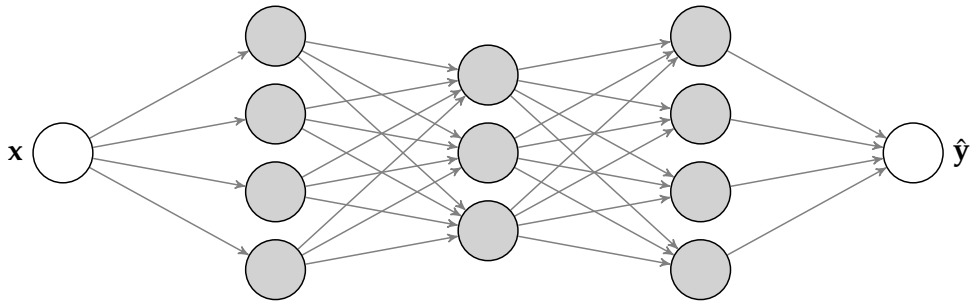


Figure 3.4: Neural network surrogate model approximating $\hat{\mathbf{y}} = f_\theta(\mathbf{x})$.

3.4.2 Neural Networks as Nonlinear Function Approximators

Used often in modern machine learning [14], neural networks can be used as surrogates to approximate complex nonlinear functions [5]. A neural network is a differential function $\hat{\mathbf{y}} = f_\theta(\mathbf{x})$ that is parameterized by a set of weights and biases θ. Because the network is differentiable, the machine learning community has focused on gradient-based methods to optimize the parameters θ [72, 76, 77].

Neural networks are primarily used for two different tasks, namely classification and regression. In multi-class classification (with K classes), the objective of a gradient-based optimization method is to minimize the *cross-entropy loss*:

$$\mathcal{L}_{\text{CE}}(y, \hat{y}; \theta) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_i^{(k)} \log \hat{y}_i^{(k)} \tag{3.21}$$

where y is the one-hot encoded true class label and $\hat{y}$ is the predicted probability distribution over the K classes for n batches. The predicted probabilities are obtained by applying a *softmax* function to the vector of logits $z = [z_1, \ldots, z_K]$ output by the neural network:

$$\hat{y}^{(k)} = \frac{\exp z_k}{\sum_{j=1}^{K} \exp z_j} \tag{3.22}$$

For single-class classification, e.g., in cases where we want to predict a failure ($y = 1$) or success ($y = 0$), the *binary cross-entropy* loss is used instead:

$$\mathcal{L}_{\text{BCE}}(y, \hat{y}; \theta) = -\frac{1}{n} \sum_{i=1}^{n} \Big(y_i \log \big(\sigma(\hat{y}_i)\big) + (1 - y_i) \log \big(1 - \sigma(\hat{y}_i)\big) \Big) \tag{3.23}$$

where $\sigma(z) = 1/(1 + \exp(-z))$ is the sigmoid function ensuring the prediction lies in $[0, 1]$.

In regression tasks, the objective is to approximate a real-valued function for outputs $y \in \mathbb{R}^d$. This is often done by minimizing the mean-squared error or mean-absolute error between the true output y and prediction $\hat{y}$:

$$\mathcal{L}_{\text{MSE}}(y, \hat{y}; \theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.24}$$

$$\mathcal{L}_{\text{MAE}}(y, \hat{y}; \theta) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{3.25}$$

Chapter 5 further details neural network surrogates and how they can be used as generative models to sample from the posterior belief given the current observations.

3.5 Safety Validation

To validate safety-critical systems, developers often perform *safety validation* [15]. Safety validation has three primary tasks shown in figure 3.5. Given a system under test $f(\mathbf{x})$, its design space $\mathbf{x} \in \mathcal{X}$, and a property specification ψ [78], where $f(\mathbf{x}) \notin \psi$ indicates the system violated the specification, we get the following tasks:

1. **Falsification:** The process of finding *any* input $\mathbf{x}$ that results in a system failure [79]:

$$\mathbf{x} \quad \text{s.t.} \quad f(\mathbf{x}) \notin \psi \tag{3.26}$$

2. **Most-likely failure analysis:** Finding the failure with the maximum likelihood under the nominal distribution $p(\mathbf{x})$ [80]:

$$\arg\max_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}) \quad \text{s.t.} \quad f(\mathbf{x}) \notin \psi \tag{3.27}$$

3. **Failure probability estimation:** Estimating the probability that a failure will occur under the nominal distribution $p(\mathbf{x})$ over the domain of the design space $\mathbf{x} \in \mathcal{X}$ [81]:

$$p_{\text{fail}} = \mathbf{E}_p\left[\mathbb{1}\{f(\mathbf{x}) \notin \psi\}\right] \tag{3.28}$$

If methods focus on failure probability estimation, then it is often the case that all three safety validation tasks can be achieved. This is because when estimating the probability of

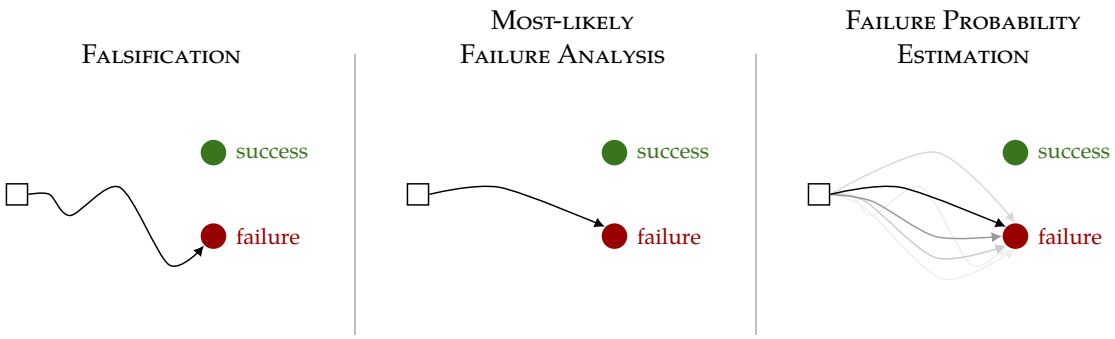


Figure 3.5: The three tasks of safety validation.

failure, samples from the normalized distribution of failures are produced. Thus, we achieve falsification by finding failures in the process of constructing the distribution and can easily compute the most-likely failure by maximizing the input likelihood across the nominal distribution. Motivated to achieve all three safety validation tasks, the work in chapter 8 develops an efficient approach to estimate probability of failure for black-box systems. For a survey of existing black-box safety validation algorithms, including falsification and most-likely failure analysis, we refer to Corso *et al.* [15].

Black-box vs. white-box validation. In the case of *black-box safety validation*, we treat the system f as a ''black box'' and attempt to perform the three tasks described above. The black-box assumption means that the only way to interact with the system is by passing inputs $\mathbf{x}$ and observing outputs $y = f(\mathbf{x})$. This is in contrast to *white-box validation* which requires information about the internals of the system to prove properties of safety [82, 83, 84]. In choosing to perform black-box validation, we can apply methods to more general systems, particularly to systems with neural network components. Although recent work has focused on verifying small neural networks [85, 86, 87], scaling to large networks remains a challenge [88]. Chapter 8 investigates an approach that attempts to address this challenge.

PART II:

PLANNING AT SCALE

Chapter 4

Batched Belief-State Planning

Ask not what your ~~country~~ [GPU] can do for you;
ask what you can do for your ~~country~~ [GPU].

John F. Kennedy [edited]

In many real-world decision making problems, an agent must select a single action at each timestep based on limited or uncertain knowledge of its current state. While the underlying environment may be modeled as a standard Markov decision process (MDP), planning under uncertainty often requires reasoning about multiple possible outcomes or future states. This motivates the use of *batched planning* methods, in which a set of sampled states are used to evaluate the expected performance of candidate actions. Such approaches have been widely adopted in belief-space planning [45, 89], distributionally robust decision making [90, 91], and batched reinforcement learning algorithms [92, 93], where parallel rollouts are used to compute uncertainty-aware value estimates. In this chapter, we formalize a batched extension of the MDP used strictly for planning: at each decision step, a single observed state s is duplicated to form a batched representation $\mathbf{s} = \{s, \ldots, s\}$, which is then used to evaluate the expected value of each action under stochastic transitions. The resulting batched model enables efficient, parallelized rollouts while ensuring that only a single action is selected and executed in the true environment. The batched planning formulation is motivated by the emphasis on modern GPU architectures for large-scale planning tasks, which can exploit the independence across batches to simulate transitions and compute rewards in parallel with significant speedups [94].

4.1 Batched Markov Decision Process

In this section we formally introduce the batched planning model. We define the *batched Markov decision process* (**B**-MDP) as the tuple $\langle \mathcal{S}^m, \mathcal{A}, \mathbf{T}, \widehat{R}, \gamma \rangle$, where the batched state transition function $\mathbf{s}' \sim \mathbf{T}(\cdot \mid \mathbf{s}, a)$ is $\mathbf{T} : \mathcal{S}^m \times \mathcal{A} \to \mathcal{S}^m$, and the set of next states is:

$$\mathbf{s}' = \{s_i'\}_{i=1}^m \quad \text{where} \quad s_i' \sim T(\cdot \mid s_i, a) \quad \forall s_i \in \mathbf{s} \tag{4.1}$$

The batched state space $\mathcal{S}^m$ is the multiset of m states:

$$\mathcal{S}^m = \{\{s_1, s_2, \ldots, s_m\} \mid s_i \in \mathcal{S}\} \tag{4.2}$$

with $\mathbf{s} \in \mathcal{S}^m$ denoting a batched state. The transition function $\mathbf{T}$ can be implemented via a parallelized (e.g., vectorized) form of the standard MDP dynamics.* The batched state-based reward function is defined as:

$$\widehat{R}(\mathbf{s}, \mathbf{a}) = \mathop{\mathbb{E}}_{s \in \mathbf{s}, a \in \mathbf{a}} \big[R(s, a)\big] = \frac{1}{m} \sum_{i=1}^{m} R(s_i, a_i) \tag{4.3}$$

where $\mathbf{a} \in \mathcal{A}^m$ denotes batched actions and $m = |\mathbf{s}| = |\mathbf{a}|$ is the batch size.

4.1.1 Optimal Batched Value Function

We show that the batched extension of an MDP preserves the optimality of the value function in the underlying MDP. The key insight is that, under the batched transition dynamics $\mathbf{T}$, each element $s_i \in \mathbf{s}$ transitions independently according to the underlying MDP transition model, as defined in equation (4.1). Similarly, the batched reward function from equation (4.3) is defined as the mean of individual rewards over the batch. This framework induces a natural decomposition of the batched value function into the average of individual value functions.

Formally, we show that this decomposition leads to the preservation of optimality, namely if π^* is an optimal policy for the underlying MDP, then the batched extension $\boldsymbol{\pi}^*$ yields a batched value function that is simply the average of optimal state values. Specifically, the batched value function $\widehat{V}^{\boldsymbol{\pi}^*}$ is pointwise optimal over the batch. We prove this result using the following theorem.

*In practice, this can be done either through a parallelized simulator, vectorizing the transition function, or a learned neural network surrogate of the transition function for easy GPU vectorization.

Theorem 1 (Batched optimal value preservation)**.** *Let $\pi : \mathcal{S} \to \mathcal{A}$ be any stationary policy over the underlying MDP, and let $\boldsymbol{\pi} : \mathcal{S}^m \to \mathcal{A}^m$ be its batched extension applied independently to each element of $\mathbf{s} \in \mathcal{S}^m$ where $\boldsymbol{\pi}(\mathbf{s}) = \{\pi(s_i)\}_{i=1}^m$. Then the batched value function satisfies:*

$$\widehat{V}^{\boldsymbol{\pi}}(\mathbf{s}) = \frac{1}{m} \sum_{i=1}^{m} V^{\pi}(s_i) \tag{4.4}$$

and in particular, if π^* is optimal for the underlying MDP, then:

$$\widehat{V}^{\boldsymbol{\pi}^*}(\mathbf{s}) = \frac{1}{m} \sum_{i=1}^{m} V^{*}(s_i) \tag{4.5}$$

That is, the batched value function preserves optimality in expectation.

Proof. The batched value function under policy $\boldsymbol{\pi}$ can be trivially derived as:

$$\widehat{V}^{\boldsymbol{\pi}}(\mathbf{s}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \widehat{R}\left(\mathbf{s}^{(t)}, \boldsymbol{\pi}(\mathbf{s}^{(t)})\right) \,\middle|\, \mathbf{s}^{(0)} = \mathbf{s} \right] \tag{4.6}$$

$$= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \left(\frac{1}{m} \sum_{i=1}^{m} R\left(s_i^{(t)}, \pi\left(s_i^{(t)}\right)\right)\right) \,\middle|\, \left\{s_1^{(0)}, \ldots, s_m^{(0)}\right\} = \mathbf{s} \right] \tag{4.7}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R\left(s_i^{(t)}, \pi\left(s_i^{(t)}\right)\right) \,\middle|\, s_i^{(0)} = s_i \right] \tag{4.8}$$

$$= \frac{1}{m} \sum_{i=1}^{m} V^{\pi}(s_i) \tag{4.9}$$

If π^* is an optimal policy for the underlying MDP, then $V^{\pi^*}(s_i) = V^*(s_i)$ for all $s_i \in \mathcal{S}$. Substituting this into the batched value expression gives:

$$\widehat{V}^{\boldsymbol{\pi}^*}(\mathbf{s}) = \frac{1}{m} \sum_{i=1}^{m} V^{*}(s_i) \tag{4.10}$$

Thus, the batched value function under the optimal batched policy $\boldsymbol{\pi}^*$ is equivalent to the average of the optimal values of the underlying MDP. Therefore, optimality is preserved under the batched extension in expectation. □

4.1.2 Batched Optimal Policy

This section details how we select optimal independent actions based on the batched value function derived in equation (4.10). Using the definition of the state-action value (i.e., the Q-value) from the underlying MDP, we get:

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a) \tag{4.11}$$

Using the results from theorem 1 and applying the definition of the value function from equation (3.10) (i.e., applying the Bellman operator), we get the following:

$$\begin{aligned}
\widehat{V}^*(\mathbf{s}) &= \frac{1}{m} \sum_{i=1}^{m} V^*(s_i) && (4.12)\\
&= \frac{1}{m} \sum_{i=1}^{m} \max_{a \in \mathcal{A}} Q^*(s_i, a) && (4.13)\\
&= \frac{1}{m} \sum_{i=1}^{m} \max_{a \in \mathcal{A}} \left(R(s_i, a) + \gamma \sum_{s'} T(s' \mid s_i, a) V^*(s') \right) && (4.14)\\
&= \frac{1}{m} \sum_{i=1}^{m} \max_{a \in \mathcal{A}} \left(R(s_i, a) + \gamma \sum_{s'} T(s' \mid s_i, a) \max_{a' \in \mathcal{A}} Q^*(s', a') \right) && (4.15)
\end{aligned}$$

Similarly, we can derive the optimal batched state-action Q-value function:

$$\begin{aligned}
\widehat{Q}^*(\mathbf{s}, a) &= \frac{1}{m} \sum_{i=1}^{m} Q^*(s_i, a) && (4.16)\\
&= \frac{1}{m} \sum_{i=1}^{m} \left(R(s_i, a) + \gamma \sum_{s'} T(s' \mid s_i, a) \max_{a' \in \mathcal{A}} Q^*(s', a') \right) && (4.17)
\end{aligned}$$

We define the optimal batched policy under independent actions as the vector of per-state optimal decisions which captures uncertainty in the per-state dynamics:

$$\boldsymbol{\pi}^*(\mathbf{s}) = \left\{ \arg\max_{a \in \mathcal{A}} Q^*(s_i, a) \mid s_i \in \mathbf{s} \right\} \tag{4.18}$$

This policy defines a batch of locally optimal actions that achieve the maximum value for each individual s_i in the batched planning model.

While $\boldsymbol{\pi}^*(\mathbf{s})$ is not a valid execution policy in environments where only one action may be taken, it serves as a useful intermediate representation for analyzing the structure of the

batched value function and for planning under independent state samples. The next section discusses how the batched planning model is used in the true single-state environment.

4.2 Batched Planning from a Single State

Although the true environment is modeled as a standard MDP $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, in which the agent is only in a single state $s \in \mathcal{S}$ at each timestep and must select a single action $a \in \mathcal{A}$, we formulated the batched planning framework in section 4.1 to enable robust and efficient evaluation of potential future outcomes. This section describes how we use the batched state representation for planning while satisfying the requirement that the agent selects only one action to execute in the true environment.

4.2.1 Constructing the Batched Planning Model

To construct a batched planning representation, we replicate the current state s a total of m times to form a batched state:

$$\mathbf{s} = \{s, s, \ldots, s\} \in \mathcal{S}^m \tag{4.19}$$

This batched state is used internally for planning purposes, such as simulating forward transitions under stochastic dynamics, evaluating expected returns, or propagating value estimates in parallel.

Transitions and rewards in the batched model follow the independent structure introduced in our formulation of the batched MDP (section 4.1). That is, for a given action $a \in \mathcal{A}$ and duplicated to get $\mathbf{a} = \{a, a, \ldots, a\} \in \mathcal{A}^m$, each element of the batch evolves independently as $s'_i \sim T(\cdot \mid s_i, a_i)$ for each $s_i = s \in \mathbf{s}$ and $a_i = a \in \mathbf{a}$. The rewards are averaged across the batch as $\widehat{R}(\mathbf{s}, \mathbf{a}) = \frac{1}{m} \sum_{i=1}^{m} R(s_i, a_i) = R(s, a)$ since all s_i and a_i are identical.

4.2.2 Selecting a Single Action

Although planning is performed over the batched state $\mathbf{s}$ and subsequent batched future states $\mathbf{s}'$, the final policy must ultimately select a single action to execute in the true environment. We therefore use the batched optimal Q-value function to express the single state-action Q-value function as:

$$\widehat{Q}^*(\mathbf{s}, a) = \frac{1}{m} \sum_{i=1}^{m} Q^*(s_i, a) = Q^*(s, a) \quad \text{for} \quad \mathbf{s} = \{s\}_{i=1}^{m} \tag{4.20}$$

with $s_i = s$ for all i. This is only true in the degenerate case where all batch elements are identical. The expansion of $\widehat{Q}^*$ is useful for the general batched planning setting defined in section 4.1, but redundant when $\mathbf{s} = \{s\}_{i=1}^m$. In the degenerate case, the resulting policy is then given by:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \widehat{Q}^*(\mathbf{s}, a) \quad \text{for} \quad \mathbf{s} = \{s\}_{i=1}^m \tag{4.21}$$

where the recursive definition of $\widehat{Q}^*$ in equation (4.16) expands each $s \in \mathbf{s}$ based on the transition dynamics. This policy selects the action that maximizes the expected value across stochastic state transitions from the single known current state. Importantly, this formulation allows us to use the computational benefits of vectorized or parallelized transitions while maintaining full compatibility with a standard single-agent MDP interface.

4.2.3 Interpretation

The batched formulation can be interpreted as a Monte Carlo approximation of the expected return under stochastic transitions. Specifically, given a single state s and candidate action a, we simulate m stochastic next states $s_i \sim T(\cdot \mid s, a)$ and compute:

$$\widehat{Q}^*(\mathbf{s}, a) \approx R(s, a) + \gamma \frac{1}{m} \sum_{i=1}^{m} V^*(s_i') \tag{4.22}$$

which approximates the Bellman backup:

$$Q^*(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim T(\cdot|s,a)} \left[V^*(s') \right] \tag{4.23}$$

for $s \in \mathbf{s}$. By evaluating this expectation over m simulated next states, the policy can make informed action choices that account for environmental uncertainty without explicitly modeling a belief state. This formulation also supports integration with planning algorithms such as Monte Carlo tree search (MCTS) [95, 96], rollout-based policy evaluation [97], or batched value iteration [98]. In the next section, we will detail how the batched planning model can be extended to belief-state planning settings.

4.3 Batched Belief-State Markov Decision Process

Given a partially observable Markov decision process (POMDP) defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$, detailed in section 3.2, we can convert the POMDP to an equivalent belief-state MDP (see section 3.2.1 and figure 4.1). Using the batched planning model we derived in section 4.1, we can define the *batched belief-state MDP* (**BB**-MDP) framework by treating a batch of beliefs $\mathbf{b} = \{b_i\}_{i=1}^m$ as states (shown in figure 4.2). The **BB**-MDP tuple is defined as $\langle \mathcal{B}^m, \mathcal{A}, \mathbf{T}_b, \widehat{R}_b, \gamma \rangle$, where the batched belief-state transition function $\mathbf{b}' \sim \mathbf{T}_b(\cdot \mid \mathbf{b}, a)$ is $\mathbf{T}_b : \mathcal{B}^m \times \mathcal{A} \to \mathcal{B}^m$, internally computing the following four steps:

$$\mathbf{s} = \{s_i\}_{i=1}^m \quad \text{where} \quad s_i \sim b_i \quad \forall b_i \in \mathbf{b} \tag{4.24}$$

$$\mathbf{s}' = \{s'_i\}_{i=1}^m \quad \text{where} \quad s'_i \sim T(\cdot \mid s_i, a) \quad \forall s_i \in \mathbf{s} \tag{4.25}$$

$$\mathbf{o} = \{o_i\}_{i=1}^m \quad \text{where} \quad o_i \sim O(\cdot \mid a, s'_i) \quad \forall s'_i \in \mathbf{s}' \tag{4.26}$$

$$\mathbf{b}' = \{b'_i\}_{i=1}^m \quad \text{where} \quad b'_i \leftarrow \textsc{Update}(b_i, a, o_i) \quad \forall i \in \{1, \ldots, m\} \tag{4.27}$$

where $b \in \mathcal{B}$ is a belief state belonging to the belief simplex $\mathcal{B}$ over the state space $\mathcal{S}$, and the batched belief state space $\mathcal{B}^m$ is the multiset of m beliefs:

$$\mathcal{B}^m = \big\{ \{b_1, b_2, \ldots, b_m\} \mid b_i \in \mathcal{B} \big\} \tag{4.28}$$

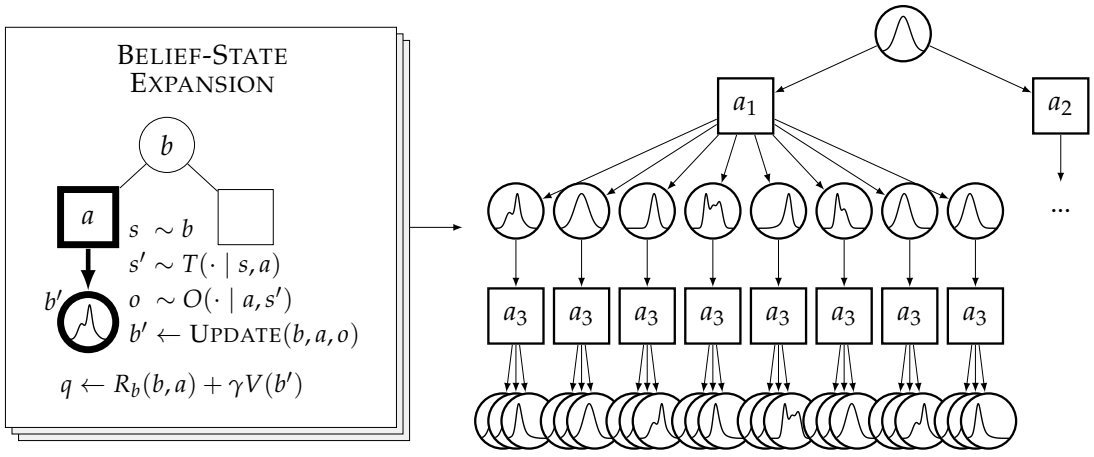


Figure 4.1: Belief-state MDP for two-step expansion.

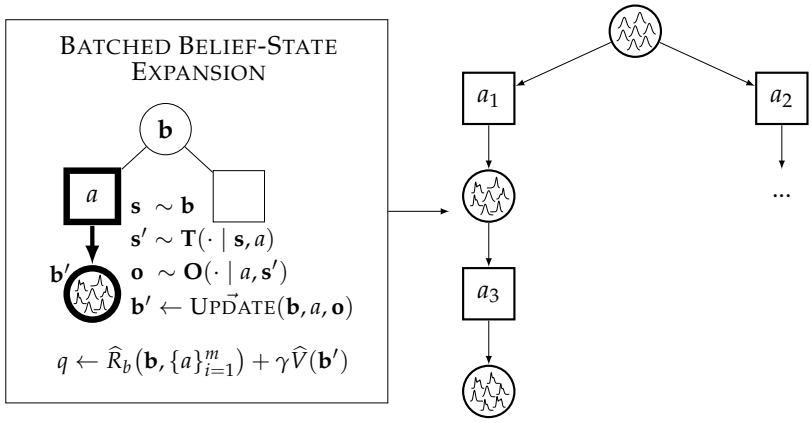


Figure 4.2: Batched belief-state MDP for two-step expansion.

with $\mathbf{b} \in \mathcal{B}^m$ denoting a batched belief-state. If the state sampling procedure, the transition function, observation function, and belief update can be done in parallel over batches (e.g., vectorized), this can be simplified to:

$$\mathbf{s} \sim \mathbf{b} \qquad \mathbf{s}' \sim \mathbf{T}(\cdot \mid \mathbf{s}, a) \qquad \mathbf{o} \sim \mathbf{O}(\cdot \mid a, \mathbf{s}') \qquad \mathbf{b}' = \vec{\textsc{Update}}(\mathbf{b}, a, \mathbf{o}) \tag{4.29}$$

for the vectorized transition function $\mathbf{T} : \mathcal{S}^m \times \mathcal{A} \to \mathcal{S}^m$, the vectorized observation function $\mathbf{O} : \mathcal{A} \times \mathcal{S}^m \to \mathcal{O}^m$, and the vectorized belief update function $\vec{\textsc{Update}} : \mathcal{B}^m \times \mathcal{A} \times \mathcal{O}^m \to \mathcal{B}^m$. The batched belief reward function is derived from equation (4.3):

$$\widehat{R}_b(\mathbf{b}, \mathbf{a}) = \mathop{\mathbb{E}}_{b \in \mathbf{b},\, a \in \mathbf{a}} \left[R_b(b_i, a_i) \right] = \frac{1}{m} \sum_{i=1}^{m} \left(\int b_i(s) R(s, a_i) \, \mathrm{d}s \right) \tag{4.30}$$

where R_b is the belief-state reward function defined in equation (3.3).

4.3.1 Batched Belief-State Value Function

To act optimally under partial observability, an agent must evaluate the expected return of a policy π given its current belief $b \in \mathcal{B}$. Kaelbling *et al.* [45] define the belief-state value function as the expectation of state values weighted by the likelihood under the belief:

$$V^\pi(b) = \int b(s) V^\pi(s) \, \mathrm{d}s \tag{4.31}$$

Extending to the batched belief-state model and using the decomposition in theorem 1, we define the batched belief-state value function as the average value across a batch of beliefs:

$$\widehat{V}^{\boldsymbol{\pi}}(\mathbf{b}) = \frac{1}{m}\sum_{i=1}^{m}\left(\int b_i(s)V^{\pi}(s)\,\mathrm{d}s\right) \tag{4.32}$$

This value reflects the expected return from executing the policy π independently from each belief in the batch, and serves as the basis for value-based planning under state uncertainty. We can similarly show that when π is the optimal policy π^*, then $\widehat{V}^{\pi} = \widehat{V}^{\pi^*}$ (following the same argument as theorem 1).

In the underlying POMDP, the optimal belief-state Q-value function is similarly defined as the expected Q-value under the belief [45]:

$$Q^*(b,a) = \int b(s)Q^*(s,a)\,\mathrm{d}s \tag{4.33}$$

Extending to the batched planning model, we define the batched belief-state Q-value function as the average over the batch of beliefs:

$$\widehat{Q}^*(\mathbf{b},a) = \frac{1}{m}\sum_{i=1}^{m}Q^*(b_i,a) = \frac{1}{m}\sum_{i=1}^{m}\left(\int b_i(s)Q^*(s,a)\,\mathrm{d}s\right) \tag{4.34}$$

This function evaluates the expected return of executing action a across all belief points in the batch $\mathbf{b}$. The corresponding optimal batched belief-state policy vector is comprised of the actions that maximize the optimal Q-value:

$$\boldsymbol{\pi}^*(\mathbf{b}) = \left\{\arg\max_{a\in\mathcal{A}} Q^*(b_i,a) \mid b_i \in \mathbf{b}\right\} \tag{4.35}$$

Action selection for a single belief. In the outer POMDP loop, when we have a single belief b, following section 4.2, we set $\mathbf{b} = \{b\}_{i=1}^{m}$ as copies of the current belief b to allow the batched planning model to expand in parallel from the current belief. The action selected at the outer POMDP loop then becomes:

$$\pi^*(b) = \arg\max_{a\in\mathcal{A}} \widehat{Q}^*(\mathbf{b},a) \quad \text{for} \quad \mathbf{b} = \{b\}_{i=1}^{m} \tag{4.36}$$

This policy selects a single action that performs well in expectation over the batch of belief states (and their future expansion), enabling robust decision making under state uncertainty.

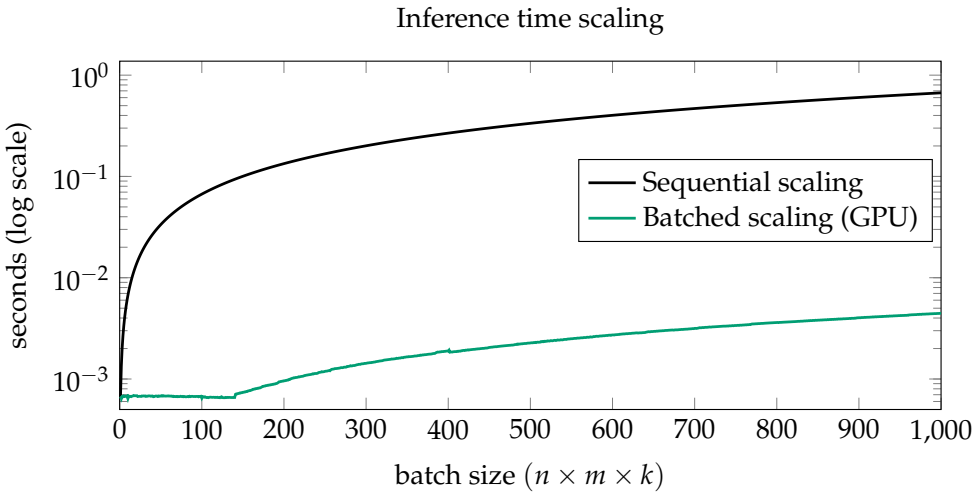


Figure 4.3: Inference time scaling in **BB**-MDPs.

4.3.2 Preliminary Results

Experimental analysis is provided in the next chapter, yet in figure 4.3 we empirically show how sequential scaling compares to batched scaling on an 80GB NVIDIA A100 GPU. The figure shows an aggregate batch for $n = |\mathcal{A}|$ actions, $m = |\mathbf{b}|$ beliefs per action, and $k = |b|$ states per belief, where we represent each belief b as a collection of state particles $b = \{s_i\}_{i=1}^{k}$.

4.4 Discussion

In this chapter, we formalized a batched extension to Markov decision processes and belief-state MDPs. Requirements on a parallelized state transition function and reward function are necessary to adapt standard MDPs to the batched setting. For POMDPs, in addition to the requirements stated for MDPs, we also require parallelized observation functions and belief updaters. Future work could focus on batched equivalents for offline planning algorithms such as value iteration (section 3.3.1) and online planning algorithms such as Monte Carlo tree search (section 3.3.2). In the next chapter, we will review methods that can be used as parallel belief updaters and introduce a generative surrogate model that approximates the posterior belief through efficient sampling.

Chapter 5

Inversion Surrogates for Belief Updating

If you want to be a good intuitive Bayesian—if you want to naturally make good predictions, without having to think about what kind of prediction rule is appropriate—you need to protect your priors.

Brian Christian

In this chapter, we address parallelized belief updating in high-dimensional observation spaces. Specifically, we introduce a method to sample from the posterior state distribution conditioned on a set of partial observations for information gathering POMDPs. We consider the case of the *purely epistemic Markov decision process* (EMDP) [99], which is a special case of a POMDP where we do not have state transition dynamics, i.e., actions do not affect the (static) state. This type of problem can be seen as an information gathering problem where uncertainty in the hidden state needs to be reduced to make some final decision. Examples of such problems include preference elicitation [99] and critical mineral exploration [8]. In this chapter, we apply our surrogate belief updater to a geological planning problem that inverts muon tomography data [100] to a distribution of subsurface intrusion states.

5.1 Problem Formulation

Using a prior dataset $\left(s^{(i)}, o_{1:t}^{(i)}\right) \in \mathcal{D}_{\text{partial}}$ over the states $s \in \mathcal{S}$ and observations $o_{1:t} \in \mathcal{O}$, we introduce a surrogate model that approximates the posterior $p(s \mid o_{1:t})$ given a set of observations $o_{1:t}$ over time. Samples from this posterior act as samples from the updated

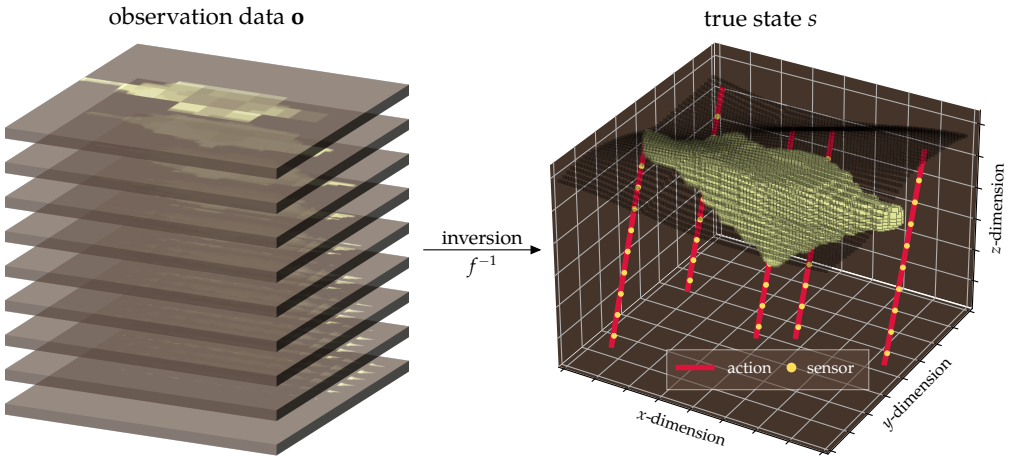


Figure 5.1: Example 3D inversion using muon tomography data (5 actions, 9 sensors each).

belief. Because we are addressing EMDP problems, we can simplify the belief update in standard POMDPs. This simplification results in an *inversion* problem [101]. Inversion is the process of going from a set of (partial) observations $o_{1:t}$ to the true state (or, ideally, a distribution of states in the probabilistic case) as shown in figure 5.1:

$$s = f^{-1}(o_{1:t}, a_{1:t}) \quad \text{or} \quad s \sim f^{-1}(o_{1:t}, a_{1:t})$$

The name stems from the inverse of the *forward* observation model $o_{1:t} = f(s, a_{1:t})$ or $O(s, a_{1:t})$ in POMDP notation (noting that in an EMDP, the static state does not transition based on actions, hence $s' = s$). To define our belief, we take m samples from the posterior $p(s \mid \mathbf{o})$ to get a finite set of posterior states $b = \{s_1, s_2, \ldots, s_m\}$ where $s_i \sim p(\cdot \mid \mathbf{o})$.* Similar to particle filtering [52], our belief is defined by a set of state particles b, yet we do not use the observation likelihood model. Instead, we only require observations generated from states for training (i.e., the forward model), without needing to evaluate their likelihood.

To develop a likelihood-free posterior sampling method for POMDP planning, we extend the *conditional variational autoencoder* (CVAE) model [102] and fine-tune to maximize mutual information across observations from the same state—thereby, allowing latent observation trajectories to be more consistent across time. As with the CVAE, we train using states and partial observations but only require partial observations during inference for conditioning.

*For brevity, we opt for the shorthand $\mathbf{o}$ instead of $o_{1:t}$ (indicating a set of observations up to time t).

5.1.1 Related Work

This work can be viewed as learning a generative model that samples states that are reconstructed from partial observations. For our work, we consider the states and observations to be images (partial images for observations); therefore, generative modeling research from the computer vision community can be directly applicable.

Probabilistic generative models. A variety of deep generative models have been proposed to learn complex data distributions and enable conditional sampling. Variational autoencoders (VAEs) [103, 104] frame the density estimation problem as approximate Bayesian inference and train an encoder-decoder architecture to maximize a variational lower bound on the data likelihood $p_\theta(s)$. Conditional VAEs (CVAEs) extend this framework by incorporating conditional information (e.g., labels or partial observations) into both the encoder and decoder, enabling conditional generation via $p_\theta(s \mid \mathbf{o})$ [102]. We detail the VAE and CVAE generative model architectures in the following section. Other approaches, such as generative adversarial networks (GANs) [105], formulate generation as a two-player minimax game between a generator and a discriminator, achieving high-fidelity samples but often suffering from instability and mode collapse [106]. Mirza *et al.* [107] introduced conditional GANs (CGANs), which extend traditional GANs by guiding the generator-discriminator framework with conditioning variables. Beyond VAEs and GANs, normalizing flows [108] provide exact likelihood evaluation through a sequence of invertible transformations, while more recent score-based and diffusion models [109, 110] demonstrate state-of-the-art sample quality by learning to reverse a data corruption process. In the context of online planning, diffusion models may be prohibitively expensive, as a single state sample typically requires on the order of 10^3 sequential neural network forward passes [109, 111, 112]. Our work extends these deep conditional generative models specifically for inversion problems, where we ensure both reconstruction consistency and guide fine-tuning using mutual information to generate state samples given partial observations. For a more detailed overview of different classes of generative models, see Cao *et al.* [113].

Deterministic inversion methods. In geological research, inversion problems [101] can be viewed as computing a function that reconstructs some hidden state of the world given some (noisy) observation, generally geophysical measurements such as gravity data [114], active seismic imaging [115], or density data from passive muon tomography [36]. Deep

learning models have been applied to geological inversion problems primarily through the use of deterministic convolutional neural networks (CNNs) [116, 117]. In these settings, a neural network is trained to perform regression $s = f_\theta(\mathbf{o}_{1:T})$ given full observations. In our work, we consider the case where the observations are only partially observed (e.g., partial borehole data or partial geophysical data), yet a full state is sampled from the model. Other approaches, such as linear and nonlinear least-squares methods as well as regularization techniques, have been explored in the deterministic inversion literature [118, 119, 120, 121]. Principal component analysis (PCA) and kernel PCA have also been used as dimensionality reduction techniques for deterministic history matching [122, 123]. Tarantola *et al.* [124] and, more recently, Zhdanov [125] provide comprehensive reviews of existing deterministic inversion methods applied to geological problems.

Probabilistic inversion methods. To quantify the uncertainty in the hidden state, probabilistic inversion methods—similar to probabilistic generative models—have been extensively studied. McAliley *et al.* [126] use a straightforward implementation of a conditional VAE to invert geophysical data and sample from a unit Gaussian $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ during inference, as is standard in most CVAE applications. Their approach omits any observation encoder as the observations are vectors of synthetic gravity. In our work, we are interested in high-dimensional observations where a separate observation encoder is useful for dimensionality reduction and we learn latent Gaussian parameters $\mu_\mathbf{o}$ and $\log \sigma^2_\mathbf{o}$ directly. Chung *et al.* [127] proposed a paired autoencoder approach but rely on an initial guess of the true state during inference and learn an observation decoder alongside a state decoder, which may not be necessary for strictly observation-to-state intrusion, as we show in our experiments. Other sampling techniques have also been studied such as Markov chain Monte Carlo (MCMC) [128, 129, 130]. Laloy *et al.* [131] condition based on expensive MCMC steps and suggest long runs to generate larger Markov chains for full exploration of the posterior, which may be expensive for planning. In a recent survey on neural network-based geological inversion, Li *et al.* [132] identify a need for efficient probabilistic inversion methods.

5.2 Background

This section provides background on probabilistic generative models that serve as the basis for our work. Specifically, we review the variational autoencoder and its conditional variant.

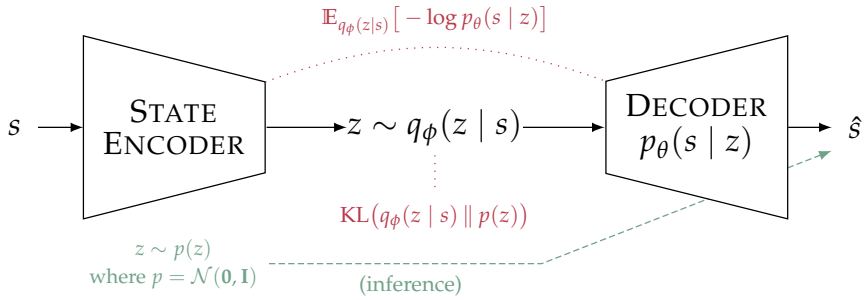


Figure 5.2: The variational autoencoder (VAE) architecture and loss.

5.2.1 Variational Autoencoder

Kingma *et al.* [103] introduced the *variational autoencoder* (VAE), a probabilistic model used for approximate inference with continuous latent variables. The VAE is designed to approximate the generative model $p(s)$ by assuming the variables s are sampled following some hidden process. This process consists of first sampling a hidden random variable $z \sim p_\theta(z)$, then generating a state sample using the latent z as a condition $s \sim p_\theta(s \mid z)$. This assumes both the prior $p_\theta(z)$ and likelihood $p_\theta(s \mid z)$ come from some parametric family of distributions [103]. To approximate the intractable true posterior density of $p_\theta(z \mid s) = p_\theta(s \mid z)p_\theta(z)/p_\theta(s)$, a probabilistic *encoder* model $q_\phi(z \mid s)$ is used (also called the *recognition model*). Similarly, $p_\theta(s \mid z)$ is referred to as the probabilistic *decoder* (or the *generation model*).

The objective when training a VAE is to maximize the marginal likelihood $p(s^{(i)})$. Kingma *et al.* [103] show that the objective can be approximated by the *variational lower bound* (ELBO):

$$\log p_\theta(s^{(i)}) \geq \mathbb{E}_{q_\phi(z|s)}\big[-\log q_\phi(z \mid s) + \log p_\theta(s, z)\big] \tag{5.1}$$

where θ and ϕ are the parameters of the generation model and recognition model, respectively. Because the objective is to maximize the marginal log-likelihood, we can minimize the following loss using standard gradient-based optimization methods [76]:

$$\mathcal{L}_{\text{VAE}}(s; \phi, \theta) = \mathbb{E}_{q_\phi(z|s)}\big[-\log p_\theta(s \mid z)\big] + \mathrm{KL}\big(q_\phi(z \mid s) \,\|\, p(z)\big) \tag{5.2}$$

Figure 5.2 visualizes the probabilistic encoder-decoder model of the variational autoencoder.

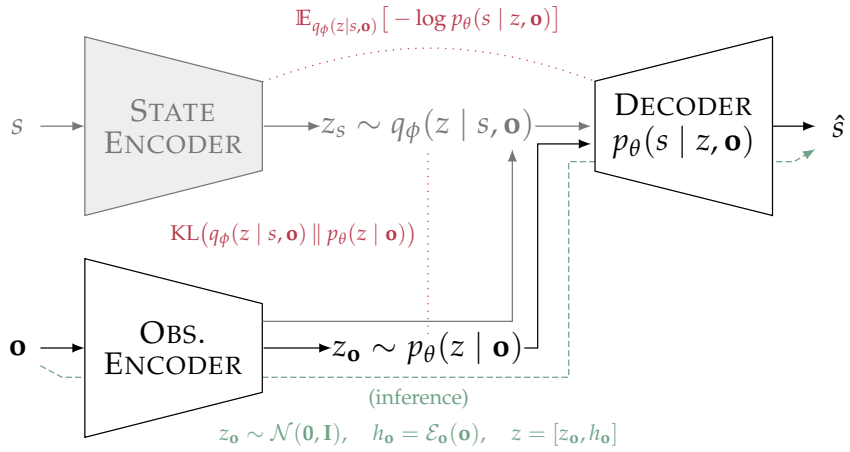


Figure 5.3: The conditional variational autoencoder (CVAE) architecture and loss.

5.2.2 Conditional Variational Autoencoder

As a natural extension to VAEs, Sohn *et al.* [102] introduced the *conditional variational autoencoder* (CVAE), illustrating the model architecture in figure 5.3. The CVAE is comprised of a recognition (state encoder) model $q_\phi(z \mid s, \mathbf{o})$, the conditional prior (observation encoder) model $p_\theta(z \mid \mathbf{o})$, and the generation (decoder) model $p_\theta(s \mid z, \mathbf{o})$. The objective of the CVAE model is to maximize $p(s \mid \mathbf{o})$, i.e., the conditional likelihood of the data s given a condition $\mathbf{o}$ (a partial observation in our case). Sohn *et al.* [102] derive a conditional extension of the variational lower bound as:

$$\begin{aligned} \log p_\theta(s \mid \mathbf{o}) &\geq \mathbb{E}_{q_\phi(z|s,\mathbf{o})}\big[-\log q_\phi(z \mid s, \mathbf{o}) + \log p_\theta(s, z \mid \mathbf{o})\big] \\ &= \mathbb{E}_{q_\phi(z|s,\mathbf{o})}\big[-\log q_\phi(z \mid s, \mathbf{o}) + \log p_\theta(z \mid \mathbf{o})\big] + \mathbb{E}_{q_\phi(z|s,\mathbf{o})}\big[\log p_\theta(s \mid z, \mathbf{o})\big] \end{aligned} \tag{5.3}$$

Minimizing the following loss approximates maximizing the conditional log-likelihood:

$$\mathcal{L}_{\text{CVAE}}(s, \mathbf{o}; \phi, \theta) = \mathbb{E}_{q_\phi(z|s,\mathbf{o})}\big[-\log p_\theta(s \mid z, \mathbf{o})\big] + \mathrm{KL}\big(q_\phi(z \mid s, \mathbf{o}) \,\|\, p_\theta(z \mid \mathbf{o})\big) \tag{5.4}$$

Sohn *et al.* [102] input both the observation $\mathbf{o}$ and an initial guess of the state $\hat{s}$ from a separate CNN for better structured output predictions [133, 134]. In our experiments, we omit this step to focus on the core model differences, yet this recurrent connection could easily be implemented for more refined generative predictions.

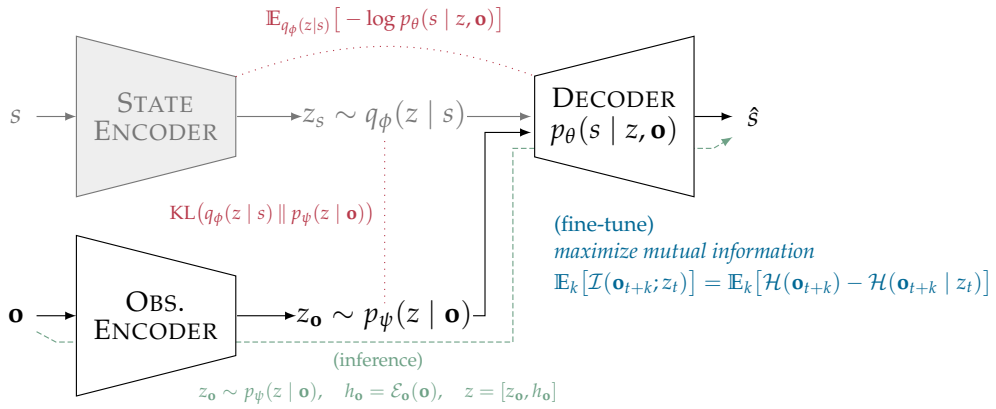


Figure 5.4: The inversion variational autoencoder ($\mathcal{I}$-VAE) architecture and loss.

5.3 Inversion Variational Autoencoder

Using conditional generative models in sequential decision making problems can act as a way to sample (hidden) states conditioning on the set of observations up to time t. Knowing that the generative model will be used during sequential planning means we will have sets of observations from different time steps that each correspond to some true state. Therefore, we extend the CVAE model in three ways: (1) simplify the network design so that the recognition model depends only on the state, (2) directly learn the parameters ψ of the conditional prior model $p_\psi(z \mid \mathbf{o})$ instead of a unit Gaussian [102, 135, 136], and (3) fine-tune the model to maximize mutual information [137] across observation trajectories, ensuring trajectories from the same state are closer together in the latent space and trajectories from differing states are further apart. In the EMDP setting, our belief update results in an inversion problem. Therefore, we propose the *inversion variational autoencoder* ($\mathcal{I}$-VAE) to address these challenges, illustrated in figure 5.4.

The following $\mathcal{I}$-VAE loss function differs from the CVAE loss simply in the proposal q_ϕ. The CVAE proposal conditions on both the state s and observation $\mathbf{o}$, namely $q_\phi(z \mid s, \mathbf{o})$, while the $\mathcal{I}$-VAE proposal removes the dependence on the observation, namely $q_\phi(z \mid s)$. The difference lies in applying conditional independence $z \perp \mathbf{o} \mid s$ which implies that once we know the state s, the observation $\mathbf{o}$ provides no additional information. This simplifies the network architecture without compromising performance (see analysis in section 5.4).

5.3.1 $\mathcal{I}$-VAE Pretraining Objective

The following pretraining loss function is derived in a similar manner to the CVAE loss, using the conditional variational lower bound (ELBO). To derive the $\mathcal{I}$-VAE loss function, we begin with the objective to approximate the following conditional distribution:

$$p(s \mid \mathbf{o}) = \int p(s, z \mid \mathbf{o}) \, \mathrm{d}z \tag{5.5}$$

Similar to the variational lower bound derivation for CVAEs [102], we get the following:

$$\begin{aligned}
\log p(s \mid \mathbf{o}) &= \log \int p(s, z \mid \mathbf{o}) \, \mathrm{d}z && (5.6)\\
&= \log \int p(s \mid z, \mathbf{o}) p(z \mid \mathbf{o}) \, \mathrm{d}z && (5.7)\\
&= \log \int q_\phi(z \mid s) \frac{p(s \mid z, \mathbf{o}) p(z \mid \mathbf{o})}{q_\phi(z \mid s)} \, \mathrm{d}z && (5.8)\\
&\approx \log \int q_\phi(z \mid s) \frac{p_\theta(s \mid z, \mathbf{o}) p_\psi(z \mid \mathbf{o})}{q_\phi(z \mid s)} \, \mathrm{d}z && (5.9)\\
&\geq \int q_\phi(z \mid s) \log \left(\frac{p_\theta(s \mid z, \mathbf{o}) p_\psi(z \mid \mathbf{o})}{q_\phi(z \mid s)} \right) \mathrm{d}z && (5.10)\\
&= \mathbb{E}_{q_\phi(z|s)} \left[\log \left(p_\theta(s \mid z, \mathbf{o}) \frac{p_\psi(z \mid \mathbf{o})}{q_\phi(z \mid s)} \right) \right] && (5.11)\\
&= \mathbb{E}_{q_\phi(z|s)} \left[\log p_\theta(s \mid z, \mathbf{o}) \right] + \mathbb{E}_{q_\phi(z|s)} \left[\log \left(\frac{p_\psi(z \mid \mathbf{o})}{q_\phi(z \mid s)} \right) \right] && (5.12)\\
&= \mathbb{E}_{q_\phi(z|s)} [\log p_\theta(s \mid z, \mathbf{o})] - \mathrm{KL} \left(q_\phi(z \mid s) \parallel p_\psi(z \mid \mathbf{o}) \right) && (5.13)
\end{aligned}$$

where equation (5.7) applies the chain rule, equation (5.8) introduces the proposal distribution $q_\phi(z \mid s)/q_\phi(z \mid s)$, equation (5.9) introduces the learnable approximations $p_\theta(s \mid z, \mathbf{o}) \approx p(s \mid z, \mathbf{o})$ and $p_\psi(z \mid \mathbf{o}) \approx p(z \mid \mathbf{o})$, equation (5.10) applies Jensen's inequality, equations (5.11) and (5.12) apply expectation and logarithmic rules, and finally equation (5.13) applies the definition of KL-divergence.

To train surrogates to approximate this distribution, we want to minimize the negative log-likelihood to get the $\mathcal{I}$-VAE (pretraining) loss function:

$$\mathcal{L}_{\mathcal{I}\text{-VAE}}(s, \mathbf{o}; \phi, \theta, \psi) = \underbrace{\mathbb{E}_{q_\phi(z|s)} \big[-\log p_\theta(s \mid z, \mathbf{o}) \big]}_{\text{Reconstruction loss}} + \underbrace{\mathrm{KL}\big(q_\phi(z \mid s) \parallel p_\psi(z \mid \mathbf{o}) \big)}_{\substack{\text{KL-divergence} \\ \text{(match latent distributions)}}} \tag{5.14}$$

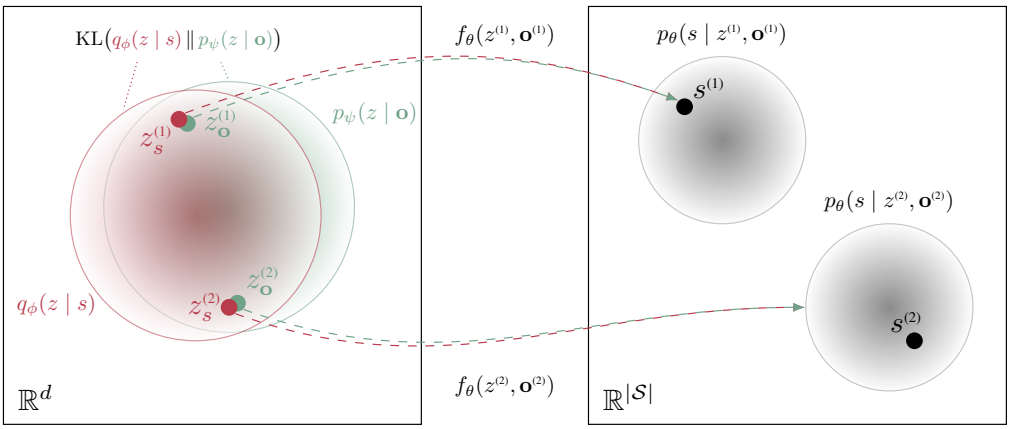


Figure 5.5: Latent space matching of z_s and $z_{\mathbf{o}}$ and mapping to $p_\theta(s \mid z, \mathbf{o})$ for the $\mathcal{I}$-VAE.

Intuitively, the $\mathcal{I}$-VAE loss function attempts to reconstruct the state s from the latent variable z and match the latent conditional distributions $q_\phi(z \mid s)$ and $p_\psi(z \mid \mathbf{o})$ by minimizing their KL-divergence. Figure 5.5 illustrates this distribution matching. The KL-divergence term acts both as a regularizer[†] [103] and as a way to learn a latent distribution that relies solely on the observations $\mathbf{o}$ during runtime inference. By penalizing their divergence, this forces both distributions to share some consistent latent structure, thus, forcing the latent distribution with partial information $\mathbf{o}$ to be close to the latent distribution with full information s. The CVAE model also only requires the observation during inference, but traditionally [102, 135, 136] samples $z_{\mathbf{o}}$ from a unit Gaussian distribution and concatenates the encoded observation $z_{\mathbf{o}}$ to get $z = [z_{\mathbf{o}}, h_{\mathbf{o}}]$, where $h_{\mathbf{o}} = \mathcal{E}_{\mathbf{o}}(\mathbf{o})$ is the encoded observation and z is passed to the decoder. The $\mathcal{I}$-VAE model explicitly learns an approximate conditional model $p_\psi(z \mid \mathbf{o})$, which uses the observation directly to shape where in the latent space to sample from. This results in matching the inference-time behavior with the training-time behavior.

Our work is related to the *variational autoencoder with arbitrary conditioning* (VAEAC) [138], but their model assumes the observed data is simply a mask of the hidden (full) data. This is true for pixel observations from images (e.g., MNIST [16], shown in figure 5.6), but breaks down in cases where the observations are over a different domain (e.g., muon tomography observations). Our model does not make this assumption about the observations, and we show that it can work in both cases in the analysis section 5.4.

[†]The KL-divergence stops $q_\phi(z \mid s)$ from mode-collapsing or becoming arbitrarily tight around each training sample s, and it prevents $p_\psi(z \mid \mathbf{o})$ from ignoring the observation $\mathbf{o}$ or becoming too broad.

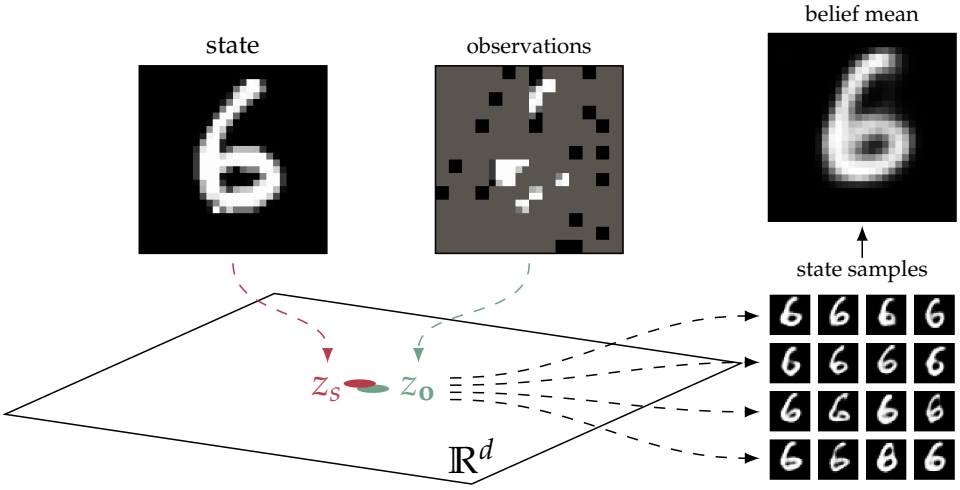


Figure 5.6: Latent space mapping of $z_{o_{1:t}}$ $(t = 30)$ to $m = 100$ state samples from $p_\theta(s \mid z, \mathbf{o})$.

Latent space mapping and runtime inference. Illustrated in figure 5.6, the goal during pretraining is to map both the latent-state variable z_s and latent-observation variable $z_{\mathbf{o}}$ to a lower-dimensional representation in $\mathbb{R}^d$. Using the KL-divergence from equation (5.14), we expect these latent variables to map to a similar area in the lower d-dimensional space. During runtime inference, the reduced-dimension latent $z_{\mathbf{o}}$ is used to sample states from $p_\theta(s \mid z, \mathbf{o})$ consistent with the observations. Figure 5.6 shows an example using masked 4×4 pixel observations for the MNIST dataset, where at time $t = 30$ the total pixel coverage given by the observations is about 15%. A set of state samples $\mathbf{s}$ can be directly used to represent samples from an updated belief in EMDP planning, and we show an example of what the mean belief may look like when sampling 100 states efficiently in a single batched forward pass of the $\mathcal{I}$-VAE model.

KL-divergence derivation. In practice, the latent distributions over z are modeled as multivariate Gaussian distributions. Given the probability density function of a multivariate Gaussian with diagonal covariance:

$$\mathcal{N}\left(z; \boldsymbol{\mu}, \operatorname{diag}\left(\boldsymbol{\sigma}^2\right)\right) = \prod_{i=1}^{|z|} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(z_i - \mu_i)^2}{2\sigma_i^2}\right) \tag{5.15}$$

we analytically derive the KL-divergence between the latent state distribution $q_\phi(z \mid s)$ and the latent observation distribution $p_\psi(z \mid \mathbf{o})$ (where $(\boldsymbol{\mu}_s, \boldsymbol{\sigma}_s)$ and $(\boldsymbol{\mu}_\mathbf{o}, \boldsymbol{\sigma}_\mathbf{o})$ are the parameters of their respective distributions) as:

$$\begin{aligned}
&\mathrm{KL}\left(q_\phi(z \mid s) \,\|\, p_\psi(z \mid \mathbf{o})\right) && (5.16)\\
&\quad= \int q_\phi(z \mid s) \log\left(\frac{q_\phi(z \mid s)}{p_\psi(z \mid \mathbf{o})}\right) \mathrm{dz} && (5.17)\\
&\quad= \int q_\phi(z \mid s) \log\left(\prod_{i=1}^{|z|} \frac{\left(2\pi\sigma_{s,i}^2\right)^{-1/2} \exp\left(-(z_i - \mu_{s,i})^2 / (2\sigma_{s,i}^2)\right)}{\left(2\pi\sigma_{\mathbf{o},i}^2\right)^{-1/2} \exp\left(-(z_i - \mu_{\mathbf{o},i})^2 / (2\sigma_{\mathbf{o},i}^2)\right)}\right) \mathrm{dz} && (5.18)\\
&\quad= \int q_\phi(z \mid s) \sum_{i=1}^{|z|} \left(\log\left(\frac{\sigma_{\mathbf{o},i}}{\sigma_{s,i}}\right) - \frac{(z_i - \mu_{\mathbf{o},i})^2}{2\sigma_{\mathbf{o},i}^2} + \frac{(z_i - \mu_{s,i})^2}{2\sigma_{s,i}^2}\right) \mathrm{dz} && (5.19)\\
&\quad= \sum_{i=1}^{|z|} \int q_\phi(z_i \mid s) \left(\log\left(\frac{\sigma_{\mathbf{o},i}}{\sigma_{s,i}}\right) - \frac{(z_i - \mu_{\mathbf{o},i})^2}{2\sigma_{\mathbf{o},i}^2} + \frac{(z_i - \mu_{s,i})^2}{2\sigma_{s,i}^2}\right) \mathrm{dz} && (5.20)\\
&\quad= \sum_{i=1}^{|z|} \left(\frac{1}{2}\Big(\log\sigma_{\mathbf{o},i}^2 - \log\sigma_{s,i}^2\Big) - \frac{1}{2} + \frac{1}{2\sigma_{\mathbf{o},i}^2}\Big(\sigma_{s,i}^2 + (\mu_{s,i} - \mu_{\mathbf{o},i})^2\Big)\right) && (5.21)\\
&\quad= \frac{1}{2}\sum_{i=1}^{|z|} \left(\log\sigma_{\mathbf{o},i}^2 - \log\sigma_{s,i}^2 + \frac{\sigma_{s,i}^2 + (\mu_{s,i} - \mu_{\mathbf{o},i})^2}{\sigma_{\mathbf{o},i}^2} - 1\right) && (5.22)
\end{aligned}$$

More complex latent distributions could instead be used and have been extensively studied in the literature, such as Gaussian mixture models [139, 140], the Gumbel-Softmax distribution for categorical latents [141], normalizing flows [108], inverse autoregressive flows [142], and diffusion models [143].

5.3.2 Fine-Tuning to Maximize Trajectory-Based Mutual Information

Not only are we interested in modeling $p(s \mid \mathbf{o})$ from equation (5.5), we are also interested in the setting where sequences of observations will be passed to the model over time. Thus, a secondary objective is to maximize mutual information between observation sequences (i.e., observation trajectories), to better align the latent space across observations correlated to the same state. Therefore, we fine-tune the pretrained $\mathcal{I}$-VAE model with a *trajectory contrastive loss* (TCL) [137, 144, 145]. The key idea is that once we have a performant generative model with trained encoders and decoder, we can further align the latent space given observation trajectories from a dataset $(s, \mathbf{o}_{1:T}) \in \mathcal{D}_{\text{traj}}$ of states and their observations over time.

The trajectory contrastive loss is a form of the InfoNCE loss [137], where we want to maximize mutual information $\mathcal{I}$ between a window of $k \in [1, \ldots, K]$ observations and the observation-inferred latent variables z_t. Namely, we want to maximize the expectation of $\mathcal{I}(\mathbf{o}_{t+k}; z_t)$ over the K-length window, or minimize the negation:

$$\begin{aligned}
\mathcal{L}_{\text{TCL}}(\mathbf{o}_{1:T}; \omega, K) &= -\mathbb{E}_k[\mathcal{I}(\mathbf{o}_{t+k}; z_t)] && (5.23)\\
&= -\mathbb{E}_k[\mathcal{H}(\mathbf{o}_{t+k}) - \mathcal{H}(\mathbf{o}_{t+k} \mid z_t)] \quad \text{for} \quad 1 \leq k \leq K && (5.24)\\
&= -\mathbb{E}_k\left[\mathbb{E}_{z_t}\left[\log \frac{p(\mathbf{o}_{t+k} \mid z_t)}{p(\mathbf{o}_{t+k})}\right]\right] && (5.25)\\
&\geq -\mathbb{E}_k\left[\mathbb{E}_{\tilde{z}}\left[\log \frac{f(\mathbf{o}^+, \tilde{z})}{\sum_{\mathbf{o} \in \Omega^-} f(\mathbf{o}, \tilde{z})}\right]\right] && (5.26)
\end{aligned}$$

where $\mathcal{I}(a, b)$ represents the mutual information between a and b, entropy is denoted by $\mathcal{H}$, the space $\Omega^- = \{\Omega \setminus \mathbf{o}^+\}$ is over all observation sequences excluding the positive observation $\mathbf{o}^+$, the latent $\tilde{z} = \Pi_\omega(\mu_{\mathbf{o}_t})$ is the output of a projection network parameterized by ω, and the function f measures the similarity between the observation $\mathbf{o}_t$ and projected latent variable $\tilde{z}$, following Oord *et al.* [137]:

$$f(\mathbf{o}_t, \tilde{z}) = \exp\left(\frac{\tilde{z}^\top h_t}{\|\tilde{z}\| \|h_t\|} / \tau\right) \quad \text{where} \quad h_t = \mathcal{E}_{\mathbf{o}}(\mathbf{o}_t) \tag{5.27}$$

where τ is a temperature parameter. For a window of length K, by minimizing the loss $\mathcal{L}_{\text{TCL}}$, we maximize mutual information for observation trajectories from $\mathbf{o}_t$ to $\mathbf{o}_{t+K}$. The idea is that we want to optimize the latent space so that observations that come from the same sequence are closer together, while observations from different sequences are further apart. In addition to fine-tuning the mutual information, we also want to preserve reconstruction and generation. Therefore, the fine-tuning $\mathcal{I}$-VAE loss function then becomes:

$$\mathcal{L}(s, \mathbf{o}_{1:T}; \omega, \theta, \phi, \psi, K) = \alpha \mathcal{L}_{\text{TCL}}(\mathbf{o}_{1:T}; \omega, K) + \mathbb{E}_t\left[\mathcal{L}^{(t)}_{\mathcal{I}\text{-VAE}}(s, \mathbf{o}_t; \theta, \phi, \psi)\right] \tag{5.28}$$

The TCL loss acts as a self-supervised consistency loss on the observation encoder, helping it produce a smooth, predictive latent path [145]. In practice, we train the projection network Π_ω to discriminate between observation trajectories.

Training and inference. The $\mathcal{I}$-VAE training procedure and forward pass are detailed in algorithms 5.1 and 5.2. During runtime inference, the input observation is encoded to get

$h_{\mathbf{o}} = \mathcal{E}_{\mathbf{o}}(\mathbf{o})$, a latent-observation variable $z_{\mathbf{o}}$ is sampled from $p_\psi(z \mid \mathbf{o})$, and a sampled state is reconstructed from the decoder $p_\theta(s \mid z, \mathbf{o})$ using $z = [z_{\mathbf{o}}, h_{\mathbf{o}}]$.

Algorithm 5.1: $\mathcal{I}$-VAE training procedure.

Require: $\theta, \phi, \psi, \omega \leftarrow$ initialize network parameters
Require: $\alpha \leftarrow$ trajectory contrastive loss weighting
Require: $\tau \leftarrow$ trajectory contrastive loss temperature
Require: $T \leftarrow$ observation horizon

for epoch $\leftarrow 1$ **to** n_{pretrain}
  Sample $(s, \mathbf{o})$ from $\mathcal{D}_{\text{partial}}$
  Forward pass $(s, \mathbf{o})$ to get $(\hat{s}, \mu_s, \log \sigma_s^2, \mu_{\mathbf{o}}, \log \sigma_{\mathbf{o}}^2)$ calling algorithm 5.2
  Compute loss $\mathcal{L}_{\mathcal{I}\text{-VAE}}(s, \mathbf{o}; \theta, \phi, \psi)$ given $(\hat{s}, \log \sigma_s^2, \mu_{\mathbf{o}}, \log \sigma_{\mathbf{o}}^2)$ with equation (5.14)
  Train θ, ϕ, and ψ
for epoch $\leftarrow 1$ **to** $n_{\text{fine-tune}}$
  Sample $(s, \mathbf{o}_{1:T})$ from $\mathcal{D}_{\text{traj}}$
  Compute μ_s and $\log \sigma_s^2$ from state encoder $\mathcal{E}_s(s)$
  for $t \leftarrow 1$ **to** T
    Encode $h_{\mathbf{o}_t}$ from observation encoder $\mathcal{E}_{\mathbf{o}}(\mathbf{o}_t)$ using $\mathbf{o}_t \in \mathbf{o}_{1:T}$
    Compute $\mu_{\mathbf{o}_t}$ and $\log \sigma_{\mathbf{o}_t}^2$ from $p_\psi(z \mid \mathbf{o}_t)$ using observation encoding $h_{\mathbf{o}_t}$
    Project $\tilde{z} = \Pi_\omega(\mu_{\mathbf{o}_t})$
    Decode $\hat{s}_t$ from $p_\theta(s \mid \tilde{z}, \mathbf{o})$
    Compute $\mathcal{L}_t = \mathcal{L}_{\mathcal{I}\text{-VAE}}(s, \mathbf{o}_t; \theta, \phi, \psi)$ from equation (5.14) using $\hat{s}_t$
  Compute combined loss $\mathcal{L} = \alpha \mathcal{L}_{\text{TCL}}(\mathbf{o}_{1:T}; \omega, K) + \mathbb{E}_t[\mathcal{L}_t]$ using equation (5.23)
  if epoch $= 1$
    Train projection network parameters ω
  else
    Train ω and ψ (optional: θ and ϕ)

Algorithm 5.2: $\mathcal{I}$-VAE forward pass.

Require: $s, \mathbf{o}$ (state and observations)

Compute h_s, μ_s, and $\log \sigma_s^2$ from state encoder $h_s = \mathcal{E}_s(s)$ and $q_\phi(z \mid s)$
Sample $z \sim q_\phi(z \mid s)$ using state encoding h_s
Compute $h_{\mathbf{o}}$, $\mu_{\mathbf{o}}$, and $\log \sigma_{\mathbf{o}}^2$ from observation encoder $h_{\mathbf{o}} = \mathcal{E}_{\mathbf{o}}(\mathbf{o})$ and $p_\psi(z \mid \mathbf{o})$
Decode $\hat{s}$ from $p_\theta(s \mid z, \mathbf{o})$ using z and observation encoding $h_{\mathbf{o}}$
return $(\hat{s}, \mu_s, \log \sigma_s^2, \mu_{\mathbf{o}}, \log \sigma_{\mathbf{o}}^2)$

5.4 Experiments and Analysis

We demonstrate the effectiveness of the $\mathcal{I}$-VAE on two sequential problems: partial MNIST observations [16] and a real-world muon tomography mineral exploration EMDP. The model code,[‡] written in PyTorch, and the muon observation EMDP code,[§] written in Julia and Python, are open-sourced to extend to other problems and for reproducibility. Details regarding hyperparameters and neural network architectures are included in the code.

5.4.1 Pixel Observation Problem: MNIST

Using the standard MNIST benchmark of hand-written digits, we are interested in measuring how well the model performs given partial observations. We partition the image into 196 chunks of 4×4 pixels each to act as a individual observations. In this problem, the observations can be interpreted as a mask applied to the true (hidden) state. Given random pixel observations over time, we refer to the ''unmasked'' observations as the observation coverage percentage (e.g., if 49 out of 196 chucks are uncovered, we have 25% coverage). We compare the $\mathcal{I}$-VAE model against a standard CVAE, a deterministic neural network (NN) baseline, and against the $\mathcal{I}$-VAE pretrained model (i.e., without fine-tuning with TCL). All of the models are simple MLPs following parameters used by Sohn *et al.* [102] for the original MNIST CVAE experiments. To measure the quality of the models, we evaluate the conditional log-likelihood (CLL) across each observation coverage percentage [102]:

$$\log p_\theta(s \mid \mathbf{o}) \approx \log \left(\frac{1}{n} \sum_{i=1}^{n} p_\theta\big(s \mid z^{(i)}, \mathbf{o}\big) \right) \quad \text{where} \quad z^{(i)} \sim p_\psi(z \mid \mathbf{o}) \tag{5.29}$$

where n is the size of the MNIST test dataset ($n = 10{,}000$). The CLL metric is a standard way to evaluate the statistical performance of the models.

Table 5.1 shows the conditional log-likelihoods across the various models for different observation coverage percentages. In the MNIST case, we observe that the $\mathcal{I}$-VAE without TCL performs just as well as the CVAE model. The benefit of additional TCL fine-tuning is most evident in regimes with low observation coverage (e.g., no observations at 0% and 25% observation coverage). Unsurprisingly, the neural network baseline performs the worst as it will deterministically infer a single state given the observations.

[‡] https://github.com/sisl/I-VAE

[§] https://github.com/sisl/MuonPOMDPs.jl

Method	Conditional log-likelihood (CLL)				
	0%	25%	50%	75%	100%
NN (baseline)	−197.11	−93.44	−74.30	−66.83	−62.43
CVAE	−192.52	−86.89	−68.76	−63.19	−59.47
$\mathcal{I}$-VAE (no TCL)	−195.07	−87.15	−68.90	−62.97	−59.02
$\mathcal{I}$-VAE	**−176.41**	**−82.39**	**−67.77**	**−62.38**	**−58.51**

Table 5.1: MNIST: CLLs given different observation coverage percentages.

The better performance with less observation coverage is also evident in figure 5.7. At observation coverages of around 30% or less, the $\mathcal{I}$-VAE model has higher conditional log-likelihood than the CVAE. Also evident in figure 5.7 is that the $\mathcal{I}$-VAE model without TCL recovers CVAE performance—which is true in the MNIST example, but as we will see in the next section, both $\mathcal{I}$-VAE models outperform the CVAE when using an information-based heuristic planner, framing the sequential information gathering problem as an EMDP.

Figure 5.8 qualitatively illustrates how the $\mathcal{I}$-VAE can better reconstruct the data when conditioning on no observations (i.e., observation $\mathbf{o}$ is solely comprised of all -1 values). In this unconditioned case, when the observation coverage is 0%, we can visually see that when sampling states with no observation information, the mean belief, shown in figures 5.8a to 5.8c, matches closer to the MNIST test data prior. We also visualize reconstructed samples, conditioned without any observation information in figures 5.8d to 5.8f.

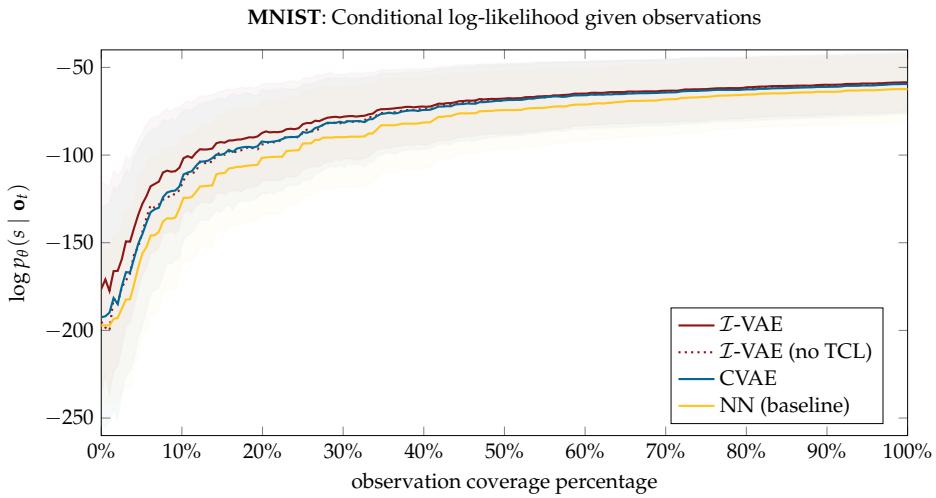
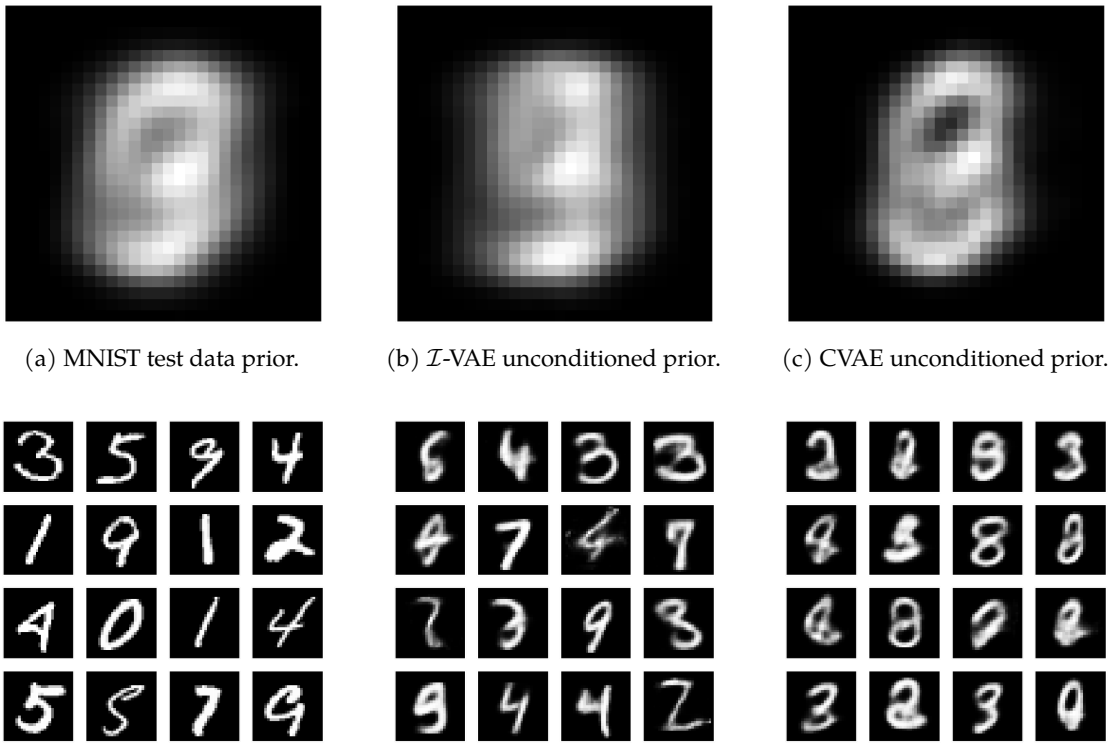


Figure 5.7: Conditional log-likelihood over the MNIST test dataset ($\pm$ one stddev).



(a) MNIST test data prior. (b) $\mathcal{I}$-VAE unconditioned prior. (c) CVAE unconditioned prior.

(d) Test data samples. (e) $\mathcal{I}$-VAE unconditioned samples. (f) CVAE unconditioned samples.

Figure 5.8: Mean priors (i.e., no observations), sampling $m = 1000$ for $\mathcal{I}$-VAE and CVAE.

Finally, figures 5.9 and 5.10 demonstrate additional qualitative analysis. The figures show representative examples of taking random pixel observation actions from 0% to 100% coverage for the $\mathcal{I}$-VAE and CVAE. We plot the mean belief over 1000 samples and several generated state samples over time. These results also indicate that the $\mathcal{I}$-VAE not only learns better and more diverse digit reconstructions, as evident by the more visually consistent predictions, but also converges to the correct state in fewer observations (about 10% in figure 5.9 and about 20% in figure 5.10). We also include the aggregate classification probabilities from a learned classification MLP[¶] in the bottom row, further indicating the confidence in the prediction when aggregating over the 1000 sampled states. In the classification plots, the shaded bars represent the correct classification label. In both models, when the observation coverage is high, the reconstruction recovers the true state.

[¶]Following Imambi *et al.* [146] for a simple MLP classification model from `https://github.com/pytorch/examples/blob/main/mnist/main.py`.

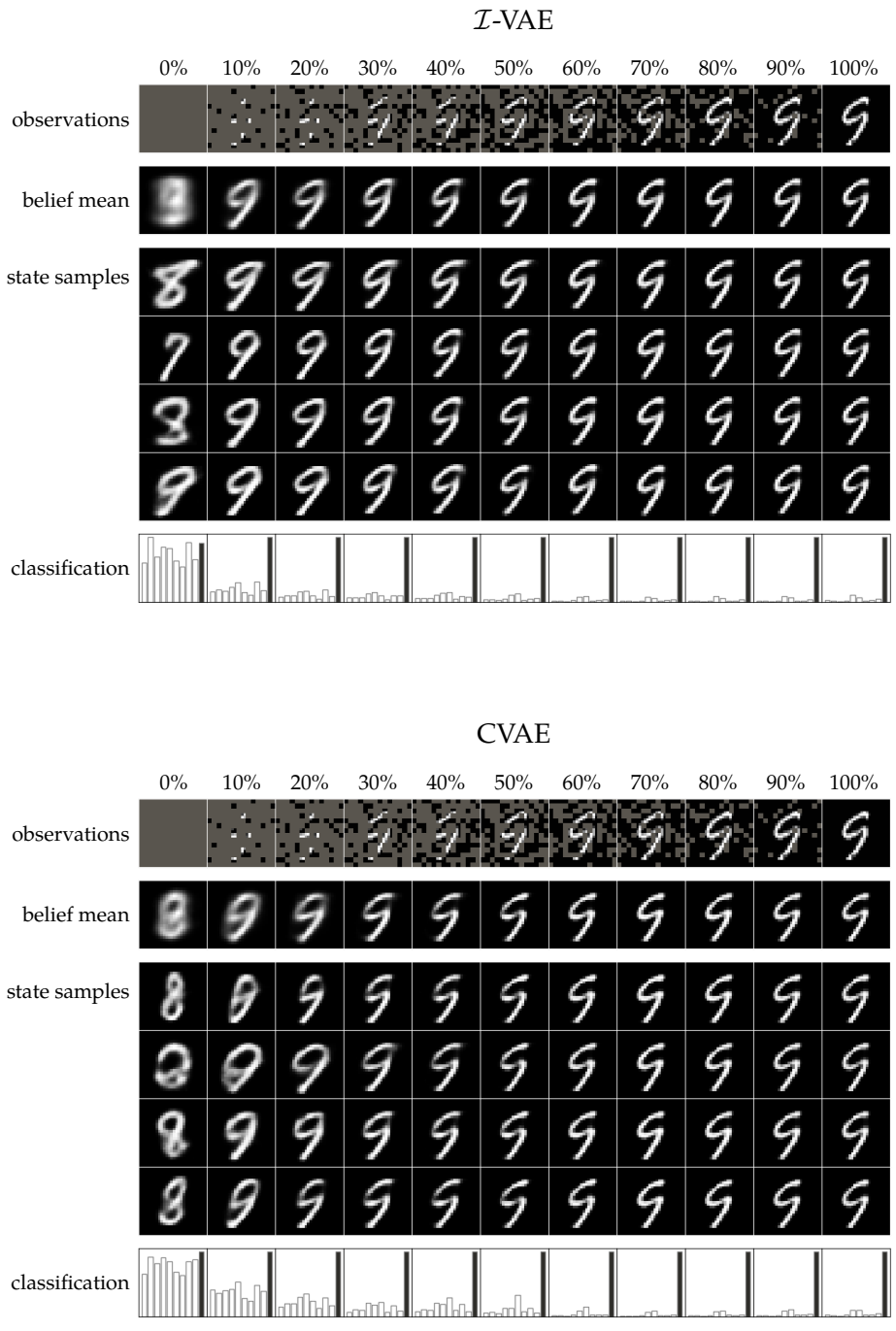


Figure 5.9: MNIST example observations over time for the $\mathcal{I}$-VAE and CVAE models.

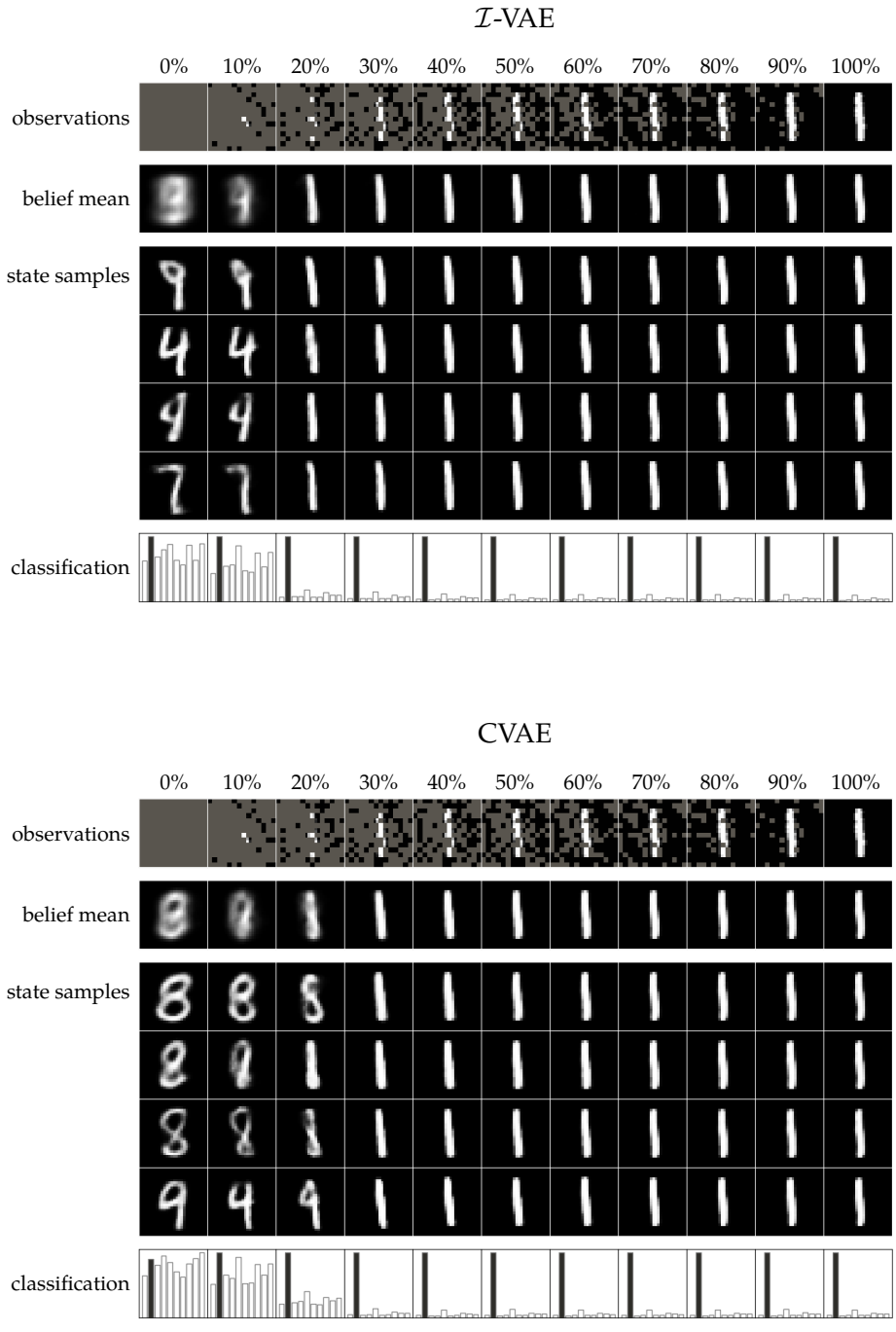


Figure 5.10: Additional MNIST observations over time for the $\mathcal{I}$-VAE and CVAE models.

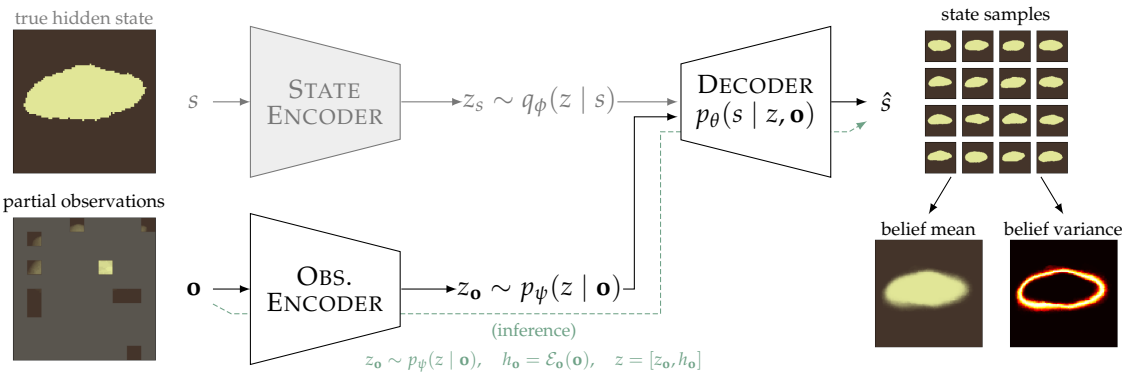


Figure 5.11: Muon observation inversion process using an $\mathcal{I}$-VAE with 10 observations.

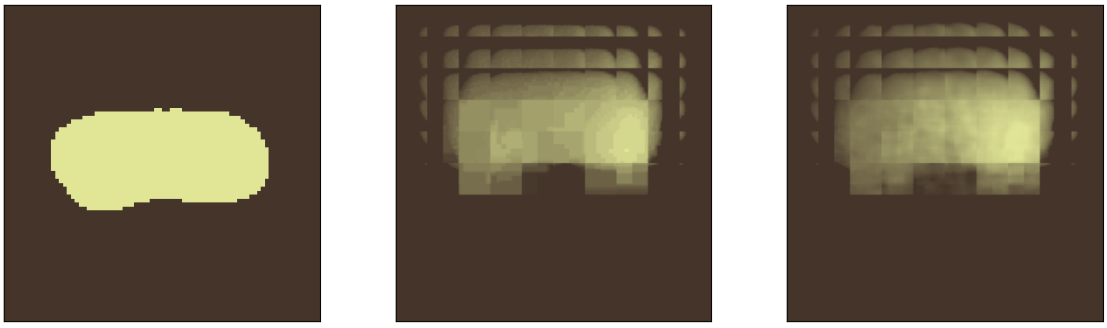
5.4.2 Indirect Observation Problem: Muon Tomography

In the previous section, we showed how the $\mathcal{I}$-VAE model operates when treating pixel observations as a state mask. In more realistic geological problems, different measurement sources, such as seismic imaging [115] or muon tomography [36, 147] (shown in figure 5.11), are used to infer the unknown state of the subsurface. Statistical models of the subsurface are particularly useful as they capture the uncertainty in what's underground [148].

In the field of mineral exploration [149], typical strategies consist of drilling or placing sensors in an exhaustive grid to collect subsurface information. More recently, Mern *et al.* [150] framed the mineral exploration problem as a POMDP and used state-of-the-art POMDP solvers to intelligently select where to drill, avoiding standard grid-based policies. Mern *et al.* [150] treated drill actions as observing borehole data to collect subsurface information. Mineral exploration is a multi-objective problem: we want to maximize information gain while simultaneously minimizing the overall cost (e.g., minimizing the number of drills). The ultimate objective of the mineral exploration problem is to make a `go/no-go` decision to `mine` or `abandon` based on some economic value of the subsurface intrusion (e.g., the mass of a subsurface ore body being worth the cost to extract). The problem can be framed as an information gathering EMDP where the true state does not transition.

Sequential Information Gathering using Batched Belief-State Planning

Crucial to the success of the POMDP solvers is the efficacy of the belief updater. Mern *et al.* [150] used a particle filter given a known observation model $O(o \mid a, s')$. In our work,

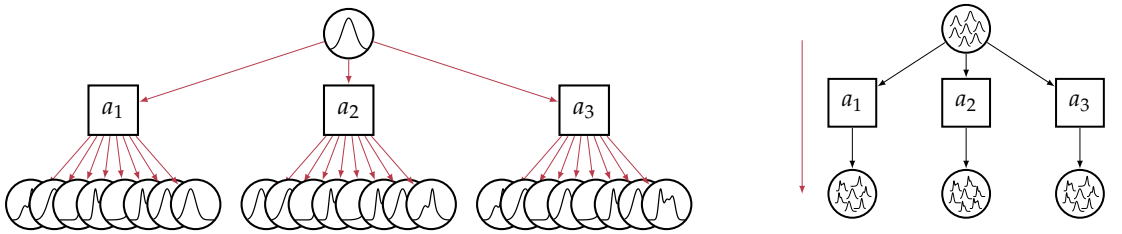


(a) Example input muon state. (b) Physics-based muon simulation. (c) U-Net observation surrogate.

Figure 5.12: Muon density U-Net surrogate model compared to the physics-based simulator.

we study the case where the observation model is not known (or difficult to compute), the observations come from muon tomography projected to 2D, and drill actions place a row of 9 passive muon sensors into the subsurface, as shown in 3D in figure 5.1. The muon tomography EMDP receives a penalty for every drill action executed, a reward proportional to the difference in the extracted intrusion mass compared to an economic threshold if the `go` action is taken, and zero reward in the case of a `no-go` decision.

Introduced in chapter 4, we convert the mineral exploration with muon tomography EMDP (termed *muon-based intrusion discovery*) into its batched equivalent. Aside from the observation function (which we cover next), the $\mathcal{I}$-VAE belief updater is designed to be parallel and, trivially, there are no state-transition dynamics. As for the forward observation function, we trained a simple U-Net [151] as a surrogate observation model (figure 5.12) using data from a physics-based muon density simulator. Therefore, we can directly convert the muon-based intrusion discovery problem into a **BB**-MDP. Using this formulation, along with the neural network surrogates, we can sample m state particles and evaluate all 100



(a) Belief-state MDP one-step lookahead over 3 actions. (b) **BB**-MDP one-step lookahead.

Figure 5.13: Indicating number of belief transition calls in red for BDMPs vs. **BB**-MDPs.

actions (corresponding to muon sensor drill locations) in one forward pass to the U-Net observation surrogate and one forward pass to the $\mathcal{I}$-VAE belief updater as a batch, shown compared to the standard belief-state MDP (BMDP) in figure 5.13.

Information-based heuristic policy. To test the performance of the $\mathcal{I}$-VAE belief updater in the batched planning setting, we use an information-based heuristic policy to maximize the expected Bayesian information gain. The heuristic policy, shown in equation (5.45), will pick drill locations $a \in \mathcal{A}$ over the 10×10 drilling sites that maximize expected information:

$$\mathcal{H}(b) - \mathbb{E}_i\big[\mathcal{H}(b'_i)\big] \approx \mathbb{E}_i\left[\sum_{s \in b'_i} b'_i(s) \log b'_i(s)\right] - \sum_{s \in b} b(s) \log b(s) \tag{5.30}$$

where $b'_i \in \mathbf{b}'$ and $\mathbf{b}' \sim \mathbf{T}_b(\cdot \mid \mathbf{b}, a)$ is an m-sized batch of beliefs, each with k state particles. In our experiments, we use $m = 3$ beliefs in the batch and $k = 100$ state samples per belief. We show that picking the action that maximizes the expected entropy reduction is equivalent to maximizing the expected KL-divergence objective (relating to the *efficacy of information* (EOI) [152]).

Theorem 2 (Equivalence of expected entropy reduction and expected KL-divergence)**.** *Let b be a prior belief over the continuous state space $\mathcal{S}$. Suppose we draw m posterior beliefs $[b'_1, \ldots, b'_m] = \mathbf{b}'$ by sampling $b'_i \sim T_b(\cdot \mid b, a, o_i)$ following equations (4.24) to (4.27), then the expected reduction in entropy is equivalent to the expected KL-divergence between $\mathbf{b}'$ and b, namely:*

$$\mathcal{H}(b) - \mathbb{E}_i\big[\mathcal{H}(b'_i)\big] = \mathbb{E}_i\big[\,\mathrm{KL}(b'_i \,\|\, b)\big] \quad \textit{where} \quad b'_i \in \mathbf{b}' \tag{5.31}$$

This results in an optimal one-step information gathering policy that maximizes Bayesian information gain. Subsequently, this can be computed using the Monte Carlo estimate:

$$\mathcal{H}(b) - \mathbb{E}_i\big[\mathcal{H}(b'_i)\big] \approx \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{s \in b'_i} b'_i(s) \log b'_i(s)\right) - \sum_{s \in b} b(s) \log b(s) \tag{5.32}$$

As $m \to \infty$ and the state particle count $k \to \infty$, this converges to the true expected KL-divergence:

$$\mathbb{E}_i\big[\,\mathrm{KL}(b'_i \,\|\, b)\big] = \lim_{\substack{m \to \infty \\ k \to \infty}} \left[\frac{1}{m} \sum_{i=1}^{m} \left(\sum_{s_j}^{k} b'_i(s_j) \log b'_i(s_j)\right) - \sum_{s_j}^{k} b(s_j) \log b(s_j)\right] \tag{5.33}$$

Proof. We show that the expected entropy reduction of the belief is exactly the expected KL-divergence between each posterior and the prior, justifying the use of this measurement when selecting actions to maximize expected information gain. For each posterior belief b_i', the KL-divergence can be expressed as:

$$\mathrm{KL}(b_i' \,\|\, b) = \int b_i'(s) \log b_i'(s)\, \mathrm{d}s - \int b_i'(s) \log b(s)\, \mathrm{d}s \tag{5.34}$$

$$= -\mathcal{H}(b_i') - \int b_i'(s) \log b(s)\, \mathrm{d}s \tag{5.35}$$

Rearranging and adding $\mathcal{H}(b)$ to both sides, we get:

$$\mathcal{H}(b) - \mathcal{H}(b_i') = \mathrm{KL}(b_i' \,\|\, b) + \int b_i'(s) \log b(s)\, \mathrm{d}s + \mathcal{H}(b) \tag{5.36}$$

Because b_i' is generated by first sampling $s \sim b$, followed by $s' \sim T(\cdot \mid s, a)$, $o \sim O(\cdot \mid a, s')$, and updated using $b' = \textsc{Update}(b, a, o)$, by the law of total expectation, we get:

$$\mathbb{E}_i\left[\int b_i'(s) \log b(s)\, \mathrm{d}s\right] = \mathop{\mathbb{E}}_{s\sim b}\left[\mathop{\mathbb{E}}_{(s',o|s)}\left[\mathbb{E}_{b_i'}\left[\log b(s)\right]\right]\right] \tag{5.37}$$

$$= \mathbb{E}_s\left[\log b(s)\right] \tag{5.38}$$

$$= \int b(s) \log b(s)\, \mathrm{d}s \tag{5.39}$$

$$= -\mathcal{H}(b) \tag{5.40}$$

Now using this result in the expanded definition of expected entropy reduction, we get:

$$\mathcal{H}(b) - \mathbb{E}_i\left[\mathcal{H}(b_i')\right] = \mathbb{E}_i\left[\mathcal{H}(b) - \mathcal{H}(b_i')\right] \tag{5.41}$$

$$= \mathbb{E}_i\left[\mathrm{KL}(b_i' \,\|\, b) + \underbrace{\int b_i'(s) \log b(s)\, \mathrm{d}s}_{-\mathcal{H}(b)} + \mathcal{H}(b)\right] \tag{5.42}$$

$$= \mathbb{E}_i\left[\mathrm{KL}(b_i' \,\|\, b)\right] \tag{5.43}$$

Therefore, we reach our desired equivalence of:

$$\mathcal{H}(b) - \mathbb{E}_i\left[\mathcal{H}(b_i')\right] = \mathbb{E}_i\left[\mathrm{KL}(b_i' \,\|\, b)\right] \tag{5.44}$$

□

Using the results from theorem 2, the information-based heuristic policy will select actions that maximize the information gain, then make a final decision based off the probability that the economic intrusion volume is above some risk threshold $1 - \Delta$. The heuristic policy is defined as:

$$\pi_{\text{heuristic}}(b) = \begin{cases} \texttt{go} & \text{if } P(v > 0) > 1 - \Delta \\ \texttt{no-go} & \text{if } P(v < 0) > 1 - \Delta \\ \arg\max_{a \in \mathcal{A}} \mathcal{H}(b) - \mathbb{E}_i\left[\mathcal{H}(b_i')\right] & \text{otherwise} \end{cases} \tag{5.45}$$

where the distribution over intrusion volume v is computed over the belief b and standardized so that the nominal (prior) volume has zero mean. When the standardized volume is above zero, this indicates the intrusion is economical to mine. This way, if we randomly took `go`/`no-go` actions, we would make the correct decision 50% of the time. The parameter Δ controls the risk tolerance in the final `go`/`no-go` decision. In the following experiments, we set $\Delta = 0.1$, resulting in making the final decision if the confidence in our estimated volume distribution is above 90%. Other criterion, such as the upper-confidence bound (UCB1) algorithm [153] or Bayes-UCB [154], could be used instead.

Baseline policies. Along with the heuristic policy, we evaluate four other baseline policies. The first two baseline policies select actions deterministically following sweeping grid-like patterns. Starting from the top-left origin of the 10×10 action space, the *horizontal grid* policy selects actions in a right-to-left, then left-to-right sweeping pattern. Similarly, the *vertical grid* policy selects actions in a downward, then upward pattern; mimicking conventional drilling strategies [149]. We also baseline against a *random* policy that will randomly select actions uniformly over the action space. Finally, given privileged information about the true state under test, we test against an *oracle* that selects the action that minimizes the error between the mean posterior belief and the true state (with access to truth):

$$\pi_{\text{oracle}}(b) = \begin{cases} \texttt{go} & \text{if } P(v > 0) > 1 - \Delta \\ \texttt{no-go} & \text{if } P(v < 0) > 1 - \Delta \\ \arg\min_{a \in \mathcal{A}} \left| \mathbb{E}[b'] - s_{\text{true}} \right| & \text{otherwise} \end{cases} \tag{5.46}$$

The oracle policy determines the best performance that each belief updater could achieve. We compare the $\mathcal{I}$-VAE with and without TCL fine-tuning, the CVAE model, and a particle filter with reinvigoration where the estimated observation likelihood measures how close each observation is to a sampled state from the belief, proportional to the mean-squared error between the observation generated from the sampled state and the environment observation.

Planning and estimation metrics. We use several metrics to compare how well the $\mathcal{I}$-VAE estimates the conditional likelihood $p(s \mid \mathbf{o})$ and its performance during planning. For estimation quality, we use the conditional log-likelihood defined in equation (5.29). To evaluate the planning performance using the heuristic and baseline policies, we measure the accuracy in the final `go/no-go` decision (where `go` should be executed if the economic volume of a test intrusion is above zero), and we measure how many actions it took to reach the final decision. Finally, we measure the error in the belief compared to the true test state over different number of actions/observations.

Muon-Based Intrusion Discovery Results and Analysis

Table 5.2 and figure 5.14 show the estimation performance of the neural network belief surrogates. When selecting drill actions, each surrogate uses the information-based heuristic policy *without* making a final decision (to ultimately evaluate the performance over all drill actions). Evident in our experiments, we see that the $\mathcal{I}$-VAE belief updater achieves better estimation qualities when compared to the CVAE and the $\mathcal{I}$-VAE without TCL. Unlike the MNIST example, in the muon tomography case, the initial belief (i.e., conditioned on no observations) is more diffuse, resulting in worse early estimation performance.

Regarding planning performance, table 5.3 indicates that the $\mathcal{I}$-VAE model produces the highest accuracy in the fewest number of actions. We also highlight that the additional TCL

Method	Conditional log-likelihood (CLL)				
	0	25	50	75	100
CVAE	$-\mathbf{563.41}$	-213.98	-169.42	-148.90	-141.17
$\mathcal{I}$-VAE (no TCL)	-692.58	-173.47	-155.47	-144.20	-127.79
$\mathcal{I}$-VAE	-657.39	$\mathbf{-159.04}$	$\mathbf{-147.93}$	$\mathbf{-136.54}$	$\mathbf{-125.45}$

Table 5.2: Muon problem: CLLs given different numbers of observations.

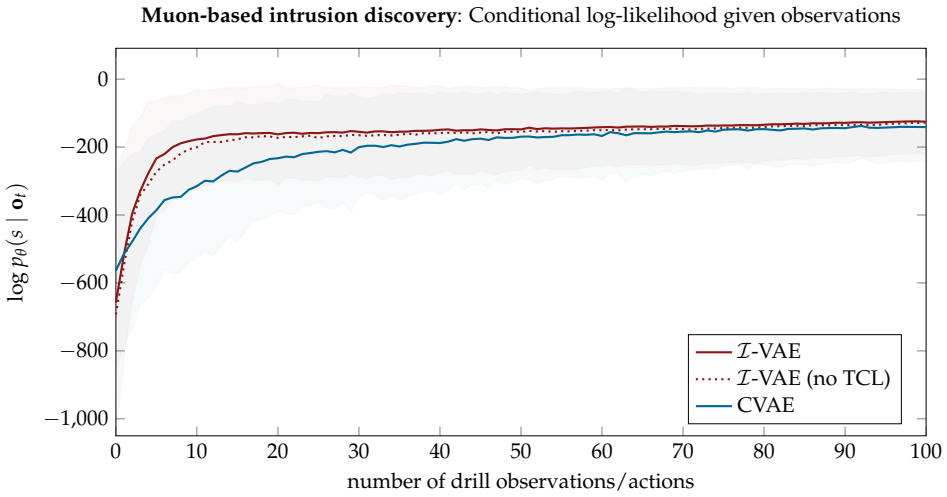


Figure 5.14: Conditional log-likelihood for the muon problem using the heuristic policy.

fine-tuning step improves the base model. Comparing the belief updating results in table 5.3 to their respective oracle column shows the theoretically one-step optimal performance achievable by each method. This is also evident when looking at the belief error in figure 5.15 and comparing the $\mathcal{I}$-VAE heuristic to the $\mathcal{I}$-VAE oracle, which shows that the information-based heuristic is approximately optimal. It is also clear that the particle filter performs poorly on this problem as it suffers from particle depletion (i.e., the belief collapses). This is

Method	Grid (horiz.)	Grid (vert.)	Random	Heuristic	Oracle
	Accuracy ↑ # Actions ↓	Accuracy ↑ # Actions ↓	Accuracy ↑ # Actions ↓	Accuracy ↑ # Actions ↓	Accuracy ↑ # Actions ↓
Particle filter†	0.75 ± 0.03 89.00 ± 1.03	0.65 ± 0.03 76.17 ± 1.13	0.67 ± 0.03 76.52 ± 1.02	— —	— —
CVAE	0.92 ± 0.02 $\mathbf{27.38 \pm 1.79}$	0.925 ± 0.02 35.40 ± 1.80	0.925 ± 0.02 24.36 ± 2.05	0.925 ± 0.02 20.65 ± 1.84	0.935 ± 0.02 21.86 ± 1.85
$\mathcal{I}$-VAE (no TCL)	$\mathbf{0.925 \pm 0.02}$ 28.85 ± 1.24	$\mathbf{0.955 \pm 0.01}$ 34.02 ± 1.53	0.935 ± 0.02 18.78 ± 1.47	0.95 ± 0.02 7.46 ± 0.97	0.975 ± 0.01 9.62 ± 1.45
$\mathcal{I}$-VAE	$\mathbf{0.925 \pm 0.02}$ 28.45 ± 1.31	$\mathbf{0.955 \pm 0.01}$ $\mathbf{31.11 \pm 1.39}$	$\mathbf{0.95 \pm 0.02}$ $\mathbf{14.53 \pm 0.90}$	$\mathbf{0.96 \pm 0.01}$ $\mathbf{6.88 \pm 0.85}$	$\mathbf{0.99 \pm 0.01}$ $\mathbf{8.15 \pm 1.22}$

† Heuristic and oracle results were omitted because the particle filter does not inherently support parallelism.

Table 5.3: Muon problem: Planning metrics comparing various policies and belief updaters.

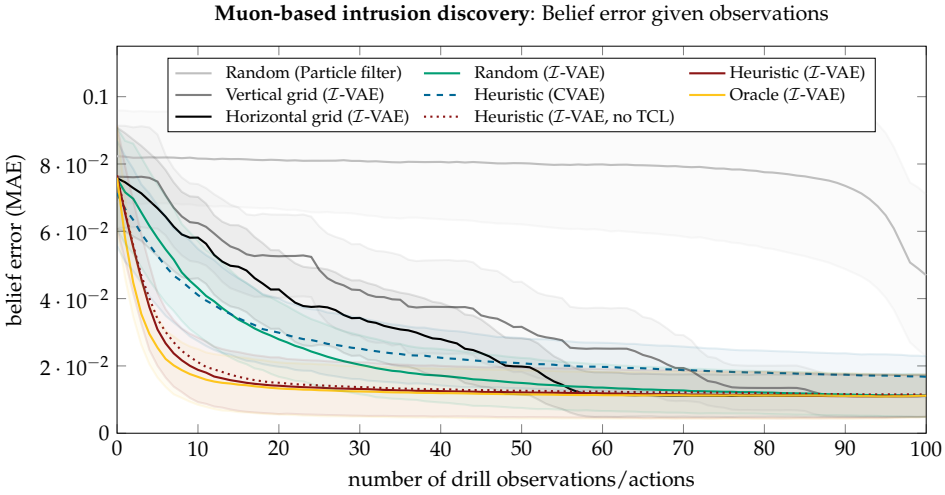
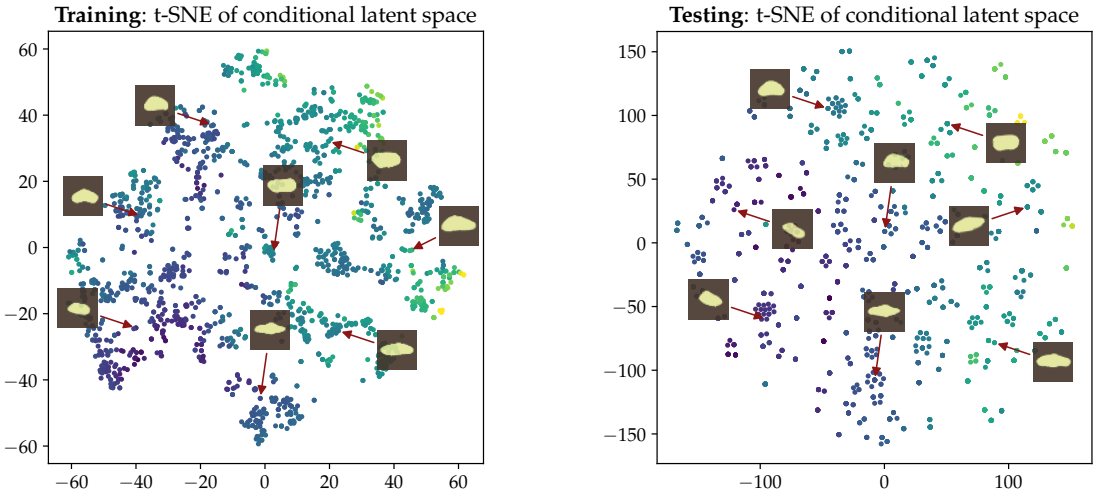


Figure 5.15: Belief error relative to holdout test states for different policies and updaters.

because the particle filter initially samples $k = 900$ state particles from the prior training data to represent its belief—using the same training states as the CVAE and $\mathcal{I}$-VAE models, given those states are the only examples each model has access to. Sampling the finite state particles results in the inability of the particle filter to generate new states that are consistent with the observations but may not directly exist in the finite prior data. This highlights the usefulness of learning generative models to act as approximate belief updaters.



(a) Conditional latent space (1800 training data).

(b) Conditional latent space (200 testing data).

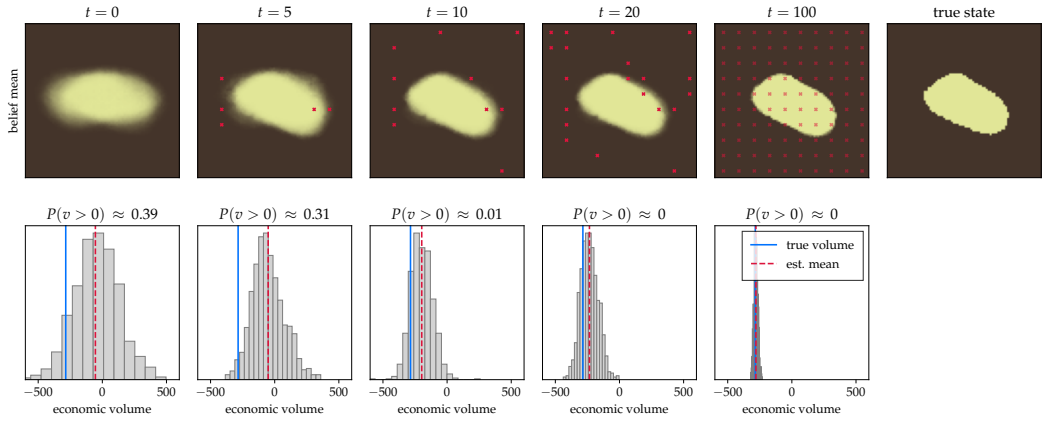
Figure 5.16: Visualizing the conditional latent space $p_\psi(z \mid \mathbf{o})$, colored by intrusion volume.

Because the $\mathcal{I}$-VAE model explicitly learns parameters $\mu_{\mathbf{o}}$ and $\log \sigma^2_{\mathbf{o}}$ for the conditional prior $p_\psi(z \mid \mathbf{o})$, instead of a unit Gaussian, we can easily visualize the latent space using the t-SNE dimensionality reduction technique [155]. Visualized in figure 5.16, we can inspect the latent space and qualitatively determine that it learns to cluster observations based on the economic volume of the reconstructed states (shown in the colors). The inlet plots show the average state reconstruction over 100 samples.

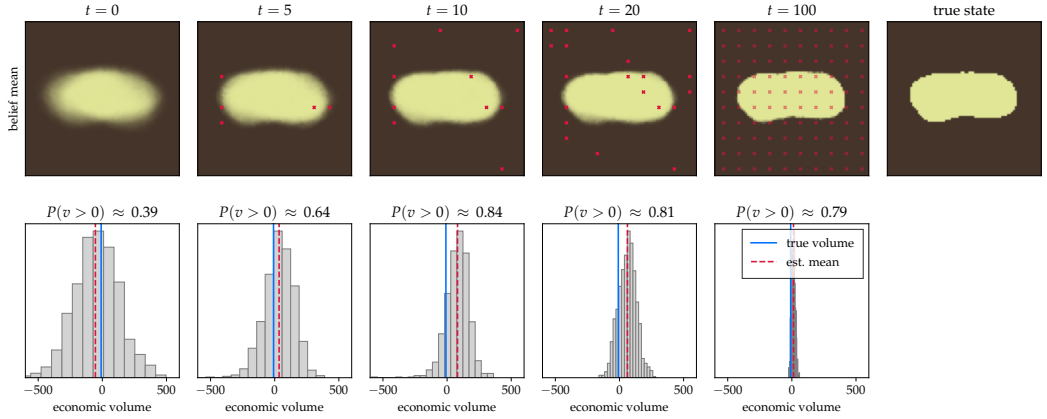
Finally, figure 5.17 illustrates three example planning trajectories showing the belief mean over 100 sampled states (where drill action locations are marked in red) and we show the economic intrusion volume distribution. Quantitatively, in each of these three cases, we see that the heuristic planner using the $\mathcal{I}$-VAE model converges close to the true intrusion volume. Qualitatively, these examples illustrate that the $\mathcal{I}$-VAE model accurately constructs a belief that converges, in shape, to the true state. The figure shows the mean belief and volume distributions after a fixed number of actions, as well as the final belief and volume distribution when all 100 drill actions are taken, where the blue lines indicate the true intrusion volume and the dashed red lines indicate the estimated mean from the belief.

5.5 Conclusions

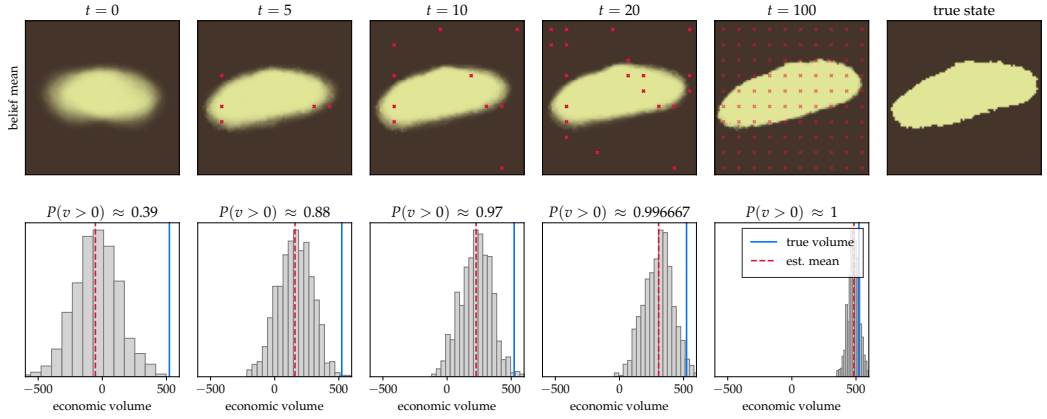
In this chapter, we introduced a conditional deep generative model to approximate the belief update in an EMDP. We framed the EMDP problem as its batched equivalent using the framework introduced in chapter 4. To plan efficiently over batches, we introduced the *inversion variational autoencoder* ($\mathcal{I}$-VAE) model that learns a latent distribution over states and a matching latent distribution over partial observations. Given our sequential decision making setting, we further refine the model during a fine-tuning step to maximize the mutual information between latent variables along the same observation trajectories. We showed that this model works well in cases where the observations are masked versions of the state and when the observations are indirectly related to the state through a forward model. We demonstrated the model performance when used as a surrogate belief updater in a muon-based intrusion discovery problem, and showed that an information-based heuristic policy works well in this setting. Insights from this work highlight that in problems with limited data, e.g., geological problems, using generative models allows for the creation of unseen states that are consistent with both the prior and the conditioning observations.



(a) Intrusion planning where the true volume is *below* the economic threshold.



(b) Intrusion planning where the true volume is *near* the economic threshold.



(c) Intrusion planning where the true volume is *above* the economic threshold.

Figure 5.17: Beliefs for the muon-based intrusion discovery problem using the $\mathcal{I}$-VAE model.

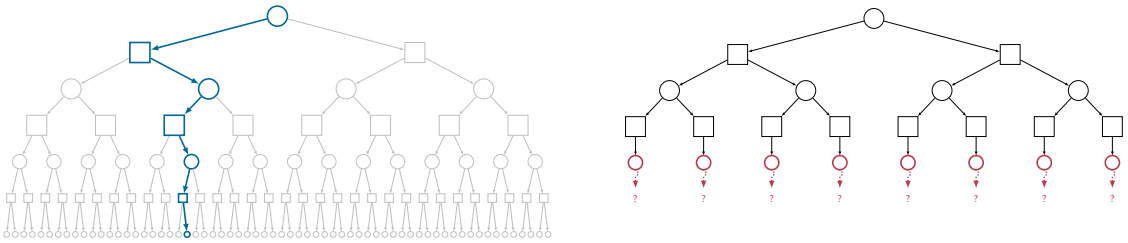
Chapter 6

Planning and Learning in Belief Space

You should consider that Imitation is the most acceptable part of Worship, and that the Gods had much rather Mankind should Resemble, than Flatter them.

Jeremy Collier and André Dacier

Real-world planning problems, including autonomous driving and sustainable energy applications like carbon storage and resource exploration, have recently been modeled as partially observable Markov decision processes (POMDPs) and solved using approximate methods. To solve high-dimensional POMDPs in practice, state-of-the-art methods use online planning with problem-specific heuristics to reduce planning horizons and make the problems tractable, as done in chapter 5. Algorithms that learn approximations to replace heuristics have recently found success in large-scale fully observable domains [17]. The key insight is to use the combination of online Monte Carlo tree search with offline neural network approximations of the optimal policy and value function. In this chapter, we bring this insight to partially observable domains and propose *BetaZero*, a belief-state planning algorithm for high-dimensional POMDPs. BetaZero learns offline approximations that replace heuristics to enable online decision making in long-horizon problems. We address several challenges inherent in large-scale partially observable domains; namely challenges of transitioning in stochastic environments, prioritizing action branching with a limited search budget, and representing beliefs as input to the network. To formalize the use of all limited search information, we train against a novel Q-weighted visit counts policy. We test BetaZero on various well-established POMDP benchmarks found in the literature



(a) Goal: Reduce tree search breadth. (b) Goal: Reduce tree search depth

Figure 6.1: Motivation to reduce tree search breadth and depth in online tree search.

and a real-world problem of critical mineral exploration. Experiments show that BetaZero outperforms state-of-the-art POMDP solvers on a variety of tasks.

6.1 Motivation

Optimizing sequential decisions in real-world settings is challenging due to uncertainties about the true state of the environment. Modeling such problems as partially observable Markov decision processes (POMDPs) has shown recent success in autonomous driving [41], robotics [57], and aircraft collision avoidance [12]. Solving large or continuous POMDPs requires approximations in the form of state-space discretizations or modeling assumptions, e.g., assuming full observability [156]. Although these approximations are useful when making decisions in a short time horizon, scaling these solutions to long-horizon problems remains challenging [157].

Recently, POMDPs have been used to model large-scale information gathering problems such as carbon capture and storage (CCS) [37, 39], remediation for groundwater contamination [158], and critical mineral exploration for battery metals [8], and are solved using online tree search methods such as DESPOT [159] and POMCPOW [61]. The performance of these online methods relies on heuristics for action selection (to reduce search tree expansion) and heuristics to estimate the value function (to avoid expensive rollouts and reduce tree search depth, illustrated in figure 6.1). Without heuristics, online methods have difficulty planning for long-term information acquisition to reason about uncertain future events. Thus, algorithms to solve high-dimensional POMDPs need to be designed to learn heuristic approximations to enable decision making in long-horizon problems.

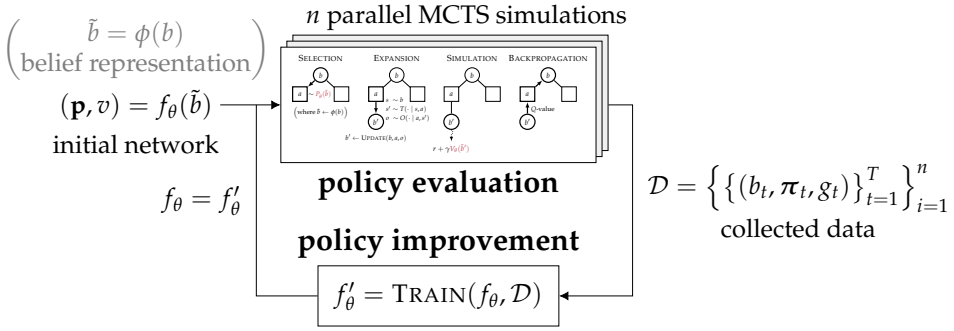


Figure 6.2: The *BetaZero* POMDP policy iteration algorithm.

6.2 Proposed Algorithm: BetaZero

In this chapter, we propose a policy iteration algorithm for POMDPs with the goal to reduce both breadth and depth of online Monte Carlo tree search (MCTS) planning by learning neural network surrogates for the action selection policy and value function, respectively. We introduce the *BetaZero* POMDP planning algorithm that replaces heuristics with learned approximations of the optimal policy and value function. BetaZero is a belief-space policy iteration algorithm with two *offline* steps (figure 6.2) that learns a network used *online*:

1. **Policy evaluation**: Evaluate the current value and policy network through n parallel episodes of MCTS (figure 6.3) and collect training data: $\mathcal{D} = \left\{ \{(b_t, \boldsymbol{\pi}_t, g_t)\}_{t=1}^{T} \right\}_{i=1}^{n}$
2. **Policy improvement**: Improve the value function and policy by retraining the neural network parameters θ with data from the n_{buffer} most recent MCTS simulations.

The policy vector over actions $\mathbf{p} = P_\theta(\tilde{b}, \cdot)$ and the value $v = V_\theta(\tilde{b})$ are combined into a single network with two output heads $(\mathbf{p}, v) = f_\theta(\tilde{b})$; we refer to P_θ and V_θ separately for convenience. During *policy evaluation*, training data is collected from the outer POMDP loop. The belief b_t and the tree policy $\boldsymbol{\pi}_t$ are collected for each time step t. At the end of each episode, the returns $g_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_i$ are computed from the set of observed rewards for all time steps up to a terminal horizon T. Traditionally, MCTS algorithms use a tree policy $\boldsymbol{\pi}_t$ that is proportional to the root node visit counts of its children actions $\boldsymbol{\pi}_t(b_t, a) \propto N(b_t, a)^{1/\tau}$. The counts are sampled after exponentiating with a temperature τ to encourage exploration but evaluated online with $\tau \to 0$ to return the maximizing action [65]. In certain settings, relying solely on visit counts may overlook crucial information (see ablation in figure 6.4). Given the collected data, the *policy improvement* step is intended to imitate the MCTS policy.

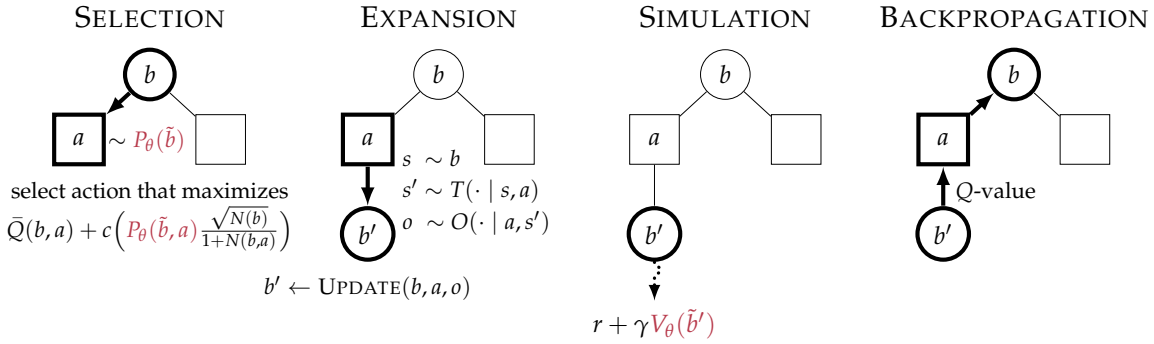


Figure 6.3: The four stages of BetaZero's belief-state MCTS with neural network surrogates.

Figure 6.3 shows the four stages of MCTS belief-state planning used in BetaZero. The learned action selection policy P_θ is used to sample new actions (if expanding on new actions according to the progressive widening procedure [66]) and to weigh the action selection criteria to emphasize promising actions as indicated by the learned policy. The value V_θ is used to avoid expensive rollouts and directly estimate the future value given an updated belief. This version of MCTS is used during the *policy evaluation* step in figure 6.2.

6.2.1 Policy Vector as Q-Weighted Counts

When planning in belief space, expensive belief updates occur in the tree search and thus may limit the MCTS budget. Therefore, the visit counts may not converge towards an optimal strategy as the budget may be spent on exploration. Danihelka *et al.* [160] and Czech *et al.* [161] suggest using knowledge of the Q-values from search in MCTS action selection. Using only tree information, we incorporate Q-values and train against the policy:

$$\pi_t(b_t, a) \propto \left(\left(\frac{\exp Q(b_t,a)}{\sum_{a'} \exp Q(b_t,a')}\right)^{z_q}\left(\frac{N(b_t,a)}{\sum_{a'} N(b_t,a')}\right)^{z_n}\right)^{1/\tau} \tag{6.1}$$

which is then normalized to get a valid probability distribution. Equation (6.1) simply weights the visit counts by the softmax Q-value distribution with parameters $z_q \in [0,1]$ and $z_n \in [0,1]$ defining the influence of the values and the visit counts, respectively. If $z_q = z_n = 1$, then the influence is equal and if $z_q = z_n = 0$, then the policy becomes uniform. Once the tree search finishes, the root node action is selected from $a \sim \pi_t(b_t, \cdot)$ and returns the argmax when the temperature $\tau \to 0$.

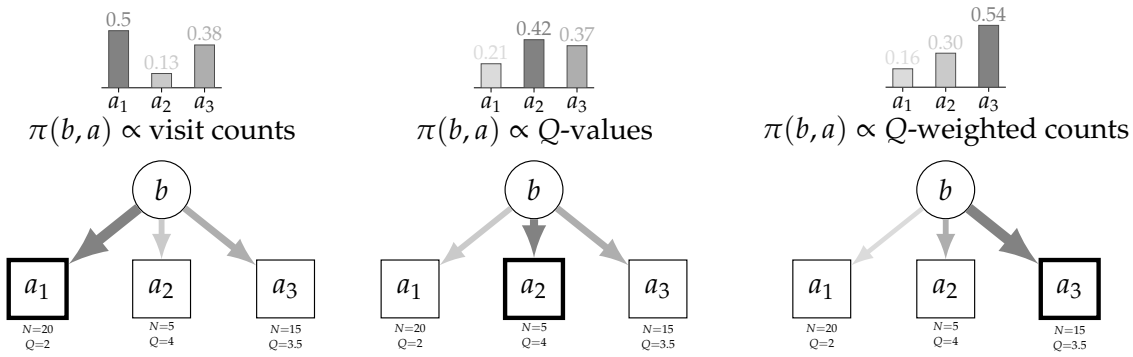


Figure 6.4: Root node imitation policies: Visit counts, Q-values, and Q-weighted visit counts.

Figure 6.4 shows an illustrative example of when collecting policy data based purely on visit counts (left) or Q-values (middle) would perform worse than weighting the visit counts based on Q-values (right). This is useful when using a small MCTS budget with high exploration. Using both the visit counts *and* Q-values, we incorporate both what the tree search *focused on* and the *values it found*. Quantitative analysis is provided in section 6.5.1.

6.2.2 BetaZero Policy Improvement Loss Function

Using the latest collected data, the *policy improvement* step retrains the policy network head using the cross-entropy loss $\mathcal{L}_{P_\theta}(\boldsymbol{\pi}_t, \mathbf{p}_t) = -\boldsymbol{\pi}_t^\top \log \mathbf{p}_t$. The value network head is simultaneously trained to fit the returns g_t using mean-squared error (MSE) or mean-absolute error (MAE) to predict the value of the belief b_t. Note that we use either MSE or MAE value losses $\mathcal{L}_{V_\theta}$ for different problems depending on the characteristics of the return distribution. In sparse reward problems, MAE is a better choice as the distribution is closer to Laplacian [162]. When the reward is distributed closer to Gaussian, then MSE is more suitable [163]. The final loss combines the value and policy losses with L_2-regularization:

$$\mathcal{L}_{\beta_0} = \mathcal{L}_{V_\theta}(g_t, v_t) + \mathcal{L}_{P_\theta}(\boldsymbol{\pi}_t, \mathbf{p}_t) + \lambda \|\theta\|^2 \tag{6.2}$$

6.2.3 Prioritized Action Widening

Explicitly planning in belief space handles state uncertainty but may incur computational overhead when performing belief updates; therefore, we avoid trying all actions at every belief node. We apply *action progressive widening* [66] to limit action expansion, which has been used in the context of continuous and large discrete action spaces [164, 165]. Browne

et al. [59] found action progressive widening to be effective in cases where favorable actions were tried first and Mern *et al.* [150] show that prioritizing actions can improve MCTS performance in large discrete action spaces. Therefore, BetaZero selects actions through progressive widening and uses information from the learned policy network to sample new actions $a \sim P_\theta(\tilde{b}, \cdot)$, as seen on line 4 in algorithm 6.1. This way, we first focus the expansion on *promising actions*, then make the final selection based on PUCT.* In section 6.4, we perform an ablation to measure the effect of using the policy P_θ to prioritize actions when widening the tree compared to uniform sampling of the action space.

Algorithm 6.1: BetaZero action progressive widening.

1 **function** ACTIONSELECTION(f_θ, b)
2 $\quad\tilde{b} \leftarrow \phi(b)$ ▷ belief representation
3 $\quad$**if** $|A(b)| \leq k_a N(b)^{\alpha_a}$ ▷ action progressive widening
4 $\quad\quad a \sim P_\theta(\tilde{b}, \cdot)$ ▷ prioritized from network
5 $\quad\quad N(b,a) \leftarrow N_0(b,a)$
6 $\quad\quad Q(b,a) \leftarrow Q_0(b,a)$ ▷ bootstrap initial Q-value
7 $\quad\quad A(b) \leftarrow A(b) \cup \{a\}$ ▷ add to visited actions $A(b)$
8 $\quad$**return** $\arg\max_{a \in A(b)} \bar{Q}(b,a) + c\left(P_\theta(\tilde{b}, a)\frac{\sqrt{N(b)}}{1+N(b,a)}\right)$ ▷ PUCT

6.2.4 Stochastic Belief-State Transitions

A challenge with partially observable domains is handling non-deterministic belief-state transitions in the tree search. The belief-state transition function T_b consists of several stochastic components, and the belief is continuous (being a probability distribution over states). To address this, we use progressive widening from Couëtoux *et al.* [66] (algorithm 6.2). Other methods for state expansion, like *state abstraction refinement* from Sokota *et al.* [67], rely on problem-specific distance metrics between states to perform a nearest neighbor search. Progressive widening avoids problem-specific heuristics by using information only available in the search tree to provide artificially limited belief-state branching. Limited branching is important as the belief updates can be computationally expensive, thus limiting the MCTS search budget in practice.

*PUCT uses normalized Q-values from 0 to 1 ($\bar{Q}$) so c can be problem independent [166].

Algorithm 6.2: BetaZero belief-state progressive widening.

function BELIEFSTATEEXPANSION(b, a)
  if $|B(b,a)| \leq k_b N(b,a)^{\alpha_b}$ ▷ belief progressive widening
    $b' \sim T_b(\cdot \mid b, a)$ ▷ equation (3.4) from chapter 3
    $B(b,a) \leftarrow B(b,a) \cup \{b'\}$ ▷ add to visited beliefs
  else
    $b' \sim B(b,a)$ ▷ sample from belief-states in the tree
  $r \leftarrow R(b,a)$ or $r \leftarrow R(b,a,b')$
  return b', r

6.2.5 Parametric Belief Representation

Inputting state histories into the network is common in the literature, in both the context of MDPs [17] and POMDPs [167]. Using only state information does not generalize to complex POMDPs (seen later in figure 6.16); therefore, a representation of the belief is required. To model beliefs, we focus on the non-parametric *particle set* that can represent a broad range of distributions [52], and Lim *et al.* [168] show that optimality guarantees exist in finite-sample particle-based POMDP approximations. Despite choosing to study particle-based beliefs, our work generalizes well to problems with parametric beliefs.

Although a particle belief is non-parametric, approximating the belief as summary statistics (e.g., mean and std) may capture enough information for value and policy estimation [169]. Approximating the particle set parametrically is easy to implement and computationally inexpensive. We show that the approximation works well across various problems and, unsurprisingly, using only the mean state is inadequate (see ablation figure 6.16). We represent the particle set b parametrically as $\phi(b) = [\mu(b), \sigma(b)]$. BetaZero plans over the full belief b in the tree and only uses the belief representation $\tilde{b} = \phi(b)$ for network evaluations.

We do not depend on the *exact* way in which the belief is represented, so long as it captures state uncertainty. Coquelin *et al.* [169] consider how to represent a particle filter belief as a finite set of features for policy gradient and suggest a mean and covariance approximation, but only consider the class of policies depending on a single feature of the mean. Their work suggests that other features, such as entropy, could also be used. Other algorithms (e.g., FORBES from Chen *et al.* [170]) could instead be used to learn this belief representation. Another example approach could use principle component analysis (PCA) to learn lower-dimensional features for belief representation [171]. Moreover, the $\mathcal{I}$-VAE latent representation from chapter 5 could also be used as the belief representation.

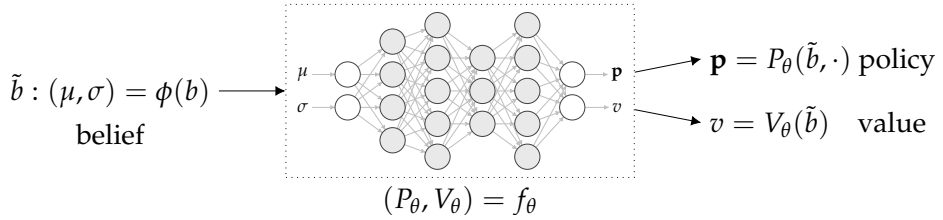


Figure 6.5: BetaZero neural network design.

Figure 6.5 illustrates the neural network design used by BetaZero, where an input belief b is converted to its belief representation $\tilde{b} = \phi(b)$ and passed into the network. The network predicts the probability distribution over all discrete actions and the estimated future value.

6.2.6 Bootstrapping Initial Q-Values

The value network V_θ is used during the *simulation* step to replace rollouts with a network lookup (line 7, alg. 6.6). When a new state-action node is added to the tree, initial Q-values can also use the value network to bootstrap the estimate:

$$Q_0(b, a) \overset{\text{def}}{=} R_b(b, a) + \gamma V_\theta(\phi(b')) \text{ where } b' \sim T_b(\cdot \mid b, a) \tag{6.3}$$

Bootstrapping occurs in algorithm 6.1 (line 6) and incurs an additional belief update through the belief-state transition T_b and may be opted for only during online execution. The bootstrapped estimate is more robust [172] and can be useful to initialize online search.[†]

6.2.7 Full BetaZero Algorithm

Algorithms 6.3 to 6.6 detail the full BetaZero policy iteration algorithm that iterates between *policy evaluation* and *policy improvement* for a total of $n_{\text{iterations}}$. The offline policy evaluation stage, or data collection process (algorithm 6.4), runs n_{data} parallel MCTS simulations over the original POMDP and collects a dataset $\mathcal{D}$ of beliefs b_t, policy vectors $\boldsymbol{\pi}_t$, and returns g_t (computed after each episode terminates). The top-level Q-weighted MCTS algorithm is shown in algorithm 6.5, which iteratively runs MCTS simulations (algorithm 6.6) for n_{online} iterations to a specified depth d. The final root node action selection policy follows the Q-weighted visit counts from equation (6.1). The descriptions of parameters ψ used in offline training and online tree search are listed in tables 6.2 to 6.4.

[†]Note that bootstrapping is also used in the model-free *MuZero* algorithm [166].

Algorithm 6.3: BetaZero offline policy iteration.

Require: $\mathcal{P} \stackrel{\text{def}}{=} \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$: POMDP
Require: ψ: Parameters (includes $n_{\text{iterations}}$, n_{data}, n_{online}, and d)

function BETAZERO($\mathcal{P}, \psi$)
  $f_\theta \leftarrow$ INITIALIZENETWORK(ψ) and let $P_\theta, V_\theta \leftarrow f_\theta$ ▷ where $(\mathbf{p}, v) \leftarrow \left(P_\theta(\tilde{b}), V_\theta(\tilde{b})\right)$
  for $i \leftarrow 1$ **to** $n_{\text{iterations}}$
    $\mathcal{D} \leftarrow$ COLLECTDATA($\mathcal{P}, f_\theta, \psi$) ▷ policy evaluation
    $f_\theta \leftarrow$ TRAIN($f_\theta, \mathcal{D}$) ▷ policy improvement
  return $\pi_{\beta_0}(\mathcal{P}, f_\theta)$ ▷ BetaZero online policy (uses alg. 6.5)

Algorithm 6.4: MCTS data collection for offline policy evaluation.

function COLLECTDATA($\mathcal{P}, f_\theta, \psi$)
  $\mathcal{D} = \emptyset$
  parallel for $i \leftarrow 1$ **to** n_{data} ▷ parallelize MCTS runs across available CPUs
    for $t \leftarrow 1$ **to** T
      $a_t \leftarrow \text{MCTS}(\mathcal{P}, f_\theta, b_t, \psi)$ ▷ select next action through online planning
      $\mathcal{D}_i^{(t)} \leftarrow \mathcal{D}_i^{(t)} \cup \left\{(b_t, \boldsymbol{\pi}_{\text{tree}}^{(t)}, g_t)\right\}$ ▷ collect belief and policy data (returns buffer)
      $s_{t+1} \sim T(\cdot \mid s_t, a_t)$
      $o_t \sim O(\cdot \mid a_t, s_{t+1})$
      $b_{t+1} \leftarrow$ UPDATE(b_t, a_t, o_t)
      $r_t \leftarrow R(s_t, a_t)$ or $R(s_t, a_t, s_{t+1})$
      } transition the original POMDP
    $g_t \leftarrow \sum_{k=t}^{T} \gamma^{(k-t)} r_k$ **for** $t \leftarrow 1$ **to** T ▷ compute returns from observed rewards
  return $\mathcal{D}$

Algorithm 6.5: Monte Carlo tree search algorithm using Q-weighed visit counts.

function MCTS($\mathcal{P}, f_\theta, b, \psi$)
  $\mathcal{M} \leftarrow \langle \mathcal{B}, \mathcal{A}, T_b, R_b, \gamma \rangle$ converted from $\mathcal{P}$ (POMDP) ▷ plan using belief-state MDP
  for $i \leftarrow 1$ **to** n_{online}
    SIMULATE(f_θ, b, d) ▷ MCTS simulated planning to a depth d (algorithm 6.6)
  $\pi_{\text{tree}}(b, a) \propto \left(\left(\frac{\exp Q(b,a)}{\sum_{a'} \exp Q(b,a')}\right)^{z_q} \left(\frac{N(b,a)}{\sum_{a'} N(b,a')}\right)^{z_n}\right)^{1/\tau}$ ▷ Q-weighted counts, eq. 6.1
  $\pi_{\text{tree}}(b, a_i) \leftarrow \pi_{\text{tree}}(b, a_i) \,/\, \sum_j \pi_{\text{tree}}(b, a_j)$ ▷ (normalize) valid probability distr.
  return $a \sim \pi_{\text{tree}}(b, \cdot)$ ▷ sample root node action (let $\tau \to 0$ to get arg max)

Algorithm 6.6: BetaZero MCTS simulation.

function SIMULATE(f_θ, b, d)
  if $d = 0$ **return** 0
  if $b \notin \mathcal{T}$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{b\}$
    $N(b) \leftarrow N_0(b)$
    $\tilde{b} \leftarrow \phi(b)$ ▷ belief representation
    return $V_\theta(\tilde{b})$ ▷ value lookup
  $N(b) \leftarrow N(b) + 1$
  $a \leftarrow$ ACTIONSELECTION(f_θ, b) ▷ selection
  $(b', r) \leftarrow$ BELIEFSTATEEXPANSION(b, a) ▷ expansion
  $q \leftarrow r + \gamma$SIMULATE($f_\theta, b', d-1$) ▷ simulation
  $N(b, a) \leftarrow N(b, a) + 1$
  $Q(b, a) \leftarrow Q(b, a) + \frac{q - Q(b,a)}{N(b,a)}$ ▷ backpropagation
  return q

6.2.8 Complexity Analysis

The runtime complexity of MCTS is $M = \mathcal{O}(ndm)$, where n is the number of MCTS iterations (denoted n_{online} in algorithm 6.6), d is the search depth, and the belief update is over m particles at each belief-state node. The full complexity of BetaZero is $\mathcal{O}(pmTM/c)$ for p parallel runs (denoted n_{data} in algorithm 6.4), an episode horizon of T (each step updating the belief over m particles), the MCTS complexity of M, and the number of CPU cores c.

The memory complexity for MCTS is $E = \mathcal{O}(k^d)$ for $k = |A(b)||B(b,a)|$ where $|B(b,a)|$ is the number of belief-action nodes and $|A(b)|$ is the number of children, which depend on progressive widening parameters. The memory complexity for BetaZero is $\mathcal{O}(TPE|\theta|)$ for the collected data sizes of the belief and returns T (same as the horizon), the policy vector size of $P = |\mathcal{A}|$ (i.e., action space size), the MCTS memory complexity of E, and the network size of $|\theta|$. Compared to standard MCTS applications to belief-state MDPs, BetaZero requires additional memory for data collection and neural network storage.

6.3 Related Work

Algorithms to solve high-dimensional, *fully observable* MDPs have been proposed that learn approximations to replace problem-specific heuristics. Notably, Silver *et al.* [17] introduced

the *AlphaZero* algorithm for large, deterministic MDPs and showed considerable success in games such as Go, chess, shogi, and Atari [17, 166]. The success is attributed to the combination of online MCTS and a neural network that approximates the optimal value function and the offline policy. Extensions of AlphaZero and the model-free variant *MuZero* [166] have already addressed several challenges when applying to broad classes of MDPs. For large or continuous action spaces, Hubert *et al.* [173] introduced a policy improvement algorithm called *Sampled MuZero* that samples an action set of an *a priori* fixed size every time a node is expanded. Antonoglou *et al.* [174] introduced *Stochastic MuZero* that extends MuZero to games with stochastic transitions but assumes a finite set of possible next states so that each transition can be associated with a chance outcome. Applying these algorithms to large or continuous spaces with partial observability remains challenging.

To handle partial observability in stochastic games, Ozair *et al.* [175] combine VQ-VAEs with MuZero to encode future discrete observations into latent variables. Other approaches handle partial observability by inputting action-observation histories directly into the network [176, 177]. Similarly, Igl *et al.* [178] introduce a method to learn a belief representation within the network when the agent is only given access to histories. Their work focuses on the *reinforcement learning* (RL) domain, and they show that a belief distribution can be represented as a latent state in the learned model. The FORBES algorithm [170] builds a normalizing flow-based belief and learns a policy through an actor-critic RL algorithm. Methods to learn the belief are necessary when a prior belief model is not available. When such models *do* exist, as is the case with many POMDPs that we study, using the models can be valuable for long-term planning. Finally, Hoel *et al.* [179] apply AlphaGo Zero [65] to an autonomous driving POMDP using the most-likely state as the network input but overlook significant uncertainty information in the belief.

6.3.1 Online POMDP Planning

Sunberg *et al.* [61] introduced the *POMCPOW* planning algorithm that iteratively builds a particle set belief within the tree, designed for fully continuous spaces. In practice, POMCPOW relies on heuristics for value function estimation and action selection (e.g., work from Mern *et al.* [8]). Wu *et al.* [180] introduced *AdaOPS* that adaptively approximates the belief through particle filtering and maintains value function bounds that are initialized with heuristics (e.g., solving the MDP or using expert policies). The major limitation of existing solvers is the reliance on heuristics to make long-horizon POMDPs tractable, which

may not scale to high-dimensional problems. Cai *et al.* [167] proposed *LeTS-Drive* applied to autonomous driving that combines planning and learning similar to BetaZero, and uses HyP-DESPOT with PUCT exploration [181] as the planning algorithm, instead of MCTS. It uses a state-history window as input to the network, which may not adequately capture the state uncertainty. LeTS-Drive expands on all actions during planning, which we show may lead to suboptimal planning under limited search budgets (shown in figures 6.17b and 6.18b). To handle long-horizon POMDPs, Mazzi *et al.* [182] propose learning logic-based rules as policy guidance in POMCP, yet domain-specific knowledge is required to define the set of features for the rules, which may not be easily generalized to complex POMDPs we study in this work. Therefore, we identified the need for a general POMDP planning algorithm that does not rely on problem-specific heuristics for good performance.

6.4 Experiments

Three benchmark problems were chosen to evaluate the performance of BetaZero against several baselines: AdaOPS [180], POMCPOW [61], DESPOT [159], and LeTS-Drive [167]. Table 6.1 details the POMDP space sizes and this section describes the experimental design.

6.4.1 POMDP Environments

In LightDark(y) from Platt Jr. *et al.* [30], the goal of the agent is to execute a `stop` action at the origin while receiving noisy observations of its true location. The noise is minimized in the *light* region $y = 5$. We also benchmark against a more challenging version with the light region at $y = 10$ from Sunberg *et al.* [61], and restrict the agent to only three actions: move `up` or `down` by one, or `stop`. The modified problem requires information gathering over longer horizons. Next is the RockSample(n, k) POMDP [31], which is a scalable information gathering problem where an agent moves in an $n \times n$ grid to observe k rocks with an

	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{O}\rvert$
LightDark(5 and 10)	$\lvert\mathbb{R}\rvert$	3	$\lvert\mathbb{R}\rvert$
RockSample(15, 15)	7,372,800	20	3
RockSample(20, 20)	419,430,400	25	3
Mineral Exploration	$\lvert\mathbb{R}^{32\times32}\rvert$	38	$\lvert\mathbb{R}_{\geq0}\rvert$

Table 6.1: POMDP state, action, and observation space dimensions.

objective to sample only the *good* rocks. Well-established POMDP benchmarks go up to $n = 15$ and $k = 15$; we also test a harder version with $n = 20$ and $k = 20$ to show the scalability of BetaZero, noting that this case has been evaluated in the multi-agent setting [181]. Finally, in the real-world MINERAL EXPLORATION problem [8], the agent drills over a 32×32 region to determine if a subsurface ore body should be mined or abandoned and the continuous ore quality is observed at the drill locations to build a belief. Drilling incurs a penalty, and if chosen to mine, then the agent is rewarded or penalized based on an economic threshold of the extracted ore mass. The problem is challenging due to reasoning over limited observations with sparse rewards. In the following sections, we provide additional details regarding each environment.

Light dark. The LIGHTDARK(y) POMDP is a one-dimensional localization problem [30]. The objective is for the agent to execute the `stop` action at the goal, which is at ± 1 of the origin. The agent is awarded 100 for stopping at the goal and -100 for stopping anywhere else, using a discount of $\gamma = 0.9$. The agent receives noisy observations of their position, where the noise is minimized in the *light* region defined by y. In the LIGHTDARK(5) problem used by Wu *et al.* [180], the noise is a zero-mean Gaussian with standard deviation of $|y - 5|/\sqrt{2} + 10^{-2}$. For the LIGHTDARK$(10)$ problem used by Sunberg *et al.* [61], the noise is a zero-mean Gaussian with standard deviation of $|y - 10| + 10^{-4}$. In both problems, we use a restricted action space of $\mathcal{A} = \{-1, 0, 1\}$ where 0 is the `stop` action. The expected behavior of the optimal policy is to first localize in the light region, then travel down to the goal. The BetaZero policy exhibits this behavior which can be seen in figure 6.6, where BetaZero (dark blue) learned to first localize in the light region at $y = 10$ before heading to the goal (at the origin, where circles indicate the final location).

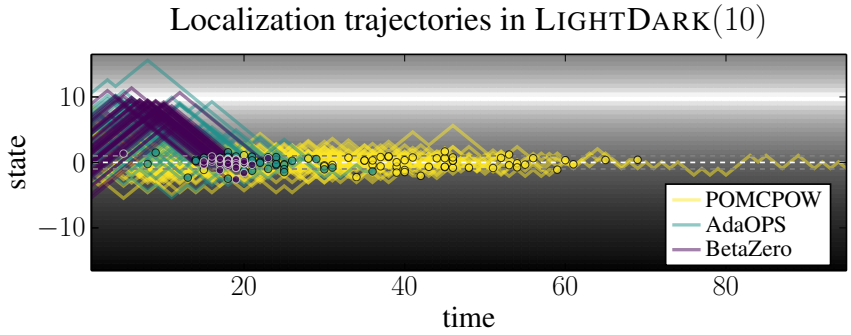


Figure 6.6: LIGHTDARK(10) trajectories from 50 episodes.

The approximately optimal solution to the light dark problems uses *local approximation value iteration* (LAVI) [58] over the discretized belief-state space (i.e., mean and std). The belief mean was discretized between the range $[-12, 12]$, and the belief std was discretized between the range $[0, 5]$, each of length 100. The LAVI solver used 100 generative samples per belief state and ran for 100 value iterations with a Bellman residual of 1×10^{-3}.

Rock sample. In the RockSample(n, k) POMDP introduced by Smith *et al.* [31], an agent has full observability of its position on an $n \times n$ grid but has to sense the k rocks to determine if they are *good* or *bad*. The agent knows *a priori* the true locations of the rocks (i.e., the rock locations $\mathbf{x}_{\text{rock}}$ are a part of the problem, not the state). The observation noise is a function of the distance to the rock:

$$\frac{1}{2}\left(1 + \exp\left(-\frac{\|\mathbf{x}_{\text{rock}} - \mathbf{x}_{\text{agent}}\|_2 \log(2)}{c}\right)\right) \tag{6.4}$$

where $c = 20$ is the sensor efficiency. The agent can move in the four cardinal directions, sense the k rocks, or take the action to `sample` a rock when it is located under the agent. The agent receives a reward of 10 for sampling a *good* rock and a penalty of -10 for sampling a *bad* rock. The terminal state is the exit at the right edge of the map, where the agent gets a reward of 10 for exiting.

Mineral exploration. The Mineral Exploration POMDP introduced by Mern *et al.* [8] is an information gathering problem with the goal of deciding whether a subsurface ore body is economical to mine or should be abandoned (calibrated so that 50% of cases are economical). The agent can drill every fifth cell of a 32×32 plot of land to determine the ore quality at that location. Therefore, the action space consists of the 36 drill locations and the final decisions to either `mine` or `abandon`. The agent receives a small cost for each `drill` action, a reward proportional to the extracted ore if it chooses to `mine` (which is negative if uneconomical), and a reward of zero if it chooses to `abandon`:

$$R(s, a) = \begin{cases} -c_{\text{drill}} & \text{if } a = \texttt{drill} \\ \sum \mathbb{1}\{s_{\text{ore}} \geq h_{\text{massive}}\} - c_{\text{extract}} & \text{if } a = \texttt{mine} \\ 0 & \text{if } a = \texttt{abandon} \end{cases} \tag{6.5}$$

where $c_{\text{drill}} = 0.1$, $h_{\text{massive}} = 0.7$, and $c_{\text{extract}} = 71$. The term $\sum \mathbb{1}(s_{\text{ore}} \geq h_{\text{massive}})$ indicates the cells that have an ore quality value above some massive ore threshold h_{massive} (which are deemed valuable). Figure 6.7 and figure 6.8 show an example of four steps of the mineral exploration POMDP.

6.4.2 Baseline Heuristics

Experiment parameters for each problem can be seen in tables 6.2 to 6.4 under the ''online'' column. For the baseline algorithms, the heuristics follow Wu *et al.* [180]. Problems that failed to run due to memory limits followed suggestions from Wu *et al.* [183] to first use the MDP solution and then use a fixed upper bound of $r_{\text{correct}} = 100$ for the light dark problems and the following for the rock sample problems:

$$V_{\max} = r_{\text{exit}} + \sum_{t=1+n-k}^{2k-n} \gamma^{t-1} r_{\text{good}} \tag{6.6}$$

where $r_{\text{good}} = r_{\text{exit}} = 10$ and the sum computes an optimistic value assuming the rocks are directly lined between the agent and the goal and assuming $n \geq k$ for simplicity.

For problems not studied by Wu *et al.* [180], we use the same heuristics as their easier counterpart (i.e., LightDark(10) uses LightDark(5) heuristics and RockSample(20, 20) uses RockSample(15, 15) heuristics). For mineral exploration, the baselines used the following heuristics. POMCPOW used a value estimator of $\max(0, R(s, a = \text{mine}))$ and when using ''no heuristic'' used a random rollout policy to estimate the value. Both AdaOPS and DESPOT used a lower bound computed as the returns if all locations were fully drilled, then made the decision to `abandon`:

$$V_{\min} = -\sum_{t=1}^{T-1} \gamma^{t-1} c_{\text{drill}} \tag{6.7}$$

The upper bound comes from an oracle π_{truth} taking the correct final action without drilling, computed over 10,000 states. Note that there is no state transition in this problem:

$$V_{\max} = \mathop{\mathbb{E}}_{s \in \mathcal{S}} \left[\max \left(0, R\left(s, \pi_{\text{truth}}(s)\right) \right) \right] \tag{6.8}$$

$$\approx \frac{1}{n} \sum_{i=1}^{n} \max \left(0, R\left(s^{(i)}, \pi_{\text{truth}}(s)\right) \right) \tag{6.9}$$

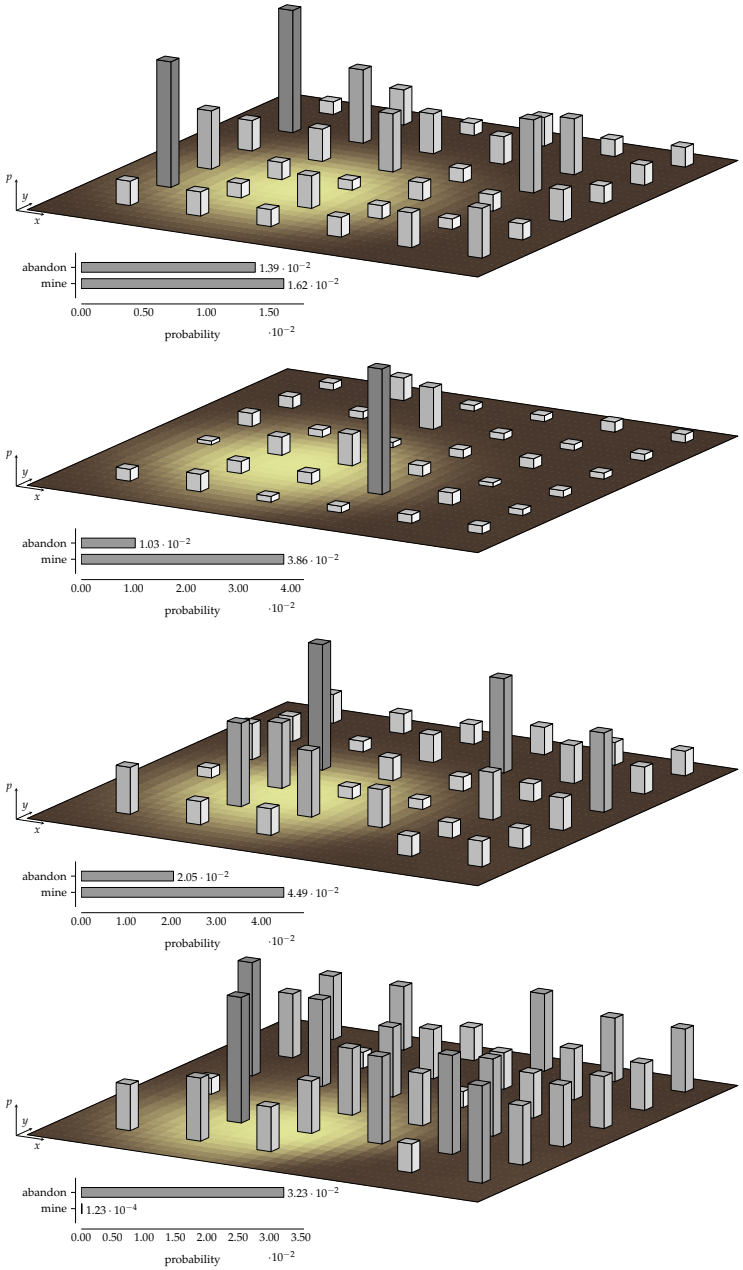


Figure 6.7: The BetaZero policy shown over the mean belief.

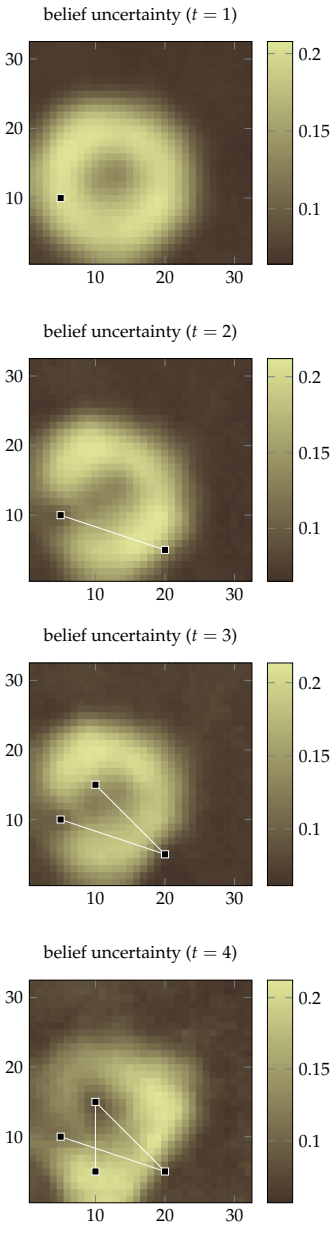


Figure 6.8: Uncertainty.

Visualized in figure 6.7, the BetaZero policy is shown over the belief mean for four steps of the MINERAL EXPLORATION POMDP. BetaZero first prioritizes the edges of the belief mean, corresponding to the belief uncertainty (shown in the figure 6.8 plots), then explores the outer regions of the subsurface; ultimately gathering information from actions with high mean and standard deviation, matching heuristics from Mern *et al.* [8]. At the initial step,

abandoning and mining have near-equal probability (bottom left bar graphs in figure 6.7) but by the fourth action, abandoning is much more likely. Figure 6.8 visualizes the selected drill actions over the belief uncertainty, showing that uncertainty collapses after drilling.

6.4.3 Particle Filter for Belief Updating

Both BetaZero and the baselines update their belief with a bootstrap particle filter using a low-variance resampler [51], with $n_{\text{particles}} \in [500, 1000, 1000]$ for the light dark, rock sample, and mineral exploration problems, respectively. The particle filter follows an update procedure of first reweighting then resampling. In mineral exploration, the observations are noiseless which could quickly result in particle depletion. Therefore, approximate Bayesian computation (ABC) [184] is used to reweight each particle using a Gaussian distribution centered at the observation with a standard deviation of $\sigma_{\text{abc}} = 0.1$.

The belief representation takes the mean and standard deviation across the $n_{\text{particles}}$. In the light dark problems, this is computed across the 500 sampled y-state values that make up the belief. The initial y-value state distribution (i.e., the initial belief) follows a Gaussian distribution, and thus the parametric representation is a good approximation of the belief.

For the rock sample problem, the belief is represented as the mean and standard deviation of the *good* rocks from the 1000 sampled states (appending the true position as it is deterministic). The rock qualities are sampled uniformly in $\{0, 1\}$ indicating if they are *good*, which makes the problem non-Gaussian, but the parametric belief approximation can model a uniform distribution by placing the mean at the center of the uniform range and stretching the variance to match the uniform.

Lastly, the mineral exploration problem flattens the 1000 subsurface 32×32 maps, each of which has associated per-pixel ore quality between $[0, 1]$, into two images: a mean and standard deviation image of the ore quality that is stacked and used as input to a convolutional neural network (CNN). The initial state distribution for the massive ore quantity closely follows a Gaussian, making the parametric belief approximation well suited.

For problems where Gaussian approximations are not well-suited to capture the belief, the parameters of other distributions could be used as a belief representation or the particles themselves could be input into a network—first passing through an order-invariant layer [178]. Scaling to larger observation spaces will not be an issue as BetaZero plans over belief states instead of observations.

6.4.4 Network Architectures

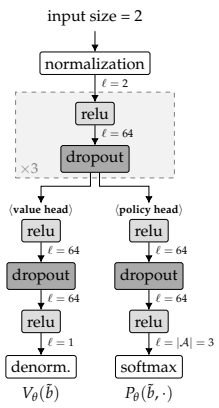


Figure 6.9: Light dark NN.

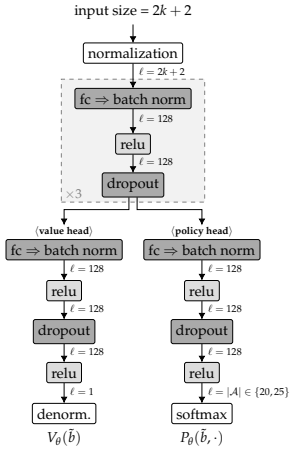


Figure 6.10: Rock sample NN.

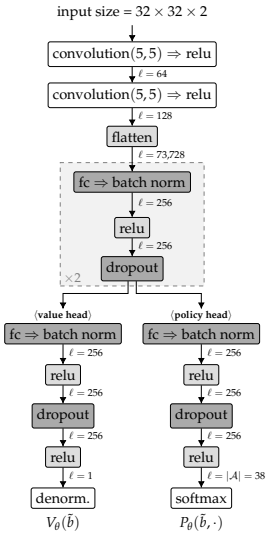


Figure 6.11: MinEx. CNN.

Figures 6.9 to 6.11 specify the neural network architectures for the three problem domains. The networks were designed to be simple so that future work could focus on incorporating more complicated architectures such as residual networks [177, 185]. Mineral exploration does not normalize the inputs and is the only problem where the input is treated as an image; thus, we use a CNN. Training occurs on normalized returns and an output denormalization layer is added to the value head to ensure proper magnitude of the predicted values.

6.4.5 Return Scaling for Output Normalization

For general POMDPs, the return can be an unbounded real-value and not conveniently in $[0, 1]$ or $[-1, 1]$, as is often the case with two-player games. Schrittwieser *et al.* [166] use a categorical representation of the value split into a discrete support to make learning more robust [186]. We instead simply normalize the target before training as:

$$\bar{g}_t = \frac{g_t - \mathbb{E}[G_{\text{train}}]}{\sqrt{\text{Var}[G_{\text{train}}]}} \tag{6.10}$$

where G_{train} is the set of returns used during training; keeping running statistics of all training data. Intuitively, this ensures that the target values have zero mean and unit variance which

is known to stabilize training [187]. After training, a denormalization layer is added to the normalized output $\bar{v}$ of the value network as:

$$v_t = \bar{v}\sqrt{\text{Var}[G_{\text{train}}]} + \mathbb{E}[G_{\text{train}}] \tag{6.11}$$

to properly scale value predictions when the network is evaluated. This scaling is done entirely internal to the network as shown in figures 6.9 to 6.11.

6.4.6 Hyperparameters and Tuning

The hyperparameters used during offline training and online execution are described in tables 6.2 to 6.4. Offline training refers to the BetaZero policy iteration steps that collect parallel MCTS data (*policy evaluation*) and then retrain the network (*policy improvement*). Online execution refers to using the BetaZero policy after offline training to evaluate its performance through additional online tree search. The main difference between these two settings is the final criteria used to select the root node action in MCTS. During offline training of problems with large action spaces (e.g., rock sample and mineral exploration), sampling root node actions according to the Q-weighted visit counts with a temperature τ ensures exploration. To evaluate the performance online, root node action selection takes the maximizing action of the Q-weighted visit counts. During training, we also evaluate a holdout set that uses the arg max criteria to monitor the performance of the learned policy.

	Parameter*	LightDark(5)		LightDark(10)		Description
		Offline	Online	Offline	Online	
BetaZero policy iteration parameters	$n_{\text{iterations}}$	30	—	30	—	Number of offline BetaZero policy iterations.
	n_{data}	500	—	500	—	Number of parallel MCTS data gen. episodes per policy iteration.
	bootstrap Q_0	`false`	`false`	`false`	`false`	Use bootstrap estimate for initial Q-value in MCTS?
Neural network parameters	n_{epochs}	50	—	50	—	Number of training epochs.
	α	1×10^{-4}	—	1×10^{-4}	—	Learning rate.
	λ	1×10^{-5}	—	1×10^{-5}	—	L_2-regularization parameter.
MCTS parameters	n_{online}	100	1300	100	1000	Number of tree search iterations of MCTS.
	c	1	1	1	1	PUCT exploration constant.
	k_a	2.0	2.0	2.0	2.0	Multiplicative action progressive widening value.
	α_a	0.25	0.25	0.25	0.25	Exponential action progressive widening value.
	k_b	2.0	2.0	2.0	2.0	Multiplicative belief-state progressive widening value.
	α_b	0.1	0.1	0.1	0.1	Exponential belief-state progressive widening value.
	d	10	10	10	10	Maximum tree depth.
	τ	0	0	0	0	Exploration temperature for final root node action selection.
	z_q	1	1	1	1	Influence of Q-values in final criteria.
	z_n	1	1	1	1	Influence of visit counts in final criteria.

* Entries with "—" denote non-applicability and "·" denotes they are disabled.

Table 6.2: *BetaZero* parameters for the LightDark problems.

	Parameter	RockSample(15, 15)		RockSample(20, 20)		Description
		Offline	Online	Offline	Online	
BetaZero policy iteration parameters	$n_{\text{iterations}}$	50	—	50	—	Number of offline BetaZero policy iterations.
	n_{data}	500	—	500	—	Number of parallel MCTS data gen. episodes per policy iteration.
	bootstrap Q_0	`false`	`true`	`false`	`true`	Use bootstrap estimate for initial Q-value in MCTS?
Neural network parameters	n_{epochs}	10	—	10	—	Number of training epochs.
	α	1×10^{-3}	—	1×10^{-3}	—	Learning rate.
	λ	1×10^{-5}	—	1×10^{-5}	—	L_2-regularization parameter.
MCTS parameters	n_{online}	100	100	100	100	Number of tree search iterations of MCTS.
	c	50	50	50	50	PUCT exploration constant.
	k_a	·	5.0	·	·	Multiplicative action progressive widening value.
	α_a	·	0.9	·	·	Exponential action progressive widening value.
	k_b	·	1.0	1.0	1.0	Multiplicative belief-state progressive widening value.
	α_b	·	0.0	0.0	0.0	Exponential belief-state progressive widening value.
	d	15	15	4	4	Maximum tree depth.
	τ	1.0	0	1.5	0	Exploration temperature for final root node action selection.
	z_q	1	0.4	1	0.5	Influence of Q-values in final criteria.
	z_n	1	0.9	1	0.8	Influence of visit counts in final criteria.

Table 6.3: *BetaZero* parameters for the RockSample problems.

	Parameter	Offline	Online	Description
BetaZero policy iteration parameters	$n_{\text{iterations}}$	20	—	Number of offline BetaZero policy iterations.
	n_{data}	100	—	Number of parallel MCTS data gen. episodes per policy iteration.
	bootstrap Q_0	`false`	`false`	Use bootstrap estimate for initial Q-value in MCTS?
Neural network parameters	n_{epochs}	10	—	Number of training epochs.
	α	1×10^{-6}	—	Learning rate.
	λ	1×10^{-4}	—	L_2-regularization parameter.
MCTS parameters	n_{online}	50	50	Number of tree search iterations of MCTS.
	c	57	57	PUCT exploration constant.
	k_a	41.09	41.09	Multiplicative action progressive widening value.
	α_a	0.57	0.57	Exponential action progressive widening value.
	k_b	37.13	37.13	Multiplicative belief-state progressive widening value.
	α_b	0.94	0.94	Exponential belief-state progressive widening value.
	d	5	5	Maximum tree depth.
	τ	1.0	0	Exploration temperature for final root node action selection.
	z_q	1	1	Influence of Q-values in final criteria.
	z_n	1	1	Influence of visit counts in final criteria.

Table 6.4: *BetaZero* parameters for the Mineral Exploration problem.

The MCTS parameters for mineral exploration were tuned using Latin hypercube sampling based on the lower-confidence bound of the returns. During rock sample training, DPW was disabled to expand on all actions and transition to a single belief. Then for online execution, we tuned the DPW parameters as shown in figures 6.17 and 6.18. The problems train with a batch size of 1024 over 80% of 100,000 samples from one round of data collection ($n_{\text{buffer}} = 1$) using p_{dropout} of 0.2, 0.5, 0.7, respectively. The neural network optimizer Adam [76] was used in LightDark(y), while RMSProp [77] was used in the others. A value function loss of MAE was used in mineral exploration; otherwise, MSE was used.

6.4.7 Compute Resources

BetaZero was designed to use a single GPU to train the network and parallelize MCTS evaluations across available CPUs. Evaluating the networks on the CPU is computationally inexpensive due to the size of the networks (shown in figures 6.9 to 6.11). This design was chosen to enable future research without a computational bottleneck. For network training, a single NVIDIA A100 was used with 80GB of memory on an Ubuntu 22.04 machine with 500 GB of RAM. Parallel data collection processes were run on 50 processes split evenly over two separate Ubuntu 22.04 machines: (1) with 40 Intel Xeon 2.3 GHz CPUs, and (2) with 56 Intel Xeon 2.6 GHz CPUs. Algorithm 6.4 (line 3) shows where CPU parallelization occurs. In practice, the MCTS data generation simulations are the bottleneck of the offline component of BetaZero and not the network training—thus, parallelization is useful.

6.4.8 Baseline Algorithms

We baseline BetaZero against several online POMDP algorithms: AdaOPS, POMCPOW, DESPOT, and LeTS-Drive (HyP-DESPOT with a learned network). In LightDark, we solve for an approximately optimal policy using *local approximation value iteration* (LAVI) [58] over a discretized parametric belief space, and for mineral exploration, the value estimates come from privileged information. For a fair comparison, parameters were set to match the total number of simulations of about one million per algorithm.

	LightDark(5)		LightDark(10)		RockSample(15, 15)		RockSample(20, 20)		Mineral Exploration	
	returns	time [*off*,*on*] s	returns	time [*off*,*on*] s	returns	time [*off*,*on*] s	returns	time [*off*,*on*] s	returns	time [*off*,*on*] s
BetaZero	**4.47 ± 0.28**	[2274, 0.014]	**16.77 ± 1.28**	[2740, 0.331]	20.15 ± 0.71	[5701, 0.477]	**13.09 ± 0.55**	[7081, 1.109]	**10.67 ± 2.25**	[22505, 5.126]
Raw Policy P_θ	4.44 ± 0.28	[2274, 0.004]	13.74 ± 1.33	[2740, 0.004]	10.96 ± 0.98	[5701, 0.018]	2.03 ± 0.34	[7081, 0.084]	8.67 ± 2.52	[22505, 0.533]
Raw Value V_θ*	3.16 ± 0.40	[2274, 0.008]	12.70 ± 1.46	[2740, 0.009]	9.96 ± 0.65	[5701, 0.158]	3.57 ± 0.40	[7081, 0.204]	9.75 ± 2.42	[22505, 1.420]
AdaOPS	3.78 ± 0.27 (3.79 ± 0.07)	[68, 0.089]	5.22 ± 1.77	[81, 0.510]	**20.67 ± 0.72** (17.16 ± 0.21)	[7, 2.768]	—	—	3.33 ± 1.95	[5, 0.112]
AdaOPS (fixed bounds)	3.70 ± 0.25	[0, 0.039]	4.98 ± 2.01	[0, 0.573]	13.37 ± 0.71	[0, 1.349]	11.66 ± 0.49	[1, 1.458]	"	"
POMCPOW	3.21 ± 0.38 (3.23 ± 0.11)	[59, 0.189]	0.68 ± 0.41	[70, 1.261]	11.14 ± 0.59 (10.40 ± 0.18)	[0, 0.929]	10.22 ± 0.47	[0, 1.480]	9.43 ± 2.19	[0, 6.728]
POMCPOW (no heuristics)	1.96 ± 0.58	[0, 0.099]	−5.90 ± 5.78	[0, 0.742]	10.17 ± 0.61	[0, 1.485]	4.03 ± 0.44	[0, 5.173]	5.38 ± 2.15	[0, 5.915]
DESPOT	2.37 ± 0.37 (2.50 ± 0.10)	[0, 0.008]	0.43 ± 0.36	[0, 0.046]	18.44 ± 0.69 (15.67 ± 0.20)	[7, 3.822]	—	—	5.29 ± 2.17	[5, 0.283]
DESPOT (fixed bounds)	2.70 ± 0.50	[0, 0.008]	0.49 ± 0.30	[0, 0.025]	4.29 ± 0.45	[0, 5.091]	0.00	[0, 5.179]	"	"
LeTS-Drive (HyP-DESPOT + f_θ)	3.05 ± 0.25	[1260, 0.019]	4.08 ± 5.48	[1529, 0.058]	11.22 ± 0.27	[48064, 1.576]	9.68 ± 0.25	[63850, 2.018]	3.17 ± 2.04	[11738, 4.613]
Approximately Optimal	4.06 ± 0.31	[18359, 0.094]	15.04 ± 1.27	[19548, 0.024]	—	—	—	—	11.90 ± 0.18	N/A

* One-step look-ahead over all actions using only the value network with 5 observations per action.
Entries with ''—'' failed to run, " are the same as the ones above, and entries in (parentheses) are from the literature.

Table 6.5: Results comparing *BetaZero* to various state-of-the-art POMDP solvers.

6.5 Empirical Results and Analysis

Table 6.5 shows that BetaZero outperforms state-of-the-art algorithms in most cases, with larger improvements when baseline algorithms do not rely on heuristics. We report the mean return and standard error over 100 seeds, and the [*offline*, *online*] timing in seconds. While BetaZero has a large offline timing component, similar to LeTS-Drive, it is significantly less than solving for the approximately optimal policy. Figure 6.12 compares the raw BetaZero value and policy network with LAVI for LightDark(10). The plots are shown over the belief space (i.e., mean and std). High uncertainty (horizontal axis) makes the agent localize `up` near $y = 10$; it then moves `down` and `stops` at the origin. Qualitatively, BetaZero learns an approximately optimal policy and value function in areas where training data was collected. Areas where BetaZero and the LVAI policy diverge may result from a lack of training data in those regions (top right corners of the policy plots). Despite this, BetaZero remains nearly optimal as those beliefs do not occur during execution. Out-of-distribution methods could quantify this uncertainty, e.g., an ensemble of networks [188].

In RockSample(15, 15), BetaZero is comparable to AdaOPS yet scales better to higher dimensional problems. AdaOPS computes an upper bound using QMDP [156] to find the optimal utility of the fully observable MDP over all $k - 1$ rock combinations, which scales exponentially in n. In problems with larger state spaces, like RockSample(20, 20), the QMDP solution is intractable. Thus, AdaOPS uses fixed bounds assuming an optimistic $V_{\max}$ from

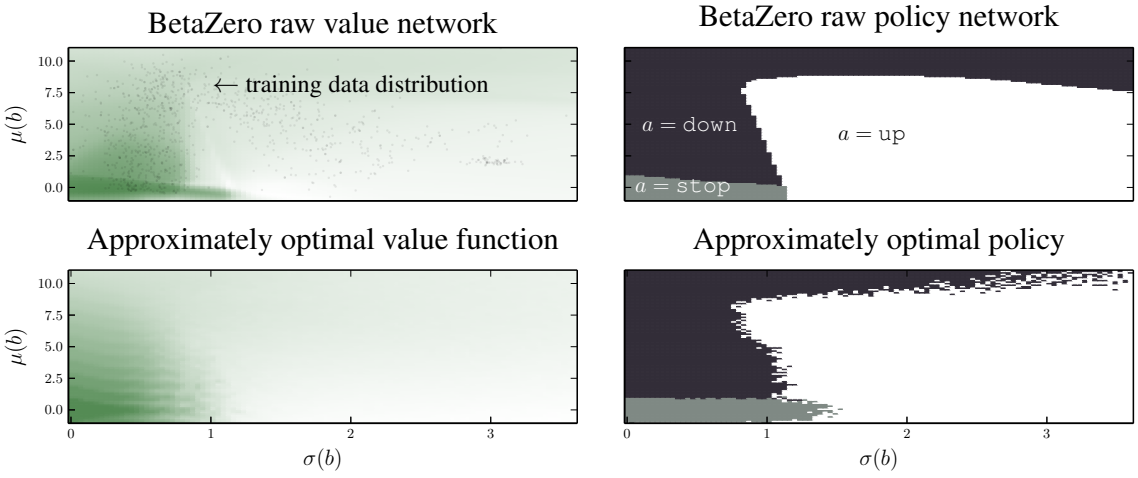


Figure 6.12: LightDark(10) value and policy plots compared to the approximately optimal.

Wu *et al.* [183]. Indicated in table 6.5, the raw networks alone perform well but outperform when combined with online planning, enabling reasoning with current information.

Additional online planning. If online algorithms ran for a large number of iterations, one might expect to see convergence to the optimal policy. In practice, this may be an intractable number as figure 6.13 shows POMCPOW has not reached the required number of iterations for RockSample(15, 15). Performance of POMCPOW with heuristics up to 10 million *online* iterations plateaus, indicating that extending online searches alone misses valuable *offline* experience. The advantage of BetaZero is that it can generalize from a more diverse set of experiences. The inability of existing online algorithms to plan over long horizons is also evident in the mineral exploration POMDP. In figure 6.14, BetaZero prioritizes uncertainty, matching heuristics from Mern *et al.* [8] (i.e., selects actions with high value and uncertainty, shown in yellow). POMCPOW ran for one million online iterations without a value estimator heuristic and BetaZero ran online for 100 iterations (using about 850,000 offline simulations). In the figure, the probability of selecting a drilling location is shown as vertical bars for each action, overlaid on the initial belief uncertainty (i.e., the std of the belief in subsurface ore quality). BetaZero learned to take actions in areas of the belief space with high uncertainty and high value (which matches the domain-specific heuristics developed by Mern *et al.* [8]), while POMCPOW fails to distinguish between the actions and resembles a uniform policy.

The most closely related algorithm, LeTS-Drive [167], which also includes an offline learning component with online tree search planning, performs better than its DESPOT counterpart without the use of offline heuristics. This is observed in all studied POMDPs except for the mineral exploration problem where the DESPOT bounds use privileged information from the approximately optimal bounds on the value function. Table 6.5

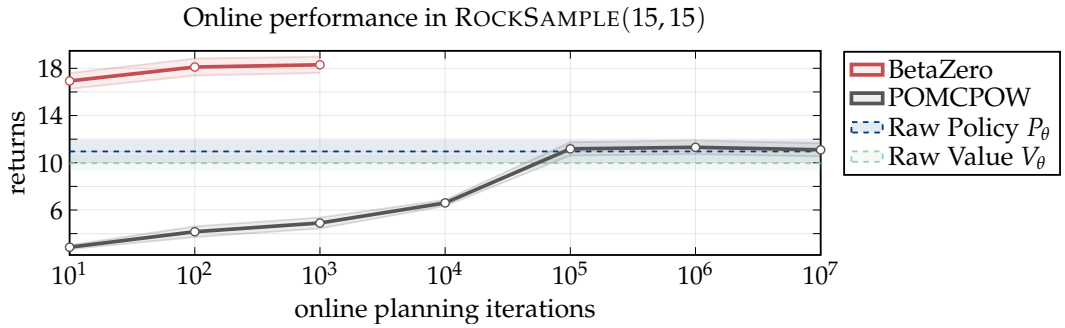


Figure 6.13: Online performance of BetaZero compared to POMCPOW.

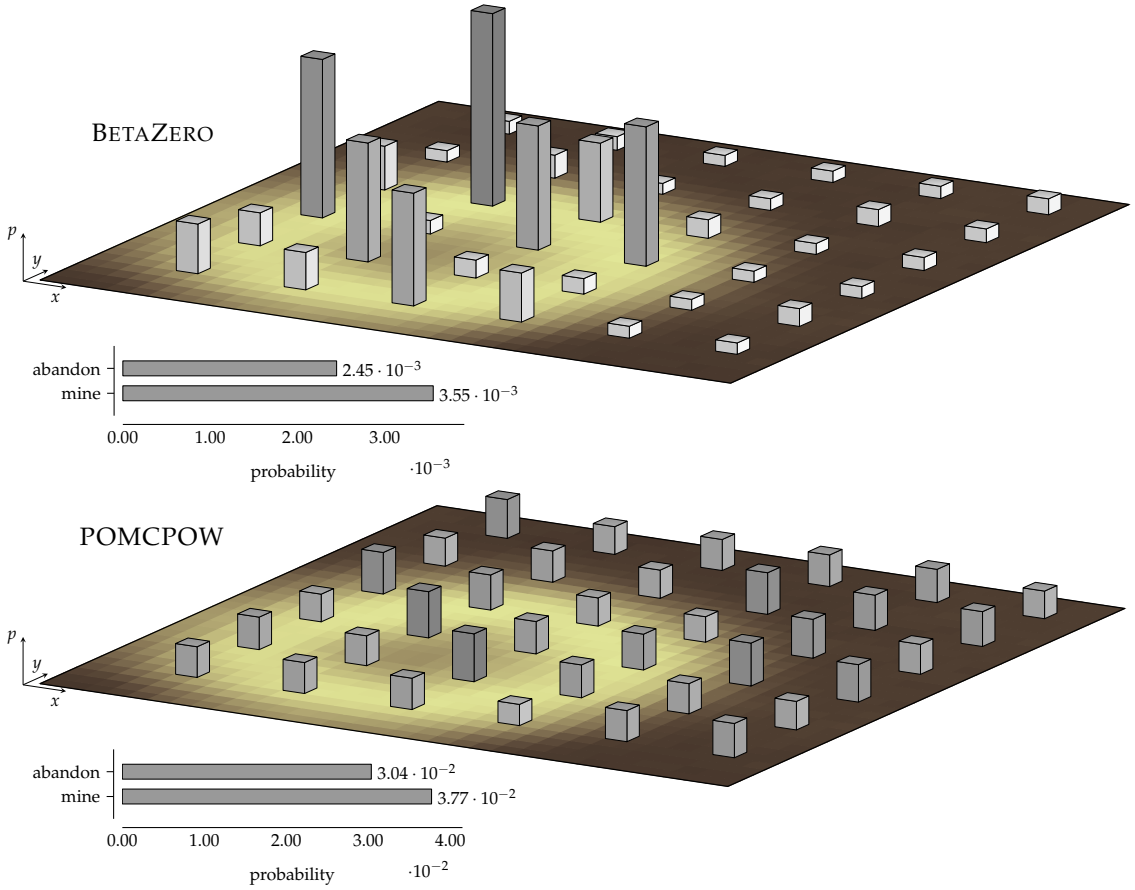


Figure 6.14: BetaZero prioritizes uncertainty, matching heuristics from Mern *et al.* [8].

highlights that LeTS-Drive is able to scale DESPOT to the RockSample(20, 20) problem, with overall similar timing results as BetaZero but worse performance. This could be attributed to the HyP-DESPOT online tree search used in LeTS-Drive that plans over observation space (similar to POMCPOW) and implicitly constructs beliefs from a set of K scenarios in the tree. Therefore, the beliefs are dependent on the number of in-tree scenarios executed, hence the comparable timing results, and not on the actual root node belief that is updated along the tree paths (where belief-state planning incurs different computational expense but with the benefit of planning directly over future uncertainty). Instead of the state history as network input, we use the in-tree belief for a better comparison. The HyP-DESPOT planner expands the tree over *all* actions instead of using progressive widening with prioritization, and, as we observe in the following ablation studies, expanding on all actions may limit the effective use of the tree search budget, potentially missing promising areas of the reachable futures.

6.5.1 Ablation Studies

To test the effect of each contribution, we run several ablation studies. The influence of value and visit count information when selecting an action is shown in figure 6.15. Each cell is the mean return for the RockSample(20, 20) problem over 100 online trials, selecting root-node actions via the arg max of equation (6.1) given z_q and z_n. The cell at $(0, 0)$ corresponds to a uniform policy and thus samples actions instead. Using only the visit counts (bottom cells) or only the values (left cells) to make decisions is worse than using a combination of the two (diagonals). The effect of Q-weighting is also shown in the leftmost figure 6.16, which suggests that it helps learn faster in LightDark(10).

Unsurprisingly, using the *state uncertainty* encoded in the belief is crucial for learning as indicated in the middle of figure 6.16. Future work could directly input the particle set into the network, first passing through an order invariant layer [189], to offload the belief approximation to the network itself. Finally, the rightmost plot in figure 6.16 suggests that when branching on actions using progressive widening, it is important to first prioritize the actions suggested by the policy network. Offline learning fails if instead we sample uniformly from the action space (even in the LightDark case with only three actions). In figure 6.16, one std is shaded from three seeds using 0.6 exponential smoothing.

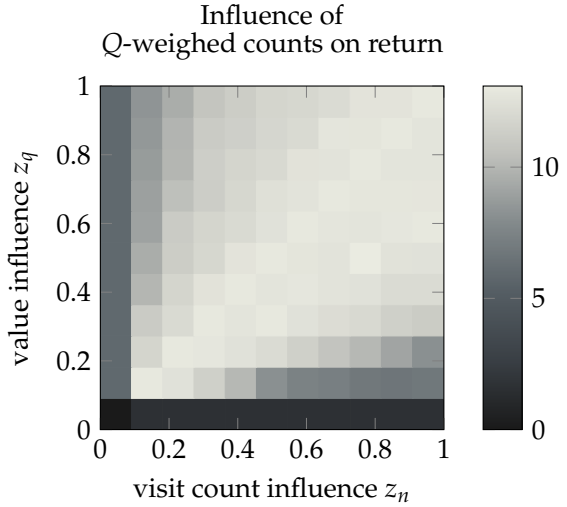


Figure 6.15: Q-weighted visit count ablation study in RockSample(20, 20).

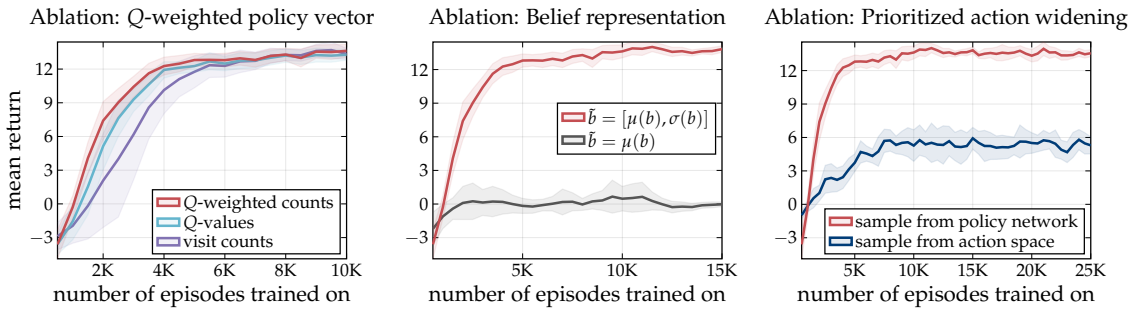


Figure 6.16: LightDark(10) ablation study results.

Bootstrapping analysis. When adding a belief-action pair (b, a) to the MCTS tree, initializing the Q-values via value-network bootstrapping may improve performance when using a small MCTS budget. Table 6.6 shows the returns and online timing results of an analysis comparing BetaZero with bootstrapping $Q_0(b, a) = R_b(b, a) + \gamma V_\theta(\tilde{b}')$ where $\tilde{b}' = \phi(b')$ and without bootstrapping $Q_0(b, a) = 0$. Each domain used the online parameters described in tables 6.2 to 6.4. Results indicate that bootstrapping was only helpful in the rock sample problems and incurs additional compute time due to the belief update done in $b' \sim T_b(b, a)$.‡ In problems with high stochasticity in the belief-state transitions, bootstrapping may be noisy during the initial search due to the transition T_b sampling a single state from the belief. Further analysis could investigate the use of multiple belief transitions to better estimate the value, at the expense of additional computation. The current b could instead be used for bootstrapping, but we would expect similar results in sparse reward problems.

	LightDark(5)		LightDark(10)		RockSample(15, 15)		RockSample(20, 20)		Mineral Exploration	
	returns	time [s]	returns	time [s]	returns	time [s]	returns	time [s]	returns	time [s]
BetaZero (bootstrap)	4.22 ± 0.31	0.014	14.45 ± 1.15	0.34	$\mathbf{20.15 \pm 0.71}$	0.48	$\mathbf{13.09 \pm 0.55}$	1.11	10.32 ± 2.38	6.27
BetaZero (no bootstrap)	$\mathbf{4.47 \pm 0.28}$	0.014	$\mathbf{16.77 \pm 1.28}$	0.33	19.50 ± 0.71	0.42	11.00 ± 0.54	0.57	$\mathbf{10.67 \pm 2.25}$	4.46

Reporting mean $\pm$ standard error over 100 seeds (i.e., episodes); timing is average per episode.

Table 6.6: Effect of Q-value bootstrapping in online *BetaZero* performance.

6.5.2 Limitations of Double Progressive Widening

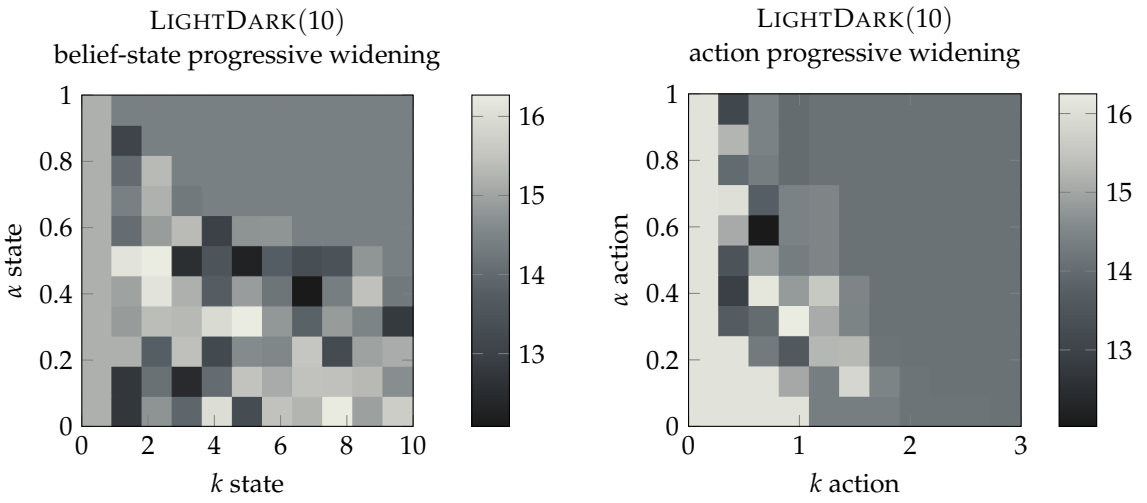
Double progressive widening (DPW) is a straightforward approach to handle large or continuous state and action spaces in Monte Carlo tree search. It is easy to implement

‡Note that bootstrapping was not used during offline training.

and only requires information available in the tree search, i.e., number of children nodes and number of node visits. It is known that MCTS performance can be sensitive to DPW hyperparameter tuning, and Sokota *et al.* [67] show that DPW ignores information about the relation between states that could provide more intelligent branching. Sokota *et al.* [67] introduced *state abstraction refinement* that uses a distance metric between states to determine if a similar state should be added to the tree, requiring a state transition every time a state-action node is visited. For our work, we want to reduce the number of expensive belief-state transitions in the tree and avoid the use of problem-specific heuristics required when defining distance metrics. Using DPW in BetaZero was motivated by simplicity and allows future work to innovate on the components of belief-state and action branching.

To analyze the sensitivity of DPW, figures 6.17a and 6.17b show a sweep over the α and k parameters for DPW in LightDark(10). Figure 6.17a shows that the light dark problem is sensitive to belief-state widening and figure 6.17b indicates that this problem may not require widening on all actions—noting that when $k = 0$, the only action expanded on is the one prioritized from the policy head $a \sim P_\theta(\tilde{b}, \cdot)$. The light dark problems have a small action space of $|\mathcal{A}| = 3$; therefore, this prioritization leads to good performance when only a single action is evaluated (left cells in figure 6.17b when $k = 0$).

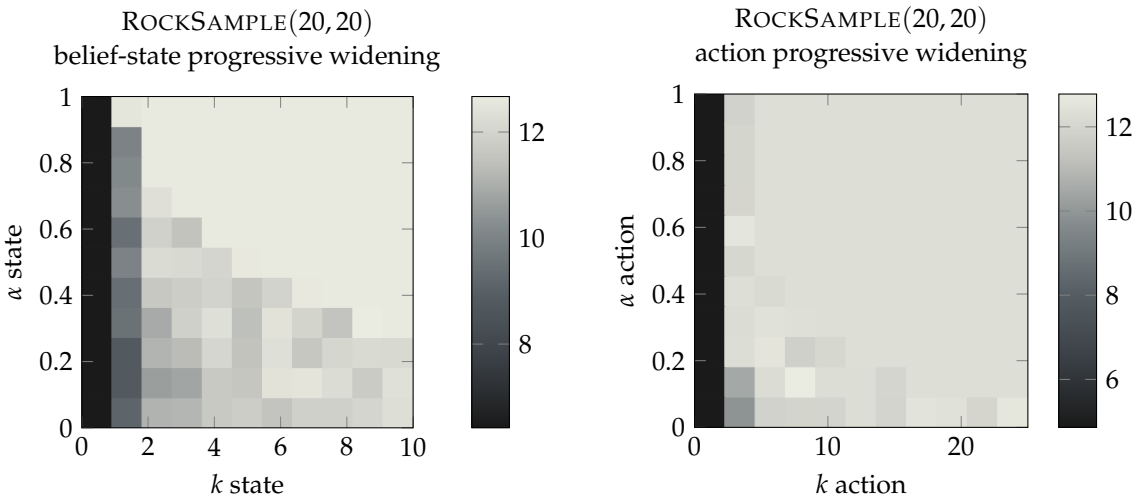
In RockSample(20, 20), figures 6.18a and 6.18b indicate that this problem benefits from a higher widening factor (top right of the figures) as the action space $|\mathcal{A}| = 25$ is larger



(a) Belief-state progressive widening sensitivity.

(b) Action progressive widening sensitivity.

Figure 6.17: Progressive widening sensitivity analyses for LightDark(10).



(a) Belief-state progressive widening sensitivity.

(b) Action progressive widening sensitivity.

Figure 6.18: Progressive widening sensitivity analyses for RockSample(20, 20).

and the belief-state transitions operate over a much larger state space. DPW uses a single branching factor throughout the tree search, and research into methods that adapt the branching based on learned information would be a valuable direction to explore.

Lim *et al.* [168] introduce a class of POMDP planning algorithms that use a fixed number of samples to branch on instead of progressive widening. The bottom rows of figures 6.17 and 6.18 (where $\alpha = 0$) can be interpreted as a fixed branching factor compared to progressive widening in the other cells. The analysis in the figures shows that there are cases where BetaZero has better performance when using progressive widening (lighter colors).

6.6 Open-Sourced Code and Experiments

The BetaZero algorithm has been open sourced and incorporated into the Julia programming language `POMDPs.jl` ecosystem [190]. Fitting into this ecosystem allows BetaZero to access existing POMDP models and can easily be compared to various POMDP solvers. The user constructs a `BetaZeroSolver` that takes parameters for policy iteration and data generation, parameters for neural network architecture and training, and parameters for MCTS (described in the tables above). The user may choose to define a method that inputs the belief b and outputs the belief representation $\tilde{b}$ used by the neural network by defining the `b̃ = input_representation(b)` interface (where the default computes the mean and std of the

belief). Given a `pomdp::POMDP` structure defining the problem, a `solver::BetaZeroSolver` is constructed and solved using the following to get the policy:

```
policy = solve(solver, pomdp)
```

which runs *offline* policy iteration (algorithm 6.3). Once you have a trained neural network, an action can then be generated *online* from the policy given a belief b using the following code (which runs algorithm 6.5):

```
a = action(policy, b)
```

All experiments, including the experiment setup for the baseline algorithms with their heuristics, are included for reproducibility. Code to run MCTS data collection across parallel processes is also included. The code and experiments from this work are available online.[§]

6.7 Conclusions

In this chapter, we proposed the *BetaZero* belief-state planning algorithm for POMDPs; designed to learn from *offline* experience to inform *online* decisions. Planning in belief space explicitly handles state uncertainty, and learning offline approximations to replace hand-crafted or expensive heuristics enables effective online planning in long-horizon POMDPs. Although belief-space planning incurs expensive belief updates in the tree search, we address the limited search budget used in practice by incorporating all information available in the search tree to (a) train the policy vector target (using the Q-weighted visit counts), and (b) sample from the policy network during action progressive widening to prioritize promising actions. Stochastic belief-state transitions in MCTS are addressed using *progressive widening* and we test a belief representation of summary statistics to allow beliefs as input to the value and policy network. Results indicate that BetaZero scales to larger problems where certain heuristics break down and, as a result, can solve large-scale POMDPs by learning to plan in belief space using zero heuristics.

[§] `https://github.com/sisl/BetaZero.jl`

6.7.1 Limitations

In standard POMDP planning algorithms, models are assumed to be known, yet this may limit the applicability to certain problems where reinforcement learning may be better suited. We chose a simplified belief representation to allow for further research innovations in using other parametric and non-parametric representations. Other limitations include compute resource requirements for training neural networks and parallelizing MCTS simulations. We designed BetaZero to use a single GPU for training and to scale based on available CPUs. Certain POMDPs may not require this training burden, especially when known heuristics perform well. BetaZero is useful for long-horizon, high-dimensional POMDPs but may be unnecessary when offline training is computationally limited. BetaZero is designed for problems where the simulation cost is the dominating factor compared to offline training time. For safety-critical problems, the standard approach would be to tune the cost components of the reward function to achieve a desired level of safety (as is standard in POMDP and RL problems). The next chapter introduces a natural extension to BetaZero to address the challenge of safe planning given a desired target level of safety.

PART III:

SAFE PLANNING

Chapter 7

Safe Planning using Failure Surrogates

If you're always aiming for perfection, you won't make anything at all.

Gabrielle Zevin

To plan safely in uncertain environments, agents must balance utility with safety constraints. Safe planning problems can be modeled as a chance-constrained partially observable Markov decision process (CC-POMDP) and solutions often use expensive rollouts or heuristics to estimate the optimal value and action-selection policy. This chapter introduces the *ConstrainedZero* policy iteration algorithm—a natural extension to BetaZero—that solves CC-POMDPs in belief space by learning neural network approximations of the optimal value and policy, with an additional network head that estimates the failure probability given a belief. This failure probability guides safe action selection during online Monte Carlo tree search (MCTS). To avoid letting failure estimates dominate the search, we introduce Δ-MCTS, which uses adaptive conformal inference to update the failure threshold during planning. The approach is tested on a safety-critical POMDP benchmark, an aircraft collision avoidance system, the sustainability problem of safe CO_2 storage, and an aerial wildfire suppression problem treated as a chance-constrained MDP. As a natural extension to BetaZero from chapter 6, results from our ConstrainedZero experiments indicate that by separating safety constraints from the objective, we can achieve a target level of safety without directly optimizing the balance between rewards and costs.

7.1 Motivation

When developing safety-critical agents to make sequential decisions in uncertain environments, planning and reinforcement learning algorithms often formulate the problem as a partially observable Markov decision process (POMDP) with the objective of maximizing a scalar-valued reward function [5]. POMDP solution methods find a policy that maximizes the expectation of this reward. To ensure adequate safety, the scalar reward is tuned to balance the goals of the agent while penalizing undesired behavior or failures. Recently, chance-constrained POMDPs (CC-POMDPs) have been used to frame the safe planning problem by separating the reward function into a constrained problem [191]. The objective of CC-POMDPs is to maximize the goal rewards while satisfying the safety constraints.

To solve CC-POMDPs, online algorithms such as RAO* use heuristic forward search to find policies that maximize the reward and estimate the risk of constraint violation [191]. RAO* plans over the reachable belief space for discrete state, action, and observation CC-POMDPs. The iterative RAO* (iRAO*) extends the heuristic search algorithm to multi-agent settings and handles continuous states and actions through Gaussian process regression and probabilistic flow tubes [7]. Lauri *et al.* [57] highlight the limitations of such chance-constrained POMDP algorithms and the need for scalable approaches to solve large-scale, long-horizon CC-POMDPs in practice.

To address scalability and applicability to continuous state and observation spaces, we introduce the *ConstrainedZero* policy iteration algorithm that combines offline neural network training of the value function, the action-selection policy, and the failure probability predictor with online Monte Carlo tree search (MCTS) to improve the policy through planning. ConstrainedZero is a direct extension to the POMDP belief-state planning algorithm BetaZero and the family of AlphaZero algorithms [17], with extensions shown in red in figure 7.1. Along with an open-source implementation,* our main contributions are:

1. We introduce Δ-MCTS, an anytime algorithm for MDPs (applied to belief-state MDPs) that estimates failure probabilities along with Q-values and adjusts the failure probability threshold using adaptive conformal inference [192]. Δ-MCTS selects actions by maximizing the Q-value while satisfying that the failure probability constraint is below the adapted threshold using the introduced CC-PUCT criterion.

*`https://github.com/sisl/ConstrainedZero.jl`

2. We introduce ConstrainedZero, a policy iteration algorithm that extends BetaZero for CC-POMDPs. ConstrainedZero includes an additional network head that estimates the failure probability given a belief and uses Δ-MCTS with the neural network surrogate to prioritize promising safe actions, replacing expensive rollouts or domain-specific heuristics. Framing the problem as a CC-POMDP means a target safety level can be specified instead of balancing penalties in the reward function.

3. We evaluate ConstrainedZero and Δ-MCTS on three challenging safety-critical benchmark CC-POMDPs: a long-horizon localization task (LightDark [30]), an aircraft collision avoidance system (modeled after ACAS X [12]), and safe CO_2 storage [39], and test on a chance-constrained MDP problem of wildfire suppression [27].

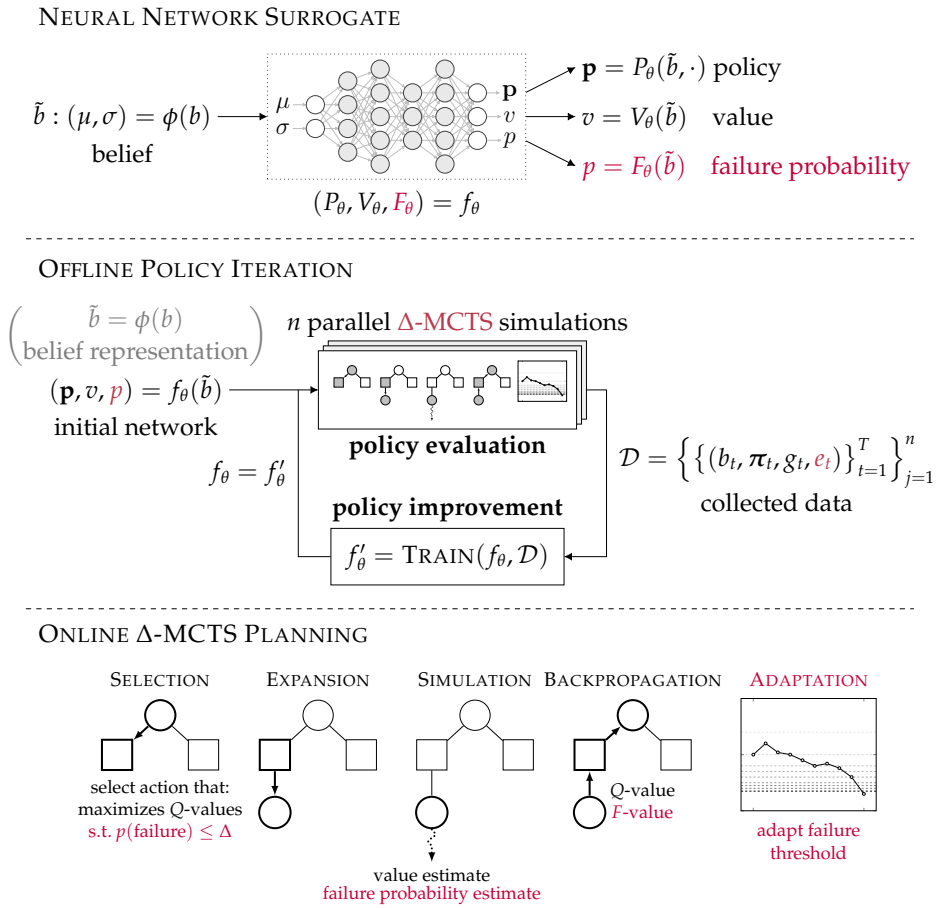


Figure 7.1: Elements of *ConstrainedZero* for CC-POMDP planning.

7.2 Problem Formulation

This section formulates the safe planning problem as a belief-state CC-MDP. Background context is also provided on the Monte Carlo tree search algorithm.

Review of POMDPs and belief-state MDPs. To formulate chance-constrained problems, we extend the POMDP framework (section 3.2). POMDPs are used for sequential decision making problems where the agent has uncertainty over their state in the environment [5]. The POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$ consisting of a state space $\mathcal{S}$, an action space $\mathcal{A}$, an observation space $\mathcal{O}$, a transition model T, a reward model R, an observation model O, and a discount factor $\gamma \in [0, 1)$. When solving POMDPs, the objective is to find a policy $\pi(b)$ given a belief b over the unobserved state and return an action $a \in \mathcal{A}$ that maximizes the *value* of the belief, which is the expected discounted sum of rewards (i.e., the expected discounted *returns*) when continuing to follow the policy π:

$$\pi(b_0) = \arg\max_{a \in \mathcal{A}} \ \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_b(b_t, a_t) \mid b_0 \right] \tag{7.1}$$

where b_0 is the initial belief, often using the initial state distribution, and the belief-based reward is defined as:

$$R_b(b, a) = \int b(s) R(s, a) \, \mathrm{d}s \tag{7.2}$$

Every POMDP can be cast as an MDP by simply treating the belief as the state. In doing so, one can construct a belief-state MDP (BMDP) with the belief space $\mathcal{B}$ of the original POMDP as the MDP state space, while using the same action space $\mathcal{A}$, the belief-based reward model R_b from equation (7.2), and a transition function $b' \sim T_b(\cdot \mid b, a)$ that takes the current belief b and action a and returns a stochastic updated belief b'. The belief transition function first samples a hidden state $s \sim b(\cdot)$ and transitions that state through the POMDP transition function $s' \sim T(\cdot \mid s, a)$. Then an observation is sampled from the observation model $o \sim O(\cdot \mid a, s')$, and finally, the belief is updated to get the posterior:

$$b'(s') \propto O(o \mid a, s') \int T(s' \mid s, a) b(s) \, \mathrm{d}s \tag{7.3}$$

The belief update may be done exactly as in equation (7.3) or using approximations such as a Kalman filter [49] or particle filter [52], as described in section 3.2.2.

The belief-state MDP tuple of $\langle \mathcal{B}, \mathcal{A}, T_b, R_b, \gamma \rangle$ can also be defined using a generative model $(b', r) \sim G_b(b, a)$ instead of an explicit belief transition model T_b and belief reward model R_b. The underlying POMDP can also use a generative model $(s', r, o) \sim G(s, a)$. Our work uses the generative POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, G, \gamma \rangle$ and the generative BMDP $\langle \mathcal{B}, \mathcal{A}, G_b, \gamma \rangle$.

7.2.1 Chance-Constrained POMDPs

When dealing with safety-critical sequential decision making problems, separating safety constraints from the objective allows for solvers to target an adequate level of safety while simultaneously maximizing rewards. This is in contrast to designing a single reward function to balance the rewards from the goals and penalties from violating safety. The chance-constrained POMDP (CC-POMDP) defines a failure set $\mathcal{F}$ that includes all state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ that fail and a bound $\Delta \in [0, 1]$ on the probability, or *chance*, of a failure event occurring. Chance constraints are intuitive for subject matter experts to define as they translate to the target failure probability of the agent, which is often the requirement for systems in industries such as aviation [193] and finance [194]. The objective when solving CC-POMDPs is to maximize the value function while ensuring that the failure probability, or the chance constraint, is below the target threshold Δ:

$$\underset{\pi}{\text{maximize}} \;\; V^\pi(b_0) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_b(b_t, a_t) \mid b_0 \right] \tag{7.4}$$

$$\text{subject to} \;\; F^\pi(b_0) = \mathbb{P}_\pi \left[\bigvee_{t=0}^{\infty} \Big((s_t, a_t) \in \mathcal{F} \Big) \mid b_0 \right] \leq \Delta \tag{7.5}$$

The failure probability $F^\pi(b_t)$ is often called the *execution risk* [191] of the policy π computed from the belief b_t. Together, the CC-POMDP is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{F}, T, R, O, \gamma, \Delta \rangle$, which may also use a generative model G to replace $\langle T, R, O \rangle$.

Chance-constrained belief-state MDPs. Our work casts the chance-constrained POMDP to a chance-constrained belief-MDP (CC-BMDP). The CC-BMDP tuple $\langle \mathcal{B}, \mathcal{A}, F_b, T_b, R_b, \gamma, \Delta \rangle$ extends BMDPs with an immediate failure probability function $F_b : \mathcal{B} \times \mathcal{A} \to [0, 1]$ and a failure probability threshold Δ. The immediate failure probability is computed as

$$F_b(b, a) = \int b(s) \mathbb{1}\{(s, a) \in \mathcal{F}\} \, \mathrm{d}s \tag{7.6}$$

using the failure set $\mathcal{F}$ and indicator function $\mathbb{1}\{\cdot\}$ returning 1 if `true` and 0 otherwise. The CC-BMDP can also be defined with a generative model $(b', r, p) \sim G_b(b, a)$ that also

returns the failure probability $p = F_b(b, a)$ with notation overloading of G_b, resulting in the tuple $\langle \mathcal{B}, \mathcal{A}, G_b, \gamma, \Delta \rangle$. In this chapter, we introduce scalable chance-constrained planning algorithms to solve high-dimensional CC-POMDPs.

Review of Monte Carlo tree search. The best-first search algorithm, Monte Carlo tree search (MCTS), is designed to solve MDPs [10] and has been applied to solve POMDPs cast as belief-state MDPs [61, 62, 195]. MCTS is an online algorithm that determines the best action to take from the current state s (or belief state b). MCTS iteratively simulates the following four steps to build out a tree of possible reachable futures to a depth d:

1. **Selection.** An action is selected from the existing children of the current state node or sampled from the action space. The selection process balances exploration and exploitation. Metrics such as UCT [63] or PUCT [65] have been used in the literature to select the action to take.

2. **Expansion.** Once an action is selected, it is executed from the current state node to expand the tree. For stochastic state transitions, methods like progressive widening [66] or state abstraction refinement [67] can be used to control when to execute the action or when to take an existing tree path.

3. **Simulation.** From the expanded state, the tree is recursively built from this new root node, and the simulation step returns an estimate of the value of the expanded state. The value estimate could use a rollout policy [68] (which may be expensive for BMDPs), or use function approximators such as neural networks [65, 195, 196].

4. **Backpropagation.** Finally, the value estimate is combined with the immediate reward to get the Q-value. This Q-value is assigned to the parent state-action node as a running mean. This process backpropagates the signal up the tree path that led to that node.

After a prescribed number of iterations, MCTS will select the best action from the children of the root node, often using the Q-values or visit counts [59].

Brázdil *et al.* [197] introduced an algorithm for CC-MDPs that uses UCT with a table-based value and risk predictor, and a linear program to compute an action distribution that satisfies an adaptive constraint. Ayton *et al.* [198] introduced an MCTS algorithm for CC-MDPs that selects actions that satisfy a local chance constraint over state histories and prune branches that violate the constraint. MCTS algorithms have also been developed

for cost-constrained POMDPs (C-POMDPs) [199], where a discounted cost is minimized. Algorithms such as C-POMCP [200], C-MCTS [201], C-POMCPOW, C-PTF-DPW, and C-POMCP-DPW [202] estimate the cost function during search using rollouts, while our approach uses *chance* constraints and estimates violation *probabilities* using surrogates.

7.3 Proposed Algorithm: ConstrainedZero

ConstrainedZero follows the BetaZero policy iteration steps of *policy evaluation* and *policy improvement* while also collecting failure event indicators to train the failure probability network head, shown in red in algorithm 7.1. During policy evaluation, n parallel Δ-MCTS executions are run and a data set $\mathcal{D}$ is collected. The data set $\mathcal{D} = \left\{\{b_t, \boldsymbol{\pi}_t, g_t, e_t\}_{t=1}^{T}\right\}_{j=1}^{n}$ is a tuple of the belief at episode time step t denoted b_t, the tree policy $\boldsymbol{\pi}_t$, the return $g_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_i$ based on the observed reward r_i and discount factor γ, and the failure event indicator e_t, where g_t and e_t are computed at the end of the trajectory for all time $t \leq T$. The failure event is computed as the disjunction of all (s, a) pairs of the CC-POMDP in the execution trajectory to ensure that if a trajectory failed at some point the full trajectory is marked as a failure:

$$e_t = \mathbb{1}\left\{\bigvee_{i=t}^{T}\Big((s_i, a_i) \in \mathcal{F}\Big)\right\} \tag{7.7}$$

where $\mathbb{1}\{E\}$ is the indicator function that returns 1 when event E is `true` and 0 otherwise.

During policy improvement, the neural network is trained to minimize the MSE or MAE loss $\mathcal{L}_{V_\theta}(g_t, v_t)$ to regress the value function $v_t = V_\theta(\tilde{b}_t)$, minimize the cross-entropy loss $\mathcal{L}_{P_\theta}(\boldsymbol{\pi}_t, \mathbf{p}_t)$ to imitate the tree policy $\mathbf{p}_t = P_\theta(\tilde{b}_t)$, and additionally minimize the binary cross-entropy loss $\mathcal{L}_{F_\theta}(e_t, p_t)$ to regress the failure probability function $p_t = F_\theta(\tilde{b}_t)$, with added regularization using the L_2-norm of the weights θ:

$$\begin{aligned}
\mathcal{L}_{V_\theta}(g_t, v_t) &= (g_t - v_t)^2 \text{ or } |g_t - v_t| \\
\mathcal{L}_{P_\theta}(\boldsymbol{\pi}_t, \mathbf{p}_t) &= -\boldsymbol{\pi}_t^\top \log \mathbf{p}_t \\
\mathcal{L}_{F_\theta}(e_t, p_t) &= -e_t \log p_t - (1 - e_t) \log(1 - p_t) \\
\mathcal{L}_{\text{CZ}} &= \mathcal{L}_{V_\theta}(g_t, v_t) + \mathcal{L}_{P_\theta}(\boldsymbol{\pi}_t, \mathbf{p}_t) + \mathcal{L}_{F_\theta}(e_t, p_t) + \lambda \|\theta\|^2
\end{aligned}$$

The failure probability head of the neural network includes a final sigmoid layer to ensure the output can be interpreted as a probability in the range $[0, 1]$.

Algorithm 7.1: Offline ConstrainedZero policy iteration.

Require: $\mathcal{P}_{\text{cc}} \stackrel{\text{def}}{=} \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{F}, G, \gamma, \Delta_0 \rangle$ ▷ Generative CC-POMDP
Require: ψ: Parameters
function PolicyIteration($\mathcal{P}_{\text{cc}}, \psi$)
  $f_\theta \leftarrow$ InitializeNetwork(ψ) ▷ $(P_\theta, V_\theta, F_\theta) \leftarrow f_\theta$
  for $i \leftarrow 1$ **to** $n_{\text{iterations}}$
    $\mathcal{D} \leftarrow$ CollectData($\mathcal{P}_{\text{cc}}, f_\theta$) ▷ policy evaluation
    $f_\theta \leftarrow$ Train($f_\theta, \mathcal{D}$) ▷ policy improvement
  return $\pi(b) \leftarrow \Delta\text{MCTS}(\mathcal{P}_{\text{cc}}, f_\theta, b)$ ▷ online policy (alg. 7.2)
function CollectData($\mathcal{P}_{\text{cc}}, f_\theta$)
  $\mathcal{D} = \emptyset$
  parallel for $j \leftarrow 1$ **to** n_{data}
    for $t \leftarrow 1$ **to** T ▷ sample a trajectory
      $a_t \leftarrow \Delta\text{MCTS}(\mathcal{P}_{\text{cc}}, f_\theta, b_t)$ ▷ record $\boldsymbol{\pi}_{\text{tree}}^{(t)}$ vector
      $s_{t+1}, r_t, o_t \sim G(s_t, a_t)$ ▷ generative model
      $b_{t+1} \leftarrow P(s_{t+1} \mid b_t, a_t, o_t)$ ▷ update belief
    for $t \leftarrow 1$ **to** T
      $g_t \leftarrow \sum_{i=t}^{T} \gamma^{(i-t)} r_i$ ▷ computed over trajectory
      $e_t \leftarrow \mathbb{1}\{\bigvee_{i=t}^{T} ((s_i, a_i) \in \mathcal{F})\}$ ▷ failure event
      $\mathcal{D}_j^{(t)} \leftarrow \mathcal{D}_j^{(t)} \cup \{(b_t, \boldsymbol{\pi}_{\text{tree}}^{(t)}, g_t, e_t)\}$
  return $\mathcal{D}$

7.3.1 Adaptive Safety Constraints in Δ-MCTS

When using online MCTS for CC-BMDP planning, two considerations have to be addressed: (1) how to estimate the observed failure probability in the tree search, and (2) how to select actions constrained by this failure probability. To address this, we introduce Δ-MCTS, an extension of MCTS for chance-constrained belief-state MDPs.

At each node for the belief-state and action (b, a), the immediate failure probability p is computed using the generative model (or by calling $p = F_b(b, a)$ directly). An estimate of the future failure probability p' can be computed using rollouts, which may be expensive for belief-state planning; thus, we use the trained neural network head for failure probability estimation $p' = F_\theta(\tilde{b}')$. Similar to the Q-value, we must compute the full failure probability of the trajectory from the immediate time step to the horizon, termed the F-value. Let E be the immediate failure event from belief b when taking action a at time t, and let E' be

the event of failing in the future (from $t+1$ to the horizon T). The probability of failing between the current time t and the horizon T becomes:

$$\begin{aligned} P(E_{t:T}) &= P(E) + P(E') - P(E \cap E') && (7.8)\\ &= P(E) + P(E') - P(E')P(E \mid E') && (7.9)\\ &= P(E) + P(E') - P(E')P(E) && (7.10)\\ &= p + p' - pp' && (7.11)\\ &= p + (1-p)p' && (7.12) \end{aligned}$$

assuming independence in equation (7.10). It is well known that a discount factor can be interpreted as the $(1-\delta)$ probability of termination on the next step [55, 203]. Therefore, we apply a discount δ to control the influence of the future failure probability:

$$p = p + \delta(1-p)p' \tag{7.13}$$

Unlike Carpin *et al.* [204], who backup F-values based on the best-case, we backpropagate the F-values up the tree similar to Q-values (line 16, alg. 7.2):

$$F(b,a) = F(b,a) + \frac{p - F(b,a)}{N(b,a)} \tag{7.14}$$

which is a running mean estimate where $F(b,a)$ is initialized using the initialization function $F_0(b,a)$, noting the F_0 subscript: which could either be zero, the immediate failure probability $F_b(b,a)$, or the bootstrapped value by taking action a to get a new belief b' and computing equation (7.13) based on the $p' = F_\theta(\tilde{b}')$ estimate. Note, the number of times the node (b,a) is visited is indicated as the visit count $N(b,a)$.

Using the estimate $F(b,a)$, a simple way to select actions that do not violate the safety constraint set by Δ would be to use the PUCT algorithm [17]:

$$\pi_{\text{naïve-PUCT}}(b) = \underset{a \in A(b)}{\arg\max}\, \bar{Q}(b,a) + c\left(P_\theta(\tilde{b},a)\frac{\sqrt{N(b)}}{1+N(b,a)}\right) \tag{7.15}$$

$$\text{s.t. } F(b,a) \leq \Delta \tag{7.16}$$

with a hard constraint on safety of only choosing actions such that $F(b,a) \leq \Delta$ is satisfied. Note that the belief node visit count is $N(b) = \sum_{a'} N(b,a')$ for children $a' \in A(b)$ and

following Schrittwieser *et al.* [166], we normalize the Q-values between zero and one, denoted $\bar{Q}$, to avoid problem-specific heuristics when selecting an exploration constant c:

$$\bar{Q}(b,a) = \frac{Q(b,a) - \min_{(b',a')\in\mathcal{T}} Q(b',a')}{\max_{(b',a')\in\mathcal{T}} Q(b',a') - \min_{(b',a')\in\mathcal{T}} Q(b',a')} \tag{7.17}$$

PUCT exploits nodes based on their observed Q-values and explores nodes based on their visit counts weighted by the action-selection policy P_θ to explore promising actions.

However, if the failure probability threshold Δ is too conservative, the action-selection process may fail to find *any* action that satisfies the constraint. Therefore, Δ-MCTS tracks an estimate of the threshold $\Delta(b)$ for each belief node and updates it using *adaptive conformal inference* (ACI) [192]. ACI is a distribution-free, sequential calibration method that guarantees valid coverage with high probability in online settings. This makes ACI well-suited for our task of adapting the failure probability threshold in an online MCTS setting. The adaptive threshold is initialized to the target tolerance $\Delta(b) = \Delta_0$ where $\Delta_0 = \Delta$ from the CC-BMDP. Each time the F-value is updated (either by equation (7.14) or initialization), the new ADAPTATION procedure is called to update the current acceptable safety threshold.

In adaptation, the error term of $\text{err} = \mathbb{1}\{F(b,a) > \Delta(b)\}$ indicates when to widen or restrict the estimated threshold $\Delta(b)$ based on whether the failure probability estimate of the most recently explored belief-action node $F(b,a)$ is above or below the current threshold. The estimated threshold is updated according to:

$$\Delta(b) = \Delta(b) + \eta(\text{err} - \Delta_0) \tag{7.18}$$

which will widen the threshold if the observed failure probability is outside the threshold (i.e., if the error is one), and will tighten the threshold otherwise:

$$\Delta(b) = \begin{cases} \Delta(b) + \eta(1 - \Delta_0) & \text{if } F(b,a) > \Delta(b) \\ \Delta(b) - \eta\Delta_0 & \text{if } F(b,a) \leq \Delta(b) \end{cases} \tag{7.19}$$

Intuitively, the update adjusts the threshold of acceptable failure probability $\Delta(b)$ based on recent experience. If the failure probability $F(b,a)$ for a recent action is higher than the current threshold $\Delta(b)$, this indicates a higher risk than expected. Thus, the threshold is increased by $\eta(1 - \Delta_0)$ for $\eta > 0$ to allow for more risk in future actions. Otherwise, if $F(b,a)$ is lower than the threshold, this means actions are safer than expected and the threshold

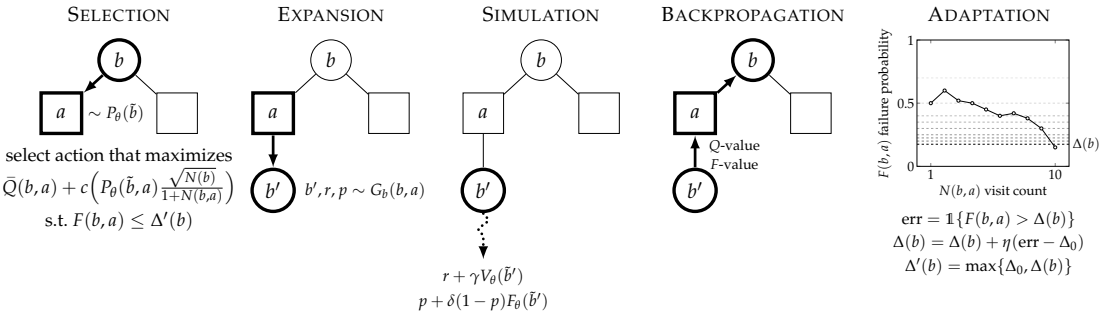


Figure 7.2: Δ-MCTS: Chance-constrained MCTS with failure threshold adaptation.

is decreased by $\eta\Delta_0$ (favoring a more reactive increase than decrease of the threshold). Notably, Gibbs *et al.* [192] prove that $\Delta(b)$ converges exactly to the desired target over time.

We clip the final threshold to the lower and upper bounds of the observed failure probability for a given belief b to restrict the change in $\Delta(b)$ and, more importantly, to guarantee that at least one action is available for selection:

$$\Delta(b) = \text{clip}\big(\Delta(b) + \eta(\text{err} - \Delta_0),\, l(b),\, u(b)\big) \tag{7.20}$$

with the lower and upper bounds of $l(b) = \min_{a'} F(b,a')$ and $u(b) = \max_{a'} F(b,a')$ for a' in children nodes $A(b)$.

The resulting criterion selects actions that satisfy the adaptive constraint of $F(b,a) \leq \Delta'(b)$ where the selection threshold $\Delta'(b) = \max\{\Delta_0, \Delta(b)\}$ upper bounds the failure probability. Together, the Δ-MCTS exploration policy becomes:

$$\pi_{\text{explore}}(b) = \underset{a \in A(b)}{\arg\max}\ \bar{Q}(b,a) + c\left(P_\theta(\tilde{b},a)\frac{\sqrt{N(b)}}{1+N(b,a)}\right) \tag{7.21}$$

$$\text{s. t. } F(b,a) \leq \Delta'(b) \tag{7.22}$$

termed the *chance-constrained PUCT* criterion (CC-PUCT). The constraint in equation (7.22) is also used in root node action selection (line 7, alg. 7.2). In practice, π_{explore} is computed using the indicator $\mathbb{1}$, returning the action $a \in A(b)$ that maximizes:

$$\mathbb{1}\{F(b,a) \leq \Delta'(b)\}\left(\bar{Q}(b,a) + c\left(P_\theta(\tilde{b},a)\frac{\sqrt{N(b)}}{1+N(b,a)}\right)\right)$$

The benefit of CC-PUCT is that when our explored samples satisfy the constraint $\Delta'(b)$ (defined over the belief rather than both belief and action) we may explore new actions

from this belief which are both safe and have the potential for higher reward. The key idea is that actions are chosen based on the balance between safety and utility, ensuring that we do not over-prioritize safety at the expense of potential rewards, while not exploiting rewards without regarding the risk.

The five stages of Δ-MCTS are shown in figure 7.2 and algorithms 7.2 to 7.6: *selection*, *expansion*, *simulation*, *backpropagation*, and *adaptation*, with extensions to BetaZero in red.

Algorithm 7.2: Δ-MCTS for online chance-constrained planning.

function ΔMCTS($\mathcal{P}_{\text{cc}}, f_\theta, b$)
  $\mathcal{M}_{\text{cc}} \leftarrow \langle \mathcal{B}, \mathcal{A}, G_b, \gamma, \Delta_0 \rangle$ ▷ convert from generative $\mathcal{P}_{cc}$
  for 1 **to** n_{online}
    SIMULATE(f_θ, b, d)
  $\pi_{\text{tree}}(b,a) \propto \left(\left(\frac{\exp Q(b,a)}{\sum_{a'} \exp Q(b,a')} \right) \left(\frac{N(b,a)}{\sum_{a'} N(b,a')} \right) \right)^{1/\tau}$ ▷ using equation (6.1)
  return $a \sim \pi_{\text{tree}}(b,a)$ ▷ let $\tau \to 0$ to get argmax
    s.t. $F(b,a) \leq \Delta'(b)$ ▷ $\Delta'(b) = \max\{\Delta_0, \Delta(b)\}$

Algorithm 7.3: Δ-MCTS recursive simulation.

function SIMULATE(f_θ, b, d)
  if TERMINAL(b) **return** $0, 0$
  if $b \notin \mathcal{T}$ **or** $d = 0$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{b\}$
    $N(b) \leftarrow N_0(b)$
    $\tilde{b} \leftarrow \phi(b)$ ▷ input belief representation
    return $V_\theta(\tilde{b}), F_\theta(\tilde{b})$ ▷ network lookup
  $N(b) \leftarrow N(b) + 1$
  $a \leftarrow$ ACTIONSELECTION(f_θ, b) ▷ selection
  $(b', r, p) \leftarrow$ BELIEFSTATEEXPANSION(b, a) ▷ expansion
  $v', p' \leftarrow$ SIMULATE($f_\theta, b', d-1$) ▷ simulation
  $q \leftarrow r + \gamma v'$
  $p \leftarrow p + \delta(1-p)p'$ ▷ observed failure probability
  $N(b,a) \leftarrow N(b,a) + 1$
  $Q(b,a) \leftarrow Q(b,a) + \frac{q - Q(b,a)}{N(b,a)}$ ▷ backpropagation: Q-value
  $F(b,a) \leftarrow F(b,a) + \frac{p - F(b,a)}{N(b,a)}$ ▷ backpropagation: F-value
  ADAPTATION(Δ, b, a) ▷ adaptation
  return q, p

Algorithm 7.4: Δ-MCTS action selection.

function ACTIONSELECTION(f_θ, b)
  $\tilde{b} \leftarrow \phi(b)$
  if $|A(b)| \leq k_a N(b)^{\alpha_a}$ ▷ action progressive widening
    $a \sim P_\theta(\tilde{b}, \cdot)$ ▷ prioritized from network
    if $a \notin A(b)$
      $N(b,a) \leftarrow N_0(b,a)$
      $Q(b,a) \leftarrow Q_0(b,a)$
      $F(b,a) \leftarrow F_0(b,a)$
      $\Delta(b) \leftarrow \Delta_0$ ▷ target tolerance Δ_0
      ADAPTATION(Δ, b, a)
      $A(b) \leftarrow A(b) \cup \{a\}$
  return $\arg\max_{a \in A(b)} \bar{Q}(b,a) + c\left(P_\theta(\tilde{b}, a)\frac{\sqrt{N(b)}}{1+N(b,a)}\right)$ ▷ CC-PUCT
      s.t. $F(b,a) \leq \Delta'(b)$

Algorithm 7.5: Δ-MCTS belief-state expansion.

function BELIEFSTATEEXPANSION(b, a)
  if $|B(b,a)| \leq k_b N(b,a)^{\alpha_b}$ ▷ belief progressive widening
    $b', r, p \sim G_b(b,a)$ ▷ generative model
    $B(b,a) \leftarrow B(b,a) \cup \{(b', r, p)\}$
  else
    $b', r, p \sim B(b,a)$
  return b', r, p

Algorithm 7.6: Δ-MCTS adaptation.

function ADAPTATION(Δ, b, a)
  $l(b) \leftarrow \min_{a' \in A(b)} F(b,a')$ ▷ update bounds
  $u(b) \leftarrow \max_{a' \in A(b)} F(b,a')$
  $\text{err} \leftarrow \mathbb{1}\{F(b,a) > \Delta(b)\}$
  $\Delta(b) \leftarrow \text{clip}\big(\Delta(b) + \eta(\text{err} - \Delta_0), l(b), u(b)\big)$

7.3.2 Connection to ACI Quantiles

In adaptive conformal inference (ACI), the algorithm provides coverage based on quantiles of an estimated value from some data distribution [192]. In our work, we simplify the ACI

formulation. To connect to the original ACI formulation, let $F(b, a)$ be the estimated failure probability (F-value) and $\widehat{Q}(\cdot)$ be the quantile function where the Δ-quantile is defined on the range $[0, 1]$.[†] Let the set of failure probabilities associated to a belief-state node b with children $A(b)$ be:

$$\mathbf{f} = \left\{ F(b, a') : a' \in A(b) \right\} \tag{7.23}$$

In this formulation, we would want to update $\Delta(b)$ based on the error as indication of miscoverage of the most recent F-value $F(b, a)$:

$$\text{err} = \begin{cases} 1 & \text{if } F(b,a) \notin \widehat{C}_\Delta \\ 0 & \text{otherwise} \end{cases} \tag{7.24}$$

where $\widehat{C}_\Delta := \left\{ p_i : p_i \leq \widehat{Q}(\Delta(b)),\ p_i \in \mathbf{f} \right\}$ is the covered set. Here we use $\Delta(b)$ instead of $1 - \Delta(b)$ from Gibbs *et al.* [192] because we want the lower quantile (i.e., we want to be below the failure probability threshold). This is equivalent to our simplification where the error is defined as the miscoverage of the F-value by the $\Delta(b)$ estimate:

$$\text{err} = \mathbb{1}\{F(b, a) > \Delta(b)\} \tag{7.25}$$

and using the same update as Δ-MCTS of:

$$\Delta(b) = \Delta(b) + \eta(\text{err} - \Delta_0) \tag{7.26}$$

Note that we reformulate the update function for our setting. In the original ACI formulation, the update is done according to $\Delta(b) = \Delta(b) + \eta(\Delta_0 - \text{err})$. In the original version, if the error is zero (indicating coverage), then $\Delta(b)$ is increased by $\eta\Delta_0$ (becoming tighter, noting this assumes the coverage is based on the upper quantile). If the error is one (indicating miscoverage), then $\Delta(b)$ is decreased by $\eta(1 - \Delta_0)$ (widening the coverage). In our version, we reverse the update to operate on the lower quantile, keeping the reactivity of the algorithm during miscoverage events, thus, increasing by $\eta(1 - \Delta_0)$ in this case, as described in equation (7.19).

[†]The quantile function is typically defined over an input set Y. But in our case, it is defined over an input range $[0, 1]$ to assess probabilities in a standardized way.

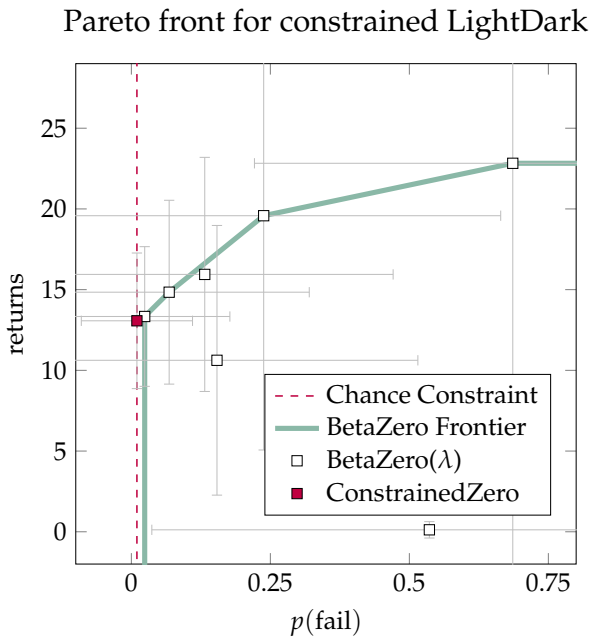


Figure 7.3: Pareto front comparing against BetaZero(λ).

7.4 Experiments

For a fair comparison, ConstrainedZero was evaluated against BetaZero using the same network and MCTS parameters. BetaZero uses a scalarized reward function to penalize failures, while ConstrainedZero omits the penalty and plans using the adaptive safety constraint instead. The BetaZero reward takes the form $\bar{R}_b(b,a) = R_b(b,a) - \lambda C(b,a)$ with a cost $C(b,a)$ scaled by a tunable parameter λ, shown in figure 7.3.

Three safety-critical CC-POMDPs were evaluated. The first is the *LightDark* POMDP, a benchmark localization task [30], where a particle filter is used to update the belief with $n_{\text{particles}} = 500$. A failure occurs if the agent stops outside of the goal (i.e., the origin).

The next CC-POMDP is the *aircraft collision avoidance* problem (CAS), modeled after ACAS X [12]. In the CAS problem, the ownship aircraft attempts to avoid a near mid-air collision (NMAC) with an intruding aircraft while minimizing the alert and reversal rates. An NMAC occurs if the ownship is within 50 meters in relative altitude h_{rel} of the intruder at the end of the encounter (i.e., when $|h_{\text{rel}}| \leq 50$ and time to collision $t_{\text{col}} = 0$). The belief is updated with an unscented Kalman filter [49] (detailed in section 3.2.2) to track the mean and covariance of the state variables.

Lastly, we study safe *carbon capture and storage* (CCS), which is a promising mitigation of global emissions that captures CO_2 and stores it in porous subsurface material [39]. A challenge of CCS is safely injecting CO_2 into the subsurface while mitigating risk of leakage and earthquakes, where any CO_2 leakage indicates a failure. The simplified CCS problem uses spillpoint analysis to model the top surface of the injection site, and a sequential importance resampling particle filter is updated with observations at drilling locations [39].

7.4.1 Empirical Results

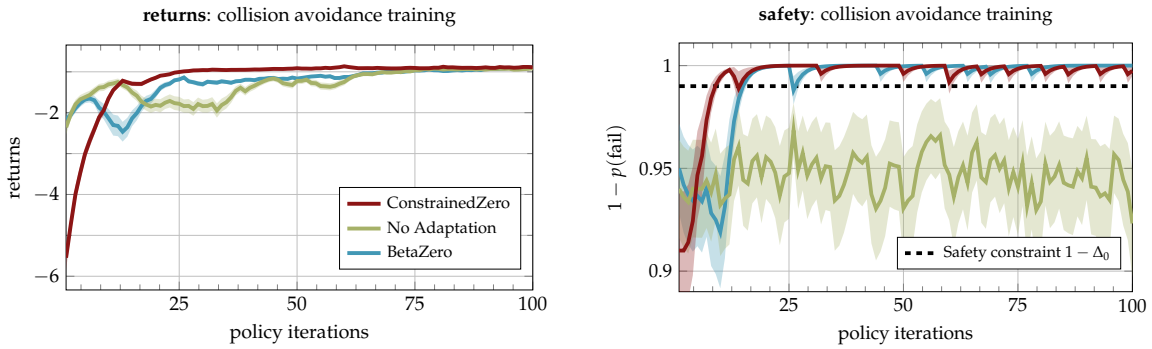
Figure 7.3 compares ConstrainedZero against BetaZero, where BetaZero uses different values of the penalty λ. The penalties were swept between -10 and -1000 with -100 being the standard for the LightDark POMDP (proportional to the goal reward of 100). A target safety level of $\Delta_0 = 0.01$ was chosen for ConstrainedZero. ConstrainedZero exceeds the BetaZero Pareto curve and achieves the target level of safety with a failure probability of 0.01 ± 0.01 computed over 100 episodes. Notably, ConstrainedZero also has less variance in the failure probability and the returns. BetaZero still achieves good performance but at the cost of sweeping the penalty values without explicitly defining a safety threshold to satisfy.

Shown in table 7.1, an ablation study is conducted for ConstrainedZero. Most notably, the adaptation procedure is crucial to enable the algorithm to properly balance safety and utility during planning (also shown in figures 7.4a and 7.4b). When comparing Δ-MCTS without

	LightDark $\Delta_0 = 0.01$		**Collision Avoidance** $\Delta_0 = 0.01$		**Spillpoint CCS** $\Delta_0 = 0.05$	
	$p(\text{fail}) \downarrow$	returns $\uparrow$	$p(\text{fail}) \downarrow$	returns $\uparrow$	$p(\text{fail}) \downarrow$	returns $\uparrow$
ConstrainedZero	$\mathbf{0.01}_{\pm \mathbf{0.01}}$	$\mathbf{13.07}_{\pm \mathbf{0.42}}$	$\mathbf{0.00}_{\pm \mathbf{0.00}}$	$\mathbf{-0.74}_{\pm \mathbf{0.03}}$	$\mathbf{0.05}_{\pm \mathbf{0.02}}$	$\mathbf{2.62}_{\pm \mathbf{0.12}}$
No Adaptation*	$0.66_{\pm 0.05}$	$27.47_{\pm 3.90}$	$0.03_{\pm 0.02}$	$-1.00_{\pm 0.00}$	$0.69_{\pm 0.04}$	$6.18_{\pm 0.36}$
Δ-MCTS (no f_θ)†	$0.01_{\pm 0.01}$	$1.86_{\pm 0.20}$	$0.32_{\pm 0.05}$	$0.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$6.87_{\pm 0.50}$
Raw Policy P_θ	$0.01_{\pm 0.01}$	$12.88_{\pm 0.46}$	$0.00_{\pm 0.00}$	$-0.86_{\pm 0.02}$	$0.06_{\pm 0.02}$	$2.45_{\pm 0.11}$
Raw Value‡ V_θ	$0.72_{\pm 0.05}$	$28.00_{\pm 4.51}$	$0.16_{\pm 0.04}$	$-0.20_{\pm 0.04}$	$0.38_{\pm 0.05}$	$4.27_{\pm 0.30}$
Raw Failure‡ F_θ	$0.80_{\pm 0.04}$	$0.05_{\pm 0.04}$	$0.00_{\pm 0.00}$	$-1.62_{\pm 0.08}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$

All results report the mean ± standard error over 100 seeds, evaluated using the arg max of equation (6.1), i.e., $\tau \to 0$.
* Trained with the same parameters as ConstrainedZero without adaptation, i.e., only a hard constraint on Δ_0.
† Δ-MCTS without the neural network for the value or failure probability and a random policy for CC-PUCT.
‡ One-step look-ahead over all actions using only the value or failure probability network head with 5 obs. per action.

Table 7.1: ConstrainedZero results. Bold indicates best results within Δ_0 threshold.



(a) Returns from policy iteration.

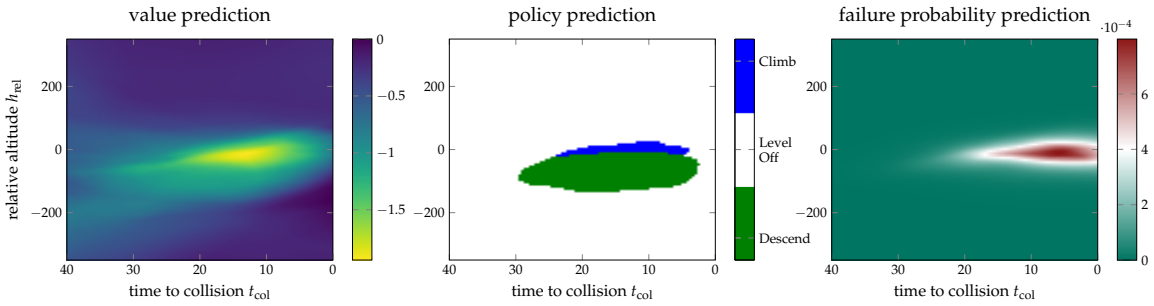
(b) Safety from policy iteration.

Figure 7.4: Utility and safety results for the collision avoidance CC-POMDP.

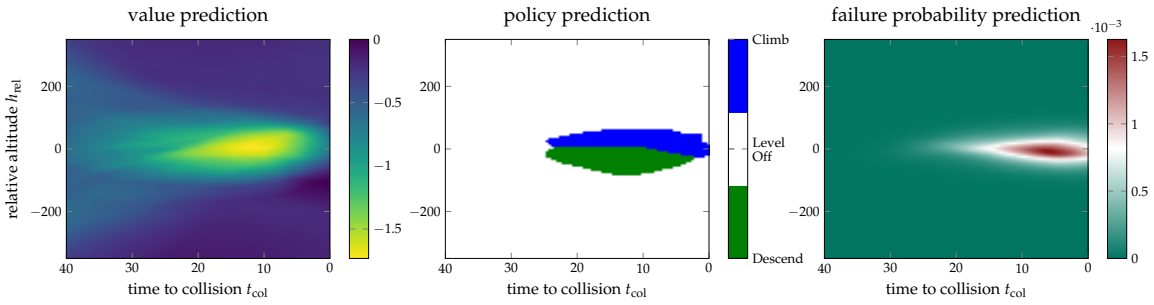
network approximators against ConstrainedZero, it is clear that offline policy iteration allows for better online planning. Using only the raw policy head P_θ achieves good performance, which is trained to imitate the tree policy. However, incorporating additional online planning with the full network yields better results overall, enabling planning over potentially unseen information. The full ConstrainedZero algorithm consistently achieves the highest return within the satisfied target level of safety Δ_0.

7.4.2 ConstrainedZero Learning and Safety Curves

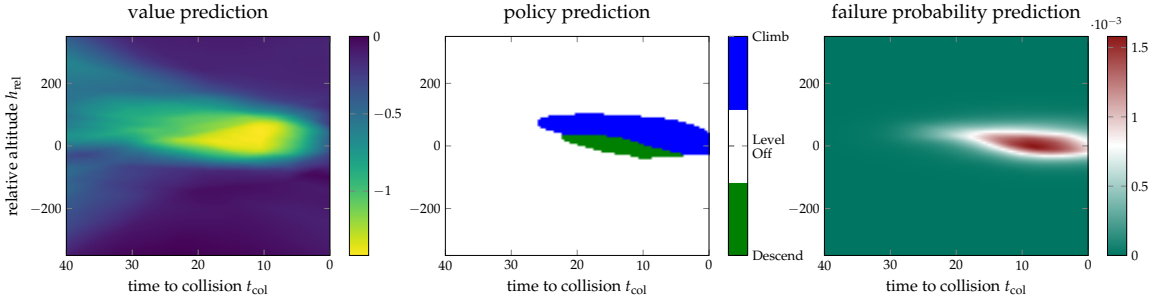
Compared to BetaZero, figure 7.4 highlights that ConstrainedZero satisfies the safety constraint earlier during policy iteration, while simultaneously maximizing returns (shown for the CAS problem). As an ablation, we show that the policy trained without adaptation learns to maximize returns but fails to satisfy the safety constraint. This is because without adaptation, the algorithm will attempt to satisfy an overly conservative fixed constraint, not taking into account the outcomes of its actions. With adaptation, ConstrainedZero adjusts the constraint in response to feedback from the environment, resulting in the algorithm becoming more capable of optimizing its performance within the bounds of the adaptive constraint. This demonstrates the importance of adaptation, as a fixed constraint may be too conservative or too risky, leading to suboptimal decision making.



(a) Learned neural network heads for a belief slice when $\dot{h}_{\text{rel}} = 3$ (climbing).



(b) Learned neural network heads for a belief slice when $\dot{h}_{\text{rel}} = 0$ (level).



(c) Learned neural network heads for a belief slice when $\dot{h}_{\text{rel}} = -3$ (descending).

Figure 7.5: Slices of the collision avoidance neural network heads.

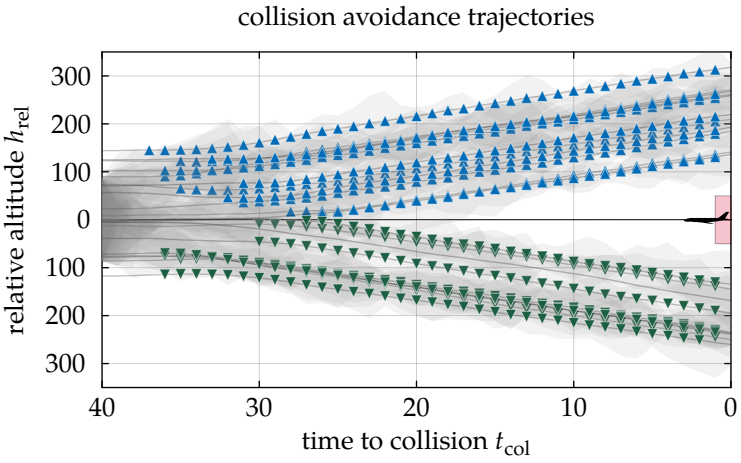


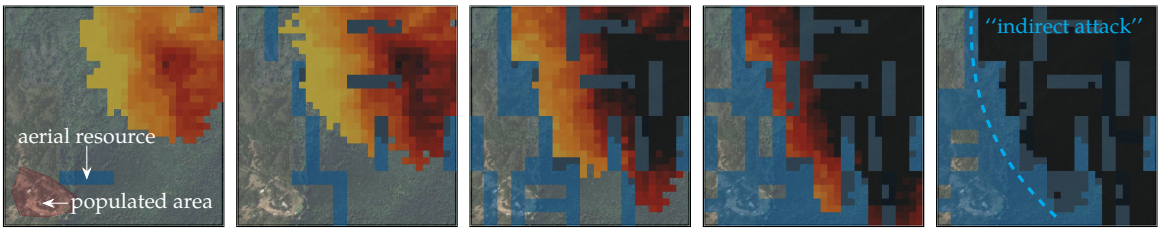
Figure 7.6: Trajectories of the collision avoidance CC-POMDP.

7.4.3 Analyzing the Learned Surrogates: Aircraft Collision Avoidance

Figure 7.5 illustrates a sweep across the belief space for the CAS problem, plotting the three neural network heads: value estimate, action selection policy, and failure probability estimate. Having a surrogate for where the system is likely to fail allows for further safety analysis to be conducted. Plotting several CAS trajectories in figure 7.6 indicates that our learned policy exhibits the ''notch'' behavior demonstrated by Kochenderfer *et al.* [12]. The behavior is evident in the beginning of the trajectories, where the policy tends to withhold alerting until the time to collision becomes smaller, indicating that when the aircraft are near co-altitude (i.e., relative altitude of zero), the policy is waiting for the state uncertainty captured in the belief to decrease, so as to not overly *reverse* the aircraft.

7.4.4 Application to CC-MDPs: Wildfire Resource Allocation

Because we reframe the CC-POMDP problem as a CC-BMDP, this means that Constrained-Zero (and subsequently, Δ-MCTS), can be applied directly to *chance-constrained MDPs* (CC-MDPs). To test this, we adapt the airborne wildfire resource allocation MDP from Griffith *et al.* [27] into a CC-MDP. The reward function applies a cost for every time step that an evolving wildfire is active and a cost for spreading to a nearby populated area (bottom left corner of the maps in figure 7.7). A failure occurs if any fire is spread to the populated area. We set the desired failure probability to be $\Delta_0 = 0.1$. Figure 7.7 shows a learned ConstrainedZero policy applied to the wildfire CC-MDP. We observed that the



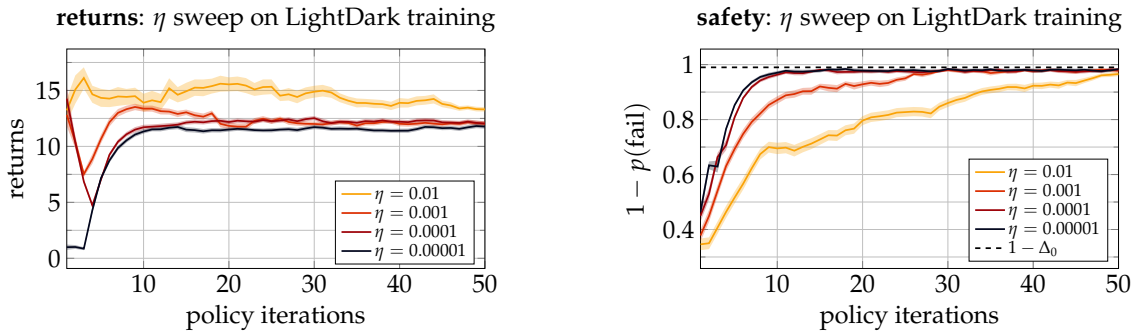
(a) Time 1 (b) Time 10 (c) Time 25 (d) Time 40 (e) Time 65

Figure 7.7: ConstrainedZero policy applied to the wildfire resource allocation CC-MDP.

policy learns to *indirectly* suppress the wildfire by placing suppression resources (in blue) between the wildfire and the populated area. This "indirect attack" is common in wildfire incident management as a strategy to get ahead of the predicted wildfire spread for better containment [26], instead of directly attempting to suppress the active wildfire.

7.4.5 Analysis of ACI Step Size η on Training

To test the sensitivity of ConstrainedZero to the ACI step size η, we swept values of η for the LightDark CC-POMDP (figure 7.8). The step size controls the reactivity of the updates in the adaptation step of Δ-MCTS, where the larger the η, the more reactive the ACI update will be. Figure 7.8a shows that with a larger step size, the reactivity of the threshold update in equation (7.18) opens the safety threshold faster, resulting in more risky behavior at the expense of higher returns. Empirically, we show that ConstrainedZero converges to an approximately Pareto-optimal policy, balancing the desired level of safety with the maximization of utility. Due to its stability, a step size of $\eta = 1 \times 10^{-5}$ was chosen for the



(a) Returns when sweeping η. (b) Safety when sweeping η.

Figure 7.8: Empirical sensitivity analysis of the ACI step size η on offline policy iteration.

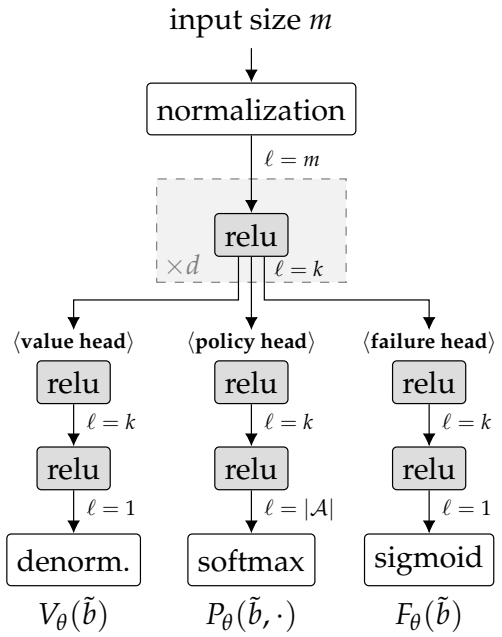


Figure 7.9: Simple network architecture used for each CC-POMDP.

final results in table 7.1. Future research may focus on methods for adapting the step size online during planning, such as the parameter-free method AgACI from Zaffran *et al.* [205].

7.4.6 Neural Network Architectures

The neural network used for each CC-POMDP is a simple fully-connected feedforward network shown in figure 7.9 and based on the network used by BetaZero in chapter 6. For the LightDark problem, an input size of $m = 2$ is used for the mean and standard deviation of the particle filter belief over the y-location state values, with an internal network depth of $d = 2$ and width of $k = 64$. For the CAS problem, an input size of $m = 20$ is used for the mean and covariance of the unscented Kalman filter belief over the state variables of h_{rel}, $\dot{h}_{\text{rel}}$, a_{prev}, and t_{col}, with a network depth of $d = 2$ and width of $k = 64$. Finally, the spillpoint CCS problem uses an input size of $m = 510$ (mean and standard deviation for the top-surface grid points, top-surface heights, porosity, injection locations, and injection depth), with a network depth of $d = 4$ and width of $k = 128$.

Following chapter 6, the value head of the network is trained on the normalized returns so that they roughly lie in the range of $[-1, 1]$. An output denormalization layer is appended

to the value head so that the predictions are properly scaled (done entirely internal to the network). Value normalization is useful for training stability so that the training targets have zero mean and unit variance [187].

Hyperparameters and open-sourced code. The parameters used for ConstrainedZero and Δ-MCTS, along with the code for all the experiments and CC-POMDP environments in this work, are included in an open-source package extending the BetaZero.jl Julia package.‡

7.5 Conclusions

This chapter introduces *ConstrainedZero*, an extension of the BetaZero POMDP planning algorithm to chance-constrained POMDPs. Along with neural network estimates of the value function and action-selection policy, we include a network head that estimates the failure probability given a belief. By formulating the safe planning problem as a CC-POMDP, we select a target level of safety to optimize towards, in contrast to traditional POMDP methods that tune the reward function to balance safety and utility as a multi-objective problem. We develop an extension to Monte Carlo tree search that includes an *adaptation* stage that adjusts the target level of safety during planning with adaptive conformal inference. The resulting Δ-MCTS algorithm modifies MCTS for CC-POMDPs and addresses the issue of overfitting to failure predictions. Additionally, a constrained action-selection criterion (CC-PUCT) was developed to enable planning under constraints.

7.5.1 Limitations

Adapting the safety level when planning with approximations may lead to deviations despite ACI convergence guarantees, but the lack of adaptation also does not guarantee the desired safety. However, our experiments show that this flexibility helps the algorithm find policies matching the targeted safety. Similar to BetaZero, ConstrainedZero may require more computing resources than existing POMDP solvers due to neural network training and parallel Δ-MCTS episodes. However, it is designed for large-scale problems with high-dimensional spaces that require long-horizon planning. We focus on real-world scenarios where transition dynamics, not policy training, are the main challenge, using past experiences to learn an approximately optimal policy offline that is refined online using tree search.

‡`https://github.com/sisl/ConstrainedZero.jl`

Chapter 8

Safety Validation using Probabilistic Surrogates

The road to wisdom?
Well, it's plain and simple to express:
Err
and err
and err again,
but less
and less
and less.

Piet Hein

Once we have a policy that we deem is safe (i.e., from algorithms presented in chapter 7), the next step is to estimate the probability of failure over the entire operating domain. Estimating the probability of failure is an important step in the certification of safety-critical systems. Efficient estimation methods are often needed due to the challenges posed by high-dimensional input spaces, risky test scenarios, and computationally expensive simulators. This chapter frames the problem of black-box safety validation as a Bayesian optimization problem and introduces a method that iteratively fits a probabilistic surrogate model to efficiently predict failures. The algorithm is designed to search for failures, compute the most-likely failure, and estimate the failure probability over an operating domain using importance sampling. We introduce three acquisition functions that aim to reduce uncertainty by covering the design space, optimize the analytically derived failure boundaries, and sample the predicted failure regions. Results show our *Bayesian safety validation* approach provides

a more accurate estimate of failure probability with orders of magnitude fewer samples compared to baselines, and performs well across various validation metrics. We test the method on three toy problems, a stochastic POMDP, and a neural network-based runway detection system. This work is currently being used to supplement the FAA certification process of the machine learning components for an autonomous cargo aircraft [25].

8.1 Motivation

Certifying safety-critical autonomous systems is an important step for their safe deployment in domains such as aviation. Examples of safety-critical aviation systems include those for detect and avoid [206, 207], collision avoidance [12], runway detection [208], and auto-land [209]. One way to provide a quantitative measure of safety is to estimate the probability of system failure. The process of estimating the probability of failure can highlight areas of weakness in the system (by uncovering failures) and can show how well the system performs in their operating environments. The rarity of failures makes it challenging to accurately estimate failure probability, especially when using computationally expensive simulators [210]. Therefore, it is important to efficiently sample the design space when searching for failures (using a minimum set of inputs) and to maximize a measure of confidence in the resulting failure probability estimate.

A standard approach for estimating this rare-event probability involves using Monte Carlo (MC) sampling to generate a set of system inputs from a likelihood model of the operating environment. Estimating this rare-event probability through MC sampling can be computationally expensive and usually requires a large number of samples to minimize the variance of the estimate [210]. A variance-reduction technique to more efficiently estimate the failure probability uses *importance sampling* [211, 212] to draw samples from a different distribution, called the *proposal*, and then re-weight the expectation based on the likelihood ratio between the operational model and the proposal. Importance sampling is especially useful in the safety-critical case due to its unbiased failure probability estimate [212].

Bayesian optimization algorithms such as the cross-entropy method (CEM) [213, 214] have been adapted to the problem of rare-event estimation through a multi-level procedure [210, 215], but rely on a real-valued system output with a defined failure threshold to adaptively narrow the search. Arief *et al.* [216] proposed a deep importance sampling approach for rare-event estimation of black-box systems (Deep-PrAE), but rely on similar

real-valued systems. In our problem, the system under test outputs a binary value indicating failure and thus cannot effectively use these methods.

Population-based methods, like population Monte Carlo (PMC) [217] and optimized population Monte Carlo (O-PMC) [218], work well for both real-valued and binary-valued systems and use adaptive importance sampling [219] to iteratively estimate the optimal proposal distribution. The PMC algorithms use self-normalized importance sampling (SNIS) to estimate the probability in question [212]. Population-based approaches often require a large number of system evaluations to adequately converge (see Luengo *et al.* [220] for a comprehensive survey of Monte Carlo estimation algorithms).

Vazquez *et al.* [221] and Wang *et al.* [222] consider the problem of failure probability estimation when dealing with computationally expensive systems. They fit a Gaussian process surrogate model to the underlying real-valued system (i.e., not the system output indication of failure) and then estimate the failure probability over this surrogate, similar to work from Renganathan *et al.* [223] for the multifidelity case. These methods may not work on binary-valued systems or scale to complex systems such as image-based neural networks. He *et al.* [224] propose a framework for analyzing safety-critical deep neural networks using Bayesian statistics to iteratively fit a decision boundary from a predefined dictionary of shapes. They use a boundary acquisition function that is based on expected improvement [225], requiring a definition of an ϵ-threshold around the predicted boundaries at $0.5 \pm \epsilon$. Our proposed approach constructs a probabilistic surrogate model such that a failure boundary can be analytically derived.

With the goal of sample efficiency, this chapter reformulates the safety validation problem [15] as a Bayesian optimization problem [72, 74, 226] and introduces a set of acquisition functions, each with their own safety validation objective. Applying a Bayesian approach allows us to fit a probabilistic surrogate model to a minimal set of design points evaluated from the true system and then estimate failure probability using importance sampling on the inexpensive surrogate. As a real-world case study, we use the proposed algorithm to estimate the failure probability for a neural network-based runway detection system where the design space consists of the glide slope angle and the distance to runway. The parametric design space is used to generate an input image of a runway in simulation, conditioned on the knowledge that the aircraft is on an approach, and the output is a binary value of failure. We classify misdetections as failures, knowing that the runway detection system is only active on an approach to land.

The goals of this work are to: (1) estimate the probability of failure for a black-box safety-critical system, (2) focus on sample efficiency using a minimal number of data points, (3) find realistic cases using a model of the environment the system will be operating in, weighting the failures based on their operational likelihood, (4) characterize the entire set of failure regions to identify model weaknesses for further development, and (5) ensure the entire design space is adequately covered. The proposed *Bayesian safety validation* (BSV) algorithm can be applied to general black-box systems to find failures, determine the most-likely failure, and estimate the overall failure probability—thus, satisfying the three safety validation tasks of *falsification*, *most-likely failure analysis*, and *failure probability estimation* [15]. An open-source Julia framework* was developed to extend this work to other black-box systems and reproduce the results in this chapter.

8.2 Problem Formulation

The Gaussian process (GP) is a probabilistic surrogate model that is particularly useful in black-box Bayesian optimization [70, 72]. In this chapter, we develop our method around the GP surrogate model to be used as a way to predict failure probabilities of an expensive black-box system. Following the GP review in the background section 3.4.1, we introduce a straightforward way to predict probabilities using a GP and develop three acquisition functions that satisfy the safety validation tasks presented in the background section 3.5.

8.2.1 Predicting Probabilities using Gaussian Process

Because our system f returns discrete values in $\{0, 1\}$ and we want to predict a real-valued probability in $[0, 1]$, we consider this a binary classification problem [227, 228]. We construct the GP to predict the logits $\hat{\mathbf{z}}$ (which we define with zero mean to indicate no prior knowledge of failures) and then apply the logistic function (i.e., sigmoid) to get the predictions $\hat{\mathbf{y}}$:

$$\hat{\mathbf{z}} \mid \text{logit}(\mathbf{y}) \sim \mathcal{N}\big(\boldsymbol{\mu}(\mathbf{X}, \mathbf{X}', \text{logit}(\mathbf{y})), \boldsymbol{\Sigma}(\mathbf{X}, \mathbf{X}')\big) \tag{8.1}$$

$$\text{logit}(y_i) = \log\left(\frac{\phi(y_i)}{1 - \phi(y_i)}\right) / s \tag{8.2}$$

$$\hat{\mathbf{y}} = \phi^{-1}\Big(\text{logit}^{-1}(\hat{\mathbf{z}})\Big) = \phi^{-1}\left(\frac{1}{1 + \exp(-s\hat{\mathbf{z}})}\right) \tag{8.3}$$

*https://github.com/sisl/BayesianSafetyValidation.jl

where $\phi(y_i) = y_i(1-\epsilon) + (1-y_i)\epsilon$ and $\phi^{-1}(\hat{y}_i) = (\hat{y}_i - \epsilon)/(1-2\epsilon)$ to ensure well defined logits and s controls the steepness of the sigmoid curve. This construction can still be used even if f already outputs values in $[0, 1]$ instead of binary indicators; the GP will fit directly to the provided failure probability of each point. When the output is binary, applying the logit transformations ensure that the prediction lies in $[0, 1]$ and can be interpreted probabilistically. Other approaches for predicting a probability using a Gaussian process explore the case where f is bounded and can be modeled as a Beta distribution [229]. The logit approach allows us to analytically derive failure boundaries.

We frame the black-box safety validation problem as a Bayesian optimization problem and use a Gaussian process surrogate model to predict failures. Bayesian optimization is a natural approach to optimize some function $f : \mathbb{R}^n \to \mathbb{R}$, e.g., a black-box system. But our problem uses a function $f : \mathbb{R}^n \to \mathbb{B}$, where $\mathbb{B}$ represents the Boolean domain (returning `true` for failures and `false` for non-failures, which can be interpreted as 1 and 0, respectively). Instead of maximizing or minimizing f, we frame the problem to find failure regions through exploration, refine failure boundaries, and refine likely failure regions through sampling the theoretically optimal failure distribution [1, 212, 230]. We introduce a set of three acquisition functions that accomplish these objectives and call the acquisition procedure *failure search and refinement* (FSAR), shown together in algorithm 8.1. Although we are primarily interested in the more restrictive case where f outputs a Boolean, our approach also works when f outputs a probabilistic value of failure (demonstrated in section 8.5.6). Throughout, we use the fact that the surrogate model provides a probabilistic interpretation of the failure predictions regardless of the type of system outputs.

Uncertainty exploration. To find failures and cover the design space $\mathcal{X}$, we want to explore areas with high uncertainty and high operational likelihood. The first proposed acquisition function searches over the uncertainty provided by the Gaussian process $\hat{\sigma}(\mathbf{x})$ weighted by the operational model $p(\mathbf{x})$ to find likely points $\mathbf{x} \in \mathcal{X}$ with maximal uncertainty:

$$\mathbf{x}_1' = \arg\max_{\mathbf{x} \in \mathcal{X}} \hat{\sigma}(\mathbf{x}) p(\mathbf{x})^{1/\alpha t} \tag{8.4}$$

The influence of the operational model is decayed by $1/\alpha t$. This will ensure that the design space $\mathcal{X}$ is fully explored in the limit [70], noting that in practice the limiting factor is

the $\mathcal{O}(n^3)$ time for the Gaussian process to fit n data points due to the $n \times n$ matrix inversion [231]. Equation 8.4 may also be sampled instead of taking the arg max where the concentration of the distribution can be controlled by a temperature parameter τ:

$$\mathbf{x}'_1 \sim \left(\hat{\sigma}(\mathbf{x})p(\mathbf{x})^{1/\alpha t}\right)^{1/\tau} \tag{8.5}$$

This distribution is normalized over the domain to compute the proposal likelihood $q(\mathbf{x}_i)$ of each sample. The likelihood is used to compute the weight $w(\mathbf{x}_i) = p(\mathbf{x}_i)/q(\mathbf{x}_i)$ for self-normalized importance sampling (see section 8.2.2).

Boundary refinement. To better characterize the areas of all failure regions, we want to refine the known failure boundaries to tighten them as much as possible. Because our surrogate $\hat{f}(\mathbf{x})$ is modeled as a logistic function (shown in equation (8.3)), we can take the derivative and get the analytical form as:

$$\mu'(\mathbf{x}) = \hat{f}(\mathbf{x})(1 - \hat{f}(\mathbf{x})) \tag{8.6}$$

where $\mu'(\mathbf{x})$ is maximal when $\hat{f}(\mathbf{x}) = 0.5$, thus giving us the failure boundary at the peaks. Therefore, the second proposed acquisition function selects the point that maximizes the upper confidence of μ' to refine the failure boundary:

$$\mathbf{x}'_2 = \arg\max_{\mathbf{x}\in\mathcal{X}} \left(\mu'(\mathbf{x}) + \lambda\hat{\sigma}(\mathbf{x})\right) p(\mathbf{x})^{1/\alpha t} \tag{8.7}$$

where upper confidence provides an over estimation, and we set $\lambda = 0.1$ in our experiments. The operational model $p(\mathbf{x})$ is used to first focus on the failure boundary with high operational likelihood, then decay the emphasis of the likelihood as a function of the current iteration t (here using an inverse decay of $1/\alpha t$). This will first acquire likely points along the boundaries, then refine all of the boundaries because as $t \to \infty$ then $p(\mathbf{x})^{1/\alpha t} \to 1$.

Similar to equation (8.5), we may also choose to sample along the boundary in equation (8.7) with a temperature parameter τ and compute the weights after normalization:

$$\mathbf{x}'_2 \sim \left(\left(\mu'(\mathbf{x}) + \lambda\hat{\sigma}(\mathbf{x})\right) p(\mathbf{x})^{1/\alpha t}\right)^{1/\tau} \tag{8.8}$$

Sampling these acquisition functions may be advantageous when the black-box system is stochastic, thus making the failure boundaries noisy (sections 8.4.2 and 8.5.5).

Failure region sampling. The optimal importance sampling distribution is $q_{\text{opt}} \propto f(\mathbf{x})p(\mathbf{x})$, which, intuitively, is the distribution of failures (when $f(\mathbf{x}) = 1$) over the likely region (weighed by $p(\mathbf{x})$) [1, 212, 230]. Yet this is exactly what we are trying to estimate, and sampling this distribution may require a prohibitive number of evaluations of f. Therefore, the third proposed acquisition function uses the surrogate to get the upper confidence of the failure prediction:

$$\hat{h}(\mathbf{x}) = \hat{f}(\mathbf{x}) + \lambda\hat{\sigma}(\mathbf{x}) \tag{8.9}$$

$$\hat{g}(\mathbf{x}) = \mathbb{1}\{\hat{h}(\mathbf{x}) \geq 0.5\} \tag{8.10}$$

and then using the estimated failure region $\hat{g}$, we draw a sample from the distribution:

$$\mathbf{x}_3' \sim \hat{g}(\mathbf{x})p(\mathbf{x}). \tag{8.11}$$

Here, we use the indicator function $\mathbb{1}\{\cdot\}$, which returns 1 when the input is `true` and 0 otherwise. Sampling from the approximate failure distribution defined by the surrogate helps refine likely failure regions to ensure a better estimate of the probability of failure. If the system f outputs a failure probability value in $[0, 1]$ instead of a binary indicator, then we can use this information and sample from the distribution that weights towards higher confidence failures:

$$\mathbf{x}_3' \sim \hat{g}(\mathbf{x})\hat{h}(\mathbf{x})p(\mathbf{x}) \tag{8.12}$$

The proposed acquisition functions work under a more restrictive binary system $f : \mathbb{R}^n \to \mathbb{B}$ and a system $f : \mathbb{R}^n \to [0, 1]$ that outputs a probabilistic value of failure (which can be interpreted as confidence or stochasticity). We define *failure region sampling* using the more general distribution in equation (8.12) because it works for both types of system outputs. When a granular measure of system failure is available, it can be used to make a more informative surrogate model. If only binary failure information is available, developers could simply focus on those failures with high likelihood. Applying to binary-valued systems is more general and thus the primary focus of this work, but we demonstrate on a probability-valued case in section 8.5.6. In the case when no failures are predicted (i.e., $\hat{g}(\mathbf{x}) = 0$, $\forall \mathbf{x} \in \mathcal{X}$), then we sample directly from $\hat{h}$.

Algorithm 8.1 describes the *failure search and refinement* (FSAR) procedure to compute the subsequent points from the three proposed acquisition functions. Figure 8.1 provides

an illustrative example of the probabilistic surrogate model with predicted failures (shown in red) and the *failure search and refinement* acquisition functions after $T = 30$ iterations (with $N = 90$ data points, showing true observations as red/green squares). Lighter colors indicate maximums, and red circles indicate the next selected point. The operational likelihood model $p(\mathbf{x})$ is shown as marginal distribution subplots. Notice the low uncertainty around the likely failure region and the influence of $p(\mathbf{x})$ on the boundary refinement; the algorithm first refines the likely boundary, and then refines the entire boundary in the limit. The system under test is an example system shown in figure 8.3a.

Algorithm 8.1: Failure search and refinement acquisition functions.

function FAILURESEARCHANDREFINEMENT($\mathcal{GP}, p, t$)

$\hat{f} \leftarrow$ MEANFUNCTION($\mathcal{GP}$)

$\hat{\sigma} \leftarrow$ STANDARDDEVIATIONFUNCTION($\mathcal{GP}$)

$$\mathbf{x}'_1 \leftarrow \begin{cases} \mathbf{x}' \leftarrow \arg\max_{\mathbf{x}\in\mathcal{X}} \hat{\sigma}(\mathbf{x}) p(\mathbf{x})^{1/\alpha t} & \text{if } \tau = 0 \\ \mathbf{x}' \sim \left(\hat{\sigma}(\mathbf{x}) p(\mathbf{x})^{1/\alpha t}\right)^{1/\tau} & \text{otherwise} \end{cases} \Bigg\} \text{uncertainty exploration}$$

$\mu'(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x})(1 - \hat{f}(\mathbf{x}))$

$$\mathbf{x}'_2 \leftarrow \begin{cases} \mathbf{x}' \leftarrow \arg\max_{\mathbf{x}\in\mathcal{X}} \left(\mu'(\mathbf{x}) + \lambda\hat{\sigma}(\mathbf{x})\right) p(\mathbf{x})^{1/\alpha t} & \text{if } \tau = 0 \\ \mathbf{x}' \sim \left(\left(\mu'(\mathbf{x}) + \lambda\hat{\sigma}(\mathbf{x})\right) p(\mathbf{x})^{1/\alpha t}\right)^{1/\tau} & \text{otherwise} \end{cases} \Bigg\} \text{boundary refinement}$$

$\hat{h}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \lambda\hat{\sigma}(x)$

$\hat{g}(\mathbf{x}) \leftarrow \mathbb{1}\{\hat{h}(\mathbf{x}) \geq 0.5\}$

$$\mathbf{x}'_3 \leftarrow \begin{cases} \mathbf{x}' \sim \hat{h}(\mathbf{x}) & \text{if } \hat{g}(\mathbf{x}) = 0,\ \forall \mathbf{x} \in \mathcal{X} \\ \mathbf{x}' \sim \hat{g}(\mathbf{x})\hat{h}(\mathbf{x}) p(\mathbf{x}) & \text{otherwise} \end{cases} \Bigg\} \text{failure region sampling}$$

$\mathbf{w}' \leftarrow$ COMPUTEWEIGHTS($\mathbf{x}'_1, \mathbf{x}'_2, \mathbf{x}'_3$)

return $\{\mathbf{x}'_1, \mathbf{x}'_2, \mathbf{x}'_3\}, \mathbf{w}'$

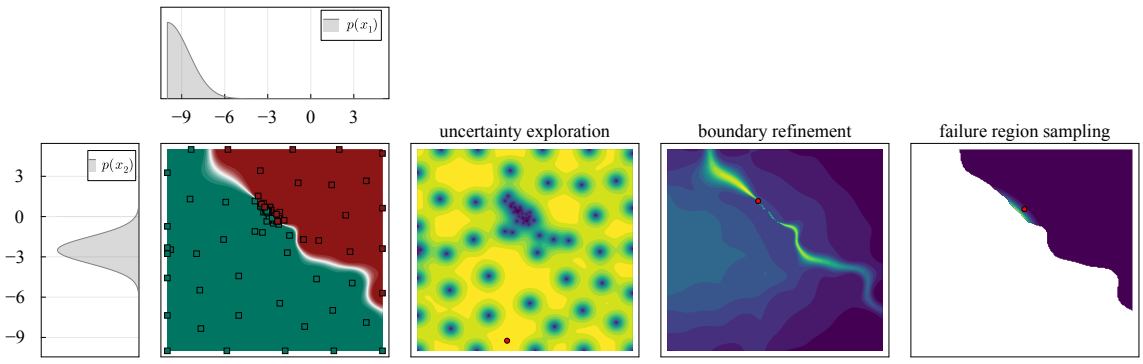


Figure 8.1: Illustrating 90 steps of the *failure search and refinement* acquisition functions.

8.2.2 Importance Sampling Estimate of Failure Probability

To compute an efficient and unbiased estimate of the probability of failure, we use *importance sampling* [212]. Probability estimation can be defined as computing the expectation of the Boolean-valued function f over the *target*, or *nominal distribution*, p (what we call the *operational likelihood model* in this work) as:

$$\mathbb{P}[f(\mathbf{x})] = \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[f(\mathbf{x})] = \int_{\mathcal{X}} p(\mathbf{x}) f(\mathbf{x})\, dx. \tag{8.13}$$

In general, the expectation of the indicator function of an event A, denoted $\mathbb{1}\{A\}$, is equal to the probability of that event occurring $\mathbb{E}[\mathbb{1}\{A\}] = \mathbb{P}[A]$. In our problem, we define $f : \mathbb{R}^n \to \mathbb{B}$ as a Boolean-valued function for convenience. Nevertheless, the following work could easily be extended to a real-valued function $v : \mathbb{R}^n \to \mathbb{R}$ where failures are defined by violating some safety threshold c, i.e., $f(\mathbf{x}) = \mathbb{1}\{v(\mathbf{x}) \geq c\}$. Now, to approximate the expectation—and therefore the probability of failure—we can use n MC samples from p:

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim p}[f(\mathbf{x})] \approx \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_i) \tag{8.14}$$

If failures are rare under the distribution p (i.e., $f(\mathbf{x}_i)$ is rarely equal to 1 when $\mathbf{x}_i \sim p$), then we may need an extremely large number of samples from p to get an accurate estimate. But this would require prohibitively many system evaluations of f. Instead, importance sampling states that we can sample from some other distribution q, called the *proposal distribution*, and re-weight the outputs of f based on the *likelihood ratio* $p(\mathbf{x})/q(\mathbf{x})$ [212]:

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim p}[f(\mathbf{x})] = \mathop{\mathbb{E}}_{\mathbf{x}\sim q}\left[\frac{p(\mathbf{x})}{q(\mathbf{x})} f(\mathbf{x})\right] \tag{8.15}$$

$$\approx \frac{1}{n}\sum_{i=1}^{n} \frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i)} f(\mathbf{x}_i) \tag{8.16}$$

Now we can use samples from q to approximate the probability over p. However, selecting an effective proposal distribution can be challenging (see Bugallo *et al.* [219]). Based on whether the black-box system is stochastic or deterministic, we can employ different importance sampling methods to estimate the proposal distribution.

Discrete proposal. In the case of deterministic systems, where every input $\mathbf{x}$ maps deterministically to an output y, we could use a uniform proposal over the design space $q = \mathcal{U}_{\mathcal{X}}$ and replace the expensive function calls to f with inexpensive evaluations of the surrogate $\hat{f}$ using orders of magnitude more samples. We let $\hat{g}(\mathbf{x}) = \mathbb{1}\{\hat{f}(\mathbf{x}) \geq 0.5\}$ to indicate failures predicted by the surrogate model. Thus, our problem gets simplified to estimating:

$$\hat{p}_{\text{fail}} = \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[\hat{g}(\mathbf{x})] = \mathop{\mathbb{E}}_{\mathbf{x}\sim q}\left[\frac{p(\mathbf{x})}{q(\mathbf{x})}\hat{g}(\mathbf{x})\right] \approx \frac{1}{n}\sum_{i=1}^{n}\frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i)}\hat{g}(\mathbf{x}_i). \tag{8.17}$$

Using a uniform distribution can induce variance in the estimate [2]; therefore, an even further simplification is to use a discretized set of n points $\bar{\mathcal{X}}$ over the range $\mathcal{X}$ as our proposal. We assigned equal likelihood to each point $\mathbf{x}_i \in \bar{\mathcal{X}}$, namely $q(\mathbf{x}_i) = 1/n\sum_{j=1}^{n} p(\mathbf{x}_j)$. Then equation (8.17) becomes:

$$\hat{p}_{\text{fail}} \approx \frac{1}{n}\sum_{i=1}^{n}\frac{p(\mathbf{x}_i)}{1/n\sum_{i=1}^{n} p(\mathbf{x}_i)}\hat{g}(\mathbf{x}_i) = \frac{\sum_{i=1}^{n} p(\mathbf{x}_i)\hat{g}(\mathbf{x}_i)}{\sum_{i=1}^{n} p(\mathbf{x}_i)} = \frac{\mathbf{w}^\top\hat{\mathbf{y}}}{\sum_{i=1}^{n} w_i} \tag{8.18}$$

where $\mathbf{w} = [p(\mathbf{x}_1), \ldots, p(\mathbf{x}_n)]$ and $\hat{\mathbf{y}} = [\hat{g}(\mathbf{x}_1), \ldots, \hat{g}(\mathbf{x}_n)]$ for $\mathbf{x}_i \in \bar{\mathcal{X}}$. Here, we are using *likelihood weighting*, which is a special case of importance sampling [1, 2]. Using a discrete set of points as the proposal distribution has lower variance than sampling the uniform space, but may not scale well to higher dimensions. For this chapter, we use a simplified 500×500 discrete grid as the proposal for two-dimensional systems. To incorporate better proposal distributions when scaling to higher dimensions, see Bugallo *et al.* [219] for adaptive importance sampling methods.

Self-normalizing importance sampling. To address the case of stochastic systems, where the output of the system is a random variable, we use the surrogate model to guide the search based on the FSAR acquisition functions and collect the weights $w(\mathbf{x})$ from the sampled points of the acquisition functions to estimate the failure probability as:

$$\hat{p}_{\text{fail}} = \frac{\sum_{i=1}^{n} w(\mathbf{x}_i)y_i}{\sum_{j=1}^{n} w(\mathbf{x}_j)} \tag{8.19}$$

which is the *self-normalized importance sampling estimate* (SNIS) and will equal the true target value in the limit [212]. The variance estimate is computed as:

$$\hat{\mathrm{Var}}(\hat{p}_{\text{fail}}) = \sum_{i=1}^{n} \bar{w}(\mathbf{x}_i)^2 (y_i - \hat{p}_{\text{fail}})^2 \tag{8.20}$$

where $\bar{w}(\mathbf{x}_i) = w(\mathbf{x}_i) / \sum_{j=1}^{n} w(\mathbf{x}_j)$ and the 99% confidence interval is $\hat{p}_{\text{fail}} \pm 2.58\sqrt{\hat{\mathrm{Var}}(\hat{p}_{\text{fail}})}$. Because failures are stochastic, applying the self-normalized weights to the true system outputs $y_i \in \mathbf{Y}$ means we can estimate the failure probability over observed failures and compute proposal weights for only the observed points using the surrogate, thus avoiding computing the surrogate over a large discrete grid proposal.

8.3 Proposed Algorithm: Bayesian Safety Validation

The proposed *Bayesian safety validation* (BSV) algorithm takes as input the black-box system $f : \mathbb{R}^n \to \mathbb{B}$ or $f : \mathbb{R}^n \to [0, 1]$ and an operational likelihood model $p : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ for inputs $\mathbf{x} \in \mathbb{R}^n$ of some parametric space, and iteratively refits a probabilistic surrogate model given selected points from the FSAR acquisition functions (section 8.2). The first acquisition, *uncertainty exploration*, explores areas with high uncertainty to provide coverage and search for failure regions. The next acquisition, *boundary refinement*, selects operationally likely points that refine the failure boundaries to better characterize likely failure regions (and includes a decaying weighted operational likelihood to refine all failure boundaries in the limit). The final acquisition, *failure region sampling*, is based on the theoretically optimal q-proposal distribution [230] and will sample from the likely failure regions to ensure a better estimate of the probability of failure. After the algorithm runs for T iterations, a total of $3T$ sampled points were used to fit the surrogate model. The three safety validation tasks are then computed (lines 9–11 of alg. 8.2). Falsification and most-likely failure analysis use only the true observations $y_i \in \mathbf{Y}$ and actual inputs $\mathbf{x}_i \in \mathbf{X}$ to find the inputs that led to failures and the most-likely failure, respectively. Then the final surrogate model or the weights are used to efficiently compute an importance sampling estimate of the failure probability. Algorithm 8.2 describes the BSV algorithm, and figure 8.2 illustrates the process.

Algorithm 8.2: Bayesian safety validation algorithm.

function BAYESIANSAFETYVALIDATION(f, p, T)
  $\mathcal{GP} \leftarrow$ INITIALIZEGAUSSIANPROCESS(m, k)
  $\mathbf{X}, \mathbf{Y}, \mathbf{W} \leftarrow \emptyset, \emptyset, \emptyset$
  for $t \leftarrow 1$ **to** T
    $\mathbf{X}', \mathbf{W}' \leftarrow$ FAILURESEARCHANDREFINEMENT($\mathcal{GP}, p, t$) ▷ select new design points
    $\mathbf{Y}' \leftarrow \{f(\mathbf{x}') \mid \mathbf{x}' \in \mathbf{X}'\}$ ▷ evaluate true system f across design points
    $\mathbf{X}, \mathbf{Y}, \mathbf{W} \leftarrow \mathbf{X} \cup \mathbf{X}', \mathbf{Y} \cup \mathbf{Y}', \mathbf{W} \cup \mathbf{W}'$ ▷ append to input, output, and weight sets
    $\mathcal{GP} \leftarrow$ FIT($\mathcal{GP}, \mathbf{X}, \mathbf{Y}$) ▷ refit surrogate model over all points
  $\mathbf{X}_{\text{fail}} \leftarrow \{\mathbf{x}_i \mid \mathbf{x}_i \in \mathbf{X},\ y_i \in \mathbf{Y},\ \mathbb{1}\{y_i\}\}$ ▷ (1) falsification
  $\mathbf{x}^* \leftarrow \arg\max_{\mathbf{x}_i \in \mathbf{X}, y_i \in \mathbf{Y}} p(\mathbf{x}_i)\mathbb{1}\{y_i\}$ ▷ (2) most-likely failure
  $\hat{p}_{\text{fail}} \leftarrow \begin{cases} \frac{1}{n}\sum_{i=1}^{n} \frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i)} \mathbb{1}\left\{\hat{f}(\mathbf{x}_i) \geq 0.5\right\} & \text{if deterministic} \\ \left(\sum_{i=1}^{n} \mathbf{W}_i \mathbf{Y}_i\right) / \sum_{i=1}^{n} \mathbf{W}_i & \text{if stochastic} \end{cases}$ ▷ (3) failure probability est.
  return $\mathbf{X}_{\text{fail}}, \mathbf{x}^*, \hat{p}_{\text{fail}}$ ▷ return all three safety validation tasks

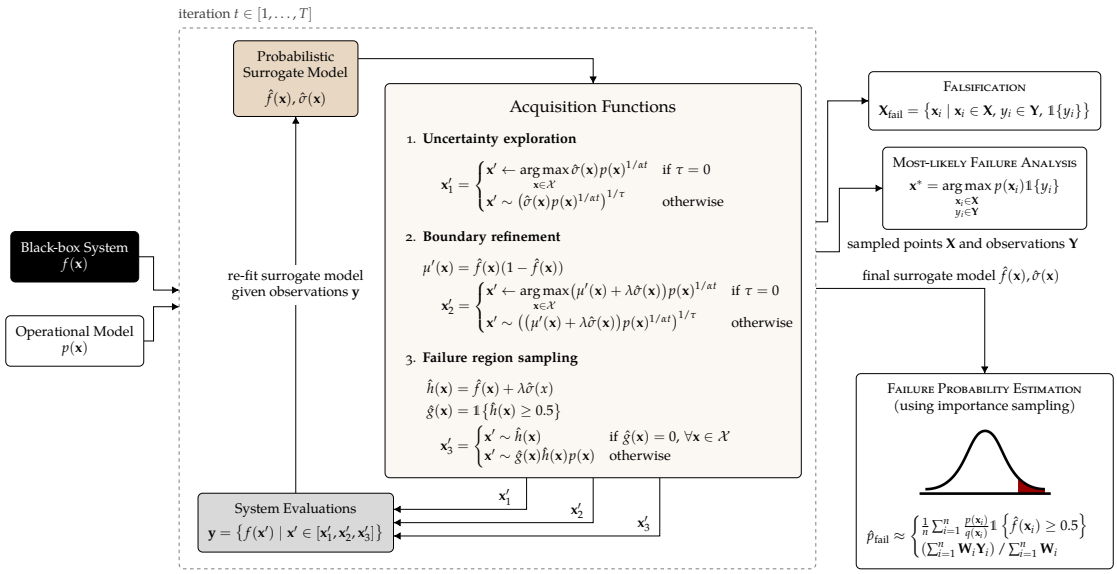


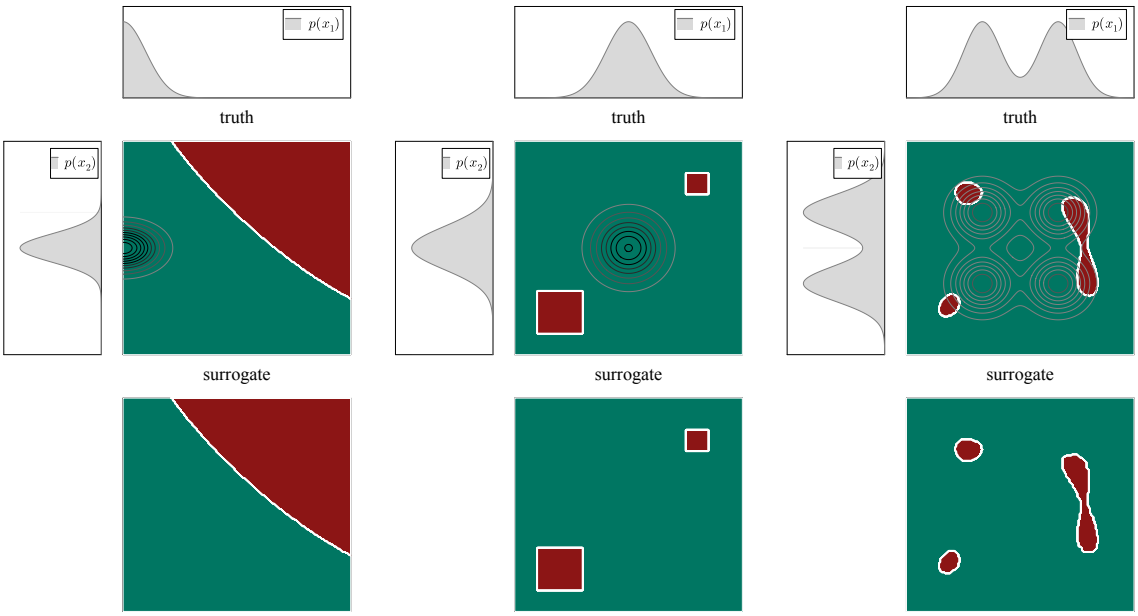
Figure 8.2: *Bayesian safety validation* used for all three safety validation tasks.

8.4 Experiments

To test the effectiveness of BSV, we ran experiments across several different test problems with varying numbers of failure modes, a stochastic sequential decision making system, and a real-world case study using a prototype neural network-based runway detection system.

8.4.1 Simplified Test Problems

Three example toy problems with access to the true value of p_{fail} were used for testing. The first problem (called REPRESENTATIVE) was chosen based on the observed shape of the failure region of the runway detection system, which is our primary case study and a system for which we do not have access to the true failure probability. The representative toy problem is modeled using Booth's function [72]: $f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \leq 200$, thresholded to make this a binary function. We define the operational parameters to be over the range $[-10, 5]$ for both x_1 and x_2 and set the operational likelihood model as $x_1 \sim \mathcal{N}_{\text{trunc}}(-10, 1.5; [-10, 5])$ and $x_2 \sim \mathcal{N}(-2.5, 1)$ where $\mathcal{N}_{\text{trunc}}(\mu, \sigma; [a, b])$ is the normal distribution truncated between $[a, b]$. The second toy problem (called SQUARES) has two,



(a) Representative problem. (b) Two square failure modes. (c) Mixture model, three modes.

Figure 8.3: Failure regions and operational models for the three test problems.

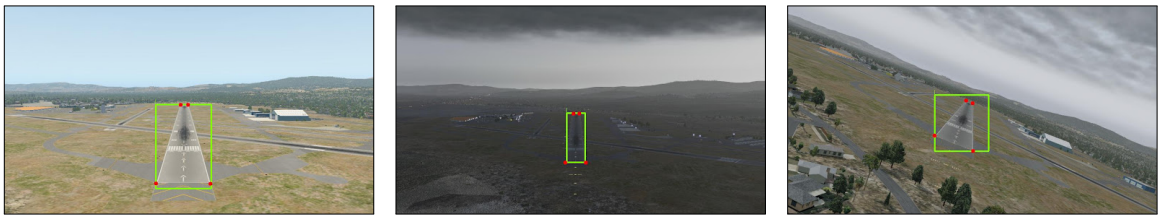
square, disjoint failure regions to test the exploration of BSV and refinement of rigid failure boundaries. The operational parameters are over the range $[0, 10]$ for both x_1 and x_2, each with the operational likelihood model of $\mathcal{N}(5, 1)$. The third toy problem (called MIXTURE) has three, smooth, disjoint failure regions and is designed to test the failure region refinement characteristic of BSV using a multimodal operational model. The operational range is over $[-6, 6]$ with identical Gaussian mixture models that have two equal components of $\mathcal{N}_{\text{trunc}}(2, 1; [-6, 6])$ and $\mathcal{N}_{\text{trunc}}(-2, 1; [-6, 6])$. Similar to the representative example, we define this last problem as the thresholded Himmelblau function [232]: $f(\mathbf{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \leq 15$. The test problems and their operational models are shown in figure 8.3, where failure regions (in red) are shown for each test problem. Operational models are illustrated as subplots and contours, and the true system is shown above the surrogate model failure classification (fit with 999 samples using BSV).

8.4.2 Stochastic Sequential Decision Making System

BSV also works in cases where system failures are stochastic, meaning that the same input could lead to a different failure outcome. To test a stochastic system, we benchmark on the *LightDark* sequential decision making problem [30], modeled as a POMDP. In the LightDark POMDP, the agent to moves `up` or `down` by one to find the light region at $y = 10$ and receives noisy observations of its true position as a function of the distance to the light region. The agent will receive a large reward of 100 when executing the `stop` action at ± 1 of the origin. If the agent stops outside the origin, then it receives a large penalty of -100. We test BSV under two cases: (1) a rare failure event when the agent never executes `stop`, and (2) a non-rare failure event when either the agent never executes `stop` or the `stop` action is executed outside the origin. The problem is one-dimensional and we use the initial state distribution of $\mathcal{N}(2, 3)$ as the operational model.

8.4.3 Neural Network-Based Runway Detection System

As a real-world case of a safety-critical subsystem, we chose a common application in autonomous flight: runway detection (RWD) using neural networks. Synthetic images were generated using the flight simulator X-Plane [233], sampled over different parameters of the approach to land (e.g., glide slope angle and distance to runway). We search over this parametric space instead of dealing directly in pixel or image space. We use an operational



(a) Nominal conditions. (b) Low-likely weather. (c) Potentially dangerous to fly.

Figure 8.4: Simulated runway conditions in X-Plane input to the runway detection system.

model of likely glide slope angles with a small standard deviation, namely $\alpha \sim \mathcal{N}(3, 0.5)$, and a model that increases the likelihood of requiring a detection as the distance to the runway decreases, namely $d \sim \mathcal{N}_{\text{trunc}}(0, 1; [0, 4])$. The parametric space is continuous in glide slope angle $\alpha \in [1, 7]$ degrees and distance to runway $d \in [0.1, 4]$ nmi. These models can be learned from historical flight data for more accurate estimates of the failure probability.

Treated as a black box, the runway detector is a convolutional neural network (CNN) that processes runway images from a front-facing RGB camera. The network predicts the runway corners and bounding box and is intended to be used to augment GPS/INS localization measurements during autonomous landing [25]. Figure 8.4 illustrates example images with detected runway corners and bounding boxes. A failure is defined as a misdetection (i.e., a false negative). Since the system is designed to be active only during the landing phase, we condition on the aircraft being on an approach. The use of a simulator means the RWD can be stressed outside the normal flight envelope to better characterize the full range of system failures, with a potentially dangerous-to-fly example shown in figure 8.4c.

8.4.4 Safety Validation Metrics

We define several metrics to measure the performance across the three safety validation tasks [15] defined in the background section 3.5.

Falsification metrics. The total number of failure cases, or more generally, the proportion of all system evaluations that resulted in failures, is the primary metric used to assess falsification (sometimes called the *failure rate*):

$$R_{\text{fail}} = \frac{\text{number of failures}}{\text{total number of evaluations}} = \frac{|\mathbf{X}_{\text{fail}}|}{|\mathbf{X}|} \tag{8.21}$$

Most-likely failure analysis metrics. The goal of most-likely failure analysis, as the name suggests, is to determine the failure with maximum operational likelihood. A natural way to assess the relative performance of this task against baselines is to compare the likelihood under the operational model of the determined most-likely failure:

$$\mathcal{L}^* = \max_{\mathbf{x}_i \in \mathbf{X}, y_i \in \mathbf{Y}} p(\mathbf{x}_i) \mathbb{1}\{y_i\} \tag{8.22}$$

Failure probability estimation metrics. Because probability of failure estimation is the primary objective of this work—capturing all three safety validation tasks—we look at the performance across several different metrics. When we have access to the true p_{fail} (e.g., in the toy examples), then we can measure the relative error in the estimated $\hat{p}_{\text{fail}}$:

$$\hat{\Delta}_{\text{fail}} = \frac{|p_{\text{fail}} - \hat{p}_{\text{fail}}|}{p_{\text{fail}}} \tag{8.23}$$

Measured as a proportion, relative error can be interpreted as the percent difference in the estimate and makes it easier to compare performance across problems. We also analyze the failure likelihood distribution $\{\log p(\mathbf{x}_i)\}_{\mathbf{x}_i \in \mathbf{X}_{\text{fail}}}$, where distributions with higher likelihood are preferred, as they cover more relevant example failures.

Coverage of design space. To measure the coverage of the design space, an average dispersion coverage metric has been used in the context of safety validation to estimate how well the sampled points cover the input space [15, 234]:

$$C_{\text{input}}(\mathbf{X}) = 1 - \frac{1}{\delta} \sum_{j=1}^{n} \frac{\min(d_j(\mathbf{X}), \delta)}{n} \tag{8.24}$$

where $C_{\text{input}}(\mathbf{X}) \in [0, 1]$ and the metric is defined over a grid of n points, separated by δ. The distance function $d_j(\mathbf{X})$ is defined as the minimum distance between the jth point in the grid to a point in $\mathbf{X}$ [15, 234].

When ground truth is available, we are also interested in the characterization of the failure and non-failure regions over the entire domain as predicted by the surrogate model. We define $C_{\text{output}} \in [0, 1]$ as the proportion of the output space that the surrogate and the true system agree upon. This can be interpreted as the surrogate classification accuracy.

8.4.5 Baseline Methods

We compare BAYESIANSAFETYVALIDATION (algorithm 8.2) against standard Monte Carlo (MC) sampling and population Monte Carlo (PMC) [217] with self-normalized importance sampling [212]. PMC requires an initial adaptive proposal q_{PMC}, which we set to be equal to the operational likelihood model for each of the example problems. The experiments were run for $T_{\max} = 100$ iterations using $N_q = 50$ samples per iteration and ran across 3 seeds. This results in $N_q T_{\max}(T_{\max} + 1)/2 = 252{,}500$ total samples per seed. Because sampling-based methods like MC and PMC tend to require many samples to adequately estimate the rare-event [210], we only test these methods on the example toy problems. Motivated by sample efficiency, we focus our comparison on the relative error in the estimated probability of failure $\hat{\Delta}_{\text{fail}}$ as a function of the number of samples, defined in equation (8.23).

8.5 Analysis and Results

We split the analysis into two sections: (1) comparison against existing methods for rare-event estimation (this tests the full Bayesian safety validation algorithm), and (2) comparison of the Gaussian process-based approach with different sampling/selection methods (this tests the failure search and refinement acquisition functions). We ran an ablation study to empirically show the influence of each acquisition function on the performance of the safety validation tasks. We demonstrate the algorithm on a stochastic system, a complex failure region problem, and a system that outputs a probabilistic value of failure (instead of strictly

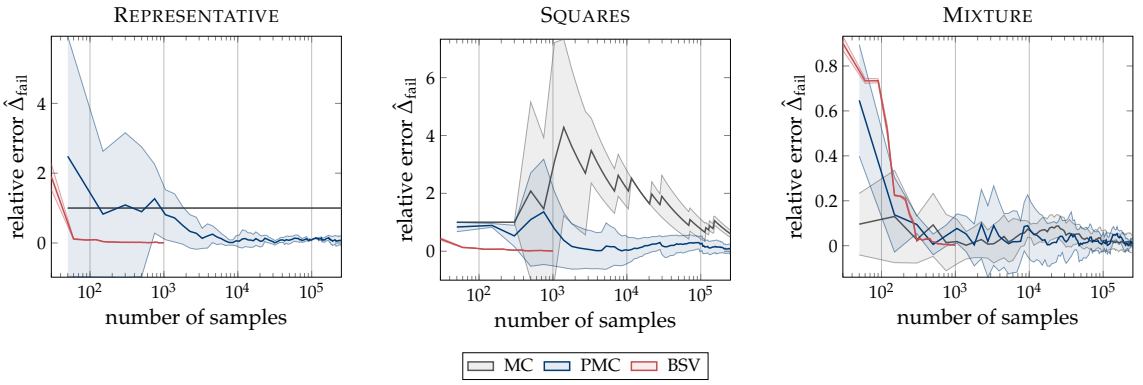


Figure 8.5: Estimator comparisons (with a log-scale horizontal axis).

binary) to show that BSV is applicable in the less restrictive problem case. Lastly, we report results on the runway detection system as a real-world case study.

8.5.1 Failure Probability Estimator Analysis

Figure 8.5 shows the error curves over the number of samples, which is equivalent to the number of system evaluations. Note that MC fails to estimate anything in the allotted number of samples in the representative example. BSV outperforms both MC and PMC in reducing the error in the probability of failure estimate using several orders of magnitude fewer samples; converging before 1000 samples in each case and closer to 100 samples in the first two problems. Results also indicate that BSV has lower variance compared to MC and PMC, which can partially be explained by the fact that two of the three acquisition functions take deterministic maximums and only one, the failure region sampling acquisition, samples predicted failure points stochastically.

8.5.2 Design Point Selection Analysis: Acquisition Functions

To test the proposed FAILURESEARCHANDREFINEMENT procedure (algorithm 8.1), we use the same GP fitting technique as in algorithm 8.2, but replace the selection process in line 5 with several baseline methods. The baselines we use are Latin hypercube sampling (LHS) [235], Sobol sequence selection [236], discrete grid selection, and uniform sampling. Each technique is defined over the entire operational domain. Importantly, we note that all methods fit the selected points to the same initial GP and use the same importance sampling procedure defined in algorithm 8.2. The FSAR approach is the only method that

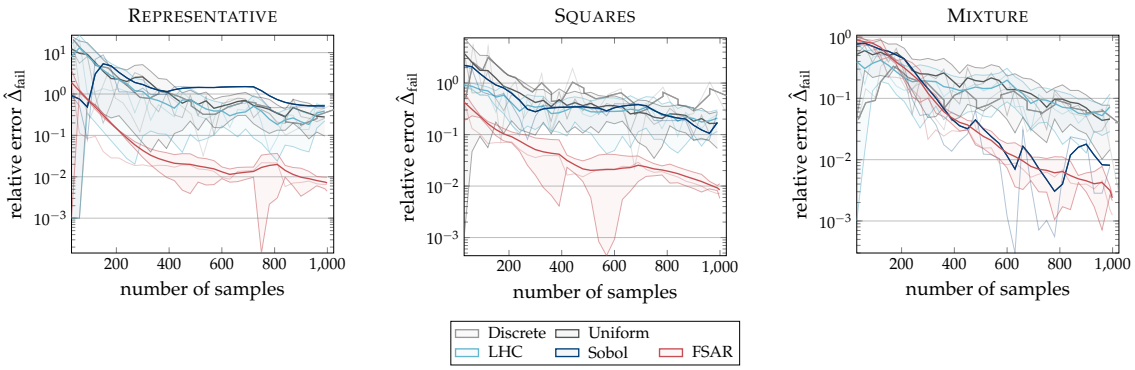


Figure 8.6: Fitting the same GP using different sampling schemes.

Example Problem	Selection Method	$R_{\text{fail}} \uparrow$	$\mathcal{L}^* \uparrow$	$\hat{\Delta}_{\text{fail}} \downarrow$	$C_{\text{input}} \uparrow$	$C_{\text{output}} \uparrow$
REPRESENTATIVE 	Discrete grid selection	0.336	5.62×10^{-8}	0.58225	**0.774**	0.9933
	Uniform sampling	0.342	4.02×10^{-8}	0.25978	0.673	0.9919
	Latin hypercube sampling	0.332	2.49×10^{-8}	0.44777	0.682	0.9922
	Sobol sequence sampling	0.335	4.98×10^{-8}	0.51179	0.719	0.9912
	Failure search and refinement (Ours)	**0.585**	$\mathbf{5.78 \times 10^{-8}}$	**0.00667**	0.638	**0.9998**
SQUARES 	Discrete grid selection	0.0439	0.00196	0.26473	**0.774**	0.9894
	Uniform sampling	0.0532	0.00086	0.17017	0.673	0.9909
	Latin hypercube sampling	0.0525	0.00192	0.24486	0.682	0.9907
	Sobol sequence sampling	0.0525	0.00142	0.27050	0.719	0.9935
	Failure search and refinement (Ours)	**0.4800**	**0.00253**	**0.00727**	0.643	**1.0**
MIXTURE 	Discrete grid selection	0.0479	0.0345	0.00279	**0.774**	0.9900
	Uniform sampling	0.0576	0.0360	0.04919	0.673	0.9871
	Latin hypercube sampling	0.0441	0.0349	0.10469	0.682	0.9864
	Sobol sequence sampling	0.0505	0.0369	0.00787	0.719	0.9881
	Failure search and refinement (Ours)	**0.4640**	**0.0383**	**0.00124**	0.663	**0.9984**

Table 8.1: Comparison against other sampling/selection methods.

uses incremental information to optimize the subsequent points. Figure 8.6 illustrates the relative error in the estimate when running BSV for $T = 333$ iterations ($N = 999$ system evaluations), run over 3 seeds with shaded regions reporting standard deviation (noting that Sobol and discrete do not use stochasticity). Exponential smoothing is applied to the curves with the raw values as thin lines of the same color. Using FSAR for acquiring subsequent points outperforms the baselines by orders of magnitude in the first two problems, and is comparable to Sobol sequence selection in the third problem but with more stability in the estimate. Table 8.1 reports the quantitative results from the baseline experiments. FSAR achieves the best performance across the various safety validation metrics and comparable input coverage relative to the baselines, notably with less than 1% error in all cases.

8.5.3 Complex Failure Region

As a visual example, we test the baseline sampling algorithms against BSV on a more complex failure region shape in figure 8.7 (where the white circles indicate the selected points for each algorithm). Using a uniform operational likelihood model, BSV is run using only the *boundary refinement* acquisition function which, by design, will explore the uncertainty when no failure boundaries are available to refine. In relatively few samples given a limited budget, BSV is able to fit to the complex failure region by focusing its search on failure boundary refinement.

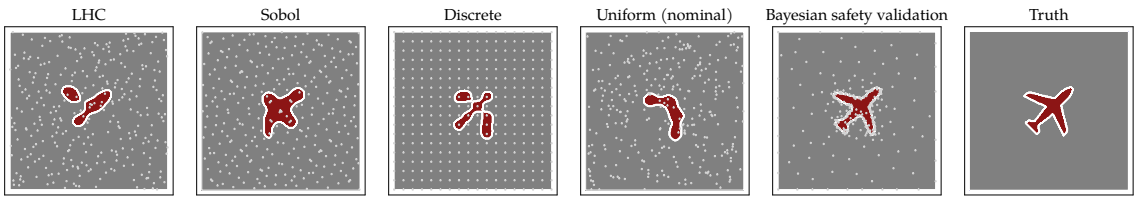


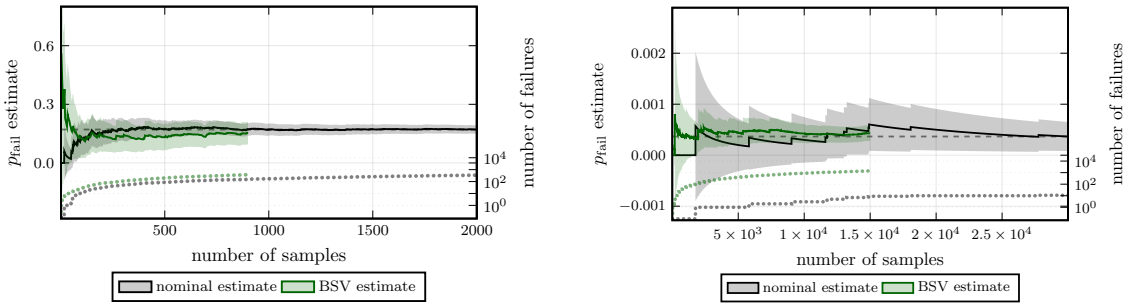
Figure 8.7: Fitting the same GP for a complex failure region using 300 selected points.

8.5.4 Ablation Study of Acquisition Functions

To empirically test the importance of all three acquisition functions, we perform an ablation study on the disjoint SQUARES problem to determine the effect of the combinations of acquisition functions across the safety validation metrics. The SQUARES example problem was chosen based on having two disjoint failure regions with precise boundaries, one of which is less likely than the other. Thus this problem requires careful balance between boundary refinement and deliberate exploration for multiple potential failure regions. Each ablation was run with 90 samples over 5 seeds for a fair comparison (i.e., when using all three acquisitions, each one gets a third of the budget). Results in table 8.2 indicate that the individual acquisitions perform well on the single metric they were designed for (i.e., *exploration* covers the input space, *failure sampling* has the highest failure rate, yet *boundary refinement* requires exploration in order to avoid exploiting a single failure mode). Using all three acquisition functions balances between the safety validation metrics and achieves the smallest error in the probability of failure estimate $\hat{\Delta}_{\text{fail}}$, while also finding a failure with the highest relative likelihood. In tables 8.1 and 8.2, arrows indicate whether the given metric is better to be high ($\uparrow$) or low ($\downarrow$).

Acquisition(s)	$R_{\text{fail}} \uparrow$	$\mathcal{L}^* \uparrow$	$\hat{\Delta}_{\text{fail}} \downarrow$	$C_{\text{input}} \uparrow$	$C_{\text{output}} \uparrow$
[1] exploration	0.044	0.00029	0.04382	**0.233**	0.9795
[2] boundary refinement	0.411	0.00025	0.93670	0.056	0.9598
[3] failure sampling	**0.707**	0.00186	0.21682	0.066	0.9604
[1, 2] exploration + boundary refinement	0.189	0.00205	0.18483	0.159	**0.9801**
[2, 3] boundary refinement + failure sampling	0.436	0.00197	0.24817	0.087	0.9647
[1, 3] exploration + failure sampling	0.318	0.00171	0.10057	0.163	0.9761
[1, 2, 3] exploration + boundary refinement + failure sampling	0.298	**0.00243**	**0.03553**	0.138	0.9777

Table 8.2: Ablation study: Effect of the *failure search and refinement* acquisition functions.



(a) Non-rare stochastic failure.

(b) Rare stochastic failure.

Figure 8.8: Failure probability estimate for the test POMDP: MC sampling vs. BSV.

8.5.5 Stochastic System Results

Figure 8.8 shows the failure probability estimation results of the stochastic LightDark POMDP when comparing BSV to nominal estimation, where nominal MC estimation samples the operational model directly. For the non-rare case in figure 8.8a, BSV is comparable to nominal yet finds more failures (41% failure rate for BSV and 15% for nominal). The non-rare failure probability estimate shown in the dashed horizontal line is computed after running nominal estimation for 5000 iterations and comes out to about $\hat{p}_{\text{fail}} = 0.15$. For the rare case in figure 8.8b, BSV finds failures early in the search and has a stable failure probability estimate with lower variance using fewer samples than the nominal. Notably, BSV finds orders of magnitude more failures than the nominal with a failure rate of about 9% compared to 0.041%. The dashed horizontal line is the nominal estimate computed over 100,000 samples and results in a rare failure of $\hat{p}_{\text{fail}} = 4.1 \times 10^{-4}$.

8.5.6 Test on Probabilistic-Valued System

The GP construction and proposed acquisition functions were designed to estimate failure probability over binary-valued systems that indicate failure, but the same techniques are applicable when the system outputs a probabilistic value of failure (that can be interpreted as confidence in the output, distance to failure boundary, or stochasticity of the system—which has been addressed by similar approaches from Gong *et al.* [237]). Using the same Himmelblau function [232] defined for the MIXTURE problem in section 8.4.1, we change the system to output a measure of failure (where $f(\mathbf{x}) \geq 0.5$ means failure): $f(\mathbf{x}) = \text{logit}^{-1}\left(c - \left((x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2\right)\right)$ for the threshold $c = 15$ (same as

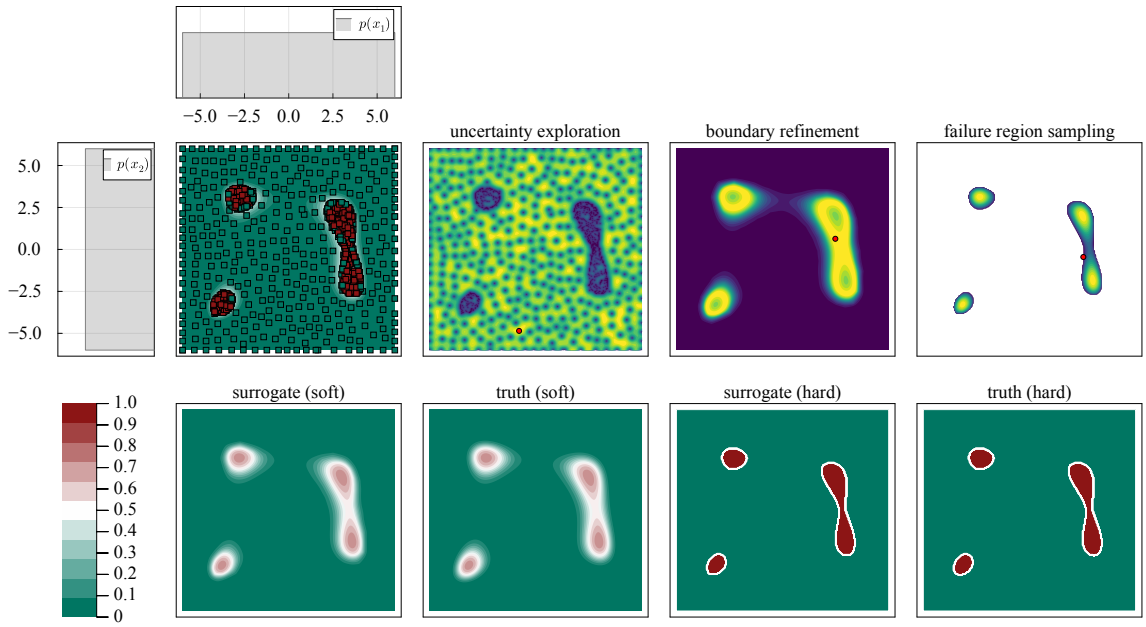


Figure 8.9: Test on a probabilistic-valued system (the MIXTURE problem).

the previous problem), passing the output through a sigmoid to interpret it as a probability with a steepness of $s = 1/c$. Figure 8.9 illustrates this example with a final relative error of $\hat{\Delta}_{\text{fail}} = 0.012$, a falsification rate of 53.6% of samples, an input coverage of $C_{\text{input}} = 0.653$, and output coverage of $C_{\text{output}} = 0.9998$. This example is tested on the MIXTURE (Himmelblau) problem, where the output is a probability value indicating distance to the failure boundary at $f(\mathbf{x}) = 0.5$. The top row illustrates BSV and the FSAR acquisition functions after $N = 999$ true observations (shown as red/green squares), with a uniform operational model p shown as subplots. The uniform model helps highlight that the failure region sampling is now more influenced by those failures that are farther away from the failure threshold c, shown as yellow peaks. The bottom row shows the surrogate model and ground truth, where ''soft'' is the probabilistic output and ''hard'' is the binary failure classification.

8.5.7 Real-World Case Study Results: Runway Detection System

After empirically validating the BSV algorithm on the example problems with access to the ground truth, we now report the performance on a real-world example: a runway detection system. Figure 8.10 shows the final surrogate after running BSV for $T = 333$ iterations (resulting in 999 sampled points). The algorithm focused the search on the likely regions of

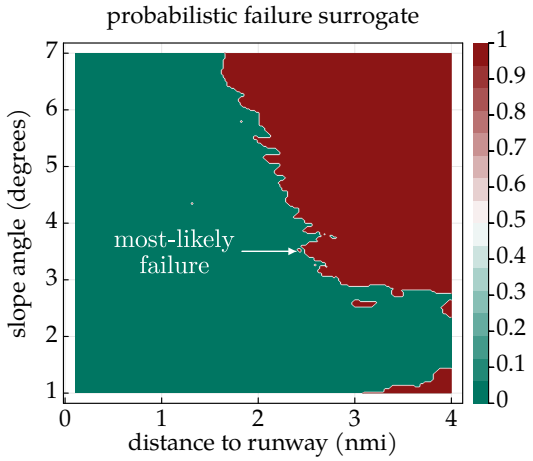


Figure 8.10: The final surrogate for the runway detection system using 999 data points.

the design space and found several failure modes. We can efficiently determine the most-likely failure, which is indicated in figure 8.10, and found 571 failures out of 999 evaluations, shown in table 8.3 as the failure rate of 57.2%. The failure probability estimate is shown to quickly converge in figure 8.11 after just above 400 system evaluations, with a final estimated failure probability of $\hat{p}_{\text{fail}} = 5.8 \times 10^{-3}$. If we instead used Monte Carlo sampling of p, we would expect to find only about 6 failures in the 999 system evaluations (compared to 571).

One way to characterize the spread of failures is to plot the log-likelihood of the observed failures under the operational model, namely $\log p(\mathbf{x})$. Shown in figure 8.11, the right-skewed peak of the distribution indicates that the failures that were found have high likelihood, and thus are more useful failures to fix first, before system deployment. Five different iterations of the BSV algorithm and acquisition functions for the RWD system are illustrated in figure 8.12. Red indicates failures predicted by the probabilistic surrogate model, and lighter colors indicate maximums for the acquisition functions. The red and green squares overlaid on the surrogate are the true system evaluations, and the red circles in the acquisition functions show the next selected point. Notice the low uncertainty in the concentration of points around the likely glide slope region (the operational models for glide slope and distance to runway are shown as subplots). The likelihood decay in the boundary refinement acquisition is illustrated as the spread of the likelihood influence as it dissipates over time. Finally, the refined failure

R_{fail}	$\mathcal{L}^*$	$\hat{p}_{\text{fail}}$	C_{input}
0.572	0.02	5.8×10^{-3}	0.681

Table 8.3: RWD safety metrics.

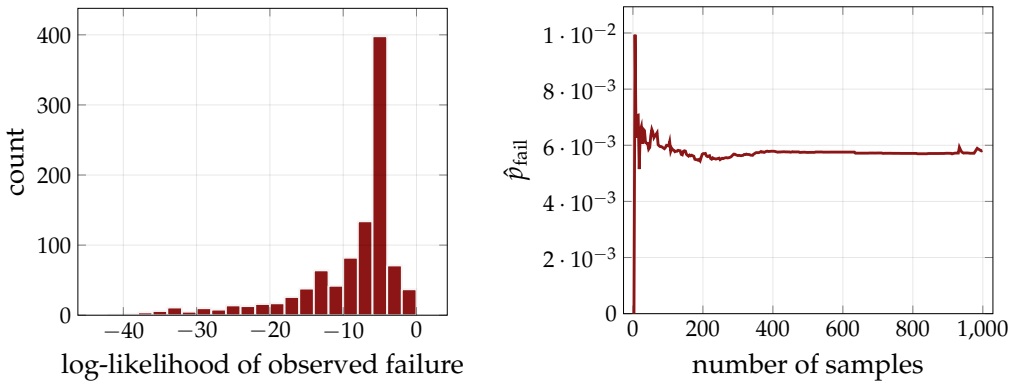


Figure 8.11: Results using Bayesian safety validation on the runway detection system.

region using $N = 999$ samples represents the predicted distribution of failures. Table 8.3 reports the safety validation metrics, noting that $\hat{\Delta}_{\text{fail}}$ and C_{output} are not reported since we do not have access to the true failure boundaries of the system.

Results show that we can characterize failure regions, generate a set of likely failures, compute the most-likely failure, and use the surrogate model to estimate the failure probability of the runway detector in a small number of samples, e.g., only using 999 samples in our experiments. Post-analysis could even further characterize the failure boundary by focusing on the likely region centered around the glide slope angle of 3 degrees. Example runway images that successfully detected the runway and failed to detect the runway are shown in figure 8.13. We demonstrate the BSV algorithm on a two-dimensional case, but this work could be scaled to higher-dimensional problems that incorporate additional environmental parameter models such as roll angle, time-of-day, and weather, and that use different airport runways. Binois *et al.* [238] provide a survey on methods, challenges, and guidelines when modeling high-dimensional problems using Gaussian processes for Bayesian optimization.

8.5.8 Validating the Simulator

A failure probability estimate is only as good as the simulator. To enhance our approach, future work could validate the simulator by taking a representative set of design points and running a flight test to compare the RWD outputs. Using flight testing to record real operations of avionics systems and compare to simulated outputs is a common way to validate simulators [239]. In the case of runway detection systems, flight testing also provides more image data for the types of runways the autonomous aircraft may encounter.

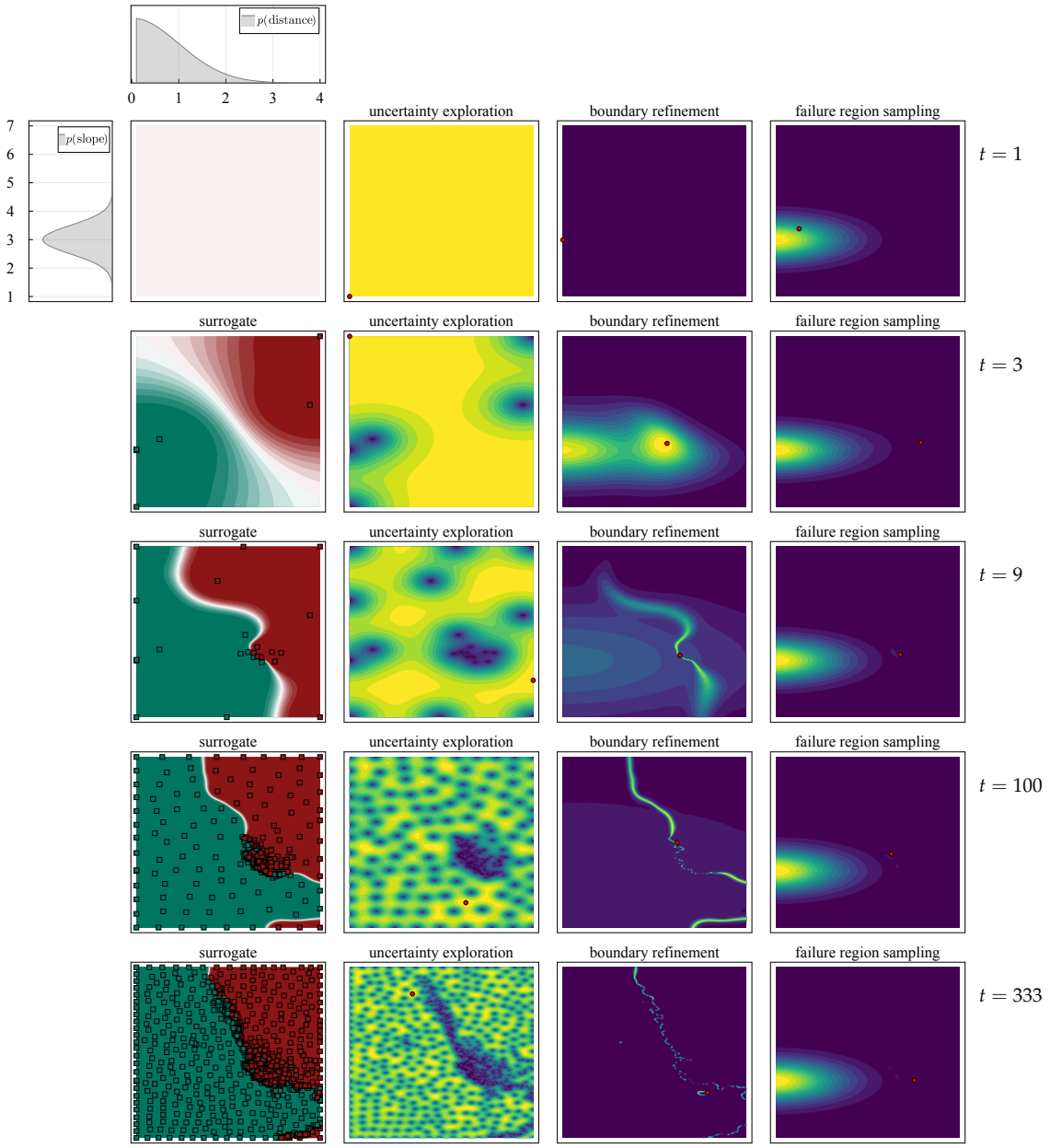
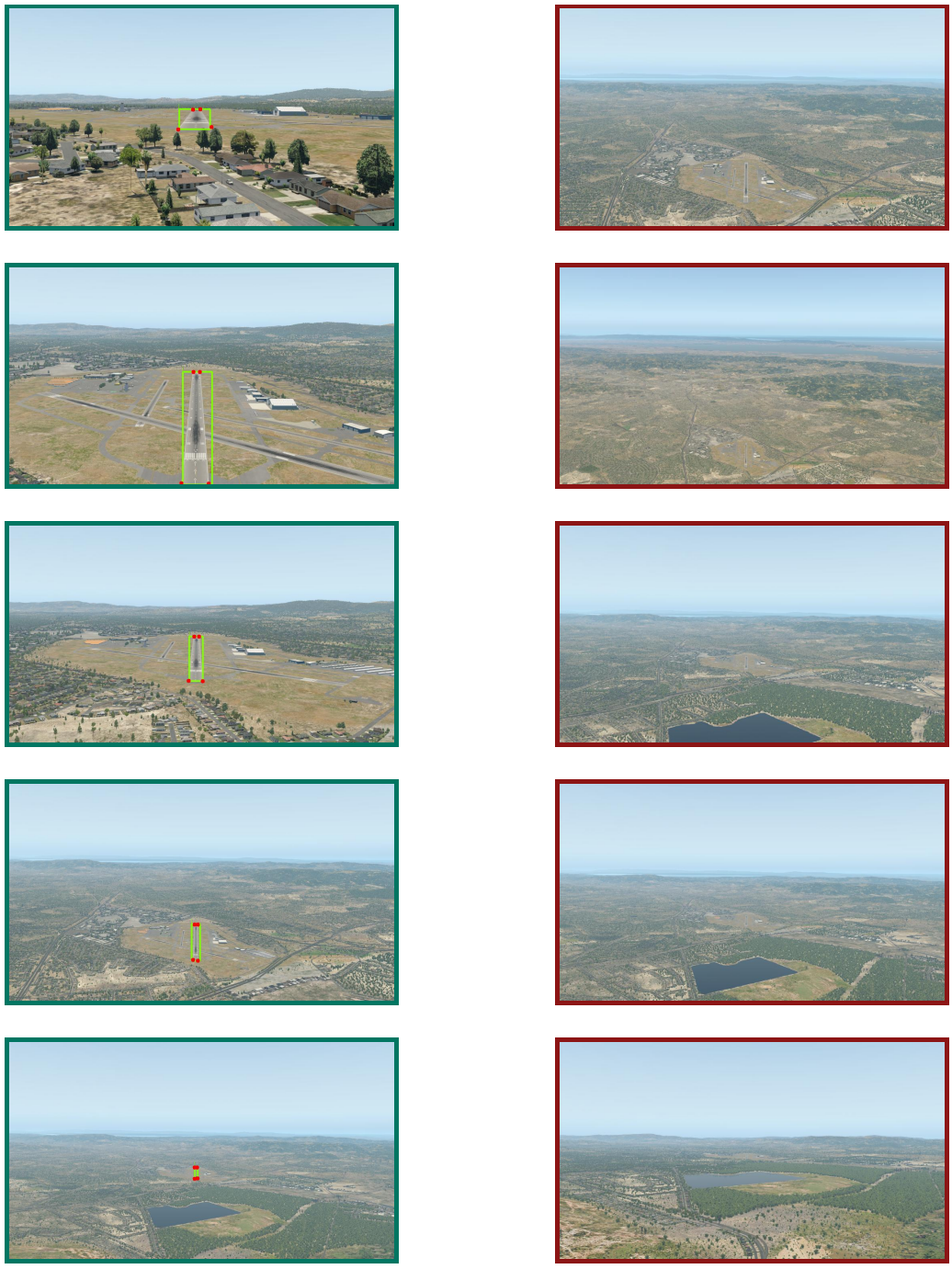


Figure 8.12: Bayesian safety validation applied to the runway detection problem.



(a) Examples of successful runway detections. (b) Examples of failing to detect the runway.

Figure 8.13: Example runway images from X-Plane found by Bayesian safety validation.

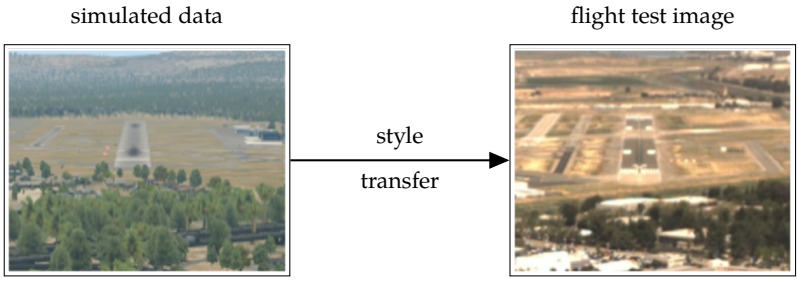


Figure 8.14: Future work: Example style transfer of simulated images from flight test data.

Other methods, such as style transfer [240, 241], as shown in figure 8.14, could close the sim-to-real gap [242] by bringing the simulated images closer to images seen during operation.

8.5.9 Open-Source Interface

Using the `GaussianProcesses.jl` [243] and `AbstractGPs.jl` packages [244], we developed an open-source Julia package† to be applied to other black-box systems. It is intended to fit into the suite of safety validation tools when considering autonomous aircraft certification. To extend to another system, a user can implement the following interface:

```
abstract type SystemParameters end
function reset() end
function initialize() end
function generate_input(sample::Vector)::Input end
function evaluate(input::Input)::Vector{Bool} end
```

Below is an example of setting up a system with a 2D operational model, running the BSV algorithm to learn the surrogate, and then computing the three safety validation tasks:

```
using BayesianSafetyValidation
system_params = RunwayDetectionSystemParameters() # defined by the user
model = [OperationalParameters("distance", [0.1, 4], TruncatedNormal(0, 1.0, 0, 4)),
         OperationalParameters("slope", [1, 7], Normal(3, 0.5))]
surrogate  = bayesian_safety_validation(system_params, model; T=100)
X_failures = falsification(surrogate.x, surrogate.y)
ml_failure = most_likely_failure(surrogate.x, surrogate.y, model)
p_failure  = p_estimate(surrogate, model)
```

†The package and experiments are available at `https://github.com/sisl/BayesianSafetyValidation.jl`.

8.6 Conclusion

In this chapter, we framed the black-box safety validation problem as a Bayesian optimization problem and introduced *Bayesian safety validation* (BSV) to build a probabilistic surrogate model that predicts system failures and uses importance sampling to efficiently estimate the failure probability. In the process, we proposed a set of acquisition functions, called *failure search and refinement* (FSAR), that each help achieve the safety validation tasks by covering the design space to search for failures and refining likely failure boundaries and regions. The Gaussian process construction allowed us to analytically derive the predicted failure boundaries, and we showed that the combination of acquisition functions is important to find more failures, find more likely failures, and minimize error in the estimate of the probability of failure.

Primarily interested in cases where the black-box system only outputs a binary indication of failure, we also showed that our method works well in the less restrictive case where the system outputs a real-valued measure of failure confidence, severity, or distance. BSV is also applicable when system failures are stochastic and the idea of deterministic ''failure regions'' becomes irrelevant. In the case of stochastic system failures, self-normalizing importance sampling was used to compute the proposal weights, which scales better to higher dimensional problems.

This technique was applied to validate an image-based neural network runway detection system in simulation. Alongside traditional DO-178C procedures [19], this work is currently being used to supplement the FAA certification process of an autonomous cargo aircraft [25]. We emphasize that the exact values of the most-likely failure likelihood and the estimated probability of failure are largely dependent on the choice of operational model, and learning these models from collected flight data would provide a more realistic understanding of the true probability of failure.

PART IV:

CONCLUSIONS

Chapter 9

Summary

It's the way I study – to understand something by trying to work it out or, in other words, to understand something by creating it. Not creating it one hundred percent, of course; but taking a hint as to which direction to go but not remembering the details. These you work out for yourself.

Richard Feynman

Designing safe policies and performing safety validation is crucial for the adoption of artificial intelligence (AI) into real-world systems. The use of AI itself in accelerating the design and validation process, whether it be using machine learning or reinforcement learning, has shown to be a significant avenue for innovation. Using surrogate models, such as neural networks, allows us to scale these safety systems over available compute to solve large, real-world problems. Along with addressing scale, accounting for uncertainty in the environment dynamics and in the state of the world itself (through the use of beliefs), is required when operating in the real world—because the world is only partially observable.* After safe policies are developed, we then have to validate their safety to find rare failure modes potentially missed during training. Because these scalable policies may rely on neural networks or expensive simulators, treating them as black boxes during validation allows the validation algorithms to be broadly applied. This thesis studies the design of large-scale planning algorithms for POMDPs, their extensions to safety-critical problems, and efficient black-box methods for safety validation.

*Following the motto of our lab SISL: *Life is a POMDP*.

We first study how to frame sequential decision making problems to scale across available compute. Therefore, we formulated the problem of planning into its batched equivalent. In doing so, we proved that in the batched planning framework, the optimal policy is preserved. Along with preserving optimality, we empirically show that, unsurprisingly, batching the lookahead process in MDPs can significantly improve performance.

Next, we address the requirement of parallelized belief updaters when formulating the decision making problem in the batched planning setting. We investigate the use of deep conditional generative models to act as surrogates for belief updating. We study how to design these surrogates to efficiently sample from the posterior belief. We operate in the *purely epistemic MDP* setting where states do not transition, turning the problem of belief updating into an inversion problem—which becomes a purely information gathering objective. We study heuristic policies that maximize the expected entropy of the belief and show that, in the limit, our heuristics result in the optimal one-step expected Bayesian information gain. We study how these surrogates would be applied in settings where observations are either direct or indirect partial state information. Through experiments, we observe that fine-tuning the surrogates to maximize mutual information along observation trajectories provides benefits for both planning and estimation.

Next, we study how to learn to replace hand-crafted heuristics and generalize POMDP planning algorithms to long-horizon problems. We build off insights from reinforcement learning to combine online Monte Carlo tree search (MCTS) with learned offline neural network surrogates for the value function and action selection policy. We address challenges relating to POMDP planning: how to handle stochastic belief-state transitions in the search tree, how to handle expensive belief updates during planning, and how to represent non-parametric beliefs as input to the neural network surrogate. We developed a scalable POMDP policy iteration algorithm, BetaZero, and applied it to several large benchmark problems. Our experiments indicate that BetaZero can learn heuristics that are comparable to expertly designed counterparts, and that BetaZero learns the approximately optimal value function and action selection policy.

Next, we extend our work on POMDP planning algorithms to the safety-critical setting by formulating the problem as a chance-constrained POMDP (CC-POMDP). We study how to design policy iteration algorithms and tree search methods that can predict failure probabilities and plan safely, and introduce the ConstrainedZero algorithm. We study the use of adaptive conformal inference as a method to estimate the failure probability threshold

during online planning, and introduce a fifth *adaptation* stage to MCTS, resulting in the Δ-MCTS algorithm. We demonstrate that by factoring the reward function to separate safety, we can achieve a target level of safety that is approximately Pareto-optimal. We study how ConstrainedZero is applied to several safety-critical CC-POMDPs and show that it can work without change in the chance-constrained MDP setting as well. Insights from this study show that directly specifying a target level of safety Δ is more interpretable than tuning a cost coefficient when developing safe policies.

Finally, we study how probabilistic surrogate models can be used to efficiently estimate the failure probability of a black-box system. We investigate the use of Gaussian processes as surrogate models and introduce three acquisition functions that each target different safety validation tasks. We introduce the Bayesian safety validation method as an approach to efficiently find failures, find likely failures, and estimate the failure probability. We study the estimation quality of our method on several toy examples where we have access to the true failure probability, and experiments show that our method can accurately estimate failure probability and uncover different failure modes. We demonstrate our method on a real-world runway detection system and provide insights into how our method is used to augment the certification of the machine learning components for an autonomous aircraft.

Chapter 10

Contributions

The impulse of the maker is hard to quell.

Mark Miller

In an attempt to improve safe planning under uncertainty and safety validation using surrogate models, we have contributed the following research:

Framework for batched belief-state planning. In chapter 4, we introduced the *batched planning* framework to scale planning problems to available compute. We then developed extensions for batched belief-state planning in POMDPs. We proved that the optimal policy is preserved in the batched planning framework, enabling faster execution without compromising policy performance.

Conditional surrogate model for belief updating. In chapter 5, we introduced the *inversion variational autoencoder* ($\mathcal{I}$-VAE) model that approximates sampling from the posterior belief, allowing us to use the batched belief-state planning framework. We study how deep conditional generative models can be used as inversion methods to go from partial observations to samples from a distribution over states. We showed that learning latent representations of the distribution over observations allows us to visualize the latent space for further analysis. We demonstrated that fine-tuning the $\mathcal{I}$-VAE model to maximize mutual information across observation trajectories results in better planning and estimation performance. We tested our approach against several benchmark belief updaters and policies, and developed an optimal one-step Bayesian information gaining heuristic. We applied our method to a challenging

planning problem of muon-based intrusion discovery and observed that the $\mathcal{I}$-VAE achieves high conditional likelihood and low belief error using only a few observations.

Method for scalable belief-state planning and learning. In chapter 6, we introduced a scalable and generalizable POMDP planning algorithm that replaces hand-crafted heuristics with learned approximations. We developed the *BetaZero* policy iteration algorithm that combines online MCTS planning with offline learned neural network surrogates to enable long-horizon planning. We addressed several challenges with partial observability and demonstrated how BetaZero can generalize to different benchmark POMDP problems. We showed that BetaZero can learn an approximately optimal value function and action selection policy and that additional online planning with the neural network surrogates outperforms both the raw networks and traditional online planning methods.

Method for safety-aware planning in partially observable problems. In chapter 7, we extended BetaZero to the chance-constrained POMDP (CC-POMDP) setting and introduced *ConstrainedZero*. We learned a neural network surrogate that also predicts the failure probability given a belief. We used this surrogate during MCTS planning and introduced the Δ-MCTS algorithm for safety-aware tree search. We showed that ConstrainedZero can learn the approximately Pareto-optimal policy by maximizing rewards within the target level of safety. We highlighted how separating safety from the reward function when formalizing the CC-POMDP allows for more interpretable objectives when developing safe policies. We demonstrated our method on several benchmark safety-critical CC-POMDPs and showed that it can easily be applied to chance-constrained MDPs.

Method for black-box safety validation using probabilistic surrogate models. In chapter 8, we introduced the *Bayesian safety validation* method to estimate the failure probability of complex black-box systems. We combined probabilistic surrogate models with three new acquisition functions to iteratively sample new design points from the inexpensive surrogate to refine the predicted failure regions. We demonstrated that Bayesian safety validation can achieve low error in the failure probability estimate while simultaneously achieving high failure rate and collecting high-likelihood failures. We tested our method on a neural network-based runway detection system used within an autonomous aircraft stack.

10.1 Open-Source Contributions

The code and experiments developed for this thesis are open-sourced and available online.

- https://github.com/sisl/I-VAE: PyTorch implementation of the $\mathcal{I}$-VAE model.
- https://github.com/sisl/MuonPOMDPs.jl: POMDP models of the muon-based intrusion discovery problem, batched belief-state planning framework, and experiments.
- https://github.com/sisl/BetaZero.jl: The BetaZero POMDP algorithm, experiments, and environments; integrated into the Julia `POMDPs.jl` ecosystem.
- https://github.com/sisl/ConstrainedZero.jl: The ConstrainedZero CC-POMDP algorithm, experiments, and environments; integrated into `POMDPs.jl`.
- https://github.com/sisl/BayesianSafetyValidation.jl: The Bayesian safety validation algorithm and experiments.

10.2 Publications

The contents of chapters 4 and 5 are new and yet to be published.

The content of chapter 6 appeared in:

> [195] R. J. Moss, A. Corso, J. Caers, and M. J. Kochenderfer, "BetaZero: Belief-State Planning for Long-Horizon POMDPs using Learned Approximations," *Reinforcement Learning Journal* (*RLJ*), vol. 1, pp. 158–181, 2024. https://rlj.cs.umass.edu/2024/papers/Paper27.html

The content of chapter 7 appeared in:

> [245] R. J. Moss, A. Corso, J. Caers, and M. J. Kochenderfer, "ConstrainedZero: Chance-Constrained POMDP Planning using Learned Probabilistic Failure Surrogates and Adaptive Safety Constraints," In *International Joint Conference on Artificial Intelligence* (*IJCAI*), pp. 6752–6760, 2024. https://doi.org/10.24963/ijcai.2024/746

[246] R. J. Moss, A. Corso, J. Caers, and M. J. Kochenderfer, ''Chance-Constrained POMDP Planning with Learned Neural Network Surrogates,'' In *IJCAI Workshop on Trustworthy Interactive Decision-Making with Foundation Models*, 2024. https://openreview.net/forum?id=w5a6C4tiat

The content of chapter 8 appeared in:

[247] R. J. Moss, M. J. Kochenderfer, M. Gariel, and A. Dubois, ''Bayesian Safety Validation for Failure Probability Estimation of Black-Box Systems,'' *AIAA Journal of Aerospace Information Systems* (*JAIS*), vol. 21, no. 7, pp. 533–546, 2024. https://doi.org/10.2514/1.I011395

[248] R. J. Moss, A. Corso, J. Caers, and M. J. Kochenderfer, ''Bayesian Safety Validation for Black-Box Systems,'' In *AIAA AVIATION Forum*, 2023. https://doi.org/10.2514/6.2023-3596

10.2.1 Non-Thesis Publications

During my time as a graduate student, I was lucky enough to explore several different research directions, including co-authoring a new textbook on *Algorithms for Validation* [249]. The following work is excluded from this thesis.

[249] M. J. Kochenderfer, S. M. Katz, A. L. Corso, and R. J. Moss, *Algorithms for Validation*. MIT Press, 2025. https://algorithmsbook.com/validation/

S. Hwang, R. J. Moss, D. Fan, and S. Follmer, ''Computational Modeling of Non-Visual Vibrotactile Touchscreen Exploration,'' *Conference on Human Factors in Computing Systems* (*CHI*), 2025. https://dl.acm.org/doi/10.1145/3706599.3719851

R. J. Moss, ''Kov: Transferable and Naturalistic Black-Box LLM Attacks Using Markov Decision Processes and Tree Search,'' *arXiv preprint arXiv:2408.08899*, 2024.
https://arxiv.org/abs/2408.08899

R. J. Moss, M. Kozlova, A. Corso, and J. Caers, ''Model-fidelity analysis for sequential decision-making systems using Simulation Decomposition: Case study of critical mineral exploration,'' *Sensitivity Analysis for Business, Technology, and Policymaking*, Routledge, 2024.
https://doi.org/10.4324/9781003453789-12

M. Kozlova, R. J. Moss, P. Roy, A. Alam, and J. S. Yeomans, ''SimDec algorithm and guidelines for its usage and interpretation,'' *Sensitivity Analysis for Business, Technology, and Policymaking*, Routledge, 2024.
https://doi.org/10.4324/9781003453789-3

M. Kozlova, R. J. Moss, J. S. Yeomans, and J. Caers, ''Uncovering heterogeneous effects in computational models for sustainable decision-making,'' *Environmental Modelling & Software*, 2024.
https://doi.org/10.1016/j.envsoft.2023.105898

[25] J.-G. Durand, A. Dubois, and R. J. Moss, ''Formal and Practical Elements for the Certification of Machine Learning Systems,'' *AIAA/IEEE Digital Avionics Systems Conference (DASC)*, 2023.
https://doi.org/10.1109/DASC58513.2023.10311201

L. J. Einstein, R. J. Moss, and M. J. Kochenderfer, ''Prioritizing emergency evacuations under compounding levels of uncertainty,'' *IEEE Global Humanitarian Technology Conference (GHTC)*, 2022.
https://doi.org/10.1109/GHTC55712.2022.9910611

[15] A. Corso, R. J. Moss, M. Koren, R. Lee, and M. J. Kochenderfer, "A Survey of Algorithms for Black-Box Safety Validation of Cyber-Physical Systems," *Journal of Artificial Intelligence Research* (*JAIR*), vol. 72, pp. 377–428, 2021.
https://doi.org/10.1613/jair.1.12716

R. J. Moss, S. Gupta, R. Dyro, K. Leung, M. J. Kochenderfer, G. X. Gao, M. Pavone, E. Schmerling, A. Corso, R. Madigan, M. Stroila, and T. Gibson, "Autonomous Vehicle Risk Assessment," *Stanford Center for AI Safety*, 2021.
https://web.stanford.edu/~mossr/pdf/av_risk_assessment.pdf

R. J. Moss, "POMDPStressTesting.jl: Adaptive Stress Testing for Black-Box Systems," *Journal of Open-Source Software* (*JOSS*), vol. 6, no. 60, 2021.
https://doi.org/10.21105/joss.02749

M. Durling, H. Herencia-Zapana, B. Meng, M. Meiners, J. Hochwarth, N. Visser, R. Lee, R. J. Moss, and V. T. Valapil, "Certification Considerations for Adaptive Stress Testing of Airborne Software," *AIAA/IEEE Digital Avionics Systems Conference* (*DASC*), 2021.
https://doi.org/10.1109/DASC52595.2021.9594395

R. J. Moss, R. Lee, N. Visser, J. Hochwarth, J. G. Lopez, and M. J. Kochenderfer, "Adaptive Stress Testing of Trajectory Predictions in Flight Management Systems," *AIAA/IEEE Digital Avionics Systems Conference* (*DASC*), 2020.
https://doi.org/10.1109/DASC50938.2020.9256730

[214] R. J. Moss, "Cross-Entropy Method Variants for Optimization," *arXiv preprint arXiv:2009.09043*, 2020.
https://arxiv.org/abs/2009.09043

Chapter 11

Future Work

Gas will expand to fill available space, work will expand to fill available time.

Keith Schwarz

This thesis presented research to address safe planning under uncertainty and efficient safety validation, but as with all research, it is only the beginning. This chapter outlines several limitations of our work, possible next steps, and potential future research directions.

Neural network verification. Despite the convergence guarantees of adaptive conformal inference in ConstrainedZero [192], planning with approximations does not provide safety guarantees. Although our experiments suggest that adaptation helps find safe policies, studying formal methods for neural network verification is important [86]. Applying nonlinear reachability methods such as Taylor models, conservative linearization, or partitioning [249, 250, 251, 252] could help make the surrogates more robust. Work from Katz *et al.* [85] could be applied to simplified neural network surrogates used in this work. Verification provides guarantees and an additional level of analysis complementary to validation, and should be studied before deploying neural networks into safety-critical systems.

Interpretable surrogate models. Especially in the safety case, having *interpretable* surrogate models [253] can be useful for stakeholders to better understand and analyze the safe decision making of the system. An important question to research is: *''How can interpretable surrogate models be used for safe planning?''* Interpretable models, such as shallow decision

trees [254], may be of interest. When replacing complex surrogates like neural networks with simpler, more interpretable models, there tends to be a trade-off between interpretability (i.e., model complexity) and performance. Studying this trade-off in the safety case is incredibly important.

Offline chance-constrained methods. Learning offline policies that guarantee satisfaction of safety constraints is another challenge that could be studied. Ono *et al.* [255] developed a method to solve chance-constrained MDPs given a target level of safety, yet their work could be extended to POMDPs and to multiple constraints. Motivated by the ACAS X safety case, Kochenderfer *et al.* [12] used QMDP [156] to train an offline policy that is augmented online with additional information. Providing offline policy guarantees can help establish the safety case in fields such as aviation. Such policies can then be verified to exhaust all combinations of a value-based lookup table [256].

Belief representations. Extensions to algorithms such as BetaZero and ConstrainedZero could focus on addressing the belief representation as input to the neural network surrogates. Future work could use the latent representation from deep conditional generative models like the $\mathcal{I}$-VAE introduced in chapter 5. Other work, such as flow-based models [170], principle component analysis [171], or order-invariant network layers [189], could also be studied as methods to represent the complex belief distributions.

Continuous action spaces. This thesis primarily deals with problems that have discrete action spaces. Yet, more complicated systems may operate over *continuous* actions. Therefore, studying methods to extend our work to the continuous action domain would be useful. Moerland *et al.* [164] introduced an extension to AlphaZero that handles continuous actions, and their work could be directly applied to the BetaZero and ConstrainedZero algorithms. Future work could therefore study how to apply our methods to safety-critical robotics tasks such as control [257] and object manipulation [258, 259].

Surrogates for environment models. Another avenue of future work is the use of surrogate models to replace expensive geological simulators, such as those used by Wen *et al.* [260] for the CCS problem. Research into approximate *world models* [261] could be directly applicable to our work and provide additional computational gains. Therefore, it is important to study how best to represent the state-transition dynamics using surrogate models.

Bibliography

[1] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[2] C. M. Bishop and N. M. Nasrabadi, *Pattern Recognition and Machine Learning*. Springer, 2006, vol. 4.

[3] P. Covington, J. Adams, and E. Sargin, "Deep Neural Networks for Youtube Recommendations," in *ACM Conference on Recommender Systems*, 2016.

[4] E. Wulczyn, N. Thain, and L. Dixon, "Ex Machina: Personal Attacks Seen at Scale," in *International Conference on World Wide Web*, 2017.

[5] M. J. Kochenderfer, T. A. Wheeler, and K. H. Wray, *Algorithms for Decision Making*. MIT Press, 2022.

[6] E. Balaban, "Health-Aware Decision Making Under Uncertainty for Complex Systems," Ph.D. dissertation, Stanford University, 2020.

[7] X. Huang, A. Jasour, M. Deyo, A. Hofmann, and B. C. Williams, "Hybrid Risk-Aware Conditional Planning with Applications in Autonomous Vehicles," in *IEEE Conference on Decision and Control (CDC)*, 2018.

[8] J. Mern and J. Caers, "The Intelligent Prospector v1.0: Geoscientific Model Development and Prediction by Sequential Data Acquisition Planning with Application to Mineral Exploration," *Geoscientific Model Development*, vol. 16, no. 1, pp. 289–313, 2023.

[9] G. Kohs, G. Krieg, J. Rosen, and K. Proudfoot, *AlphaGo*, Google DeepMind, 2017.

[10] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *Computers and Games*, 2007.

[11] R. Bellman, "Dynamic Programming," *Princeton University Press*, 1957.

[12] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos, "Next-Generation Airborne Collision Avoidance System," *Lincoln Laboratory Journal*, vol. 19, no. 1, 2012.

[13] A. Forrester, A. Sobester, and A. Keane, *Engineering Design via Surrogate Modelling: A Practical Guide*. John Wiley & Sons, 2008.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[15] A. Corso, R. J. Moss, M. Koren, R. Lee, and M. J. Kochenderfer, "A Survey of Algorithms for Black-Box Safety Validation of Cyber-Physical Systems," *Journal of Artificial Intelligence Research*, vol. 72, pp. 377–428, 2021.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[18] C. V. Oster Jr., J. S. Strong, and C. K. Zorn, "Analyzing aviation safety: Problems, challenges, opportunities," *Research in Transportation Economics*, vol. 43, no. 1, pp. 148–164, 2013.

[19] RTCA, *Software Considerations in Airborne Systems and Equipment Certification*, DO-178C, 2011.

[20] RTCA, *Minimum Operational Performance Standards for Airborne Collision Avoidance System X (ACAS X) (ACAS Xa and ACAS Xo) Volume I and II*, DO-385A, 2023.

[21] RTCA, *Minimum Operational Performance Standards for Traffic Alert and Collision Avoidance System II (TCAS II) Airborne Equipment*, DO-185B, 2008.

[22] F. A. Administration, *Introduction to TCAS II*, Version 7.1, 2011.

[23] A. Gautam, P. B. Sujit, and S. Saripalli, "A Survey of Autonomous Landing Techniques for UAVs," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014.

[24] Airbus A^3, *How Wayfinder is Using Neural Networks for Vision-Based Autonomous Landing*, `https://acubed.airbus.com/blog/wayfinder/how-wayfinder-is-using-neural-networks-for-vision-based-autonomous-landing/`, 2019.

[25] J.-G. Durand, A. Dubois, and R. J. Moss, "Formal and Practical Elements for the Certification of Machine Learning Systems," in *AIAA/IEEE Digital Avionics Systems Conference (DASC)*, 2023.

[26] S. J. Pyne, P. L. Andrews, and R. D. Laven, *Introduction to Wildland Fire*. John Wiley & Sons, 1996.

[27] J. D. Griffith, M. J. Kochenderfer, R. J. Moss, V. V. Mišic, V. Gupta, and D. Bertsimas, "Automated Dynamic Resource Allocation for Wildfire Suppression," *Lincoln Laboratory Journal*, vol. 22, no. 2, pp. 38–59, 2017.

[28] D. Boychuk, W. J. Braun, R. J. Kulperger, Z. L. Krougly, and D. A. Stanford, "A stochastic forest fire growth model," *Environmental and Ecological Statistics*, vol. 16, pp. 133–151, 2009.

[29] J. Guiochet, M. Machin, and H. Waeselynck, "Safety-critical advanced robots: A survey," *Robotics and Autonomous Systems*, vol. 94, pp. 43–52, 2017.

[30] R. Platt Jr., R. Tedrake, L. Kaelbling, and T. Lozano-Perez, "Belief Space Planning Assuming Maximum Likelihood Observations," *Robotics: Science and Systems VI*, 2010.

[31] T. Smith and R. Simmons, "Heuristic Search Value Iteration for POMDPs," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[32] M. E. Boot-Handford, J. C. Abanades, E. J. Anthony, M. J. Blunt, S. Brandani, N. Mac Dowell, J. R. Fernández, M.-C. Ferrari, R. Gross, J. P. Hallett, R. S. Haszeldine, P. Heptonstall, A. Lyngfelt, Z. Makuch, E. Mangano, R. T. J. Porter, M. Pourkashanian, G. T. Rochelle, N. Shah, J. G. Yao, and P. S. Fennell, "Carbon capture and storage update," *Energy and Environmental Science*, vol. 7, pp. 130–189, 1 2014.

[33] K. E. Aikin and C. E. Johnson, "Enhanced Precision in Geothermal Reservoir Characterization and Exploitation," US Department of Energy Office of Science, Tech. Rep., 2019.

[34] B. K. Sovacool, S. H. Ali, M. Bazilian, B. Radley, B. Nemery, J. Okatz, and D. Mulvaney, "Sustainable minerals and metals for a low-carbon future," *Science*, vol. 367, no. 6473, pp. 30–33, 2020.

[35] S. J. Gibowicz, "Seismicity Induced by Mining: Recent Research," *Advances in Geophysics*, vol. 51, pp. 1–53, 2009.

[36] A. Lechmann, D. Mair, A. Ariga, T. Ariga, A. Ereditato, R. Nishiyama, C. Pistillo, P. Scampoli, F. Schlunegger, and M. Vladymyrov, "Muon tomography in geoscientific research – A guide to best practice," *Earth-Science Reviews*, vol. 222, 2021.

[37] Y. Wang, M. Zechner, G. Wen, A. L. Corso, J. M. Mern, M. J. Kochenderfer, and J. K. Caers, "Optimizing Carbon Storage Operations for Long-Term Safety," *arXiv preprint arXiv:2304.09352*, 2023.

[38] J. S. Lee and E. C. Choi, "CO_2 leakage environmental damage cost – A CCS project in South Korea," *Renewable and Sustainable Energy Reviews*, vol. 93, pp. 753–758, 2018.

[39] A. Corso, Y. Wang, M. Zechner, J. Caers, and M. J. Kochenderfer, "A POMDP Model for Safe Geological Carbon Sequestration," *NeurIPS Workshop on Tackling Climate Change with Machine Learning*, 2022.

[40] W. Zhang and H. Wang, "Diagnostic Policies Optimization for Chronic Diseases Based on POMDP Model," *Healthcare*, vol. 10, no. 2, 2022.

[41] K. H. Wray, B. Lange, A. Jamgochian, S. J. Witwicki, A. Kobashi, S. Hagaribommanahalli, and D. Ilstrup, "POMDPs for Safe Visibility Reasoning in Autonomous Vehicles," in *International Conference on Intelligence and Safety for Robotics (ISR)*, 2021.

[42] A. R. Sankar, P. Doshi, and A. Goodie, "Evacuate or Not? A POMDP Model of the Decision Making of Individuals in Hurricane Evacuation Zones," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020.

[43] S. Cho, S. Cho, and K. C. Yow, "A Robust Time Series Prediction Model Using POMDP and Data Analysis," *Journal of Advances in Information Technology (JAIT)*, 2017.

[44] A. R. Cassandra, "Exact and Approximate Algorithms for Partially Observable Markov Decision Processes," Ph.D. dissertation, Brown University, 1998.

[45] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[46] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, 1960.

[47] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," *University of North Carolina at Chapel Hill*, 1995.

[48] S. J. Julier and J. K. Uhlmann, "New Extension of the Kalman Filter to Nonlinear Systems," in *Signal Processing, Sensor Fusion, and Target Recognition*, vol. 3068, 1997.

[49] E. A. Wan and R. Van Der Merwe, "The Unscented Kalman Filter for Nonlinear Estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium (AS-SPCC)*, 2000.

[50] J. Uhlmann, *First-Hand: The Unscented Transform*, `https://ethw.org/First-Hand:The_Unscented_Transform`, 2021.

[51] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," in *IEEE for Radar and Signal Processing*, vol. 140, 1993.

[52] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[53] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *NIPS Deep Learning Workshop*, 2013.

[54] C. J. C. H. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, King's College, 1989.

[55] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[56] J. Subramanian, A. Sinha, R. Seraj, and A. Mahajan, "Approximate Information State for Approximate Planning and Reinforcement Learning in Partially Observed Systems," *Journal of Machine Learning Research (JMLR)*, vol. 23, no. 1, pp. 483–565, 2022.

[57] M. Lauri, D. Hsu, and J. Pajarinen, "Partially Observable Markov Decision Processes in Robotics: A Survey," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 21–40, 2022.

[58] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[59] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[60] D. Silver and J. Veness, "Monte-Carlo Planning in Large POMDPs," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 23, 2010.

[61] Z. N. Sunberg and M. J. Kochenderfer, "Online Algorithms for POMDPs with Continuous State, Action, and Observation Spaces," in *International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 28, 2018.

[62] J. Fischer and Ö. S. Tas, "Information Particle Filter Tree: An Online Algorithm for POMDPs with Belief-Based Rewards on Continuous Domains," in *International Conference on Machine Learning (ICML)*, 2020.

[63] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *European Conference on Machine Learning (ECML)*, 2006.

[64] C. D. Rosin, "Multi-Armed Bandits with Episode Context," *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 3, pp. 203–230, 2011.

[65] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, 2017.

[66] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous Upper Confidence Trees," in *Learning and Intelligent Optimization*, 2011.

[67] S. Sokota, C. Y. Ho, Z. Ahmad, and J. Z. Kolter, "Monte Carlo Tree Search with Iteratively Refining State Abstractions," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021.

[68] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, 2016.

[69] F. C. Schadd, "Monte-Carlo Search Techniques in the Modern Board Game Thurn and Taxis," *Maastricht University: Maastricht, The Netherlands*, 2009.

[70] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[71] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[72] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. MIT Press, 2019.

[73] P. I. Frazier, "A Tutorial on Bayesian Optimization," *arXiv:1807.02811*, 2018.

[74] R. Garnett, *Bayesian Optimization*. Cambridge University Press, 2023.

[75] M. G. Genton, "Classes of Kernels for Machine Learning: A Statistics Perspective," *Journal of Machine Learning Research (JMLR)*, vol. 2, pp. 299–312, 2002.

[76] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[77] G. Hinton, N. Srivastava, and K. Swersky, *Neural Networks for Machine Learning. Lecture 6a: Overview of Mini-Batch Gradient Descent*, `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, 2014.

[78] A. Donzé and O. Maler, "Robust Satisfaction of Temporal Logic Over Real-Valued Signals," in *International Conference on Formal Modeling and Analysis of Timed Systems*, 2010.

[79] A. Donzé, "Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems," in *Computer Aided Verification*, 2010.

[80] R. Lee, O. J. Mengshoel, A. Saksena, R. W. Gardner, D. Genin, J. Silbermann, M. Owen, and M. J. Kochenderfer, "Adaptive Stress Testing: Finding Likely Failure Events with Reinforcement Learning," *Journal of Artificial Intelligence Research (JAIR)*, vol. 69, pp. 1165–1201, 2020.

[81] T. A. Gooley, W. Leisenring, J. Crowley, and B. E. Storer, "Estimation of Failure Probabilities in the Presence of Competing Risks: New Representations of Old Estimators," *Statistics in Medicine*, vol. 18, no. 6, pp. 695–706, 1999.

[82] J. M. Schumann, *Automated Theorem Proving in Software Engineering*. Springer Science & Business Media, 2001.

[83] M. Fitting, *First-Order Logic and Automated Theorem Proving*. Springer Science & Business Media, 2012.

[84] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*. Springer, 2018.

[85] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," in *Computer Aided Verification*, 2017.

[86] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer, "Algorithms for Verifying Deep Neural Networks," *Foundations and Trends® in Optimization*, vol. 4, no. 3-4, pp. 244–404, 2021.

[87] C. Sidrane, A. Maleki, A. Irfan, and M. J. Kochenderfer, "OVERT: An Algorithm for Safety Verification of Neural Network Control Policies for Nonlinear Systems," *Journal of Machine Learning Research* (*JMLR*), vol. 23, no. 117, pp. 1–45, 2022.

[88] M. König, A. W. Bosman, H. H. Hoos, and J. N. van Rijn, "Critically Assessing the State of the Art in Neural Network Verification," *Journal of Machine Learning Research* (*JMLR*), vol. 25, no. 12, pp. 1–53, 2024.

[89] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *International Joint Conference on Artificial Intelligence* (*IJCAI*), vol. 3, 2003.

[90] A. Nilim and L. El Ghaoui, "Robust control of Markov decision processes with uncertain transition matrices," *Operations Research*, vol. 53, no. 5, pp. 780–798, 2005.

[91] G. Iyengar, "Robust dynamic programming," *Mathematics of Operations Research*, vol. 30, no. 2, pp. 257–280, 2005.

[92] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 29, 2016.

[93] K. Lee, Y. Lee, S. Lee, S. Hwang, and P. Abbeel, "SUNRISE: A simple unified framework for ensemble learning in reinforcement learning," in *International Conference on Machine Learning* (*ICML*), 2021.

[94] D. Steinkraus, I. Buck, and P. Y. Simard, "Using GPUs for Machine Learning Algorithms," in *International Conference on Document Analysis and Recognition* (*ICDAR*), 2005.

[95] J.-B. Grill, F. Altché, Y. Tang, T. Hubert, M. Valko, I. Antonoglou, and R. Munos, "Monte-Carlo tree search as regularized policy optimization," in *International Conference on Machine Learning* (*ICML*), 2020.

[96] T. Cazenave, "Batch Monte Carlo Tree Search," in *International Conference on Computers and Games*, 2022.

[97] D. Bertsekas, *Rollout, Policy Iteration, and Distributed Reinforcement Learning*. Athena Scientific, 2021.

[98] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-Based Batch Mode Reinforcement Learning," *Journal of Machine Learning Research* (*JMLR*), vol. 6, 2005.

[99] R. Sabbadin, J. Lang, and N. Ravoanjanahary, "Purely Epistemic Markov Decision Processes," *AAAI Conference on Artificial Intelligence*, vol. 22, no. 2, pp. 1057–1062, 2007.

[100] K. Morishima, M. Kuno, A. Nishio, N. Kitagawa, Y. Manabe, M. Moto, F. Takasaki, H. Fujii, K. Satoh, H. Kodama, K. Hayashi, S. Odaka, S. Procureur, D. Attié, S. Bouteille, D. Calvet, C. Filosa, P. Magnier, I. Mandjavidze, M. Riallot, B. Marini, P. Gable, Y. Date, M. Sugiura, Y. Elshayeb, T. Elnady, M. Ezzy, E. Guerriero, V. Steiger, N. Serikoff, J.-B. Mouret, B. Charlès, H. Helal, and M. Tayoubi, "Discovery of a big void in Khufu's Pyramid by observation of cosmic-ray muons," *Nature*, vol. 552, no. 7685, pp. 386–390, 2017.

[101] A. Tarantola, *Inverse Problem Theory and Methods for Model Parameter Estimation*. SIAM, 1987.

[102] K. Sohn, H. Lee, and X. Yan, "Learning Structured Output Representation using Deep Conditional Generative Models," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 28, 2015.

[103] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *International Conference on Learning Representations* (*ICLR*), 2013.

[104] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic Backpropagation and Approximate Inference in Deep Generative Models," in *International Conference on Machine Learning* (*ICML*), 2014.

[105] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 27, 2014.

[106] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled Generative Adversarial Networks," in *International Conference on Learning Representations* (*ICLR*), 2017.

[107] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *arXiv preprint arXiv:1411.1784*, 2014.

[108] D. Rezende and S. Mohamed, "Variational Inference with Normalizing Flows," in *International Conference on Machine Learning* (*ICML*), 2015.

[109] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 33, 2020.

[110] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-Based Generative Modeling through Stochastic Differential Equations," in *International Conference on Learning Representations* (*ICLR*), 2021.

[111] A. Q. Nichol and P. Dhariwal, "Improved Denoising Diffusion Probabilistic Models," *International Conference on Machine Learning* (*ICML*), 2021.

[112] M. Janner, J. Fu, and S. Levine, "Planning with Diffusion for Flexible Behavior Synthesis," in *International Conference on Machine Learning* (*ICML*), 2022.

[113] Y. Cao, S. Li, Y. Liu, Z. Yan, Y. Dai, P. S. Yu, and L. Sun, "A Comprehensive Survey of AI-Generated Content (AIGC): A History of Generative AI from GAN to ChatGPT," *arXiv preprint arXiv:2303.04226*, 2023.

[114] R. Huang, S. Liu, R. Qi, and Y. Zhang, "Deep Learning 3D Sparse Inversion of Gravity Data," *Journal of Geophysical Research: Solid Earth*, vol. 126, no. 11, 2021.

[115] C. Deng, S. Feng, H. Wang, X. Zhang, P. Jin, Y. Feng, Q. Zeng, Y. Chen, and Y. Lin, "OpenFWI: Large-scale Multi-structural Benchmark Datasets for Full Waveform Inversion," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 35, 2022.

[116] V. Puzyrev, "Deep learning electromagnetic inversion with convolutional neural networks," *Geophysical Journal International*, vol. 218, no. 2, pp. 817–832, 2019.

[117] Z. Hu, S. Liu, X. Hu, L. Fu, J. Qu, H. Wang, and Q. Chen, "Inversion of magnetic data using deep neural networks," *Physics of the Earth and Planetary Interiors*, vol. 311, no. 106653, 2021.

[118] W. Menke, *Geophysical Data Analysis: Discrete Inverse Theory*. Academic Press, 1989.

[119] A. N. Tikhonov, "Solution of Incorrectly Formulated Problems and the Regularization Method," *Soviet Mathematics Doklady*, vol. 4, pp. 1035–1038, 1963.

[120] S. C. Constable, R. L. Parker, and C. G. Constable, "Occam's Inversion: A Practical Algorithm for Generating Smooth Models from Electromagnetic Sounding Data," *Geophysics*, vol. 52, no. 3, pp. 289–300, 1987.

[121] R. L. Parker, *Geophysical Inverse Theory*. Princeton University Press, 1994.

[122] A. C. Reynolds, N. He, L. Chu, and D. S. Oliver, "Reparameterization techniques for generating reservoir descriptions conditioned to variograms and well-test pressure data," *SPE Journal*, vol. 1, no. 04, pp. 413–426, 1996.

[123] M. Esmaeili, M. Ahmadi, and A. Kazemi, "Kernel-based two-dimensional principal component analysis applied for parameterization in history matching," *Journal of Petroleum Science and Engineering*, vol. 191, no. 107134, 2020.

[124] A. Tarantola and B. Valette, "Inverse Problems in Geophysics," *Geophysics*, vol. 50, no. 3, pp. 493–503, 1985.

[125] M. S. Zhdanov, *Geophysical Inverse Theory and Regularization Problems*, 2nd. Elsevier, 2015.

[126] W. A. McAliley and Y. Li, "Stochastic inversion of geophysical data by a conditional variational autoencoder," *Geophysics*, vol. 89, no. 1, WA219–WA232, 2024.

[127] M. Chung, E. Hart, J. Chung, B. Peters, and E. Haber, "Paired autoencoders for likelihood-free estimation in inverse problems," *Machine Learning: Science and Technology*, vol. 5, no. 4, 2024.

[128] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[129] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, pp. 97–109, 1 1970.

[130] K. Mosegaard and A. Tarantola, "Monte Carlo sampling of solutions to inverse problems," *Journal of Geophysical Research: Solid Earth*, vol. 100, no. B7, pp. 12 431–12 447, 1995.

[131] E. Laloy, R. Hérault, J. Lee, D. Jacques, and N. Linde, "Inversion using a new low-dimensional representation of complex binary geological media based on a deep neural network," *Advances in Water Resources*, vol. 110, pp. 387–405, 2017.

[132] M. Li, X.-s. Yan, and M.-z. Zhang, "A comprehensive review of seismic inversion based on neural networks," *Earth Science Informatics*, vol. 16, no. 4, pp. 2991–3021, 2023.

[133] P. Pinheiro and R. Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling," in *International Conference on Machine Learning (ICML)*, 2014.

[134] K. Sohn, W. Shang, and H. Lee, "Improved Multimodal Deep Learning with Variation of Information," in *Advances in Neural Information Processing Systems* (*NIPS*), vol. 27, 2014.

[135] J. Walker, C. Doersch, A. Gupta, and M. Hebert, "An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders," in *European Conference on Computer Vision* (*ECCV*), 2016.

[136] M. Babaeizadeh, C. Finn, D. Erhan, R. H. Campbell, and S. Levine, "Stochastic Variational Video Prediction," in *International Conference on Learning Representations* (*ICLR*), 2018.

[137] A. van den Oord, Y. Li, and O. Vinyals, "Representation Learning with Contrastive Predictive Coding," *arXiv preprint arXiv:1807.03748*, 2018.

[138] O. Ivanov, M. Figurnov, and D. Vetrov, "Variational Autoencoder with Arbitrary Conditioning," in *International Conference on Learning Representations* (*ICLR*), 2019.

[139] N. Dilokthanakul, P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan, "Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders," in *International Conference on Learning Representations* (*ICLR*), 2016.

[140] J. M. Tomczak and M. Welling, "VAE with a VampPrior," in *International Conference on Artificial Intelligence and Statistics* (*AISTATS*), vol. 84, 2018.

[141] E. Jang, S. Gu, and B. Poole, "Categorical Reparametrization with Gumble-Softmax," in *International Conference on Learning Representations* (*ICLR*), 2017.

[142] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved Variational Inference with Inverse Autoregressive Flow," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 29, 2016.

[143] D. Kingma, T. Salimans, B. Poole, and J. Ho, "Variational Diffusion Models," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 34, 2021.

[144] M. Halawa, O. Hellwich, and P. Bideau, "Action-based Contrastive Learning for Trajectory Prediction," in *European Conference on Computer Vision* (*ECCV*), 2022.

[145] R. Zheng, X. Wang, Y. Sun, S. Ma, J. Zhao, H. Xu, H. Daumé III, and F. Huang, "TACO: Temporal Latent Action-Driven Contrastive Loss for Visual Reinforcement Learning," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 36, 2023.

[146] S. Imambi, K. B. Prakash, and G. R. Kanagachidambaresan, "PyTorch," *Programming with TensorFlow*, pp. 87–104, 2021.

[147] L. J. Schultz, G. S. Blanpied, K. N. Borozdin, A. M. Fraser, N. W. Hengartner, A. V. Klimenko, C. L. Morris, C. Orum, and M. J. Sossong, "Statistical Reconstruction for Cosmic Ray Muon Tomography," *IEEE Transactions on Image Processing*, vol. 16, no. 8, pp. 1985–1993, 2007.

[148] J. Kaipio and E. Somersalo, *Statistical and Computational Inverse Problems*. Springer Science & Business Media, 2006, vol. 160.

[149] S. K. Haldar, *Mineral Exploration: Principles and Applications*. Elsevier, 2018.

[150] J. Mern, A. Yildiz, L. Bush, T. Mukerji, and M. J. Kochenderfer, "Improved POMDP Tree Search Planning with Prioritized Action Branching," *AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, pp. 11 888–11 894, 2021.

[151] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention* (*MICCAI*), 2015.

[152] J. Caers, C. Scheidt, Z. Yin, L. Wang, T. Mukerji, and K. House, "Efficacy of Information in Mineral Exploration Drilling," *Natural Resources Research*, vol. 31, no. 3, pp. 1157–1173, 2022.

[153] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, pp. 235–256, 2002.

[154] E. Kaufmann, O. Cappé, and A. Garivier, "On Bayesian Upper Confidence Bounds for Bandit Problems," in *International Conference on Artificial Intelligence and Statistics* (*AISTATS*), 2012.

[155] A. Platzer, "Visualization of SNPs with t-SNE," *PLOS One*, vol. 8, no. 2, pp. 1–6, 2013.

[156] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Machine Learning Proceedings*, Elsevier, 1995, pp. 362–370.

[157] G. Shani, J. Pineau, and R. Kaplow, "A survey of point-based POMDP solvers," *Autonomous Agents and Multi-Agent Systems*, vol. 27, pp. 1–51, 2013.

[158] Y. Wang, M. Zechner, J. M. Mern, M. J. Kochenderfer, and J. K. Caers, "A sequential decision-making framework with uncertainty quantification for groundwater management," *Advances in Water Resources*, vol. 166, no. 104266, 2022.

[159] N. Ye, A. Somani, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP Planning with Regularization," *Journal of Artificial Intelligence Research (JAIR)*, vol. 58, pp. 231–266, 2017.

[160] I. Danihelka, A. Guez, J. Schrittwieser, and D. Silver, "Policy Improvement by Planning with Gumbel," in *International Conference on Learning Representations (ICLR)*, 2022.

[161] J. Czech, P. Korus, and K. Kersting, "Improving AlphaZero Using Monte-Carlo Graph Search," in *International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 31, 2021.

[162] T. O. Hodson, "Root-mean-square error (RMSE) or mean absolute error (MAE): When to use them or not," *Geoscientific Model Development*, vol. 15, no. 14, pp. 5481–5487, 2022.

[163] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature," *Geoscientific Model Development*, vol. 7, no. 3, pp. 1247–1250, 2014.

[164] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "A0C: Alpha Zero in Continuous Action Space," *arXiv preprint arXiv:1805.09613*, 2018.

[165] T. Yee, V. Lisỳ, M. H. Bowling, and S. Kambhampati, "Monte Carlo Tree Search in Continuous Action Spaces with Execution Uncertainty," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

[166] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[167] P. Cai and D. Hsu, "Closing the Planning–Learning Loop With Application to Autonomous Driving," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 998–1011, 2022.

[168] M. H. Lim, T. J. Becker, M. J. Kochenderfer, C. J. Tomlin, and Z. N. Sunberg, "Optimality Guarantees for Particle Belief Approximation of POMDPs," *Journal of Artificial Intelligence Research (JAIR)*, vol. 77, pp. 1591–1636, 2023.

[169] P.-A. Coquelin, R. Deguest, and R. Munos, "Particle Filter-Based Policy Gradient in POMDPs," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 21, 2008.

[170] X. Chen, Y. M. Mu, P. Luo, S. Li, and J. Chen, "Flow-Based Recurrent Belief State Learning for POMDPs," in *International Conference on Machine Learning (ICML)*, 2022.

[171] N. Roy, G. Gordon, and S. Thrun, "Finding Approximate POMDP Solutions Through Belief Compression," *Journal of Artificial Intelligence Research (JAIR)*, vol. 23, pp. 1–40, 2005.

[172] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine, "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[173] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver, "Learning and Planning in Complex Action Spaces," in *International Conference on Machine Learning (ICML)*, 2021.

[174] I. Antonoglou, J. Schrittwieser, S. Ozair, T. K. Hubert, and D. Silver, "Planning in Stochastic Environments with a Learned Model," in *International Conference on Learning Representations (ICLR)*, 2021.

[175] S. Ozair, Y. Li, A. Razavi, I. Antonoglou, A. Van Den Oord, and O. Vinyals, "Vector Quantized Models for Planning," in *International Conference on Machine Learning (ICML)*, 2021.

[176] T. Kimura, K. Sakamoto, and T. Sogabe, "Development of AlphaZero-Based Reinforcment Learning Algorithm for Solving Partially Observable Markov Decision Process (POMDP) Problem," *Bulletin of Networking, Computing, Systems, and Software*, vol. 9, no. 1, pp. 69–73, 2020.

[177] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[178] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson, "Deep Variational Reinforcement Learning for POMDPs," in *International Conference on Machine Learning (ICML)*, 2018.

[179] C.-J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, "Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 2, pp. 294–305, 2019.

[180] C. Wu, G. Yang, Z. Zhang, Y. Yu, D. Li, W. Liu, and J. Hao, "Adaptive Online Packing-guided Search for POMDPs," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021.

[181] P. Cai, Y. Luo, D. Hsu, and W. S. Lee, "HyP-DESPOT: A Hybrid Parallel Algorithm for Online Planning under Uncertainty," *The International Journal of Robotics Research*, vol. 40, no. 2-3, pp. 558–573, 2021.

[182] G. Mazzi, D. Meli, A. Castellini, and A. Farinelli, "Learning Logic Specifications for Soft Policy Guidance in POMCP," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2023.

[183] C. Wu, G. Yang, Z. Zhang, Y. Yu, D. Li, W. Liu, and J. Hao, *OpenReview Response: Adaptive Online Packing-guided Search for POMDPs*, `https://openreview.net/forum?id=0zvTBoQb5PA&noteId=bJTC_0sZPzr`, 2021.

[184] K. Csilléry, M. G. B. Blum, O. E. Gaggiotti, and O. François, "Approximate Bayesian Computation (ABC) in practice," *Trends in Ecology & Evolution*, vol. 25, no. 7, pp. 410–418, 2010.

[185] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network," *Physica D: Nonlinear Phenomena*, vol. 404, 2020.

[186] J. Schrittwieser, *MuZero Intuition*, `https://www.furidamu.org/blog/2020/12/22/muzero-intuition`, 2020.

[187] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade*, Springer, 2002, pp. 9–50.

[188] M. Salehi, H. Mirzaei, D. Hendrycks, Y. Li, M. H. Rohban, and M. Sabokrou, "A Unified Survey on Anomaly, Novelty, Open-Set, and Out of-Distribution Detection: Solutions and Future Challenges," *Transactions of Machine Learning Research*, 2022.

[189] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep Sets," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[190] M. Egorov, Z. N. Sunberg, E. Balaban, T. A. Wheeler, J. K. Gupta, and M. J. Kochenderfer, "POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty," *Journal of Machine Learning Research (JMLR)*, vol. 18, no. 26, pp. 1–5, 2017.

[191] P. Santana, S. Thiébaux, and B. Williams, "RAO*: An Algorithm for Chance-Constrained POMDPs," *AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[192] I. Gibbs and E. Candes, "Adaptive Conformal Inference Under Distribution Shift," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 1660–1672, 2021.

[193] A. C. Busch, *Methodology for Establishing A Target Level of Safety*. Federal Aviation Administraton Technical Center, 1985.

[194] M. J. Flannery, "Capital Regulation and Insured Banks Choice of Individual Loan Default Risks," *Journal of Monetary Economics*, vol. 24, no. 2, pp. 235–258, 1989.

[195] R. J. Moss, A. Corso, J. Caers, and M. J. Kochenderfer, "BetaZero: Belief-State Planning for Long-Horizon POMDPs using Learned Approximations," *Reinforcement Learning Journal (RLJ)*, vol. 1, pp. 158–181, 2024.

[196] J. Fischer, E. Bührle, D. Kamran, and C. Stiller, "Guiding Belief Space Planning with Learned Models for Interactive Merging," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2022.

[197] T. Brázdil, K. Chatterjee, P. Novotnỳ, and J. Vahala, "Reinforcement Learning of Risk-Constrained Policies in Markov Decision Processes," *AAAI Conference on Artificial Intelligence*, vol. 34, pp. 9794–9801, 2020.

[198] B. J. Ayton and B. C. Williams, "Vulcan: A Monte Carlo Algorithm for Large Chance Constrained MDPs with Risk Bounding Functions," *arXiv:1809.01220*, 2018.

[199] J. D. Isom, S. P. Meyn, and R. D. Braatz, "Piecewise Linear Dynamic Programming for Constrained POMDPs," *AAAI Conference on Artificial Intelligence*, vol. 1, pp. 291–296, 2008.

[200] J. Lee, G.-H. Kim, P. Poupart, and K.-E. Kim, "Monte-Carlo Tree Search for Constrained POMDPs," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[201] D. Parthasarathy, G. Kontes, A. Plinge, and C. Mutschler, "C-MCTS: Safe Planning with Monte Carlo Tree Search," *arXiv:2305.16209*, 2023.

[202] A. Jamgochian, A. Corso, and M. J. Kochenderfer, "Online Planning for Constrained POMDPs with Continuous Spaces through Dual Ascent," *International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 33, no. 1, pp. 198–202, 2023.

[203] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning*, Elsevier, 1994, pp. 157–163.

[204] S. Carpin and T. C. Thayer, "Solving Stochastic Orienteering Problems with Chance Constraints Using Monte Carlo Tree Search," in *International Conference on Automation Science and Engineering (CASE)*, 2022.

[205] M. Zaffran, O. Féron, Y. Goude, J. Josse, and A. Dieuleveut, "Adaptive Conformal Predictions for Time Series," in *International Conference on Machine Learning (ICML)*, 2022.

[206] RTCA, *Minimum Operational Performance Standards (MOPS) for Detect and Avoid (DAA) Systems*, DO-365, 2017.

[207] M. P. Owen, A. Panken, R. Moss, L. Alvarez, and C. Leeper, "ACAS Xu: Integrated Collision Avoidance and Detect and Avoid Capability for UAS," in *IEEE/AIAA Digital Avionics Systems Conference (DASC)*, 2019.

[208] K. Abu-Jbara, W. Alheadary, G. Sundaramorthi, and C. Claudel, "A Robust Vision-based Runway Detection and Tracking Algorithm for Automatic UAV Landing," in *International Conference on Unmanned Aircraft Systems*, 2015.

[209] G. Balduzzi, M. Ferrari Bravo, A. Chernova, C. Cruceru, L. van Dijk, P. de Lange, J. Jerez, N. Koehler, M. Koerner, C. Perret-Gentil, Z. Pillio, R. Polak, H. Silva, R. Valentin, I. Whittington, and G. Yakushev, "Neural Network Based Runway Landing Guidance for General Aviation Autoland," United States. Department of Transportation. Federal Aviation Administration, Tech. Rep., 2021.

[210] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A Tutorial on the Cross-Entropy Method," *Annals of Operations Research*, vol. 134, pp. 19–67, 2005.

[211] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*. Springer, 1999, vol. 2.

[212] A. B. Owen, *Monte Carlo Theory, Methods and Examples*. Stanford University, 2013.

[213] R. Y. Rubinstein and D. P. Kroese, *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer, 2004, vol. 133.

[214] R. J. Moss, "Cross-Entropy Method Variants for Optimization," *arXiv:2009.09043*, 2020.

[215] C. Miller, J. N. Corcoran, and M. D. Schneider, "Rare Events via Cross-Entropy Population Monte Carlo," *IEEE Signal Processing Letters*, vol. 29, pp. 439–443, 2021.

[216] M. Arief, Z. Huang, G. K. S. Kumar, Y. Bai, S. He, W. Ding, H. Lam, and D. Zhao, "Deep Probabilistic Accelerated Evaluation: A Robust Certifiable Rare-Event Simulation Methodology for Black-Box Safety-Critical Systems," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

[217] O. Cappé, A. Guillin, J.-M. Marin, and C. P. Robert, "Population Monte Carlo," *Journal of Computational and Graphical Statistics*, vol. 13, no. 4, pp. 907–929, 2004.

[218] V. Elvira and E. Chouzenoux, "Optimized Population Monte Carlo," *IEEE Transactions on Signal Processing*, vol. 70, pp. 2489–2501, 2022.

[219] M. F. Bugallo, V. Elvira, L. Martino, D. Luengo, J. Miguez, and P. M. Djuric, "Adaptive Importance Sampling: The Past, the Present, and the Future," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 60–79, 2017.

[220] D. Luengo, L. Martino, M. Bugallo, V. Elvira, and S. Särkkä, "A survey of Monte Carlo methods for parameter estimation," *EURASIP Journal on Advances in Signal Processing*, vol. 2020, no. 1, pp. 1–62, 2020.

[221] E. Vazquez and J. Bect, "A sequential Bayesian algorithm to estimate a probability of failure," *IFAC Symposium on System Identification*, vol. 42, no. 10, pp. 546–550, 2009.

[222] H. Wang, G. Lin, and J. Li, "Gaussian process surrogates for failure detection: A Bayesian experimental design approach," *Journal of Computational Physics*, vol. 313, pp. 247–259, 2016.

[223] A. Renganathan, V. Rao, and I. Navon, "Multifidelity Gaussian processes for failure boundary and probability estimation," in *AIAA SCITECH 2022 Forum*, 2022.

[224] Y. He and J. Schumann, "A Framework for the Analysis of Deep Neural Networks in Aerospace Applications using Bayesian Statistics," in *International Joint Conference on Neural Networks (IJCNN)*, 2020.

[225] P. Ranjan, D. Bingham, and G. Michailidis, "Sequential Experiment Design for Contour Estimation from Complex Computer Codes," *Technometrics*, vol. 50, no. 4, pp. 527–541, 2008.

[226] J. Mockus and J. Mockus, "Global Optimization and the Bayesian Approach," *Bayesian Approach to Global Optimization. Mathematics and Its Applications*, vol. 37, 1989.

[227] C. K. Williams and D. Barber, "Bayesian Classification with Gaussian Processes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, pp. 1342–1351, 1998.

[228] H. Nickisch and C. E. Rasmussen, "Approximations for Binary Gaussian Process Classification," *Journal of Machine Learning Research (JMLR)*, vol. 9, no. Oct, pp. 2035–2078, 2008.

[229] B. S. Jensen, J. B. Nielsen, and J. Larsen, "Bounded Gaussian Process Regression," in *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2013.

[230] H. Kahn and A. W. Marshall, "Methods of Reducing Sample Size in Monte Carlo Computations," *Journal of the Operations Research Society of America*, vol. 1, no. 5, pp. 263–278, 1953.

[231] J. Quinonero-Candela and C. E. Rasmussen, "A Unifying View of Sparse Approximate Gaussian Process Regression," *Journal of Machine Learning Research (JMLR)*, vol. 6, pp. 1939–1959, 2005.

[232] D. M. Himmelblau, *Applied Nonlinear Programming*. McGraw Hill, 1972.

[233] Laminar Research, *X-Plane Flight Simulator*, `https://www.x-plane.com/`, 2023.

[234] J. M. Esposito, J. Kim, and V. Kumar, "Adaptive RRTs for Validating Hybrid Robotic Control Systems," *Algorithmic Foundations of Robotics VI*, vol. 17, pp. 107–121, 2005.

[235] M. D. McKay, R. J. Beckman, and W. J. Conover, "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[236] I. M. Sobol', "On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals," *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, vol. 7, no. 4, pp. 784–802, 1967.

[237] X. Gong and Y. Pan, "Sequential Bayesian experimental design for estimation of extreme-event probability in stochastic input-to-response systems," *Computer Methods in Applied Mechanics and Engineering*, vol. 395, no. 114979, 2022.

[238] M. Binois and N. Wycoff, "A survey on high-dimensional Gaussian process modeling with application to Bayesian optimization," *ACM Transactions on Evolutionary Learning and Optimization*, vol. 2, no. 2, pp. 1–26, 2022.

[239] J. W. Gregory and T. Liu, *Introduction to Flight Testing*. John Wiley & Sons, 2021.

[240] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[241] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," in *European Conference on Computer Vision (ECCV)*, 2016.

[242] J. Rüter, U. Durak, and J. C. Dauer, "Investigating the Sim-to-Real Generalizability of Deep Learning Object Detection Models," *Journal of Imaging*, vol. 10, no. 259, 2024.

[243] J. Fairbrother, C. Nemeth, M. Rischard, J. Brea, and T. Pinder, "GaussianProcesses.jl: A Nonparametric Bayes Package for the Julia Language," *Journal of Statistical Software*, vol. 102, pp. 1–36, 2022.

[244] D. Widmann, W. Tebbutt, st--, T. Galy-Fajou, S. Yalburgi, H. Ge, S. C. Surace, D. Vicente, J. Vaverka, J. Skovbekk, N. Bosch, N. Schmitz, R. Viljoen, T. Wright, Vikram, and A. Koher, *JuliaGaussianProcesses/AbstractGPs.jl*, Zenodo, 10.5281/zenodo.8377741, 2023.

[245] R. J. Moss, A. Jamgochian, J. Fischer, A. Corso, and M. J. Kochenderfer, "Constrained-Zero: Chance-Constrained POMDP Planning using Learned Probabilistic Failure Surrogates and Adaptive Safety Constraints," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2024.

[246] R. J. Moss, A. Jamgochian, J. Fischer, A. Corso, and M. J. Kochenderfer, "Chance-Constrained POMDP Planning with Learned Neural Network Surrogates," in *IJCAI Workshop on Trustworthy Interactive Decision-Making with Foundation Models (TIDMwFM)*, 2024.

[247] R. J. Moss, M. J. Kochenderfer, M. Gariel, and A. Dubois, "Bayesian Safety Validation for Failure Probability Estimation of Black-Box Systems," *AIAA Journal of Aerospace Information Systems (JAIS)*, vol. 21, no. 7, pp. 533–546, 2024.

[248] R. J. Moss, M. J. Kochenderfer, M. Gariel, and A. Dubois, "Bayesian Safety Validation for Black-Box Systems," in *AIAA AVIATION Forum*, 2023.

[249] M. J. Kochenderfer, S. M. Katz, A. L. Corso, and R. J. Moss, *Algorithms for Validation*. MIT Press, 2025.

[250] M. Althoff, G. Frehse, and A. Girard, "Set Propagation Techniques for Reachability Analysis," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 369–395, 2021.

[251] K. Makino and M. Berz, "Taylor Models and Other Validated Functional Inclusion Methods," *International Journal of Pure and Applied Mathematics*, vol. 6, pp. 239–316, 2003.

[252] M. Everett, G. Habibi, C. Sun, and J. P. How, "Reachability Analysis of Neural Feedback Loops," *IEEE Access*, vol. 9, pp. 163 938–163 953, 2021.

[253] N. Bougie, T. Onishi, and Y. Tsuruoka, "Interpretable Imitation Learning with Symbolic Rewards," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 1, pp. 1–34, 2023.

[254] B. De Ville, "Decision trees," *WIREs Computational Statistics*, vol. 5, no. 6, pp. 448–455, 2013.

[255] M. Ono, M. Pavone, Y. Kuwata, and J. Balaram, "Chance-constrained dynamic programming with application to risk-aware robotic space exploration," *Autonomous Robots*, vol. 39, pp. 555–571, 2015.

[256] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer, "Formal verification of ACAS X, an industrial airborne collision avoidance system," in *International Conference on Embedded Software (EMSOFT)*, 2015.

[257] S. C. W. Ong, S. W. Png, D. Hsu, and W. S. Lee, "Planning under Uncertainty for Robotic Tasks with Mixed Observability," *The International Journal of Robotics Research*, vol. 29, no. 8, 2010.

[258] J. Pajarinen and V. Kyrki, "Robotic Manipulation of Multiple Objects as a POMDP," *Artificial Intelligence*, vol. 247, pp. 213–228, 2017.

[259] J. Pajarinen, J. Lundell, and V. Kyrki, "POMDP Planning Under Object Composition Uncertainty: Application to Robotic Manipulation," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 41–56, 2022.

[260] G. Wen, C. Hay, and S. M. Benson, "CCSNet: A deep learning modeling suite for CO_2 storage," *Advances in Water Resources*, vol. 155, no. 104009, 2021.

[261] D. Ha and J. Schmidhuber, "World Models," *arXiv preprint arXiv:1803.10122*, 2018.