Counterfactual Regret Evolutionary Agents (CREA)

Aman Kumar Anand

Abstract

Financial decision-making is increasingly challenged by non-stationarity, tail risk, ambiguous information, and causal complexity. Traditional quantitative models often optimize policies against historical returns or point forecasts, but such optimization can become fragile when market regimes shift or when observed data conceal unobserved causal drivers. This project develops Counterfactual Regret Evolutionary Agents (CREA), a mathematical framework for adaptive decision-making in financial systems under deep uncertainty.

The proposed framework models the market on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, represents returns as stochastic processes, and uses a structural causal view of latent market worlds. Its central contribution is the Counterfactual Regret Landscape (CRL), an objective that evaluates each trading policy against alternative actions across possible causal market realities. Instead of asking only how a strategy performed historically, CREA asks how much causal opportunity was lost under plausible counterfactual states.

CREA integrates probability theory, stochastic processes, statistical inference, fuzzy logic, causal modeling, and evolutionary optimization into a unified agent system. A Market World Generator constructs counterfactual trajectories from latent causal variables. In the advanced extension, this generator is no longer only assumption-driven: structural causal discovery methods such as NOTEARS, DAG-GNN, PCMCI, and temporal causal discovery are used to learn directed market mechanisms from data. A probabilistic modeling layer estimates conditional return distributions and uncertainty. Statistical risk measures, including Value at Risk and Conditional Value at Risk, regulate exposure to adverse tail events. Fuzzy membership functions model ambiguous market signals such as uncertain volatility regimes, weak trends, and imprecise investor behavior. Evolutionary agents then search the policy space using mutation, crossover, selection, and meta-evolutionary adaptation, with fitness defined by counterfactual regret rather than by historical profit alone.

The result is a causally-aware and risk-sensitive framework for discovering strategies that remain stable across multiple possible futures. The report further upgrades CREA with Wasserstein distributionally robust optimization, transformer memory architectures, multi-agent market ecology, and information-geometric regret analysis. These additions connect CREA to modern robust optimization, adversarial finance, temporal deep learning, reflexive market modeling, and differential-geometric views of policy search.

1 Introduction

Financial markets are stochastic, adaptive, and causally entangled systems. Asset prices respond not only to historical prices and returns, but also to latent macroeconomic variables, liquidity constraints, institutional behavior, investor sentiment, policy interventions, and reflexive feedback between agents. A trading or portfolio policy that appears optimal on past data may fail when the hidden data-generating process changes. This creates a central mathematical problem: how can a decision-maker learn robust strategies when the future is non-stationary, uncertainty is deep, and the relevant causal variables are only partially observed?

Classical quantitative finance has developed powerful tools for modeling risk and return. Markowitz portfolio theory formalizes mean-variance allocation, Black-Scholes-Merton style models describe idealized asset price dynamics, and statistical risk measures quantify downside exposure. However, these approaches often work with observed distributions or assumed stochastic dynamics. Modern machine learning extends this by fitting high-dimensional predictive models, but prediction alone does not necessarily answer the causal decision question: what would have happened if a different action had been taken under the same latent market conditions?

The framework therefore builds on portfolio selection [12], option-pricing and stochastic price modeling [5, 3], risk-adjusted performance measurement [21], coherent and tail-risk measures [2, 17], fuzzy set theory [25], structural causal reasoning [15, 19], continuous causal discovery [26], temporal causal discovery [18], distributionally robust optimization [4, 13], and transformer sequence modeling [22, 11, 10].

This report proposes Counterfactual Regret Evolutionary Agents (CREA), a hybrid mathematical and causal framework for adaptive financial decision-making. CREA combines counterfactual reasoning, probabilistic modeling, fuzzy logic, tail-risk control, and evolutionary search. Its objective is not merely to maximize observed returns, but to minimize regret across counterfactual market worlds generated by latent causal variables.

1.1 Motivation

The motivation for CREA arises from five limitations of purely historical optimization:

1. Historical returns represent only one realized market path, while financial decisions

are exposed to many possible unrealized paths.

2. Regime shifts can invalidate policies fitted to stationary assumptions.

3. Market signals are often ambiguous rather than crisp, especially during crises or transitions.

4. Strategies can be locally profitable but globally fragile when evaluated under causal interventions and tail events.

5. Single-agent models ignore reflexivity: large strategies can alter liquidity, volatility, and other agents' behavior.

CREA addresses these limitations by shifting the learning target from observed performance to counterfactual robustness. A policy is preferred when it performs well across multiple plausible market realities and loses little opportunity compared with the best action that could have been taken in each latent state.

1.2 Problem Statement

Let $\mathcal{S}$ be a financial state space, $\mathcal{A}$ an action space, and Π a class of admissible policies $\pi : \mathcal{S} \to \mathcal{A}$. The goal is to find a policy that is robust to uncertainty, ambiguity, and causal regime changes. In conventional empirical optimization, one may solve

$$\max_{\pi \in \Pi} \mathbb{E}[R(\pi(s))],$$

where R denotes realized return. In non-stationary financial systems this objective can overfit the observed path. CREA instead evaluates policy quality through counterfactual regret:

$$\min_{\pi \in \Pi} \mathbb{E}\big[\mathcal{R}_{CF}(\pi)\big],$$

where $\mathcal{R}_{CF}(\pi)$ measures the loss of choosing $\pi(s)$ relative to the best feasible counterfactual action under latent market conditions.

1.3 Objectives

The objectives of this project are:

1. To formulate a Counterfactual Regret Landscape for financial decision-making.

2. To integrate probability, stochastic processes, statistical risk measures, fuzzy logic, and causal modeling into one mathematical framework.
3. To design an evolutionary agent system that searches for robust trading policies.
4. To define an evaluation protocol for testing performance under regime shifts, volatility spikes, noise, and out-of-distribution scenarios.

1.4 Key Contributions

The major contributions of this report are:

1. A counterfactual regret objective for evaluating financial policies across possible market realities.
2. A hybrid objective that combines causal regret, CVaR-based tail-risk control, fuzzy ambiguity penalties, and expected utility.
3. A Market World Generator for producing counterfactual trajectories from latent causal variables.
4. A meta-evolutionary adaptation mechanism for controlling exploration and exploitation across changing market regimes.
5. An advanced extension that learns causal structure from data, adds Wasserstein distributional robustness, uses transformer memory for long-range temporal context, and models multi-agent market ecology.

2 Preliminaries

This section presents the mathematical concepts needed for the CREA framework: probability spaces, stochastic processes, asset price models, conditional expectations, risk measures, fuzzy sets, causal modeling, and evolutionary optimization.

2.1 Probability Space and Financial Returns

A financial system is modeled as a probability space

$$(\Omega, \mathcal{F}, \mathbb{P}),$$

where Ω is the sample space of possible market outcomes, $\mathcal{F}$ is a sigma-algebra of events, and $\mathbb{P}$ is a probability measure. A market return at time t is represented by a random variable

$$R_t : \Omega \to \mathbb{R}.$$

The filtration $\{\mathcal{F}_t\}_{t\geq 0}$ represents the information available up to time t. Conditional expectations such as

$$\mathbb{E}[R_{t+1} \mid \mathcal{F}_t]$$

represent the expected future return given current market information.

2.2 Stochastic Price Dynamics

Under a simplified geometric Brownian motion model, the asset price process satisfies

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

where S_0 is the initial price, μ is the drift parameter, σ is volatility, and $\{W_t\}_{t\geq 0}$ is a Wiener process. The stochastic differential form is

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

Although this model is mathematically elegant, financial markets often exhibit volatility clustering, jumps, regime shifts, and heavy tails. CREA therefore treats such models as a baseline rather than as a complete description of market reality. The stochastic modeling perspective is rooted in early mathematical finance and option-pricing theory [3, 5], while volatility clustering motivates extensions such as GARCH-type processes [6].

2.3 Statistical Inference

Let $r_1, r_2, \ldots, r_n$ denote observed returns. The empirical mean and variance are

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} r_i, \qquad \hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(r_i - \hat{\mu})^2.$$

These estimates are useful but unstable when the return distribution changes over time. CREA therefore combines empirical inference with counterfactual generation and robustness testing.

2.4 Risk Measures

The Sharpe ratio is defined as

$$SR = \frac{\mathbb{E}[R_p] - R_f}{\sigma_p},$$

where R_p is portfolio return, R_f is the risk-free rate, and σ_p is portfolio volatility. Value at Risk at confidence level α is

$$VaR_\alpha(L) = \inf\{\ell : \mathbb{P}(L \leq \ell) \geq \alpha\},$$

where L denotes loss. Conditional Value at Risk is

$$CVaR_\alpha(L) = \mathbb{E}[L \mid L \geq VaR_\alpha(L)].$$

CVaR is especially important for CREA because it penalizes severe tail losses that may occur in counterfactual stress worlds. The Sharpe ratio is used for risk-adjusted performance evaluation [21], while VaR and CVaR connect the framework to coherent and convex risk-control methods [2, 17].

2.5 Fuzzy Sets and Ambiguity

Classical logic assigns membership as either true or false. In financial markets, signals are frequently ambiguous. A market may be partly volatile, partly trending, or partly illiquid. A fuzzy set $\tilde{A}$ on universe X is

$$\tilde{A} = \{(x, \mu_A(x)) \mid x \in X\},$$

where $\mu_A(x) \in [0, 1]$ is the membership degree of x in $\tilde{A}$.

For example, a volatility signal v_t may be mapped to a high-volatility membership score

$$\mu_{\text{high}}(v_t) = \frac{1}{1 + \exp[-k(v_t - c)]},$$

where c is a threshold and k controls the steepness of the transition. This lets CREA handle gradual regime boundaries rather than imposing hard classifications. This fuzzy representation follows the foundational view that uncertain membership can be modeled continuously rather than through binary classification [25].

2.6 Structural Causal Models

A structural causal model represents variables through functional relations:

$$X_i = f_i(\mathrm{Pa}(X_i), U_i),$$

where $\mathrm{Pa}(X_i)$ are causal parents and U_i are exogenous noise variables. In a financial setting, latent causal variables may include monetary policy shocks, liquidity states, institutional demand, macroeconomic surprises, and investor sentiment.

Counterfactual reasoning asks what would have happened under a different action while holding relevant latent conditions fixed. This can be written using the intervention operator $do(a)$. A counterfactual return may be represented as

$$R_{t+1}(a, U_t) = R_{t+1} \mid do(A_t = a), U_t,$$

where U_t denotes the latent market world at time t. The causal interpretation follows structural causal models and potential-outcome reasoning [15, 19].

2.7 Structural Causal Discovery

The basic CREA framework can specify latent causal variables from economic theory. The advanced framework strengthens this assumption by learning a causal graph from data. Let

$$X_t = (R_t, \sigma_t, \ell_t, m_t, q_t, z_t)$$

collect returns, volatility, liquidity, macroeconomic indicators, order-flow proxies, and sentiment variables. A directed causal graph can be represented by a weighted adjacency matrix W, where $W_{ij} \neq 0$ suggests that variable X_i directly influences X_j.

For contemporaneous causal discovery, NOTEARS formulates graph learning as a continuous constrained optimization problem:

$$\min_W \ \mathcal{L}(W; X) + \rho \|W\|_1 \quad \text{subject to} \quad h(W) = \mathrm{tr}\left(e^{W \circ W}\right) - d = 0,$$

where $h(W) = 0$ enforces acyclicity for a d-variable graph [26]. Neural extensions such as DAG-GNN replace linear structural equations with graph neural encoders and decoders, allowing nonlinear causal mechanisms [24].

Financial systems are also temporal. For lagged causal effects, CREA may use PCMCI or related time-series discovery methods to estimate whether

$$X_{t-\ell}^{(i)} \to X_t^{(j)}$$

after conditioning on other relevant lagged variables [18]. The learned graph becomes the causal skeleton of the Market World Generator, so counterfactual worlds are generated from empirically discovered dependencies rather than only from manually assumed latent factors.

2.8 Distributionally Robust Optimization

Historical data provide an empirical distribution $\widehat{P}$, but future markets may be generated by a nearby distribution $Q \neq \widehat{P}$. Distributionally robust optimization protects the policy against such misspecification. Let $\mathcal{U}_\varepsilon(\widehat{P})$ denote a Wasserstein ambiguity set:

$$\mathcal{U}_\varepsilon(\widehat{P}) = \left\{ Q : W_c(Q, \widehat{P}) \leq \varepsilon \right\},$$

where W_c is a Wasserstein distance induced by transportation cost c. The robust CREA objective is

$$\min_{\pi \in \Pi} \sup_{Q \in \mathcal{U}_\varepsilon(\widehat{P})} \mathbb{E}_Q \left[\mathcal{R}_{CF}(\pi) \right].$$

This converts counterfactual regret minimization into an adversarial uncertainty problem: the policy must remain effective even when the market distribution moves within a statistically plausible neighborhood of the observed sample. This connects CREA with modern robust optimization and data-driven ambiguity sets [4, 13].

2.9 Transformer Memory Models

CREA can represent a policy as a transformer memory architecture instead of a simple feature rule. Let

$$H_t = (s_{t-L}, a_{t-L}, r_{t-L}, \ldots, s_t)$$

be a rolling history of states, actions, rewards, regimes, and causal graph signals. A transformer policy uses self-attention to compute

$$\pi_\theta(a_t \mid H_t) = \text{softmax}\left(f_\theta^T(H_t) \right),$$

where f_θ^T is a temporal transformer. Temporal Fusion Transformers are useful when static covariates, known future inputs, and interpretable attention weights are required [11]. Decision Transformers can instead condition actions on desired return-to-go, making portfolio control resemble sequence modeling over trajectories [10]. Under this extension, CREA becomes a Counterfactual Causal Evolutionary Transformer Agent framework.

2.10 Multi-Agent Market Ecology

A single optimized agent assumes the market is an external environment. In real financial systems, agents interact and can change the environment itself. Let M agents use policies

$$\Pi^M = \{\pi_{\theta_1}^1, \ldots, \pi_{\theta_M}^M\}.$$

The state transition becomes

$$s_{t+1} \sim P\left(s_{t+1} \mid s_t, a_t^1, \ldots, a_t^M\right),$$

and agent i receives reward $G_i(a_t^i, a_t^{-i}, U_t)$, where a_t^{-i} denotes the actions of other agents. This creates a market ecology with competing liquidity takers, liquidity providers, trend followers, arbitrageurs, and adversarial agents. Evolution then operates at the population level:

$$\mathcal{P}_{g+1}^{1:M} = \mathcal{E}\left(\mathcal{P}_g^{1:M}, \mathcal{M}_g\right),$$

where $\mathcal{M}_g$ is the endogenous market environment produced by the current population. The purpose is to model reflexivity, adaptive adversaries, crowding, and emergent regimes.

2.11 Regret Geometry and Information Geometry

The policy class $\Pi = \{\pi_\theta : \theta \in \Theta\}$ can be treated as a statistical manifold. The local geometry is described by the Fisher information matrix

$$I(\theta) = \mathbb{E}_{s,a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s)^\top\right].$$

Instead of updating parameters by Euclidean descent, a natural-gradient or Riemannian update follows

$$\theta_{k+1} = \theta_k - \alpha I(\theta_k)^{-1} \nabla_\theta J(\theta_k).$$

In this view, high regret can be interpreted as curvature on the policy manifold: some regions are sensitive to small distributional or causal perturbations, while flatter regions correspond to policies whose counterfactual performance is stable. This gives CREA a mathematical-finance interpretation beyond heuristic optimization and links it to information geometry and natural-gradient learning [1].

2.12 Evolutionary Optimization

Evolutionary algorithms maintain a population of candidate solutions and improve them through selection, mutation, and crossover. In CREA, each individual is an agent policy π_θ parameterized by genome θ. The important distinction is that fitness is not defined only by historical return; it is defined by counterfactual regret, tail risk, and ambiguity handling. The population-based search mechanism follows classical genetic adaptation and multiobjective evolutionary optimization principles [8, 7].

3 Materials and Methods

This section develops the CREA framework. The methodology is divided into the Market World Generator, structural causal discovery layer, probabilistic modeling layer, risk layer, fuzzy logic layer, Wasserstein robust objective, transformer memory policy, evolutionary agent system, multi-agent market ecology, regret-geometry layer, and meta-evolutionary adaptation mechanism.

3.1 System Overview

At each decision time t, the observed market state is denoted by $s_t \in \mathcal{S}$. A policy π_θ maps the state into an action $a_t \in \mathcal{A}$, such as buy, sell, hold, rebalance, or allocate portfolio weights. CREA evaluates this action not only in the observed trajectory but also across counterfactual worlds generated from latent causal variables.

The architecture is summarized as follows:

1. The structural causal discovery layer learns directed market mechanisms from observed and lagged variables.

2. The Market World Generator uses the learned or specified causal structure to produce counterfactual trajectories.

3. The probabilistic layer estimates conditional distributions of returns and losses.

4. The risk layer computes Sharpe ratio, VaR, CVaR, maximum drawdown, and stability metrics.

5. The fuzzy layer assigns degrees of ambiguity to market signals.

6. The Wasserstein robust layer evaluates the policy under adversarially shifted distributions.

7. The transformer memory layer encodes long-range temporal dependence and regime history.

8. The evolutionary layer searches for policies with low counterfactual regret and controlled downside risk.

9. The multi-agent ecology layer allows policies to compete and alter market dynamics.

10. The regret-geometry layer studies sensitivity on the policy manifold.

11. The meta-evolutionary layer adapts mutation, crossover, and exploration rates in response to market regimes.

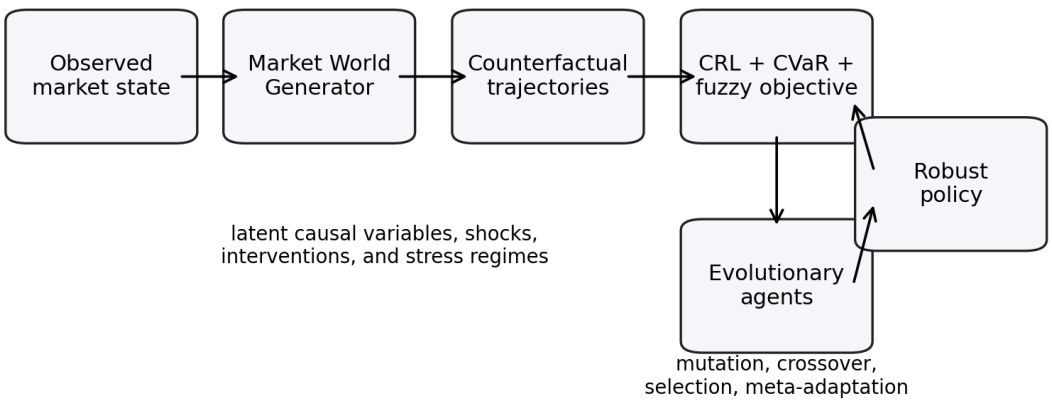

Figure 1: CREA system architecture linking market observation, counterfactual world generation, hybrid regret-risk-fuzzy evaluation, and evolutionary policy search.

3.2 Market World Generator

The Market World Generator (MWG) represents the causal mechanism through which latent variables generate observed market states. Let

$$U_t = (U_t^{macro}, U_t^{liq}, U_t^{sent}, U_t^{vol}, U_t^{policy})$$

denote a vector of latent market drivers. The observed state is modeled as

$$s_t = g(U_t, \epsilon_t),$$

where ϵ_t is observation noise. The future return under action a is modeled as

$$G_{t+1}(a, U_t) = h(s_t, a, U_t, \xi_{t+1}),$$

where ξ_{t+1} is future shock noise and G_{t+1} denotes policy reward or portfolio return.

In an implementation, the MWG may be specified through probabilistic graphical models, variational autoencoders, diffusion models, bootstrapped stochastic simulators, or regime-switching models. Variational autoencoders provide one possible generative modeling route for latent-world simulation [9]. The essential requirement is that the generator can produce alternative trajectories under interventions:

$$\tau_a^{(k)} = \{s_t, a, G_{t+1}^{(k)}, s_{t+1}^{(k)}, \ldots\}, \qquad k = 1, \ldots, K.$$

3.3 Structural Causal Discovery Layer

The advanced CREA pipeline learns the causal structure used by the MWG. Let $X \in \mathbb{R}^{n \times d}$ be the data matrix of observed market variables and let W be a weighted adjacency matrix. A possible discovery procedure is:

1. Estimate a contemporaneous graph W_0 using NOTEARS or DAG-GNN.
2. Estimate lagged links W_ℓ for $\ell = 1, \ldots, L$ using PCMCI or another temporal causal discovery method.
3. Combine economic priors with learned edges by constraining impossible directions, such as future variables causing past variables.

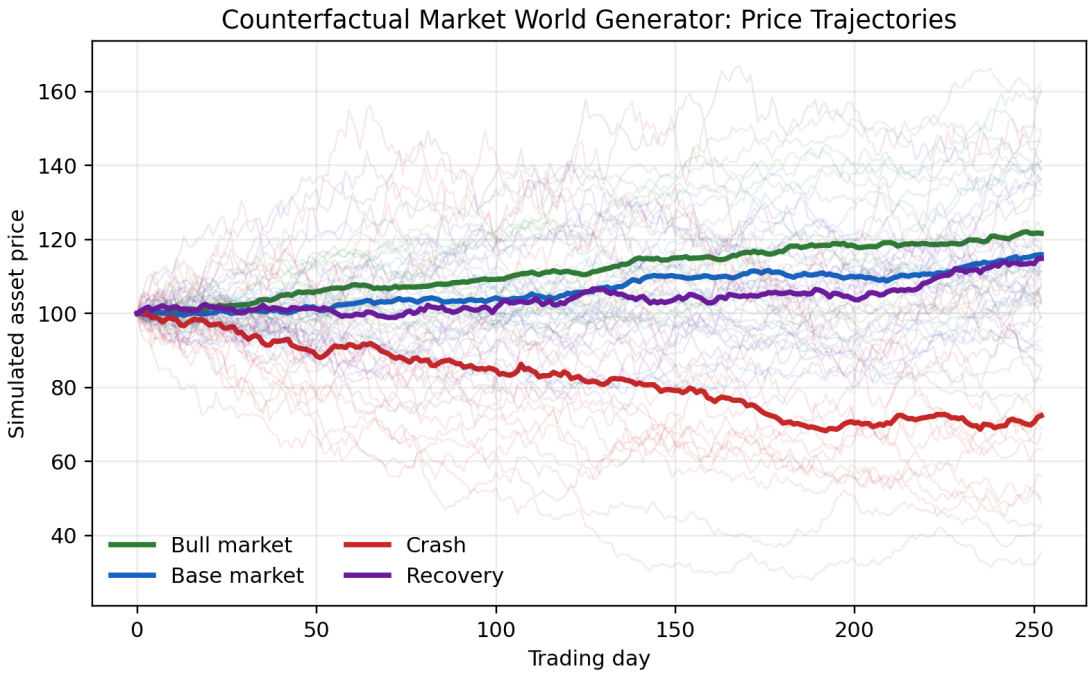

Figure 2: Illustrative counterfactual price trajectories generated under bull, base, crash, and recovery regimes using stochastic price dynamics. Thin lines represent individual simulated worlds and thick lines represent regime averages.

4. Use bootstrap resampling to retain only stable edges with high selection frequency.

5. Pass the learned graph $\mathcal{G}_{CF} = (V, E)$ to the MWG for intervention-based trajectory simulation.

The learned structural equations may be written as

$$X_t^{(j)} = f_j\left(\{X_t^{(i)} : i \in \mathrm{Pa}_0(j)\}, \{X_{t-\ell}^{(i)} : i \in \mathrm{Pa}_\ell(j), 1 \le \ell \le L\}, \epsilon_t^{(j)}\right).$$

Under an intervention $do(A_t = a)$, CREA modifies the action node while preserving the learned causal mechanisms for macro, liquidity, volatility, and sentiment channels. Thus the counterfactual worlds become partly data-discovered rather than purely assumed.

3.4 Counterfactual Regret Landscape

The central mathematical object in CREA is the Counterfactual Regret Landscape (CRL). For a state s and latent market world U, let $G(a, U)$ be the reward obtained by action a. The counterfactual regret of a policy π is defined by

$$\mathcal{R}_{CF}(\pi; s) = \mathbb{E}_{U|s}\left[\max_{a' \in \mathcal{A}} G(a', U) - G(\pi(s), U)\right].$$

The global regret is

$$\mathcal{R}_{CF}(\pi) = \mathbb{E}_{s\sim d_\pi}\left[\mathcal{R}_{CF}(\pi; s)\right],$$

where d_π is the state distribution induced by policy π.

This objective changes the optimization question. A conventional policy may be rewarded for high return on a realized path. CREA rewards a policy for having low opportunity loss when evaluated across possible causal worlds. This counterfactual framing is consistent with causal decision analysis in which interventions are compared under shared latent conditions [15, 19].

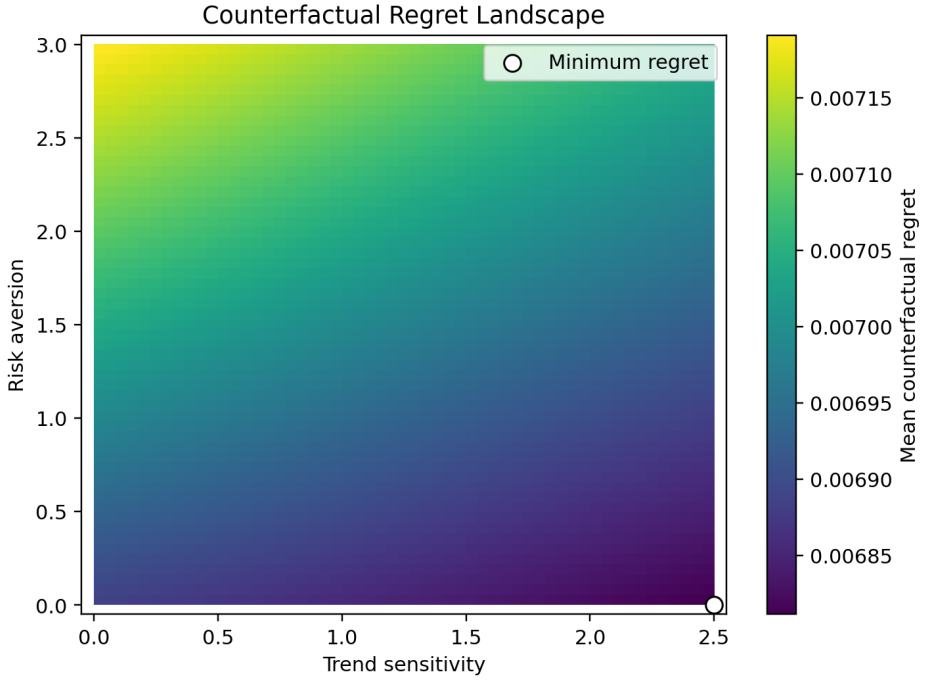

Figure 3: A simulated Counterfactual Regret Landscape over trend sensitivity and risk-aversion parameters. Darker regions indicate lower mean counterfactual regret.

3.5 Probabilistic Modeling Layer

The probabilistic layer estimates expected returns, variances, and conditional distributions:

$$\mathbb{E}[R], \qquad \mathrm{Var}(R), \qquad \mathbb{E}[R_{t+1} \mid \mathcal{F}_t].$$

For counterfactual trajectories generated by the MWG, the empirical estimate of expected reward for policy π is

$$\widehat{\mathbb{E}}[G_\pi] = \frac{1}{K}\sum_{k=1}^{K} G(\pi(s), U^{(k)}).$$

The empirical counterfactual regret is

$$\widehat{\mathcal{R}}_{CF}(\pi; s) = \frac{1}{K} \sum_{k=1}^{K} \left[\max_{a' \in \mathcal{A}} G(a', U^{(k)}) - G(\pi(s), U^{(k)}) \right].$$

3.6 Fuzzy Logic Modeling Layer

Let z_t denote a vector of observed market indicators such as volatility, momentum, liquidity, drawdown, volume, and sentiment. CREA defines fuzzy memberships:

$$\mu_{\text{vol}}(z_t), \quad \mu_{\text{trend}}(z_t), \quad \mu_{\text{liq}}(z_t), \quad \mu_{\text{panic}}(z_t).$$

These memberships produce an ambiguity score

$$\mathcal{A}_F(s_t) = \sum_{j=1}^{m} w_j \, \mu_j(s_t)\big(1 - \mu_j(s_t)\big),$$

where $w_j \geq 0$ are importance weights. The term $\mu_j(1 - \mu_j)$ is largest when membership is uncertain and smallest when the signal is clearly absent or clearly present.

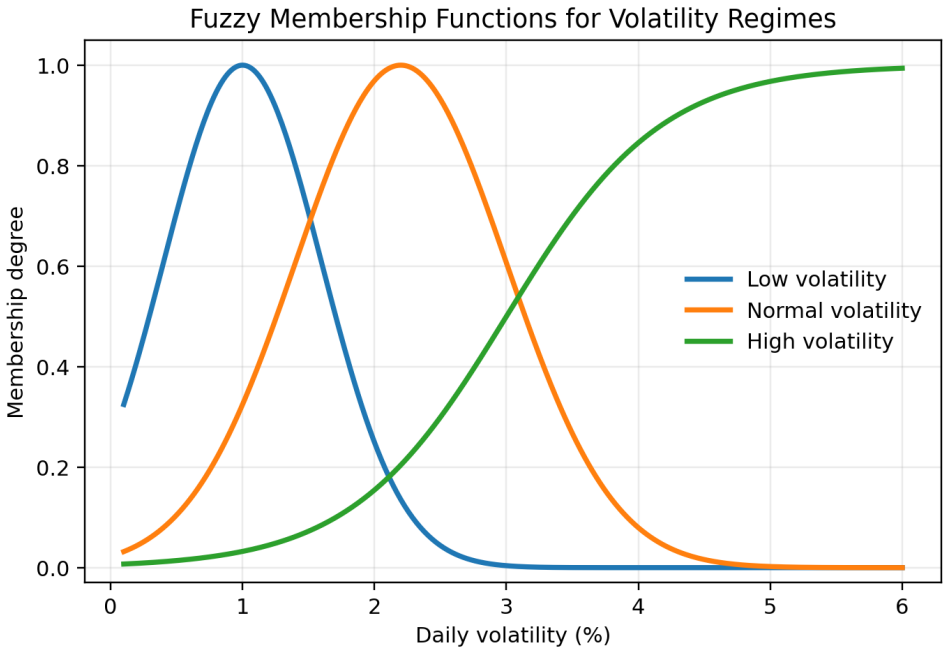

Figure 4: Fuzzy volatility membership functions used to represent low, normal, and high volatility regimes without imposing hard regime boundaries.

3.7 Hybrid Mathematical Objective

The CREA optimization problem combines counterfactual regret, expected performance, tail risk, and fuzzy ambiguity:

$$\min_{\theta} J(\theta) = \mathbb{E}\big[\mathcal{R}_{CF}(\pi_\theta)\big] + \lambda\, CVaR_\alpha(L_{\pi_\theta}) + \gamma\, \mathbb{E}[\mathcal{A}_F(s)] - \eta\, \mathbb{E}[G_{\pi_\theta}],$$

where $\lambda, \gamma, \eta \geq 0$ control the trade-off between regret minimization, risk control, ambiguity handling, and reward maximization. The loss variable may be defined as

$$L_{\pi_\theta} = -G_{\pi_\theta}.$$

This objective formalizes the central design principle of CREA:

Component	Purpose
Counterfactual regret	Reduce causal opportunity loss across alternative worlds
CVaR	Control severe downside risk and crash exposure
Fuzzy ambiguity	Penalize unclear or unstable signal regimes
Expected reward	Preserve economic performance

The objective combines causal comparison [15], fuzzy ambiguity modeling [25], and CVaR-based tail-risk control [17].

3.8 Wasserstein Robust CREA Objective

To protect against distribution shift, the hybrid objective can be lifted from the empirical distribution $\widehat{P}$ to a Wasserstein ambiguity set:

$$J_{\mathrm{DRO}}(\theta) = \sup_{Q \in \mathcal{U}_\varepsilon(\widehat{P})} \mathbb{E}_Q \left[\mathcal{R}_{CF}(\pi_\theta) + \lambda L^{tail}_{\pi_\theta} + \gamma \mathcal{A}_F(s) - \eta G_{\pi_\theta}\right],$$

where $L^{tail}_{\pi_\theta}$ may be a CVaR-compatible loss term. The advanced optimization problem is therefore

$$\min_{\theta} J_{\mathrm{DRO}}(\theta).$$

The radius ε controls conservatism. A small radius behaves like empirical optimization, while a large radius asks the agent to survive stronger adversarial distributional shifts.

In financial terms, this is equivalent to asking whether the policy remains stable when correlations, volatility, liquidity, or tail probabilities move away from their historical estimates.

Layer	Advanced role in CREA
Causal discovery	Learn market graph rather than assume all latent links
Wasserstein DRO	Defend against adversarial distribution shift
Transformer memory	Use long-range temporal context for policy selection
Multi-agent ecology	Model reflexivity, crowding, and adaptive opponents
Information geometry	Search along stable directions of the policy manifold

3.9 Evolutionary Agent System

CREA maintains a population

$$\mathcal{P}_g = \{\pi_{\theta_1}^{(g)}, \pi_{\theta_2}^{(g)}, \ldots, \pi_{\theta_N}^{(g)}\}$$

at generation g. Each policy may be represented by a vector of strategy parameters, a decision tree, a neural network, a portfolio rule, or a hybrid symbolic-numeric genome.

The fitness of agent i is defined as

$$F_i = -J(\theta_i).$$

Higher fitness corresponds to lower counterfactual regret, lower tail risk, lower ambiguity penalty, and higher expected reward.

The evolutionary operations are:

1. **Selection:** Choose high-fitness policies using tournament selection, rank selection, or probabilistic selection.
2. **Crossover:** Combine parameter subsets from two parent policies to produce offspring.

3. **Mutation:** Perturb policy parameters to explore new regions of the strategy space.

4. **Elitism:** Preserve a small number of best agents to prevent loss of robust strategies.

These operations are standard in genetic algorithms and evolutionary multiobjective optimization [8, 7].

3.10 Transformer Evolutionary Policy

In the advanced design, each evolutionary individual may encode a transformer policy rather than a simple parameter vector. The genome becomes

$$\theta_i = (\theta_i^T, \theta_i^{risk}, \theta_i^{fuzzy}, \theta_i^{exec}),$$

where θ_i^T are transformer weights or low-rank adapter parameters, θ_i^{risk} controls tail-risk preferences, θ_i^{fuzzy} controls fuzzy memberships, and θ_i^{exec} controls execution rules.

The transformer receives a history window H_t and produces either portfolio weights or action probabilities:

$$a_t \sim \pi_{\theta_i}(a \mid H_t, \mathcal{G}_{CF}, \widehat{U}_t).$$

The causal graph $\mathcal{G}_{CF}$ and inferred latent state $\widehat{U}_t$ can be supplied as additional tokens or side information. Evolutionary search then acts as an outer optimizer that selects robust transformer policies, while gradient-based fine-tuning may update transformer parameters inside each generation.

3.11 Multi-Agent Market Ecology

CREA can be extended from one agent to a co-evolving population of market participants. Let each agent i choose action a_t^i and receive a counterfactual reward

$$G_i\left(a_t^i, a_t^{-i}, U_t\right).$$

The individual regret is

$$\mathcal{R}_{CF}^i = \mathbb{E}_{U,a^{-i}}\left[\max_{a_i'} G_i(a_i', a^{-i}, U) - G_i(\pi_i(s), a^{-i}, U)\right].$$

The ecology-level objective can combine individual robustness and systemic stability:

$$J_{\text{eco}} = \sum_{i=1}^{M} \omega_i J_i + \chi \, \text{Fragility}(\mathcal{M}),$$

where $\mathcal{M}$ is the simulated market environment and χ penalizes destabilizing collective behavior. This formulation allows the model to study adaptive adversaries, crowded trades, liquidity spirals, and emergent regimes.

3.12 Regret Geometry Layer

The regret landscape can be studied through its local curvature. Around a policy parameter θ, define the second-order approximation

$$J(\theta + \Delta) \approx J(\theta) + \nabla J(\theta)^\top \Delta + \frac{1}{2} \Delta^\top H_J(\theta) \Delta.$$

Large eigenvalues of $H_J(\theta)$ indicate directions where small parameter changes produce large regret changes. Using the Fisher metric $I(\theta)$, CREA may update policies along a natural direction

$$\widetilde{\nabla} J(\theta) = I(\theta)^{-1} \nabla J(\theta),$$

which respects the information geometry of the policy distribution. Policies that lie in flatter, lower-curvature basins are preferred because they are less sensitive to noise, graph uncertainty, and adversarial distribution shifts.

3.13 Meta-Evolutionary Adaptation

In a non-stationary market, fixed evolutionary parameters may be inefficient. CREA therefore introduces meta-parameters

$$\phi_g = (m_g, c_g, e_g),$$

where m_g is mutation rate, c_g is crossover rate, and e_g is exploration intensity. These parameters evolve based on regime uncertainty and population diversity.

One possible adaptation rule is

$$m_{g+1} = \text{clip}\left[m_g + \kappa_1 \Delta_{regime} - \kappa_2 D_g, m_{\min}, m_{\max}\right],$$

where Δ_{regime} measures regime instability and D_g measures population diversity. Mutation increases when the market appears unstable and decreases when the population already explores sufficiently.

3.14 CREA Algorithm

The algorithmic procedure can be written as:

1. Initialize a population of policies $\mathcal{P}_0$.
2. For each generation $g = 0, 1, \ldots, G$:
   (a) Learn or update the causal graph using structural and temporal causal discovery.
   (b) Estimate latent market variables using the graph-conditioned MWG.
   (c) Generate K counterfactual trajectories for each relevant state.
   (d) Evaluate each policy using $\widehat{\mathcal{R}}_{CF}$, $CVaR$, fuzzy ambiguity, expected reward, and Wasserstein robust loss.
   (e) If transformer policies are used, encode the history H_t and graph $\mathcal{G}_{CF}$ before action selection.
   (f) If multi-agent ecology is enabled, simulate competing agents and endogenous market feedback.
   (g) Estimate regret-curvature or Fisher-metric diagnostics for policy stability.
   (h) Select parent policies according to fitness.
   (i) Apply crossover and mutation to create offspring.
   (j) Update meta-evolutionary parameters using regime and diversity diagnostics.
   (k) Form the next population with elitism.
3. Return the policy with the best out-of-sample counterfactual robustness.

4 Results and Discussion

This project is primarily a mathematical framework and methodological design. Therefore, the results are presented as expected analytical behavior, validation criteria, and an experimental protocol rather than as claims from a completed empirical backtest.

4.1 Python Simulation Outputs

To support the mathematical framework, a reproducible Python simulation was developed. The code generates counterfactual market worlds, evaluates a policy by counterfactual regret, computes VaR and CVaR, and runs a simple evolutionary search over policy parameters. The simulation is illustrative rather than an empirical trading claim; its purpose is to demonstrate how the CREA objective can be computed and visualized.

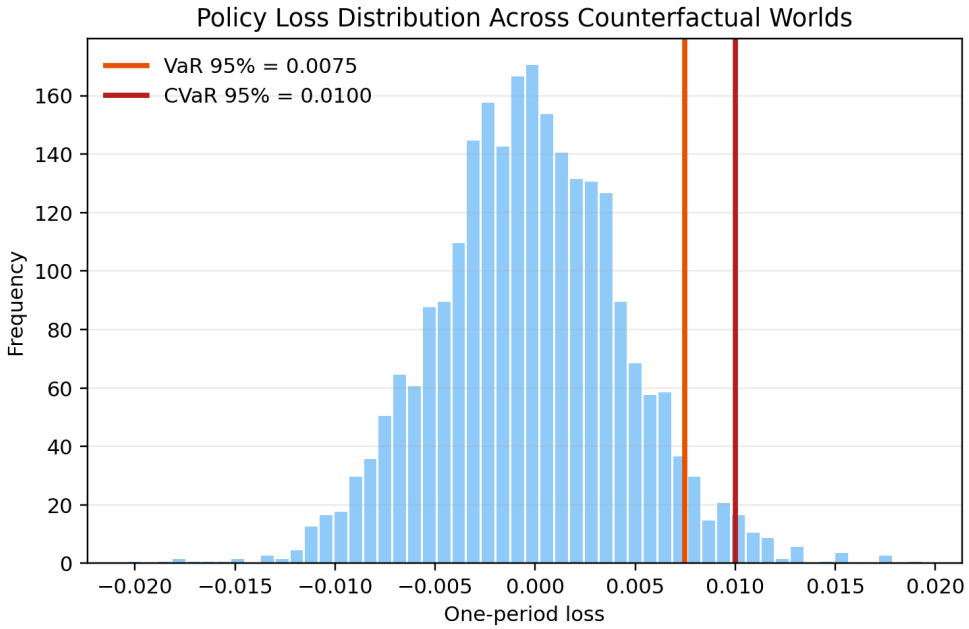

Figure 5: Loss distribution of the evolved policy across generated counterfactual worlds. The VaR and CVaR lines identify the tail-risk region controlled by the hybrid objective.

Metric	Simulated value
Mean reward	0.000321
Mean counterfactual regret	0.007191
VaR at 95 percent	0.007462
CVaR at 95 percent	0.009999
Mean allocation	0.305077
Best trend sensitivity	0.000000
Best risk aversion	3.000000

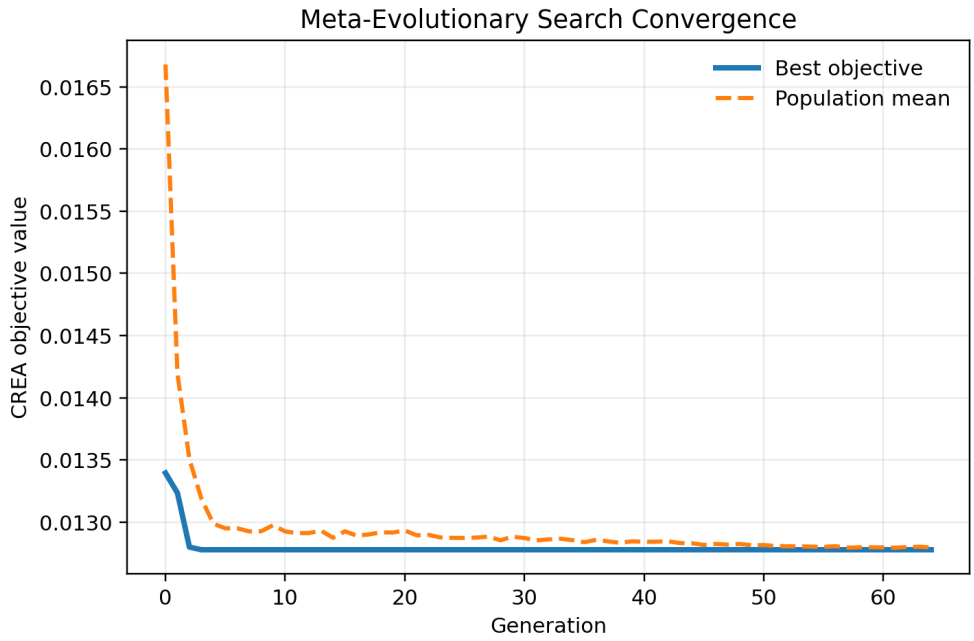

Figure 6: Meta-evolutionary search convergence. The best policy objective decreases rapidly as selection, crossover, and mutation refine the policy population.

4.2 Real Market Experiments: NIFTY 50, S&P 500, BTC, and ETH

To address practical financial validation, the framework was also tested on four real market series: NIFTY 50 (`^NSEI`), S&P 500 (`^GSPC`), Bitcoin (`BTC-USD`), and Ethereum (`ETH-USD`). Daily adjusted closing prices were downloaded using `yfinance` from Yahoo Finance [16, 23]. The available data range in the experiment is 2 January 2020 to 2 May 2026. The training period ends on 31 December 2024, and the out-of-sample test period runs from 1 January 2025 to 2 May 2026, giving 1,805 training observations and 487 testing observations.

Because crypto assets trade continuously while equity indices trade only on exchange business days, missing equity-index observations on non-trading days were forward filled before computing daily portfolio returns. This creates a common calendar for cross-market portfolio comparison. The experiment is intended as an academic backtest, not as an investment recommendation.

The following strategies were compared:

1. **CREA:** an evolutionary fuzzy/regret policy using momentum, volatility, one-day return, and drawdown features.

2. **Mean-variance:** a long-only Markowitz portfolio optimized on the training sample [12].

3. **DQN:** a lightweight neural Q-network baseline inspired by Deep Q-Networks [14].

4. **PPO:** a lightweight clipped policy-gradient baseline inspired by Proximal Policy Optimization [20].

5. **Buy and Hold:** an equal-initial-weight buy-and-hold allocation across the four-asset universe.

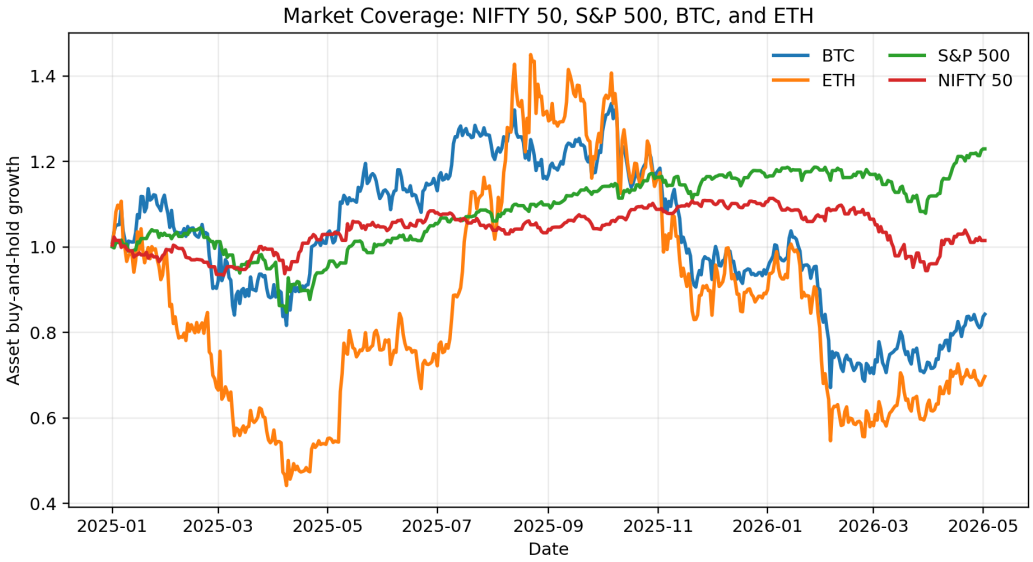

Figure 7: Out-of-sample market coverage for NIFTY 50, S&P 500, BTC, and ETH during the test period from 1 January 2025 to 2 May 2026.

Strategy	Total (%)	Annual (%)	Sharpe	MDD (%)	CVaR 95% (%)
CREA	6.35	4.72	0.322	-46.75	5.27
Mean-variance	-0.13	-0.10	0.091	-23.94	2.34
DQN	-53.12	-43.32	-0.835	-62.60	7.26
PPO	-29.50	-23.04	-0.029	-62.88	7.91
Buy and Hold	-5.84	-4.41	-0.023	-32.36	3.38

Asset	Total (%)	Annual (%)	Sharpe	MDD (%)	CVaR 95% (%)
NIFTY 50	1.49	1.12	0.149	-15.18	1.70
S&P 500	22.93	16.73	0.966	-18.90	2.17
BTC	-15.75	-12.05	-0.056	-49.74	5.29
ETH	-30.30	-23.70	0.005	-62.29	8.53

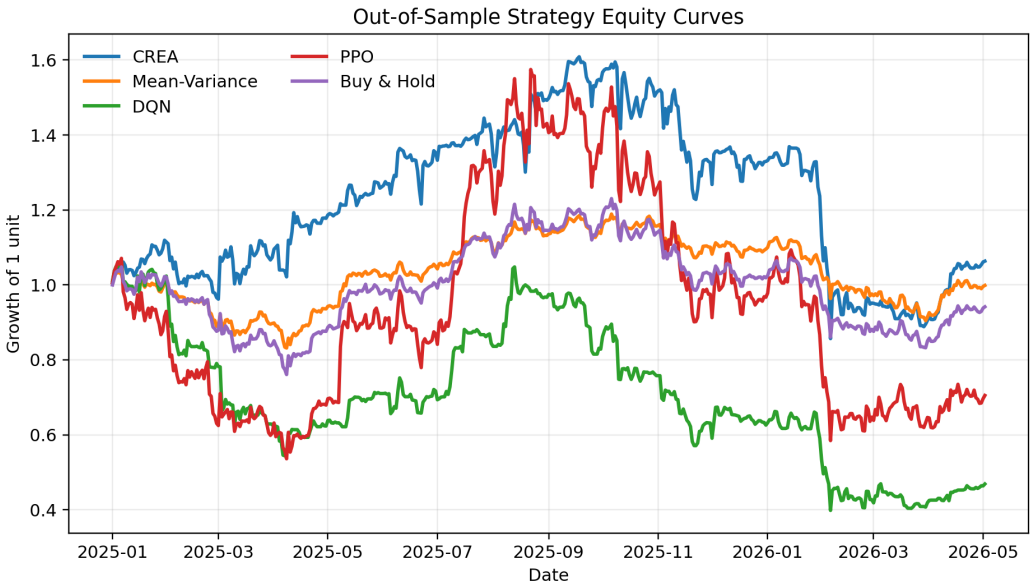

Figure 8: Out-of-sample strategy equity curves comparing CREA, mean-variance, DQN, PPO, and buy-and-hold baselines.

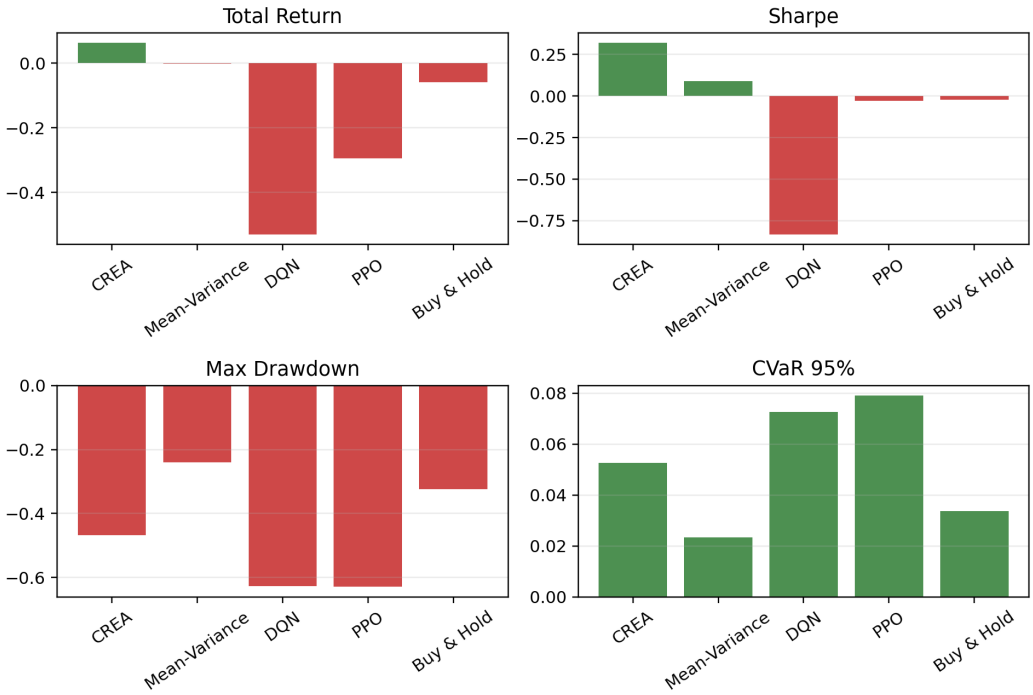

Figure 9: Metric comparison across the market experiment. CREA achieves the highest total return and Sharpe ratio in this run, while mean-variance has the lowest drawdown and CVaR.

The experiment shows why the proposed framework emphasizes robustness rather than raw prediction. During the selected out-of-sample period, the S&P 500 was the strongest single asset, NIFTY 50 was relatively stable, and BTC/ETH experienced large drawdowns. CREA produced the highest total return and Sharpe ratio among the five tested strategies, but it still experienced a large drawdown because the cross-market universe included volatile crypto assets. Mean-variance had the lowest drawdown and CVaR, but its realized return was nearly flat. The DQN and PPO baselines underperformed in this lightweight configuration, which suggests that reinforcement-learning strategies require careful reward design, transaction-cost handling, and hyperparameter tuning before they can be reliable in noisy multi-market financial environments.

4.3 Expected Behavioral Properties

CREA is expected to improve robustness in three ways. First, the CRL objective discourages policies that exploit only one historical path. Second, CVaR limits exposure to extreme losses in generated stress trajectories. Third, fuzzy ambiguity scores prevent excessive confidence when market signals are unclear or transitional.

A policy trained only on realized returns may maximize

$$\widehat{\mathbb{E}}[R_\pi]$$

while having high regret under counterfactual worlds. CREA instead seeks a policy for which

$$\widehat{\mathcal{R}}_{CF}(\pi)$$

remains small across regimes. This shifts evaluation from predictive accuracy to causal decision quality.

4.4 Evaluation Framework

The model can be evaluated using historical financial data, simulated regimes, and synthetic counterfactual worlds. A complete validation design should include:

1. **Training period:** Estimate policy parameters and learn the MWG.
2. **Validation period:** Tune λ, γ, η, and evolutionary meta-parameters.

3. **Test period:** Measure out-of-sample performance under unseen regimes.

4. **Counterfactual stress set:** Generate crash, rebound, liquidity shock, and volatility spike worlds.

4.5 Statistical Testing

Let M_{CREA} be a performance metric for the CREA policy and M_0 be the metric for a baseline strategy. A hypothesis test may be formulated as

$$H_0 : \mathbb{E}[M_{CREA} - M_0] \leq 0,$$

$$H_1 : \mathbb{E}[M_{CREA} - M_0] > 0.$$

Confidence intervals can be computed using block bootstrap methods to preserve temporal dependence in financial returns. The same testing structure can be applied to Sharpe ratio improvement, drawdown reduction, CVaR reduction, and regret stability. The selected performance metrics connect the experiment to standard risk-adjusted return and tail-risk evaluation [21, 17].

4.6 Robustness Analysis

Robustness should be evaluated under:

1. **Regime shifts:** abrupt changes from low volatility to high volatility or from bull to bear markets.

2. **Volatility spikes:** sudden increases in σ_t or realized variance.

3. **Noise injection:** corrupted signals, delayed prices, and noisy sentiment indicators.

4. **Out-of-distribution testing:** scenarios not represented in the training sample.

5. **Liquidity shocks:** widening spreads and reduced executable volume.

4.7 Advanced Validation Protocol

The upgraded CREA model requires additional validation beyond ordinary backtesting:

1. **Causal graph stability:** run causal discovery on bootstrap samples and rolling windows, then measure edge persistence and orientation stability.

2. **Temporal causality checks:** test whether discovered lagged links improve out-of-sample counterfactual prediction without introducing look-ahead bias.

3. **Wasserstein sensitivity:** evaluate policy performance over a grid of robustness radii ε to identify the cost of robustness.

4. **Transformer ablation:** compare simple feature policies, recurrent policies, Temporal Fusion Transformer policies, and Decision Transformer policies.

5. **Ecology stress tests:** simulate adversarial agents, crowded strategies, liquidity withdrawal, and feedback from large portfolio rebalancing.

6. **Regret-geometry diagnostics:** measure Hessian eigenvalues, Fisher-scaled gradient norms, and regret variance across perturbed policies.

These tests are important because a more expressive model can fail in more sophisticated ways. Causal discovery may learn unstable edges, robust optimization may become overly conservative, transformer policies may overfit long histories, and multi-agent simulations may depend strongly on behavioral assumptions. The validation protocol therefore treats originality and falsifiability as equally important.

4.8 Performance Metrics

The main metrics for judging CREA are:

$$SR, \quad VaR_\alpha, \quad CVaR_\alpha, \quad \text{MDD}, \quad \widehat{\mathcal{R}}_{CF}, \quad \text{Stability}_{CF}.$$

Maximum drawdown is

$$\text{MDD} = \max_t \left(\frac{\max_{\tau \le t} V_\tau - V_t}{\max_{\tau \le t} V_\tau} \right),$$

where V_t is portfolio value. Counterfactual stability may be measured by the dispersion of regret across generated worlds:

$$\text{Stability}_{CF}(\pi) = \frac{1}{1 + \text{Var}_k(\mathcal{R}_{CF}^{(k)}(\pi))}.$$

4.9 Discussion

The main advantage of CREA is that it treats financial optimization as a causal decision problem rather than a purely predictive problem. This is important because two policies with similar historical returns may differ sharply in counterfactual regret. One policy may have succeeded because the realized path was favorable, while another may remain more stable across many plausible worlds.

The fuzzy layer is also useful because financial regimes rarely change at precise boundaries. A volatility regime may be partially high and partially normal, and sentiment may be mixed across market participants. Fuzzy membership functions allow the policy to respond gradually. This follows the fuzzy-set principle that uncertain categories can be represented by graded membership values [25].

The advanced extension substantially increases the originality of the framework. Structural causal discovery makes the counterfactual generator data-adaptive. Wasserstein robustness turns model error into an explicit adversarial optimization problem. Transformer memory allows policies to use long-horizon temporal structure rather than only current features. Multi-agent ecology models reflexivity and strategic interaction. Regret geometry provides a theoretical language for policy stability, curvature, and sensitivity.

However, CREA also has limitations. Counterfactual worlds are only as reliable as the MWG and causal assumptions used to construct them. If latent variables are misspecified, the regret landscape may reward the wrong form of robustness. Learned causal graphs are not automatically causal truth; they require temporal ordering, economic interpretation, stability checks, and sensitivity analysis. Wasserstein robustness can reduce return if the ambiguity radius is too conservative. Transformer policies can overfit if sequence length, regularization, and data splits are not carefully controlled. Multi-agent ecology introduces behavioral assumptions that must be made transparent. Such caution is necessary because causal conclusions depend on the assumed structural model and intervention semantics [15].

5 Conclusion or Future Work

This report developed Counterfactual Regret Evolutionary Agents (CREA), a hybrid mathematical framework for adaptive decision-making in financial systems. The core idea is to evaluate strategies by their regret across counterfactual market worlds rather than

by historical returns alone. CREA integrates probability theory, stochastic processes, statistical inference, fuzzy logic, causal modeling, and evolutionary optimization into one unified agent architecture.

The framework begins with a probabilistic model of market returns and extends it using latent causal variables. A Market World Generator produces alternative trajectories under possible interventions. The upgraded version learns part of this causal structure using NOTEARS, DAG-GNN, PCMCI, and temporal causal discovery. The Counterfactual Regret Landscape measures policy quality by comparing the chosen action with the best feasible counterfactual action in each latent world. A hybrid objective then adds CVaR-based tail-risk control, fuzzy ambiguity penalties, expected reward, and Wasserstein distributional robustness. Evolutionary agents search the strategy space, while transformer memory models encode long-range temporal context and meta-evolutionary adaptation adjusts exploration as market regimes change.

The principal conclusion is that robust financial intelligence should learn from multiple possible futures, not only from one historical past. CREA provides a mathematically grounded route toward this goal by combining causal decision intelligence with risk-sensitive evolutionary learning. With the proposed advanced layers, CREA can be interpreted as a Counterfactual Causal Evolutionary Transformer Agent architecture: a system that learns causal market structure, reasons through counterfactual worlds, optimizes against adversarial distribution shifts, competes in a multi-agent ecology, and studies regret through the geometry of the policy manifold.

Future work may include:

1. Full empirical implementation on equity, index, currency, cryptocurrency, rates, and macroeconomic data.

2. Benchmarking NOTEARS, DAG-GNN, PCMCI, and Granger-style temporal discovery on rolling market windows.

3. Comparing Temporal Fusion Transformer, Decision Transformer, and financial time-series transformer policies.

4. Building multi-agent exchange simulations with transaction costs, slippage, order-book liquidity, and adaptive adversaries.

5. Estimating regret curvature and Fisher information for policy-stability certification.

6. Formal convergence and generalization analysis of the counterfactual regret objective.

References

[1] Amari S (1998) Natural gradient works efficiently in learning. Neural Computation 10(2):251-276.

[2] Artzner P, Delbaen F, Eber JM, Heath D (1999) Coherent measures of risk. Mathematical Finance 9(3):203-228.

[3] Bachelier L (1900) Theorie de la speculation. Annales Scientifiques de l'Ecole Normale Superieure 17:21-86.

[4] Ben-Tal A, El Ghaoui L, Nemirovski A (2009) Robust Optimization. Princeton University Press.

[5] Black F, Scholes M (1973) The pricing of options and corporate liabilities. Journal of Political Economy 81(3):637-654.

[6] Bollerslev T (1986) Generalized autoregressive conditional heteroskedasticity. Journal of Econometrics 31(3):307-327.

[7] Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2):182-197.

[8] Holland JH (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press.

[9] Kingma DP, Welling M (2014) Auto-encoding variational Bayes. International Conference on Learning Representations.

[10] Chen L, Lu K, Rajeswaran A, Lee K, Grover A, Laskin M, Abbeel P, Srinivas A, Mordatch I (2021) Decision Transformer: Reinforcement learning via sequence modeling. Advances in Neural Information Processing Systems 34:15084-15097.

[11] Lim B, Arik SO, Loeff N, Pfister T (2021) Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. International Journal of Forecasting 37(4):1748-1764.

[12] Markowitz H (1952) Portfolio selection. The Journal of Finance 7(1):77-91.

[13] Mohajerin Esfahani P, Kuhn D (2018) Data-driven distributionally robust optimization using the Wasserstein metric: Performance guarantees and tractable reformulations. Mathematical Programming 171:115-166.

[14] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature 518:529-533.

[15] Pearl J (2009) Causality: Models, Reasoning and Inference. Cambridge University Press.

[16] Ranaroussi R (2026) yfinance documentation: download market data from Yahoo Finance. Available at `https://ranaroussi.github.io/yfinance/`. Accessed 3 May 2026.

[17] Rockafellar RT, Uryasev S (2000) Optimization of conditional value-at-risk. Journal of Risk 2(3):21-41.

[18] Runge J, Nowack P, Kretschmer M, Flaxman S, Sejdinovic D (2019) Detecting and quantifying causal associations in large nonlinear time series datasets. Science Advances 5(11):eaau4996.

[19] Rubin DB (1974) Estimating causal effects of treatments in randomized and nonrandomized studies. Journal of Educational Psychology 66(5):688-701.

[20] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv:1707.06347.

[21] Sharpe WF (1966) Mutual fund performance. The Journal of Business 39(1):119-138.

[22] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. Advances in Neural Information Processing Systems 30.

[23] Yahoo Finance (2026) Historical market data for NIFTY 50, S&P 500, Bitcoin USD, and Ethereum USD. Available at `https://finance.yahoo.com/`. Accessed 3 May 2026.

[24] Yu Y, Chen J, Gao T, Yu M (2019) DAG-GNN: DAG structure learning with graph neural networks. International Conference on Machine Learning.

[25] Zadeh LA (1965) Fuzzy sets. Information and Control 8(3):338-353.

[26] Zheng X, Aragam B, Ravikumar P, Xing EP (2018) DAGs with NO TEARS: Continuous optimization for structure learning. Advances in Neural Information Processing Systems 31.

Appendices

Appendix A: Notation

Symbol	Meaning
Ω	Sample space of possible market outcomes
$\mathcal{F}$	Sigma-algebra of events
$\mathbb{P}$	Probability measure
R_t	Market return at time t
S_t	Asset price at time t
W_t	Wiener process
U_t	Latent causal market variables
s_t	Observed market state
a_t	Action selected at time t
π_θ	Policy parameterized by θ
$G(a, U)$	Reward under action a and latent world U
$\mathcal{R}_{CF}$	Counterfactual regret
$CVaR_\alpha$	Conditional Value at Risk at confidence level α
$\mathcal{A}_F$	Fuzzy ambiguity score
W	Learned causal adjacency matrix
$\mathcal{G}_{CF}$	Learned counterfactual causal graph
$\mathcal{U}_\varepsilon(\widehat{P})$	Wasserstein ambiguity set around empirical distribution $\widehat{P}$
H_t	Transformer history window at time t
$I(\theta)$	Fisher information matrix of policy π_θ

Appendix B: Compact CREA Objective

The final CREA objective is

$$\min_\theta \left\{ \mathbb{E}\big[\mathcal{R}_{CF}(\pi_\theta)\big] + \lambda\, CVaR_\alpha(L_{\pi_\theta}) + \gamma\, \mathbb{E}[\mathcal{A}_F(s)] - \eta\, \mathbb{E}[G_{\pi_\theta}] \right\}.$$

This expression summarizes the full framework: causal opportunity loss, tail-risk control, ambiguity awareness, and economic reward are optimized together.

The advanced robust form is

$$\min_{\theta} \sup_{Q \in \mathcal{U}_{\varepsilon}(\widehat{P})} \mathbb{E}_{Q}\left[\mathcal{R}_{CF}(\pi_{\theta}) + \lambda L^{tail}_{\pi_{\theta}} + \gamma \mathcal{A}_{F}(s) - \eta G_{\pi_{\theta}}\right],$$

with optional transformer, multi-agent, and information-geometric policy layers.

Appendix C: Python Coding

The following Python excerpts show the core simulation and policy-evaluation code used to generate the graphs in this report. The complete runnable script is saved as `crea_simulation.py`.

Listing 1: Latent world generation, policy allocation, and CREA objective evaluation.

```python
def generate_latent_worlds(rng: np.random.Generator, n_worlds: int = 2500) -> pd.DataFrame:
    regimes = rng.choice(
        ["bull", "base", "crash", "recovery"],
        size=n_worlds,
        p=[0.25, 0.42, 0.13, 0.20],
    )
    params = {
        "bull": (0.0010, 0.010),
        "base": (0.0003, 0.014),
        "crash": (-0.0026, 0.032),
        "recovery": (0.0014, 0.024),
    }
    means = np.array([params[r][0] for r in regimes])
    vols = np.array([params[r][1] for r in regimes])
    trend = rng.normal(means * 240, vols * 8)
    liquidity = rng.beta(5, 2, size=n_worlds)
    sentiment = rng.normal(trend, 0.45, size=n_worlds)
    returns = rng.normal(means + 0.00035 * sentiment, vols)
    ambiguity = 0.35 * (1 - liquidity) + 0.65 * sigmoid(12 * (vols - 0.018))
    return pd.DataFrame(
        {
            "regime": regimes,
            "return": returns,
            "vol": vols,
            "trend": trend,
            "liquidity": liquidity,
            "sentiment": sentiment,
            "ambiguity": ambiguity,
        }
    )


def policy_allocation(theta: np.ndarray, worlds: pd.DataFrame) -> np.ndarray:
    trend_weight, risk_aversion = theta
    score = 1.2 * trend_weight * worlds["trend"].to_numpy()
```

```
    score -= 18.0 * risk_aversion * worlds["vol"].to_numpy()
    score += 0.5 * worlds["sentiment"].to_numpy()
    return sigmoid(score)


def reward_for_allocation(allocation: np.ndarray, worlds: pd.DataFrame) -> np.ndarray:
    ret = worlds["return"].to_numpy()
    vol = worlds["vol"].to_numpy()
    ambiguity = worlds["ambiguity"].to_numpy()
    transaction_penalty = 0.00025 * np.abs(allocation - 0.50)
    risk_penalty = 0.15 * (allocation**2) * (vol**2)
    ambiguity_penalty = 0.0007 * allocation * ambiguity
    return allocation * ret - risk_penalty - ambiguity_penalty - transaction_penalty


def evaluate_policy(theta: np.ndarray, worlds: pd.DataFrame) -> tuple[float, dict[str, float]]:
    allocations = policy_allocation(theta, worlds)
    rewards = reward_for_allocation(allocations, worlds)
    action_grid = np.linspace(0, 1, 21)
    reward_grid = np.vstack(
        [reward_for_allocation(np.full(len(worlds), action), worlds) for action in action_grid]
    )
    best_rewards = reward_grid.max(axis=0)
    regret = best_rewards - rewards
    losses = -rewards
    var95 = np.quantile(losses, 0.95)
    cvar95 = losses[losses >= var95].mean()
    objective = regret.mean() + 0.55 * cvar95 + 0.0004 * worlds["ambiguity"].mean() - 0.25 * rewards.mean()
    return objective, {
        "mean_reward": rewards.mean(),
        "mean_regret": regret.mean(),
        "var95": var95,
        "cvar95": cvar95,
        "mean_allocation": allocations.mean(),
    }
```

Listing 2: Evolutionary search loop and figure-generation driver.

```
def run_evolution(rng: np.random.Generator, worlds: pd.DataFrame) -> tuple[np.ndarray, pd.DataFrame]:
    pop_size = 70
    generations = 65
    population = np.column_stack(
        [
            rng.uniform(0.0, 2.5, pop_size),
            rng.uniform(0.0, 2.8, pop_size),
        ]
    )
    history = []
    for generation in range(generations):
        scores = np.array([evaluate_policy(ind, worlds)[0] for ind in population])
```

```
        order = np.argsort(scores)
        best = population[order[0]].copy()
        history.append(
            {
                "generation": generation,
                "best_objective": scores[order[0]],
                "mean_objective": scores.mean(),
                "best_trend": best[0],
                "best_risk": best[1],
            }
        )
        elite = population[order[:8]]
        next_population = [elite[i].copy() for i in range(len(elite))]
        mutation_scale = max(0.04, 0.32 * (1 - generation / generations))
        while len(next_population) < pop_size:
            parents = elite[rng.choice(len(elite), 2, replace=True)]
            alpha = rng.uniform(0.25, 0.75)
            child = alpha * parents[0] + (1 - alpha) * parents[1]
            child += rng.normal([0, 0], [mutation_scale, mutation_scale * 0.55])
            child[0] = np.clip(child[0], 0.0, 2.5)
            child[1] = np.clip(child[1], 0.0, 3.0)
            next_population.append(child)
        population = np.array(next_population)
    history_df = pd.DataFrame(history)
    best_theta = history_df.sort_values("best_objective").iloc[0][["best_trend", "
        best_risk"]].to_numpy()
    history_df.to_csv(FIG_DIR / "evolution_history.csv", index=False)
    return best_theta, history_df


def plot_evolution(history: pd.DataFrame) -> None:
    plt.figure(figsize=(7.2, 4.6))
    plt.plot(history["generation"], history["best_objective"], linewidth=2.6, label="
        Best objective")
    plt.plot(history["generation"], history["mean_objective"], linewidth=2.0, linestyle=
        "--", label="Population mean")
    plt.title("Meta-Evolutionary Search Convergence")
    plt.xlabel("Generation")
    plt.ylabel("CREA objective value")
    plt.grid(True, alpha=0.25)
    plt.legend(frameon=False)
    savefig("evolution_convergence.png")


def main() -> None:
    rng = np.random.default_rng(SEED)
    worlds = generate_latent_worlds(rng)
    worlds.to_csv(FIG_DIR / "counterfactual_worlds.csv", index=False)

    plot_price_paths(rng)
    plot_architecture()
    plot_regret_landscape(worlds)
    plot_fuzzy_membership()
    best_theta, history = run_evolution(rng, worlds)
```

```
    plot_loss_distribution(worlds, best_theta)
    plot_evolution(history)

    objective, metrics = evaluate_policy(best_theta, worlds)
    summary = pd.DataFrame([{**metrics, "objective": objective, "trend_weight":
        best_theta[0], "risk_aversion": best_theta[1]}])
    summary.to_csv(FIG_DIR / "best_policy_summary.csv", index=False)
    print(summary.round(6).to_string(index=False))
```

Appendix D: Real-Market Experiment Python Coding

The complete real-market experiment is saved as `crea_market_experiments.py`. The following excerpts show the data download, evaluation metrics, CREA/mean-variance construction, and DQN/PPO baselines.

Listing 3: Market data download, features, and evaluation metrics.

```
def download_prices() -> pd.DataFrame:
    raw = yf.download(
        list(TICKERS.values()),
        start=START_DATE,
        end=END_DATE,
        auto_adjust=True,
        progress=False,
        threads=False,
    )
    close = raw["Close"].rename(columns={v: k for k, v in TICKERS.items()})
    close = close.sort_index().ffill().dropna()
    close.index = pd.to_datetime(close.index).tz_localize(None)
    close.to_csv(FIG_DIR / "market_prices.csv")
    return close


def make_features(prices: pd.DataFrame) -> Tuple[pd.DataFrame, pd.DataFrame]:
    returns = prices.pct_change().dropna()
    features = []
    for asset in prices.columns:
        frame = pd.DataFrame(index=prices.index)
        frame[(asset, "ret1")] = prices[asset].pct_change()
        frame[(asset, "mom5")] = prices[asset].pct_change(5)
        frame[(asset, "mom20")] = prices[asset].pct_change(20)
        frame[(asset, "vol20")] = returns[asset].rolling(20).std()
        frame[(asset, "drawdown20")] = prices[asset] / prices[asset].rolling(20).max() -
            1.0
        features.append(frame)
    feature_frame = pd.concat(features, axis=1).dropna()
    feature_frame.columns = pd.MultiIndex.from_tuples(feature_frame.columns)
    return returns, feature_frame


def align_state_reward(
```

```
    returns: pd.DataFrame,
    features: pd.DataFrame,
    start: str,
    end: str | None = None,
) -> Tuple[pd.DataFrame, pd.DataFrame]:
    state = features.shift(1)
    reward = returns.copy()
    idx = reward.loc[start:end].index.intersection(state.dropna().index)
    return state.loc[idx], reward.loc[idx]


def standardize_features(
    train_state: pd.DataFrame,
    test_state: pd.DataFrame,
) -> Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]:
    train_values = train_state.to_numpy()
    test_values = test_state.to_numpy()
    mean = train_values.mean(axis=0)
    std = train_values.std(axis=0) + 1e-8
    train_z = (train_values - mean) / std
    test_z = (test_values - mean) / std
    n_assets = len(TICKERS)
    n_features = train_state.shape[1] // n_assets
    return (
        train_z.reshape(len(train_state), n_assets, n_features),
        test_z.reshape(len(test_state), n_assets, n_features),
        mean,
        std,
    )


def max_drawdown(equity: pd.Series) -> float:
    peak = equity.cummax()
    drawdown = equity / peak - 1.0
    return float(drawdown.min())


def cvar_loss(returns: pd.Series, alpha: float = 0.95) -> float:
    losses = -returns
    var = losses.quantile(alpha)
    return float(losses[losses >= var].mean())


def metrics(returns: pd.Series) -> Dict[str, float]:
    returns = returns.dropna()
    equity = (1 + returns).cumprod()
    total = float(equity.iloc[-1] - 1.0)
    ann_return = float(equity.iloc[-1] ** (ANNUALIZATION / len(returns)) - 1.0)
    ann_vol = float(returns.std() * np.sqrt(ANNUALIZATION))
    sharpe = float((returns.mean() * ANNUALIZATION) / ann_vol) if ann_vol > 0 else np.
        nan
    return {
        "Total Return": total,
        "Annual Return": ann_return,
```

```
        "Annual Volatility": ann_vol,
        "Sharpe": sharpe,
        "Max Drawdown": max_drawdown(equity),
        "CVaR 95%": cvar_loss(returns),
    }
```

Listing 4: CREA evolutionary policy and mean-variance baseline.

```
def crea_weights_from_theta(theta: np.ndarray, feature_cube: np.ndarray, index: pd.Index
    , columns: pd.Index) -> pd.DataFrame:
    scores = (
        theta[0] * feature_cube[:, :, 0]
        + theta[1] * feature_cube[:, :, 1]
        + theta[2] * feature_cube[:, :, 2]
        - abs(theta[3]) * feature_cube[:, :, 3]
        + theta[4] * feature_cube[:, :, 4]
    )
    weights = softmax(scores)
    return pd.DataFrame(weights, index=index, columns=columns)


def optimize_crea(train_x: np.ndarray, train_returns: pd.DataFrame) -> np.ndarray:
    rng = np.random.default_rng(SEED)
    pop_size = 80
    generations = 55
    population = rng.normal(0, 0.7, size=(pop_size, 5))

    def objective(theta: np.ndarray) -> float:
        weights = crea_weights_from_theta(theta, train_x, train_returns.index,
            train_returns.columns)
        daily = portfolio_returns_from_weights(weights, train_returns)
        eq = (1 + daily).cumprod()
        reward = daily.mean() * ANNUALIZATION
        risk = daily.std() * np.sqrt(ANNUALIZATION)
        tail = cvar_loss(daily)
        dd = abs(max_drawdown(eq))
        return -(reward - 0.45 * risk - 8.0 * tail - 0.35 * dd)

    for generation in range(generations):
        scores = np.array([objective(theta) for theta in population])
        elite = population[np.argsort(scores)[:10]]
        children = [row.copy() for row in elite]
        mutation = max(0.04, 0.35 * (1 - generation / generations))
        while len(children) < pop_size:
            p1, p2 = elite[rng.choice(len(elite), 2, replace=True)]
            blend = rng.uniform(0.2, 0.8)
            child = blend * p1 + (1 - blend) * p2 + rng.normal(0, mutation, size=5)
            children.append(np.clip(child, -3.0, 3.0))
        population = np.array(children)
    final_scores = np.array([objective(theta) for theta in population])
    return population[np.argmin(final_scores)]


def mean_variance_weights(train_returns: pd.DataFrame) -> pd.Series:
```

```
    mu = train_returns.mean().to_numpy() * ANNUALIZATION
    cov = train_returns.cov().to_numpy() * ANNUALIZATION
    n_assets = len(mu)

    def neg_sharpe(w: np.ndarray) -> float:
        port_mu = float(w @ mu)
        port_vol = float(np.sqrt(w @ cov @ w))
        return -port_mu / (port_vol + 1e-9)

    result = minimize(
        neg_sharpe,
        np.repeat(1 / n_assets, n_assets),
        method="SLSQP",
        bounds=[(0, 1)] * n_assets,
        constraints={"type": "eq", "fun": lambda w: w.sum() - 1},
    )
    weights = result.x if result.success else np.repeat(1 / n_assets, n_assets)
    return pd.Series(weights, index=train_returns.columns, name="Mean-Variance")
```

Listing 5: DQN-style and PPO-style neural reinforcement-learning baselines.

```
def train_dqn(train_x: np.ndarray, train_returns: pd.DataFrame, test_x: np.ndarray) ->
    np.ndarray:
    if keras is None:
        return np.argmax(test_x[:, :, 1], axis=1)

    tf.random.set_seed(SEED)
    states = train_x.reshape(len(train_x), -1).astype("float32")
    q_targets = train_returns.to_numpy().astype("float32") * 100.0
    n_actions = train_returns.shape[1]

    model = build_mlp(n_actions)
    model.compile(optimizer=keras.optimizers.Adam(0.001), loss="mse")
    # Offline one-step Q approximation: Q(s, a) is trained toward next-day reward.
    # This keeps the DQN baseline reproducible and lightweight for a thesis report.
    model.fit(states, q_targets, epochs=35, batch_size=128, verbose=0)
    test_states = test_x.reshape(len(test_x), -1).astype("float32")
    return np.argmax(model.predict(test_states, verbose=0), axis=1)


def train_ppo(train_x: np.ndarray, train_returns: pd.DataFrame, test_x: np.ndarray) ->
    np.ndarray:
    if keras is None:
        return np.argmax(test_x[:, :, 1] - test_x[:, :, 3], axis=1)

    tf.random.set_seed(SEED + 1)
    rng = np.random.default_rng(SEED + 1)
    states = train_x.reshape(len(train_x), -1).astype("float32")
    rewards = train_returns.to_numpy().astype("float32") * 100.0
    n_actions = train_returns.shape[1]

    policy = build_mlp(n_actions, activation="softmax")
    value = build_mlp(1)
    policy_opt = keras.optimizers.Adam(0.0008)
```

```
    value_opt = keras.optimizers.Adam(0.001)
    clip_eps = 0.18

    for _ in range(14):
        probs = policy.predict(states, verbose=0)
        actions = np.array([rng.choice(n_actions, p=p / p.sum()) for p in probs])
        old_probs = probs[np.arange(len(states)), actions]
        chosen_rewards = rewards[np.arange(len(states)), actions]
        values = value.predict(states, verbose=0).reshape(-1)
        advantages = chosen_rewards - values
        advantages = (advantages - advantages.mean()) / (advantages.std() + 1e-8)

        dataset = tf.data.Dataset.from_tensor_slices(
            (states, actions.astype("int32"), old_probs.astype("float32"), advantages.
                astype("float32"), chosen_rewards.astype("float32"))
        ).shuffle(2048, seed=SEED).batch(128)

        for s, a, old_p, adv, rew in dataset:
            with tf.GradientTape() as tape:
                current_probs = policy(s, training=True)
                action_mask = tf.one_hot(a, n_actions)
                new_p = tf.reduce_sum(current_probs * action_mask, axis=1)
                ratio = new_p / (old_p + 1e-8)
                unclipped = ratio * adv
                clipped = tf.clip_by_value(ratio, 1 - clip_eps, 1 + clip_eps) * adv
                entropy = -tf.reduce_mean(tf.reduce_sum(current_probs * tf.math.log(
                    current_probs + 1e-8), axis=1))
                loss = -tf.reduce_mean(tf.minimum(unclipped, clipped)) - 0.01 * entropy
            grads = tape.gradient(loss, policy.trainable_variables)
            policy_opt.apply_gradients(zip(grads, policy.trainable_variables))

            with tf.GradientTape() as tape:
                pred = tf.squeeze(value(s, training=True), axis=1)
                value_loss = tf.reduce_mean((pred - rew) ** 2)
            grads = tape.gradient(value_loss, value.trainable_variables)
            value_opt.apply_gradients(zip(grads, value.trainable_variables))

    test_states = test_x.reshape(len(test_x), -1).astype("float32")
    probs = policy.predict(test_states, verbose=0)
    return np.argmax(probs, axis=1)
```