Randomized Policy Optimization for Optimal Stopping

Ethan Babel, Scott Klosen, Simon Rupp

April 2025

1 Introduction

This project investigates the optimal stopping problem in stochastic systems, with a focus on applications to financial derivatives such as American options. We provide a high-level overview of the optimal stopping framework, introduce key exact and approximate solution methods, and conduct a comparative analysis through practical implementation and experimentation.

This work is motivated by a recent paper on Randomized Policies for Optimal Stopping (Guan and Misic 2022), which proposes a randomized policy optimization (RPO) approach for solving high-dimensional optimal stopping problems via simulation-based learning. We compare this approach to the well-established Least Squares Monte Carlo (LSM) method (Longstaff and Schwartz 2001), which approximates continuation values through regression.

In the course of this project, we implement both methods, explore their theoretical foundations, and evaluate their performance across a range of simulated and data-driven financial scenarios. Our findings illustrate the strengths and trade-offs of each approach and highlight the potential of randomized policy optimization in high-dimensional settings.

2 Background

In this section, we will introduce concepts in stochastic systems that are relevant to the Optimal Stopping Problem (Section 2.1), briefly explain the problem at a high level (Section 2.2), and explain its application to financial derivatives markets (Section 2.3).

2.1 Overview of Stochastic Processes and Systems

Stochastic processes form the mathematical foundation for modeling systems that evolve randomly over time. A stochastic process is defined as a collection of random variables $\{X_t\}_{t \in T}$ indexed by time t, where each X_t takes values in

a state space $\mathcal{X}$. The index set T may be discrete (e.g., $T = \{0, 1, \ldots, T\}$) or continuous (e.g., $T = [0, T]$), depending on the application.

A fundamental building block of continuous-time stochastic processes is Brownian motion. It is the standard model of random movement and serves as the driving noise in many stochastic differential equations.

Definition: A stochastic process $\{W_t\}_{t \geq 0}$ is called a standard Brownian motion if it satisfies the following properties:

1. $W_0 = 0$ almost surely
2. W_t is almost surely continuous
3. W_t has independent increments: for any $0 \leq t_0 < t_1 < \ldots < t_n$, the random variables $W_{t_1} - W_{t_0}, W_{t_2} - W_{t_1}, \ldots, W_{t_n} - W_{t_{n-1}}$ are independent
4. W_t has normally distributed increments: for any $s, t \geq 0$,

$$W_{t+s} - W_s \sim \mathcal{N}(0, t - s),$$

Another common type of stochastic process is geometric Brownian motion (GBM), which is often used to model stock prices under the assumption of continuous time and log-normal returns. GBM is defined by the stochastic differential equation:

$$dS_t = \mu S_t \, dt + \sigma S_t \, dW_t,$$

where S_t is the asset price at time t, μ is the drift rate (the deterministic trend over time), σ is the volatility (representing the magnitude of random fluctuations), and $\{W_t\}_{t \geq 0}$ is a standard Brownian motion. The closed-form solution to this SDE is given by:

$$S(t) = S_0 \cdot \exp\left(\left(\mu - \frac{1}{2}\sigma^2\right) t + \sigma W(t)\right),$$

where S_0 is the initial asset price. This expression shows that under GBM, the natural logarithm of the asset price follows a normal distribution, and the asset price itself follows a log-normal distribution. GBM captures both the compounding effect of returns and the randomness introduced by Brownian motion, making it a foundational model in mathematical finance and option pricing. Under the risk-neutral measure, μ is replaced by the risk-free rate r, enabling arbitrage-free pricing of derivatives.

A system governed by a stochastic process is referred to as a stochastic system. In many settings, the evolution of the process is governed by a Markov property, meaning that the future state depends only on the current state and not on the full history of the process. Formally, this is written as:

$$\mathbb{P}(X_{t+1} \in A \mid X_0, X_1, \ldots, X_t) = \mathbb{P}(X_{t+1} \in A \mid X_t), \quad \text{for all measurable } A \subseteq \mathcal{X}.$$

More generally, stochastic systems are used to model decision-making under uncertainty. In such systems, an agent observes the evolving state and must make decisions that impact the future evolution and outcomes.

2.2 Introduction to the Optimal Stopping Problem

The optimal stopping problem is characterized by a stochastic system that transitions probabilistically at each discrete time step, and a reward function $g(t, x)$ that assigns a payoff for stopping in state x at time t. At each time step $t \in \{0, 1, \ldots, T\}$, an agent observes the current state and decides between the following:

- Stop the process now and collect the current reward
- Continue to the next time step, deferring the decision to a future time and forgoing current reward in hope of a higher future payoff

The goal is to identify a stopping strategy (called a policy) that maximizes the expected reward over the finite horizon.

2.3 Applications to American Options and Other Financial Derivatives

Optimal stopping theory plays a central role in the valuation and exercise of many financial derivatives. The most well-known example is the American option, which grants the holder the right to exercise at any time prior to a fixed maturity date. The decision of when to exercise is naturally formulated as an optimal stopping problem, where the agent must choose the time that maximizes the expected discounted payoff.

In the case of an American put option, for instance, we model the underlying asset price as a discrete or continuous stochastic process $\{S_t\}_{t \in T}$ where T is the set of time steps until maturity ($T = \{0, 1, \ldots, T\}$ in the discrete case and $T = [0, T]$ in the continuous case). In the continuous case, we typically model the underlying asset price as a GBM. The reward function is given by $g(t, S_t) = (K - S_t)^+$, where K is the strike price. Note that for American options $g(t, S_t) \geq 0$ always, as exercising an out-of-the-money option produces zero reward. The option holder faces a trade-off between exercising now for immediate payoff and holding the option in anticipation of a more favorable future value. This framework extends to a variety of other financial instruments, including callable bonds, convertible securities, employee stock options, and certain types of exotic options such as Bermudan or swing options.

3 Optimal Stopping Problem

In this section, we formalize the optimal stopping problem and review established methods for solving it in low-dimensional settings. We begin by defining the problem rigorously in discrete time and finite horizon (Section 3.1), which will serve as the foundation for both exact and approximate approaches. We then present classical solution methods (Section 3.2), including dynamic programming in discrete settings (Section 3.2.1) and PDE-based approaches in continuous time (Section 3.2.2), such as the free-boundary Black-Scholes

framework. Finally, we discuss the limitations of these exact methods in high-dimensional problems due to the curse of dimensionality (Section 3.2.3), which motivates the use of approximate and simulation-based techniques explored later in the report.

3.1 Formal Problem Definition

We now present the general formulation of the finite-horizon optimal stopping problem, following the setup introduced by Guan and Mišić (Guan and Misic 2022).

Let $x(t)$ denote the state of the system at time $t \in [T]$. At each time t, the decision-maker observes the current state $x(t)$ and must choose whether to stop the process and collect a reward, or to continue to the next time step.

The decision to stop is encoded in a stopping time $\tau \in [T]$, which is a random variable. To determine τ, the decision maker must specify a policy $\pi : [T] \times \mathcal{X} \to \mathbb{R}$, which maps a period and state to an action $A \in \{\text{stop}, \text{continue}\}$. The stopping time defined by the policy is then

$$\tau_\pi = \min\{t \in [T] : \pi(t, X_t) = \text{stop}\}.$$

Letting Π be the set of all possible policies, the objective is to find the policy π that determines a stopping time τ_π that maximizes the expected reward:

$$\sup_{\pi \in \Pi} \mathbb{E}[g(\tau_\pi, x(\tau_\pi))],$$

where g is a reward function.

In the context of financial derivatives, $x(t)$ typically represents the state of an underlying asset (e.g., the stock price), and $g(t, x)$ corresponds to the payoff received upon early exercise of an option. We typically model $x(t)$ as $\{X_t\}_{t=0}^T$, a discrete-time, finite-horizon stochastic process with values in a state space $\mathcal{X} \subseteq \mathbb{R}^d$.

In the following subsections, we review exact methods for solving this problem in low-dimensional settings.

3.2 Low-Dimensional Exact Methods

When the state space is low-dimensional and the system dynamics are known, the optimal stopping problem can often be solved exactly using classical analytical or numerical techniques. These methods leverage dynamic programming principles or partial differential equation (PDE) formulations to compute the value function and determine the optimal stopping rule. While highly effective in one or two dimensions, their applicability breaks down as dimensionality increases due to exponential growth in computational complexity.

3.2.1 Dynamic Programming Approach (Discrete Case)

In discrete-time settings with a finite time horizon and low-dimensional state space, the optimal stopping problem can be solved exactly using dynamic programming. The key idea is to recursively compute the optimal value function by working backward from the final time step.

Let $V(t, x)$ denote the maximum expected reward achievable from state x at time t. At the terminal time T, the value function is simply equal to the reward:

$$V(T, x) = g(T, x).$$

For earlier time steps $t = T - 1, T - 2, \ldots, 0$, the value function satisfies the dynamic programming recursion:

$$V(t, x) = \max \left\{ g(t, x(t)),\ \mathbb{E}[V(t + 1, x(t + 1)) \mid x(t) = x] \right\}.$$

At each step, the agent chooses between stopping and receiving the immediate reward $g(t, x)$, or continuing and receiving the expected future reward. The recursion proceeds backward in time and, when combined with a model of the transition dynamics of x, yields both the optimal value function and an associated stopping policy.

This approach is exact and tractable in problems where the state space can be discretized finely enough to capture relevant structure without incurring prohibitive computational cost. In practice, it is commonly used in low-dimensional financial applications, such as pricing American options on a single underlying asset.

3.2.2 Free-Boundary Black-Scholes, Finite Difference Method (Continuous Case)

In continuous-time settings, optimal stopping problems arising from financial derivatives are often formulated as free-boundary problems involving partial differential equations (PDEs). A classic example is the pricing of American-style options, where the holder may choose to exercise at any time before expiration.

For instance, in the case of an American put option, the value function $V(t, S_t)$, which represents the price of the option when the underlying asset has value S_t at time t, satisfies a modified version of the Black-Scholes PDE. The optimal stopping region, where early exercise is optimal, is determined as part of the solution and defines a time-dependent exercise boundary $S^*(t)$, also known as a free boundary.

Because closed-form solutions do not exist in most cases, numerical methods such as finite difference schemes are used to approximate the option value and the free boundary. These methods discretize time and asset price into a grid and solve the PDE iteratively using techniques such as the implicit method or Crank–Nicolson scheme.

Although accurate and widely used in low-dimensional problems, these techniques do not scale to high-dimensional settings due to the rapid growth in

computational complexity. As a result, they are primarily limited to problems with one or two state variables.

3.3 Curse of Dimensionality

While exact methods such as dynamic programming and finite difference schemes are highly effective in low-dimensional problems, their applicability becomes severely limited as the dimensionality of the state space increases. This challenge, known as the curse of dimensionality, refers to the exponential growth in computational and memory requirements as the number of state variables increases.

For example, if the state space is d-dimensional and each dimension is discretized into n points, the total number of grid points grows with n^d. As a result, exact methods become infeasible even for moderately sized problems (e.g., $d \geq 4$).

To illustrate the severity of the curse of dimensionality, consider a discretization with just $n = 100$ grid points per dimension. For $d = 2$ (e.g., an option on two underlying assets), this results in $100^2 = 10^4$ grid points—easily manageable. However, for $d = 4$, the number of points grows to $100^4 = 10^8$, and for $d = 8$, it reaches $100^8 = 10^{16}$, which far exceeds the storage and computational capabilities of standard numerical methods. This exponential scaling makes exact approaches intractable for even moderately high-dimensional problems.

In practical terms, this means that traditional PDE solvers and backward induction algorithms cannot be used for options dependent on multiple underlying assets, path-dependent features, or additional stochastic factors such as volatility or interest rates.

This limitation motivates the use of simulation-based approximation methods, such as Least Squares Monte Carlo (LSM) and Randomized Policy Optimization (RPO), which avoid explicit enumeration of the state space and instead operate on sampled trajectories. These methods are the focus of the next section.

4 High Dimensional Optimal Stopping

As discussed in the previous section, the exponential growth of the discretized state space with the number of dimensions renders dynamic programming and PDE-based approaches impractical.

To address this challenge, a variety of approximation methods have been developed that rely on simulation and statistical learning. These methods operate on sampled trajectories rather than the full state space, enabling tractable solutions even in complex, high-dimensional environments. In this section, we focus on two such approaches: the classical Least Squares Monte Carlo (LSM) method introduced by Longstaff and Schwartz, and the more recent Randomized Policy Optimization (RPO) method proposed by Guan and Mišić. We describe both

techniques in detail and compare their theoretical underpinnings, strengths, and limitations.

4.1 Linear Policies

Before introducing the LSM or RPO method, we must adapt our optimal stopping problem into something more solvable via optimization. Both this section and the next section (Section 4.2 Sample-Average Approximation Simplification) are directly motivated by sections 3.2 and 3.3 in Guan and Mišić's paper (Guan and Misic 2022).

In theory, the optimal stopping problem allows for an unrestricted class of stopping policies, including highly non-linear or history-dependent decision rules. However, most practical methods in the literature, including the one considered in this work, restrict attention to parametric policy classes that are more tractable to optimize. In particular, we focus on linear policies, as introduced by Guan and Mišić (Guan and Misic 2022).

A linear stopping policy considers a set of pre-defined basis functions $\Phi : \mathcal{X} \to \mathbb{R}^k$, where $\Phi = (\phi_1, \phi_2, \ldots, \phi_k)$ and $\phi_i : \mathcal{X} \to \mathbb{R}\ \forall i \in [k]$. The policy evaluates Φ at a given state $x(t)$, and applies a linear decision rule.

Restricting to linear policies simplifies optimization and generalizes well in high dimensions when paired with carefully chosen basis functions.

4.2 Deterministic Linear Policies

As mentioned above, a linear policy considers evaluates a pre-defined set of basis functions Φ at a given state $x(t)$ and applies a linear decision rule. Specifically, for a deterministic linear policy, the policy stops at time t if and only if:

$$b_t^\top \Phi(x(t)) \geq 0,$$

where $b_t = (b_0, b_1, \ldots, b_k) \in \mathbb{R}^k$ is a vector of weights to be learned.

Let $\mathcal{B} \subseteq \mathbb{R}^{kT}$ be the set of all feasible parameter vectors. Let

$$\Pi_{\mathcal{B}} = \{\pi_b : b \in \mathcal{B}\}$$

be the corresponding set of linear policies. Then the objective function of our optimization over b, which we'll denote $J_D(b)$, is as follows:

$$J_D(b) = \sup_{\pi \in \Pi_{\mathcal{B}}} \mathbb{E}[g(\tau_\pi, x(\tau_\pi))]$$

4.3 Randomized Linear Policies

In contrast to deterministic policies, a randomized linear policy does not deterministically stop when the decision boundary is crossed. Instead, it stops at time t with a probability determined by applying a sigmoid activation function

to the linear combination of basis functions. Specifically, a randomized linear policy stops at time t with probability:

$$\sigma(b_t^\top \Phi(x(t))) = \frac{1}{1 + e^{-b_t^\top \Phi(x(t))}}$$

where $\sigma(\cdot)$ is the standard logistic sigmoid function, and $b_t \in \mathbb{R}^k$ is the parameter vector at time t.

Let $\mathcal{B} \subseteq \mathbb{R}^{kT}$ be the set of feasible parameter vectors, and define the set of all randomized linear policies as:

$$\Pi_{\mathcal{B}}^R = \{\pi_b^R : b \in \mathcal{B}\},$$

where each π_b^R specifies a probability distribution over stopping times induced by the randomized stopping rule defined above.

We define the objective function of the optimization over randomized linear policies as:

$$J_R(b) = \sup_{\pi \in \Pi_{\mathcal{B}}^R} \mathbb{E}[g(\tau_\pi, x(\tau_\pi))],$$

where τ_π is the (random) stopping time generated by the randomized policy π_b^R, and where the expectation is taken over both the stochastic process $\{x(t)\}_{t=1}^T$ and the random stopping decisions.

This formulation generalizes the deterministic linear policy framework by allowing for smooth decision boundaries and introduces randomness into the stopping rule, which can improve optimization tractability and robustness when sample sizes are limited or gradients are noisy.

4.4 The Sample Average Approximation (SAA) Simplification

Even when restricting the policy class to deterministic or randomized linear policies, evaluating the objective function exactly remains challenging. If the underlying stochastic process is complex or not fully known in closed form, computing the expected reward requires integrating over an intractable or unknown distribution. In practice, this makes direct optimization over the true objective computationally expensive or infeasible.

To address this, we adopt the Sample Average Approximation (SAA) framework, which replaces the expectation in the objective function with an empirical average over a set of simulated trajectories. Let $\{x^{(1)}, \ldots, x^{(n)}\}$ denote a sample of n independent trajectories generated from the stochastic process. The SAA versions of the deterministic and randomized objective functions are given as follows:

Deterministic policy SAA objective:

$$\hat{J}_D(b) = \frac{1}{n} \sum_{i=1}^{n} g(\tau_{\pi_b}^{(i)}, x^{(i)}(\tau_{\pi_b}^{(i)})),$$

where $\tau_{\pi_b}^{(i)}$ is the stopping time induced by the deterministic policy π_b, defined by the parameter vector b, on trajectory $x^{(i)}$.

Randomized policy SAA objective:

$$\hat{J}_R(b) = \frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T} g(t, x^{(i)}(t)) \cdot \sigma(b_t^\top \Phi(x^{(i)}(t))) \cdot \prod_{s=0}^{t-1}\left(1 - \sigma(b_s^\top \Phi(x^{(i)}(s)))\right),$$

where the inner expression represents the probability that the randomized policy first chooses to stop at time t on trajectory $x^{(i)}$.

The SAA objective serves as a computationally tractable proxy for the true expectation, enabling us to evaluate and optimize policy performance using only sampled trajectories. Throughout the remainder of this report, we will often use the terms "objective function" and "SAA objective" interchangeably, as the SAA problem becomes a stand-in for the original in practical implementations.

4.5 Statistical Properties and Computational Complexity of Randomized SAA

Given that the SAA objective is an approximation of the true expectation, it is important to understand its statistical behavior. In this section, we present two key results: the convergence of the Randomized SAA objective to the true objective as the number of samples increases (Section 4.5.1), and bounds on the error between the SAA objective and the true objective when using a finite number of samples (Section 4.5.2). We also discuss the computational complexity issues associated with the SAA framework, and our method for addressing them (Section 4.5.3).

4.5.1 Convergence of the Randomized SAA Objective

Let $J_R(b)$ denote the true expected reward under a randomized policy and $\hat{J}_{R,n}(b)$ the corresponding SAA estimate over n simulated trajectories. Then, for any fixed policy parameter $b \in \mathcal{B}$,

$$\hat{J}_{R,n}(b) \xrightarrow{a.s.} J_R(b) \quad \text{as } n \to \infty.$$

The proof of this can be found in (Guan and Misic 2022) section 4.1. That is, the SAA objective converges with probability one to the true objective by the Law of Large Numbers. As a result, the SAA optimization problem defined by maximizing $\hat{J}_R(b)$ converges to the true problem as the sample size grows, under mild regularity conditions (e.g., bounded reward function g and compact parameter space $\mathcal{B}$).

4.5.2 Bounds on the Randomized SAA Objective

In the finite-sample setting, it is useful to understand how close the optimal value of the SAA problem is to the true optimal value. Let b_n^* be the optimizer of the SAA problem (i.e., $\hat{J}_R(b_n^*) \geq \hat{J}_R(b)$ for all $b \in \mathcal{B}$), and let b^* be the

true optimizer of $J_R(b)$ over $\mathcal{B}$. Then, with probability $1-\delta$ where $\delta > 0$, the following uniform bound holds:

$$\sup_{b\in\mathcal{B}} \left| \hat{J}_R(b) - J_R(b) \right| \leq \mathcal{O}\left(\mathfrak{R}_n(\mathcal{B}) + \sqrt{\frac{\log(1/\delta)}{n}} \right),$$

where $\mathfrak{R}_n(\mathcal{B})$ is the empirical Rademacher complexity of the policy class. The proof of this can be found in (Guan and Misic 2022) section 4.2. This bound implies that with a sufficiently large sample size n, the SAA estimate uniformly approximates the true objective across all candidate policies in $\mathcal{B}$.

These statistical guarantees made in subsections 4.5.2 and 4.5.3 justify the use of the SAA approximation in practice, particularly when simulation is the only viable way to evaluate policy performance. They also provide theoretical assurances that performance gains observed during empirical optimization are likely to generalize beyond the training sample.

4.5.3 Complexity of the Randomized SAA Problem

Although the randomized policy formulation introduces smoothness into the stopping rule and allows for gradient-based optimization, the resulting Sample Average Approximation (SAA) problem remains computationally challenging. In particular, Guan and Mišić (Guan and Misic 2022) show (via reduction from MAX-3SAT) in section 5.1 and the electronic companion that even in this simplified setting, where the policy is linear and the objective function is approximated via simulation, the SAA problem is, in general, non-convex and NP-hard to solve.

This hardness result holds even when the basis functions are simple, the number of time periods is small, and the state space is low-dimensional. The core challenge arises from the non-convexity of the objective function $\hat{J}_R(b)$, which is composed of nested sigmoid functions and product terms that encode the probability of stopping at each time step. These structures introduce multiple local maxima, making global optimization intractable in worst-case scenarios.

Despite this theoretical hardness, heuristic methods such as stochastic gradient ascent or coordinate search often perform well in practice, especially when the number of basis functions is moderate and the sample size is large. Consequently, while no polynomial-time algorithm is guaranteed to solve the randomized SAA problem globally, empirical optimization remains a practical and effective approach for many real-world applications.

4.6 Equivalence of Deterministic and Randomized Linear Policies

Although deterministic and randomized linear policies differ in their structure and optimization behavior, they are ultimately equivalent in terms of the maximum achievable expected reward as shown by (Guan and Misic 2022) in section

3.5. That is, while randomized policies introduce smoothness and probabilistic stopping rules to facilitate optimization, they do not expand the expressive power of the policy class relative to deterministic policies.

Formally, let $J_D^* = \sup_{b \in \mathcal{B}} J_D(b)$ denote the optimal value under the deterministic linear policy class, and let $J_R^* = \sup_{b \in \mathcal{B}} J_R(b)$ denote the optimal value under the randomized linear policy class. Then the two objectives are equivalent:

$$J_D^* = J_R^*.$$

This result follows from the fact that we can design a randomized linear policy that mimics a deterministic policy defined by the weight vector b, simply by scaling b by some arbitrarily large constant such that the sigmoid function outputs the same decision as the deterministic policy almost surely. Conversely, deterministic policies can be viewed as a limiting case of randomized policies where the sigmoid becomes a step function.

Therefore, although randomized policies are often easier to optimize, especially under the SAA formulation, they do not improve the achievable optimal value over deterministic linear policies. As a result, both formulations can be used interchangeably when the goal is policy evaluation or comparison of value functions.

4.7 Randomized Policy Optimization (RPO) Method

The Randomized Policy Optimization (RPO) method is a simulation-based approach to solving optimal stopping problems that directly optimizes the reward of the policy itself. Introduced by Guan and Mišić (Guan and Misic 2022) in section 5.2, the method operates within the class of randomized linear policies and seeks to find the parameter vector $b \in \mathcal{B}$ that maximizes the sample-average approximation (SAA) of the expected reward.

As described earlier, a randomized linear policy stops at time t with probability:

$$\sigma(b_t^\top \Phi(x(t))) = \frac{1}{1 + e^{-b_t^\top \Phi(x(t))}},$$

and induces a probability distribution over stopping times. Given a set of simulated trajectories $\{x^{(i)}(t)\}_{i=1}^n$, the SAA objective function $\hat{J}_R(b)$ is:

$$\hat{J}_R(b) = \frac{1}{n} \sum_{i=1}^{n} \sum_{t=0}^{T} g(t, x^{(i)}(t)) \cdot \sigma(b_t^\top \Phi(x^{(i)}(t))) \cdot \prod_{s=0}^{t-1} \left(1 - \sigma(b_s^\top \Phi(x^{(i)}(s)))\right).$$

This expression represents the expected reward across all sample trajectories under the randomized stopping policy defined by b, where the stopping probability at time t is weighted by the probability of not stopping in previous periods.

The optimization problem becomes:

$$\max_{b \in \mathcal{B}} \hat{J}_R(b),$$

which is typically solved using stochastic gradient ascent, coordinate search methods, or other optimization algorithms like the Adam optimizer. Although the objective function is non-convex and may contain multiple local optima, the smoothness introduced by the sigmoid function enables the use of gradient-based techniques for scalable optimization.

Unlike methods such as Least Squares Monte Carlo (LSM), which rely on fitting the continuation value and then deriving a stopping rule from it, RPO optimizes policy performance directly. This makes RPO particularly appealing in high-dimensional settings, where value-function approximation is more difficult and direct policy learning may offer better empirical results.

In summary, the RPO method provides a flexible and powerful framework for learning stopping policies through direct optimization, and is especially effective when combined with the linear policy structure and SAA formulation introduced in earlier sections.

4.8 Least Squares Monte Carlo (LSM) Method

The Least Squares Monte Carlo (LSM) method, introduced by Longstaff and Schwartz (Longstaff and Schwartz 2001) and discussed by Guan and Mišić (Guan and Misic 2022) in section 5.3, is one of the most widely used approaches for approximating solutions to optimal stopping problems via simulation. Unlike the RPO framework, which directly parameterizes and optimizes over stopping policies maximizing for expected reward, LSM approximates the value function associated with continuation decisions, and uses this estimate to guide the stopping rule.

The LSM method starts by simulating a set of sample trajectories from the underlying stochastic process. At each timestep, starting from the penultimate period and moving backward in time, LSM performs a regression of the observed future rewards against a set of basis functions evaluated at the current state. This regression estimates the continuation value (the expected reward of continuing rather than stopping) which is then compared to the immediate reward to decide whether to stop at that point in the simulation.

Let $\{x^{(i)}(t)\}_{i=1}^{n}$ denote the simulated state trajectories and $\Phi(x^{(i)}(t))$ the basis function representation of each state. The estimated continuation value $\hat{C}_t(x)$ is obtained by solving a least squares regression:

$$\hat{C}_t(x) = \hat{\beta}_t^\top \Phi(x),$$

where $\hat{\beta}_t$ minimizes the squared error between future observed rewards and the predicted continuation value.

Once the continuation value is estimated, the policy is defined implicitly by stopping whenever the immediate reward exceeds the estimated continuation value:

$$\text{Stop at time } t \text{ if } g(t, x(t)) \geq \hat{C}_t(x(t)).$$

While LSM is computationally efficient and easy to implement, especially in low to moderate dimensions, it has several limitations. Most notably, the

quality of the stopping policy is limited by the ability of the regression model to accurately approximate the continuation value, which can be difficult in high-dimensional or non-linear settings. In contrast, the RPO framework directly optimizes policy performance rather than fitting value functions, which can lead to improved results when the function approximation quality is poor or sample sizes are small.

Despite these differences, LSM remains a strong baseline and is widely used in practice, particularly in the context of pricing American-style derivatives.

4.9 LSM vs. RPO Performance in Application Comparison

Guan and Mišić (Guan and Misic 2022) empirically compare the performance of the LSM and RPO methods across a range of optimal stopping problems, including classical financial applications and synthetic high-dimensional benchmarks. Their experiments evaluate both methods under varying conditions such as the number of basis functions, the number of simulated trajectories, and the dimensionality of the state space.

At a high level, their results show that RPO consistently outperforms LSM in scenarios where the continuation value is difficult to approximate or when the dimensionality of the problem is high. Because LSM relies on regression to estimate continuation values, its performance degrades when the approximation quality of the regression model is poor, particularly when the number of basis functions is limited or the structure of the value function is highly nonlinear. In contrast, RPO directly optimizes policy performance, enabling it to more effectively learn stopping rules even when the underlying value function is difficult to approximate.

Overall, the empirical findings validate the theoretical advantages of RPO and highlight its promise as a more reliable alternative to LSM in complex optimal stopping problems, particularly those with high-dimensional or nonlinear structure.

5 Single-Asset Options Experimentation and Application

In this section, we investigate the behavior and performance of Randomized Policy Optimization (RPO) and Least-Squares Monte Carlo (LSM) in the context of American-style options on a single asset. Our motivation stems from a desire to understand how the distinct methodologies employed by RPO and LSM translate into concrete differences in exercise policies—particularly, the resulting exercise boundaries over time. By working within the single-asset setting, we are able to clearly visualize and compare these boundaries across time steps and price levels. This controlled environment provides a valuable testing ground for interpreting the mechanics of each method and benchmarking their behavior against known financial intuition.

We restrict ourselves initially to single-asset options due to the relative tractability of their dynamics. In contrast to the high-dimensional multi-asset setting, where complexity can obscure the learning signal and limit interpretability, the single-asset case offers a clearer lens through which we can analyze the structure of the learned stopping policies. It also allows us to isolate the effect of basis function choice, training regime, and probabilistic modeling within RPO without the confounding influence of cross-asset correlations or systemic shocks.

Beyond experimental clarity, this single-asset study also provides a stepping stone toward real-world applications. In practical trading contexts, the price of a stock or asset is driven by a variety of observable and latent factors—such as volatility regimes, macroeconomic events, earnings reports, and liquidity constraints. Extensions of LSM and RPO can be naturally adapted to accommodate these factors by incorporating them as part of the state vector, enabling the modeling of more realistic optimal stopping problems. This has direct implications for tasks like optimal liquidation, the exercise of employee stock options under uncertainty, and algorithmic trade execution where timing is critical and decisions are path-dependent.

Ultimately, our experiments lay the foundation for deeper insights into how randomized versus regression-based policies perform in simple and interpretable environments, and how such policies may be expanded for use in complex, data-driven trading systems.

5.1 Experimentation Design

To compare optimal stopping methods in a structured and reproducible manner, we first established a diverse and representative dataset of underlying asset price trajectories. We modeled the dynamics of the asset using a Geometric Brownian Motion (GBM) process, a standard choice in financial mathematics for capturing the continuous-time evolution of asset prices under stochastic volatility and drift.

For our experiments, we focused on a discrete-time setting with $T = 100$ steps, corresponding to the maturity of the option. At each time step, the asset price evolves according to the GBM equation:

$$S_{t+1} = S_t \cdot \exp\left((\mu - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t} \cdot Z_t\right)$$

where μ is the drift, σ is the volatility, and $Z_t \sim \mathcal{N}(0, 1)$ is standard Gaussian noise. We set $\Delta t = 1$ to match the granularity of our discrete time horizon.

To simulate a wide range of market conditions, we constructed GBM trajectories across a grid of initial prices S_0, drifts μ, and volatilities σ. Specifically, we generated paths for all combinations in the following ranges:

- $S_0 \in \{80, 85, 90, \ldots, 115\}$
- $\mu \in \{-0.005, -0.004, \ldots, 0.004\}$
- $\sigma \in \{0.001, 0.002, \ldots, 0.009\}$

For each parameter configuration (S_0, μ, σ), we generated 5 independent GBM paths. With 8 initial prices, 10 drift values, and 9 volatility values, this produced a total of $8 \times 10 \times 9 \times 5 = 3{,}600$ trajectories for training and evaluation. Our SAA problem was then comprised of these 3,600 trajectories, as shown below.

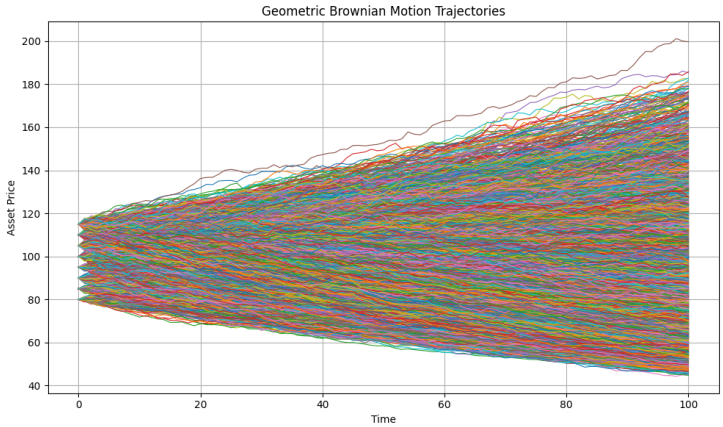

Figure 1: GBM Trajectories Used

We focused on the optimal stopping problem for Bermudan (discretely exercisable) single-asset American call options with strike price $K = 100$. At each time step, the holder of the option decides whether to exercise the option—receiving the payoff $\max(S_t - K, 0)$—or to continue holding the option in anticipation of greater gains later. Our goal was to learn and compare exercise strategies that maximize the expected return over the sampled GBM scenarios.

5.2 Basis Function Selection

The choice of basis functions plays a critical role in the performance of both Least Squares Monte Carlo (LSM) and Randomized Policy Optimization (RPO) methods. Basis functions serve as the features upon which we approximate the continuation value at each time step, and hence must be expressive enough to capture the relevant structure in the value function while maintaining computational tractability.

In this study, we selected a compact yet expressive set of four basis functions designed to reflect the geometry of the American option payoff and the behavior of the underlying asset around the strike price. The basis functions are:

1. **Constant term:** $\phi_1(x) = 1.0$
   This provides a bias term, allowing the model to shift the value function vertically and capture a nonzero intercept. It is crucial for flexibility in regression-based approaches.

2. **Scaled payoff:** $\phi_2(x) = \frac{x-K}{x+\epsilon}$
   This function grows approximately linearly with moneyness but normalizes the input and avoids division by zero (where $\epsilon = 10^{-5}$). It emphasizes how far in- or out-of-the-money the option is, and provides smooth behavior around the strike.

3. **Quadratic term:** $\phi_3(x) = \left(\frac{x-K}{x+\epsilon}\right)^2$
   This captures nonlinear effects and curvature in the value function, especially useful when the value function is convex with respect to price, as is often the case near-the-money.

4. **Exponential decay around strike:** $\phi_4(x) = \exp\left(-\frac{|x-K|}{K}\right)$
   This feature captures rapid changes in the optimal policy around the strike price, introducing local sensitivity and enabling the model to learn sharper decision boundaries in that critical region.

Together, these basis functions offer a blend of linearity, curvature, and localized nonlinearity tailored to the structure of American call options. Importantly, this basis set is compact enough to avoid overfitting or instability in the optimization process, but expressive enough to approximate a wide variety of continuation value shapes across different moneyness regimes.

While richer or more domain-specific basis sets (e.g., involving implied volatility or historical price features) could further improve real-world trading applications, our choices here reflect a balance between interpretability, tractability, and empirical effectiveness in the single-asset setting.

Comparison to Prior Work. In prior work, such as Guan and Mišić (2022), the authors primarily employ simple basis functions—most commonly the payoff function and a constant term. This choice is often sufficient in the multi-asset setting, where the optimization model has access to a high-dimensional state space, including multiple underlying asset prices. These multiple dimensions provide rich informational signals that help guide the estimation of continuation values even with a limited basis set.

In contrast, our study focuses on single-asset American options, where each state is characterized by a single price value. This restriction implies a significantly lower-dimensional feature space, and therefore requires greater care in engineering the basis functions. To compensate for the lack of inherent dimensional richness, we deliberately augment our basis set with nonlinear features that can extract more nuanced information from the price series—especially around the exercise boundary. This design choice ensures that the optimization algorithm has sufficient representational capacity to learn complex policies and provides the granularity needed to effectively visualize differences in the exercise regions between LSM and RPO.

5.3 LSM Implementation

To implement the Longstaff-Schwartz least squares Monte Carlo method (LSM), we constructed a custom stopping policy class (`LSMPolicy`) that performs backward induction using regression to estimate continuation values. The implementation closely follows the original design proposed by Longstaff and Schwartz, but includes several tailored modifications to better suit our experimentation.

Model Structure. We initialize the LSM policy with a set of basis functions, a user-defined payoff function $g(t, x_t)$, and a horizon T. The core of the implementation is a list of regression models—one for each time step—which are learned sequentially in backward order. These models are stored in the `self.models` array and trained only on paths where the option is in-the-money (i.e., when $g(t, x_t) > 0$).

Regression on In-the-Money Paths. As is standard in the LSM literature, we restrict the regression fitting to only those trajectories where the option has intrinsic value at time t. This decision avoids introducing noise from out-of-the-money paths, where the continuation value is less informative. For each in-the-money state, we evaluate the basis functions to produce a feature vector, and regress the discounted future cashflows (continuation values) against these features using Ridge regression.

Ridge Regression and Non-Negative Continuation Values. Instead of ordinary least squares, we use Ridge regression with $\alpha = 1.0$. This choice introduces L_2 regularization, improving stability and robustness in the presence of collinearity or overfitting, especially when many basis functions are used. Additionally, we enforce a non-negativity constraint on the predicted continuation values to reflect the non-negative nature of option payoffs.

Exercise Strategy and Cashflow Updates. At each time step t, for each path, we compare the immediate payoff to the estimated continuation value. If the immediate payoff is higher, we update the cashflow at that path to reflect early exercise. Otherwise, we preserve the previously stored cashflow, which corresponds to deferring the stopping decision to a later time. This backward update continues until $t = 0$.

Stopping Policy and Evaluation. Once training is complete, the policy determines whether to exercise or continue at any state x_t based on whether the immediate payoff exceeds the estimated continuation value from the learned regression model. We also provide utility methods for evaluating the policy value along a trajectory and for extracting the learned regression weights, which facilitate boundary visualization and comparison to RPO-based methods.

Design Rationale. We opted for a transparent, easily debuggable implementation using standard regression techniques (e.g., scikit-learn's `Ridge`) rather than neural networks or differentiable approximations. This approach allows precise control over model behavior and reproducibility of results. Moreover, our code is structured to support flexible substitution of basis functions and clear separation between training, prediction, and stopping logic—key to comparing policy behavior across different basis sets and trajectory distributions.

5.4 RPO Implementation

To implement the Randomized Policy Optimization (RPO) method, we developed a custom class `RPOPolicy`, which trains a sequence of randomized stopping rules using backward optimization. Unlike LSM, which uses regression to estimate continuation values and derives a deterministic exercise rule, RPO directly optimizes the expected reward by learning probabilistic stopping rules at each time step. Each rule is parameterized by a logistic function applied to a linear combination of basis functions.

Parameterization and Initialization. The stopping policy at each time step t is defined by a weight vector $\mathbf{w}_t$, which maps the basis function outputs $\phi(x_t)$ to a probability via the sigmoid function $\sigma(\mathbf{w}_t^\top \phi(x_t))$. These weights are initialized to zero by default but may optionally be warm-started using weights derived from an LSM policy. The number of weights matches the number of basis functions used.

Training via Backward Optimization. The RPO policy is trained using a nested optimization loop. In the outer loop, we iterate backward in time from $T-1$ to 1. For each time step t, we perform several inner epochs of stochastic gradient ascent (using the Adam optimizer) to maximize the expected reward. The continuation value for each trajectory is initialized to the terminal payoff and recursively updated as training proceeds backward in time.

Algorithm 1 *

Backwards optimization algorithm for approximately solving the randomized policy SAA problem

1: Initialize $\mathbf{b}_t \leftarrow \mathbf{0}$ for all $t \in [T]$
2: Initialize $c_T(\omega) \leftarrow 0$ for all $\omega \in [\Omega]$
3: **for** $t = T, \ldots, 1$ **do**
4: Compute $p_t(\omega)$ as:

$$p_t(\omega) = \prod_{t'=1}^{t-1} \left(1 - \sigma\left(\mathbf{b}_{t'} \cdot \Phi(\mathbf{x}(\omega, t'))\right)\right)$$

5: Solve:

$$\max_{\mathbf{b}_t \in \mathcal{B}_t} \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} p_t(\omega) \cdot [g(t, \mathbf{x}(\omega, t)) \cdot \sigma(\mathbf{b}_t \cdot \Phi(\mathbf{x}(\omega, t))) + c_t(\omega) \cdot (1 - \sigma(\mathbf{b}_t \cdot \Phi(\mathbf{x}(\omega, t))))]$$

6: Update continuation values:

$$c_{t-1}(\omega) = g(t, \mathbf{x}(\omega, t)) \cdot \sigma(\mathbf{b}_t^* \cdot \Phi(\mathbf{x}(\omega, t))) + c_t(\omega) \cdot (1 - \sigma(\mathbf{b}_t^* \cdot \Phi(\mathbf{x}(\omega, t))))$$

7: **end for**

Iterative Refinement. While the RPO algorithm is designed to run a single backward pass from T to 1, we found it beneficial to repeat this process for multiple global iterations. This is because the weights $\mathbf{b}_t$ at later time steps depend on weights at earlier time steps via $p_t(\omega)$, which represents the probability of not having exercised before time t. Conversely, the optimal weights at earlier time steps depend on accurate estimates of future continuation values $c_t(\omega)$, which are only updated in later steps. Therefore, iterating over the backward pass allows both the stopping probabilities and continuation values to progressively improve across time steps and global iterations.

Trajectory Filtering and Objective Masking. To make the training more focused and numerically stable, we restrict optimization at each time step t to in-the-money trajectories only—those for which the immediate payoff $g(t, x_t) > 0$. We also compute the probability that each trajectory has not yet been exercised up to time t by accumulating past exercise probabilities. This masking ensures that RPO only learns from paths where the stopping decision is relevant, in line with techniques used in LSM.

Loss Function. At each time step, the expected reward is computed as:

$$\mathbb{E}[\text{reward}] = p_t \cdot \left(\text{payoff}_t \cdot \sigma + \text{cont}_t \cdot (1 - \sigma)\right),$$

where p_t is the probability that a trajectory has not been exercised before time t, σ is the stopping probability at t, and cont_t is the continuation value. We negate this expected reward to define a loss that can be minimized via standard PyTorch autograd and gradient descent tools.

Training Features and Safeguards. To promote numerical stability, we apply gradient clipping with a maximum norm of 1.0. We also save the learned weights after training for reuse or analysis. Optionally, continuation values can be reset at each outer loop iteration to reduce contamination across time steps—although in some setups, initializing them from the previous iteration can help accelerate convergence.

Stochastic Decision Boundaries. Once trained, the RPO policy provides a soft, probabilistic exercise strategy. We offer utilities for computing the stopping probability at a given state, evaluating the total payoff from a trajectory, and visualizing approximate exercise boundaries by thresholding the stopping probabilities. This visualization allows direct comparison with LSM and helps reveal the smoothness and flexibility of RPO-based policies.

Design Rationale. The RPO implementation is designed to align closely with the theoretical formulation proposed by Guan and Mišić (2023), while also being modular and extensible. We intentionally omit minibatching and train on the full trajectory set to maintain theoretical fidelity and better control optimization noise. Our architecture supports both linear and nonlinear basis function combinations, making it adaptable to richer state representations in multi-asset or more realistic market environments.

5.5 Challenges

Throughout our experimentation, we encountered several technical and methodological challenges that shaped our design decisions. One of the primary difficulties was reconciling the structural differences between the RPO and LSM frameworks. RPO optimizes over randomized policies using continuation values and full trajectory expectations, whereas LSM relies on regression against cashflows only at in-the-money (ITM) points. This difference initially led to unstable RPO training and unintuitive decision boundaries. Additionally, the recursive dependency in RPO—where weights at each time step affect future continuation values and are themselves affected by previous exercise probabilities—necessitated multiple passes over the entire time horizon to stabilize learning. We also observed that initialization mattered significantly; starting RPO from LSM-derived weights helped, but the divergence in objective structure between the two approaches made warm-starting non-trivial. Other practical issues included low objective values during early iterations, challenges around training

only on ITM trajectories without distorting gradient signals, and ensuring numerical stability in basis function design. These obstacles required us to iteratively refine the training loop, adjust optimization targets, and carefully select basis functions to enable meaningful learning dynamics in the single-asset setting.

5.6 Results

After training both the LSM and RPO policies on the generated GBM trajectories and evaluating their exercise strategies, we visualized the exercise boundaries to better understand their decision-making processes. Figures 2 and 3 present the exercise boundaries for LSM and RPO, respectively.

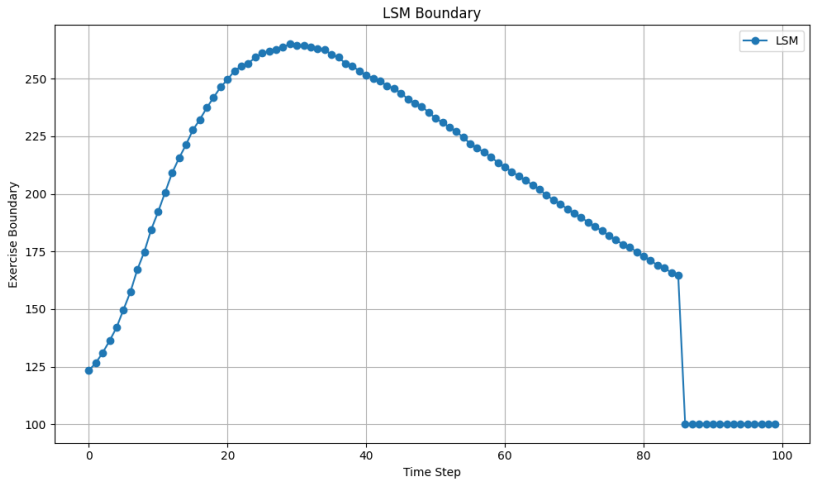

Figure 2: Exercise boundaries for the LSM-trained policy.

Comparison and Analysis. The LSM boundary shows a clear exercise strategy that rises and then sharply falls near maturity. This behavior is expected in optimal stopping for American-style options, as the opportunity cost of delaying exercise grows near maturity, especially when the asset is deeply in-the-money.

In contrast, the RPO boundaries rise more steadily and exhibit a smoother curvature, but lack the same sharp decline near maturity seen in the LSM results. This may be attributed to the nature of randomized stopping, where the decision to stop is governed by probabilities instead of deterministic comparisons of immediate payoff vs. continuation value. The probabilistic exercise structure means RPO can naturally smooth out boundary transitions.

One likely contributor to these differences is how the two methods are trained. In RPO, weights are trained via backward optimization with a stochastic policy model, where the stopping probability at each time step depends on previous weights (through p_t) and continuation values (which depend on future

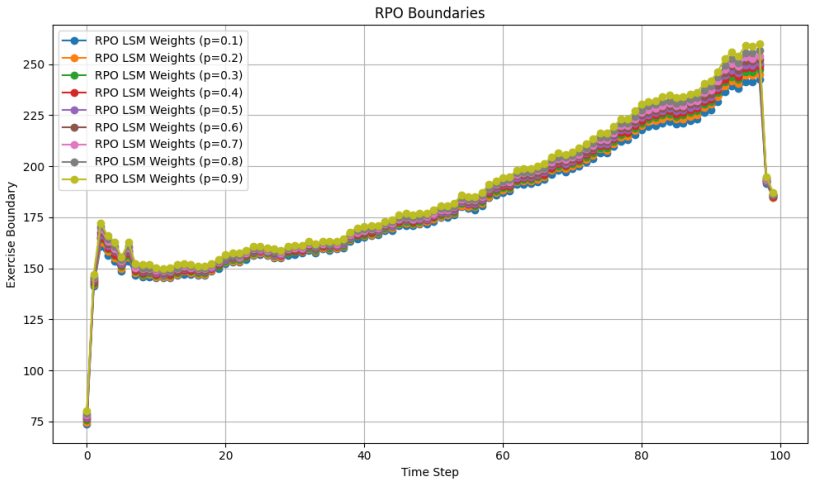

Figure 3: Exercise boundaries for the RPO-trained policy, shown for various threshold probabilities $p \in \{0.1, 0.2, ..., 0.9\}$.

weights). This recursive interdependence can slow convergence, especially in early time steps. LSM, on the other hand, only requires learning conditional expected continuation values, and thus converges faster.

It's worth noting that we warm-started the RPO policy from the LSM weights to mitigate this slow convergence, in line with the methodology outlined in Guan and Mišić (2022). Even so, LSM generally achieves sharper boundaries faster, while RPO is better suited for modeling environments where uncertainty and smooth transitions in stopping behavior are preferable, such as trading strategies incorporating risk aversion or transaction costs.

Contextualization with Guan and Mišić. Our findings mirror some of the behavior described in Guan and Mišić, particularly the ability of RPO to learn smooth stopping boundaries. However, unlike their multi-asset setting, our single-asset model provides fewer high-dimensional features, making it more difficult for the RPO optimizer to extract structure. This required us to augment our basis function set and iterate the backward pass multiple times to reach satisfactory convergence. In future work, extending this comparison to multi-asset derivatives may provide a closer parallel to their setup and yield more decisive advantages for RPO.

6 Conclusion

In this work, we implemented and compared two powerful approaches for solving optimal stopping problems in the context of single-asset American options: the Least Squares Monte Carlo (LSM) method and the Randomized Policy

Optimization (RPO) framework introduced by Guan and Mišić. Our experimentation focused on visualizing exercise boundaries learned by each method, leveraging synthetic data generated through a wide range of geometric Brownian motion (GBM) trajectories.

The results reveal that while LSM yields sharp and interpretable boundaries with relatively fast convergence, RPO offers a more probabilistic and flexible modeling approach that could better accommodate real-world complexities such as transaction costs, risk preferences, or noisy environments. Nonetheless, the single-asset setting presented challenges for RPO due to limited input dimensionality, emphasizing the importance of thoughtful basis function design and careful tuning of the training loop.

Future work may involve expanding this framework to multi-asset derivatives, where RPO's advantage in modeling high-dimensional uncertainty and flexible exercise behavior is more apparent. Additionally, incorporating transaction costs, regime-switching dynamics, or reinforcement learning-inspired architectures could further bridge the gap between theoretical optimal stopping and practical financial decision-making. Exploring empirical backtests or integrating real-world features from historical price data are also natural next steps in making these models viable tools for algorithmic trading and financial engineering applications.

References

Guan, Xinyi and Velibor Misic (Jan. 2022). "Randomized Policy Optimization for Optimal Stopping". In: *SSRN Electronic Journal*. DOI: 10.2139/ssrn.4066178.

Longstaff, Francis and Eduardo Schwartz (Feb. 2001). "Valuing American Options by Simulation: A Simple Least-Squares Approach". In: *Review of Financial Studies* 14, pp. 113–47. DOI: 10.1093/rfs/14.1.113.