Path Dependent Learning Dynamics of ICM PPO with Epstein Zin Utility in Time Series Reinforcement Learning

GitHub: https://github.com/utilityfog/EZ_Optimization

Define the Problem

Big picture

Financial decision making over time is usually framed as predicting returns and then choosing an action based on those predictions. Classical models treat each decision in isolation and optimize a simple expected utility of wealth. Modern reinforcement learning (RL) instead treats the investor as an agent that repeatedly interacts with a stochastic environment and learns a policy that maximizes long run reward.

In this project we study a curiosity driven RL agent that allocates wealth between a risky asset and cash using a single asset time series. The agent uses Proximal Policy Optimization (PPO) together with an Intrinsic Curiosity Module (ICM). The value function is motivated by Epstein Zin recursive utility, which separates risk aversion from intertemporal substitution. Our question is not only whether this agent can earn good returns, but also how its learning path in early training episodes influences its long run behavior.

Specific problem and hypothesis

We focus on the following questions.

1. Can an ICM PPO agent trained on a univariate financial time series learn a stable policy that produces positive risk adjusted returns out of sample

2. Are the learning dynamics path dependent in the sense that small stochastic differences during the first few episodes drive the agent either toward a good regime or into a degenerate regime

3. How do key state variables and hyperparameters relate to performance

    - fractionally differenced return feature

    - rolling volatility

    - curiosity bonus scale

Our working hypothesis is that the training dynamics of the agent are path dependent. In particular, we posit the existence of at least two basins of attraction in policy space. In a favorable regime, the stochastic trajectories observed in the first few episodes produce value targets where intrinsic rewards and realized external returns are positively aligned, so gradient updates push the policy toward nontrivial trading behavior and stable Epstein Zin value estimates. In an unfavorable regime, early trajectories yield value targets dominated by noisy or misaligned intrinsic rewards, so the same architecture and hyperparameters drive the policy toward near zero allocation and effectively shut down exploration. Once the parameters enter one of these regions, subsequent PPO updates remain in the corresponding basin, so later episodes cannot reliably move the agent from the degenerate regime to the favorable regime.

Data Collection

Source and structure

All raw data are loaded in load_raw_data() inside src/data_preprocessing.py. The sources are

- Daily S&P 500 OHLCV (open, high, low, close, volume) data stored in sp500_df.csv
- Monthly personal savings rate stored in psavert_df.csv
- Monthly unemployment rate stored in unemploy_df.csv

The preprocessing code converts each raw date column to a proper timestamp, then aggregates the daily S&P 500 data to month end by taking the last trading day in each calendar month. From the month end close prices it computes the monthly percentage return SP500_Returns. Savings and unemployment are already monthly series and are merged on the same month index.

The resulting merged dataset has the following columns for each month:

- Month end date
- S&P 500 open, high, low, close, and volume
- Personal savings rate
- Unemployment rate
- S&P 500 monthly return

After dropping the first row, whose return is undefined, we obtain a contiguous monthly panel. With the default chronological split of 80 percent for training and 20 percent for testing, the script reports

- 391 training months with 7 numeric features

- 98 test months with 7 numeric features

and stores

- features.npy and returns.npy for the train set
- features_test.npy and returns_test.npy for the test set

Sampling limitations

The dataset reflects one historical period for a single broad equity index and two macro variables. There is strong temporal dependence, and the sample is not representative of all possible future regimes. Results therefore cannot be interpreted as general evidence that the strategy works on every asset or time period. The purpose of the project is to study learning dynamics in a controlled setting rather than to propose a ready to trade strategy.

Data Preparation

All data preparation steps are implemented in src/data_preprocessing.py.

Cleaning and filtering

- The first row is dropped because SP500_Returns is NaN there.
- Any remaining rows with non finite values in SP500_Returns are removed, though for this dataset there are no additional missing values after the first drop.
- The merged dataset is sorted chronologically so that all subsequent transformations respect time order.

Feature construction

The feature matrix features_raw consists of the following seven columns:

1. S&P 500 open price (month end)
2. S&P 500 high price
3. S&P 500 low price
4. S&P 500 close price
5. S&P 500 trading volume

6. Personal savings rate

7. Unemployment rate

All seven columns are cast to float32 and stacked into an array of shape (n_months, 7).

The target array gross_returns is defined as

$$
\text { gross_returns }_t=1+\text { SP500_Returns }{ }_t
$$

so each element represents the gross portfolio return of investing in the S&P 500 between month t and t+1.

Expanding window normalization

Rather than computing a single global mean and standard deviation, the script uses an expanding window z score transform implemented in expanding_normalize().

For each feature dimension j and time index t:

- It computes the cumulative mean of feature j over months 0 through t.

- It computes the cumulative variance using cumulative sums of values and squared values.

- It defines the standard deviation as the square root of the variance, with a small floor of 10^{-8} to avoid division by zero.

- It normalizes the current value by

$$
z_{t, j}=\frac{x_{t, j}-\mu_{t, j}}{\sigma_{t, j}} .
$$

This produces an output matrix of the same shape where each row is normalized using only past and present information. From a time series perspective this is important, because it prevents the agent from peeking at future statistics during training.

Train test split

After normalization, the script performs a chronological split:

- The first $\lfloor 0.8 n\rfloor$ = 391 months form the training set.

- The remaining 98 months form the test set.

Training features and returns are saved as features.npy and returns.npy. Test arrays are saved as features_test.npy and returns_test.npy. The RL environment only sees the training arrays during learning. The test arrays are reserved for out-of-sample evaluation.

There are no categorical variables, so no encoding is required.

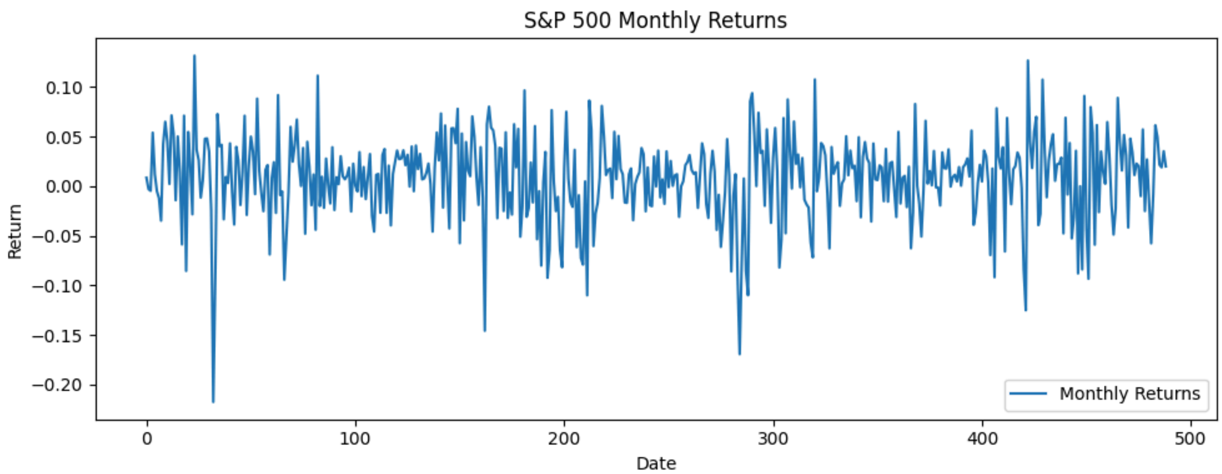
Data Exploration

We explore the processed training dataset before training the RL agent. All visualizations mentioned here are generated in the accompanying code.

Univariate patterns

1. S&P 500 monthly gross return

Visualization: histogram of gross_returns - 1.



Observation: The histogram of S&P 500 monthly excess returns is centered slightly above zero, with most months between about −5% and +5%, a small positive mean around 0.8% per month, and a few extreme negative and positive outliers. This suggests a mildly right-shifted but heavy-tailed distribution, consistent with the idea that typical months are modestly positive while occasional shocks dominate long run performance.

2. Covariance Matrix of features and returns

Visualization: table of Pearson correlations between features and returns

```

===== SUMMARY STATISTICS =====
```

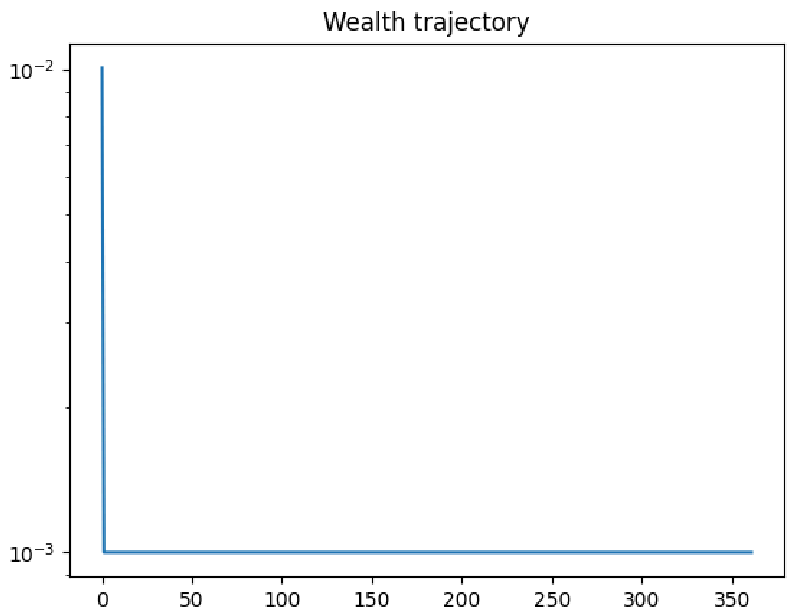
```
            open         high          low  SP500_Close       volume  Personal_Savings_Rate
Unemployment  SP500_Returns
count   489.000000   489.000000   489.000000   489.000000  4.890000e+02
363.000000    363.000000     489.000000
mean   1617.766602  1627.014526  1607.516113  1617.237183  2.387948e+09
6.684023   8589.178711       0.008431
std    1419.480103  1426.659668  1409.792480  1418.759399  2.034217e+09
1.799131   2429.459961       0.043631
min     179.539993   180.630005   178.860001   179.830002  1.499000e+07            2.200000
5481.000000     -0.217630
25%     513.640015   515.289978   510.899994   514.710022  3.172200e+08            5.400000
6932.500000     -0.017004
50%    1211.369995  1222.739990  1204.520020  1218.890015  1.799770e+09
6.800000   7946.000000       0.012209
75%    2073.169922  2075.760010  2057.939941  2065.300049  4.167160e+09
8.100000   9135.000000       0.035794
max    6860.500000  6880.750000  6820.689941  6822.339844  8.926480e+09
12.000000  15352.000000       0.131767

===== COVARIANCE MATRIX =====
                          open          high          low   SP500_Close       volume
Personal_Savings_Rate  Unemployment  SP500_Returns
open                 2.014924e+06  2.025040e+06  2.001089e+06  2.013699e+06  2.025135e+12
-5.046762e+02  2.601919e+05   2.938607e+00
high                 2.025040e+06  2.035358e+06  2.011179e+06  2.024000e+06  2.037605e+12
-5.086552e+02  2.600097e+05   2.886005e+00
low                  2.001089e+06  2.011179e+06  1.987515e+06  2.000047e+06  2.008718e+12
-5.010075e+02  2.583550e+05   3.070056e+00
SP500_Close          2.013699e+06  2.024000e+06  2.000047e+06  2.012878e+06
2.024120e+12         -5.040928e+02  2.586988e+05   3.026843e+00
volume               2.025135e+12  2.037605e+12  2.008718e+12  2.024120e+12  4.138041e+18
-1.048551e+09  2.809062e+12  -5.477668e+06
Personal_Savings_Rate -5.046762e+02 -5.086552e+02 -5.010075e+02 -5.040928e+02
-1.048551e+09          3.236871e+00  7.837916e+02   6.771722e-03
Unemployment         2.601919e+05  2.600097e+05  2.583550e+05  2.586988e+05
2.809062e+12          7.837916e+02  5.902276e+06   2.757018e+00
SP500_Returns        2.938607e+00  2.886005e+00  3.070056e+00  3.026843e+00
-5.477668e+06          6.771722e-03  2.757018e+00   1.903655e-03
```

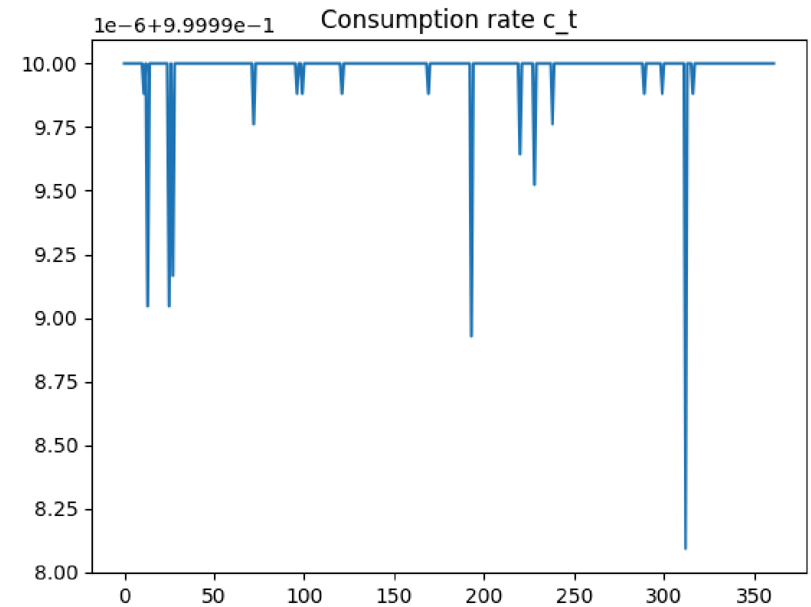
Observation: The covariance matrix shows that the four price levels (open, high, low, close) and trading volume move almost perfectly together, with covariances on the order of 10^6 between prices and 10^{12} between prices and volume, indicating strong comovement and multicollinearity among these level variables. In contrast, the personal savings rate and unemployment have much smaller variances and covariances that reflect slow-moving macro trends. Covariances involving SP500_Returns are close to zero relative to its own variance (about 1.9 x 10^{-3}), which suggests that one-month returns are only weakly linearly related to price levels, volume, or the macro variables. This supports treating return prediction as a difficult problem where simple linear structure is limited.

3. Training Wealth Trajectory and Policy Decisions (illustrates path dependence)

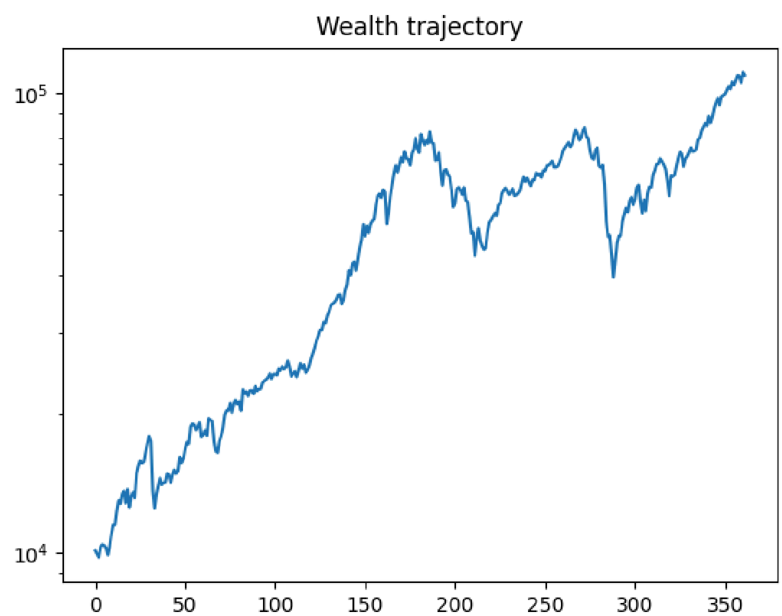
Visualization: Wealth Trajectory in a bad-seed round in the final episode



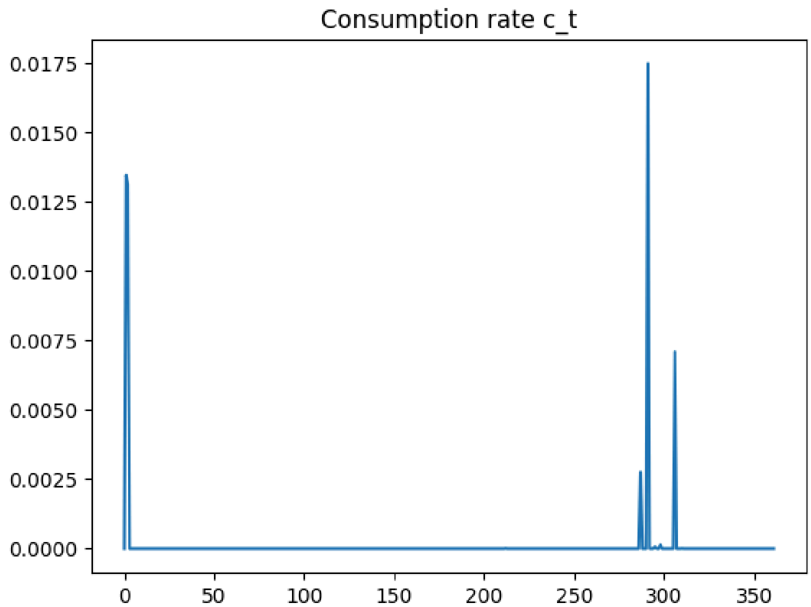
Visualization: Policy actions in a bad seed round in the final episode



Visualization: Wealth Trajectory in a good-seed round in the final episode



Visualization: Policy actions in a good seed round in the final episode



Observation: The wealth and policy plots exhibit a clear path-dependence phenomenon. In the bad-seed run, wealth falls by several orders of magnitude early in the episode and then hovers near the numerical floor, while the policy quickly drives the consumption fraction toward its upper bound, effectively liquidating wealth and preventing recovery. In the good-seed run, wealth grows steadily over the episode, ending far above the initial level, and the policy maintains moderate consumption rates that only rise after wealth has increased. With identical architecture and hyperparameters, the only difference between these runs is the stochastic training path, so the divergence in wealth and consumption policies provides visual evidence of multiple basins of attraction in the learning dynamics.

These explorations satisfy the requirement to examine at least three variables and their associations. They also motivated our final feature set: we retain the full 7 dimensional feature vector, but in this writeup we focus on the economic interpretation of returns, savings rate, unemployment, and volume.

Model Building

The key hyperparameters include:

- Curiosity weight that scales the intrinsic reward.
- Learning rate for the optimizer.
- Fractional differencing parameter used in feature construction.

These act as the three main “design variables” we vary in experiments, alongside the observed state features themselves.

src/config.py
```
class Config:
    # EZ parameters
    beta = 0.96
    psi = 1.5
    gamma_risk = 5.0

    # fracdiff preprocessing (still used by data_preprocessing)
    frac_d = 0.4
    frac_tol = 1e-4
    frac_max_lag = 512

    # learnable fractional differencing (inference-time)
    use_learnable_fracdiff = True
    fd_window = 12              # 12 months window in the state
    fracdiff_max_lag = 12       # kernel length in LearnableFracDiff1D
    fracdiff_init_d = 0.4

    # PPO
    gamma = 0.99
    gae_lambda = 0.95
    clip_ratio = 0.1 # 0.2
    vf_coeff = 1.0 # 0.5
    ent_coeff = 0.1
    ppo_epochs = 10
    batch_size = 128
    lr = 1e-2 # Changing this to any other value, such as 1e-4, fucks performance.

    # training
    num_episodes = 10
    start_wealth = 10000.0
    k_terminal: float = 1000.0

    device = "mps"
```

Why this model

Standard DATA110 models such as linear regression or decision trees operate on independent and identically distributed samples and output a one step prediction. Our problem is inherently sequential. Actions taken today change both wealth and the state distribution tomorrow. PPO is a natural choice because:

- It can handle continuous actions in bounded ranges.
- It uses a clipped objective that stabilizes policy updates.
- It is widely used in practice and relatively easy to implement.

The ICM adds an exploration pressure that is important when external rewards are sparse or noisy, which is very common in financial decision problems.

0) PURPOSE & SCOPE

This document **replaces CRRA** with **Epstein–Zin (EZ)** recursive preferences and **adds a Learnable Fractional Differencing (FracDiff) layer** in the feature pipeline, while preserving the **PPO + ICM** training stack and environment mechanics. It is **drop-in**: policy parameterization, buffers, rollout loop, exact log-probabilities, and PPO machinery remain intact. Only the **preference aggregator** (reward/value semantics) and **feature memory module** (FracDiff) change.

What stays the same (do not touch):

- Time/indexing, assets, returns, risk-free, ~~turnover/transaction cost~~, budget identity.
- Actor heads (consumption squashed Gaussian; ~~risky weights Dirichlet/softmax~~), exact log-prob math. (No more portfolio optimization)
- Critic backbone mechanics (but we output two EZ heads; see §5).
- ICM encoder/forward (and optional inverse) and curiosity reward wiring.
- PPO: ratio, clipping, GAE, epochs/minibatching, entropy, optimizers.
- Data split, standardization, rollout collection, buffer contents.

What changes:

1. **Utility/Value:** CRRA is replaced by **Epstein–Zin** with a numerically stable target in (z)-space (§4–§6).
2. **Features:** Insert **Learnable FracDiff** over returns before feature construction (§3).

---

1) Core definitions (time, single risky asset, wealth, consumption)

We now consider a **single risky asset** (S&P) and remove portfolio optimization entirely.

1.1 Time and assets

- Discrete time ($t = 0, 1, 2, \ldots, T-1$).
- One risky asset with **gross** return ($R_{t+1} \in R_{>0}$) between (t) and ($t+1$).
- No explicit risk–free asset and no portfolio weights – all *unconsumed* wealth is automatically invested in the risky asset.

1.2 Wealth, consumption, normalization

- Wealth at start of step (t): ($W_t > 0$).
- Consumption fraction (action): ($c_t \in (0,1)$); **dollar consumption** ($C_t := c_t \cdot W_t$).
- Running max wealth ($M_t := \max_{0 \le \tau \le t} W_\tau$); normalized wealth ($\tilde{W}_t := W_t / M_t \in (0,1]$).

1.3 Budget identity (wealth transition)

In the simplified world, after consuming ($C_t = c_t W_t$), the remaining wealth ($(1-c_t)W_t$) is fully invested in the risky asset, which realizes a gross return (R_{t+1}) over ($[t, t+1]$).

The **wealth transition** is

$$([W_{t+1} = (1 - c_t) W_t R_{t+1} \, .])$$

We may optionally clip (W_{t+1}) below by a small floor $(\varepsilon_W > 0)$ for numerical stability. Running max wealth is updated as

$$([M_{t+1} := \max(M_t, W_{t+1}) \, .])$$

---

2) OBSERVATIONS, FEATURES, STATE (BASELINE PIPELINE)

2.1 Observables at time (t)

- (W_t) and a causal feature vector $(x_t \in R^d)$ built **only** from data $(\leq t)$.
- Standardize (x_t) via train-set $((\mu, \sigma))$ to $(\tilde{x}_t)$ (store (μ, σ) from training only).

2.2 State to networks

- **State:** $(s_t := \text{concat}\left(\tilde{W}_t, \tilde{x}_t\right) \in R^{1+d+n})$ (fixed order).

At time (t) the agent observes:

- Normalized wealth $(\tilde{W}_t = W_t \ / \ M_t)$.
- A standardized feature vector $(\tilde{x}_t \in R^d)$, built from the FracDiff pipeline and other signals, using only information up to time (t).

The **state** fed to the policy and critic is

$$(s_t := \text{concat}\left(\tilde{W}_t, \tilde{x}_t\right) \in R^{1+d} \, .)$$

There is no (w_{t-1}) term in the state any more, since there is no portfolio decision.

---

3) Learnable Fractional Differencing (returns-domain feature module)

3.1 Goal & parameter

Learn a memory depth $(d_{\text{target}} \in [d_{\min}, d_{\max}])$ (e.g., $([0, 1])$) that controls the fractional differencing of returns to **capture long memory** while promoting **stationarity**.

3.2 Placement in pipeline

- Input raw **log-returns** per asset: $(r_t \in R^n)$ (or windows).
- Apply a FracDiff operator with effective exponent (d_{eff}):
  - **Mode "direct"**: apply $((1 - L)^{d_{\text{target}}})$ to returns.
  - **Mode "price_equiv"**: apply $((1 - L)^{d_{\text{target}} - 1})$ to returns (equivalent to price fracdiff of (d_{target}) without reconstructing prices).

- Truncate the kernel to length (K) (auto-chosen from (d_{eff}) and a tolerance). Outputs lose the first (K) steps.

3.3 State augmentation & alignment

- Build usual statistics **from** the FD output (lags, MAs, vol, PCA, cross-sectional transforms).
- **Shift** all time-aligned targets by (K) (drop first (K) steps) so shapes match.
- Optionally append $(\text{stop}_{\text{grad}}(d_{\text{target}}))$ and (K) as scalar features so the policy/critic can adapt to memory depth.

3.4 Regularization & constraints

- Keep (d_{target}) within bounds via a sigmoid reparameterization.
- Add a small L2 penalty if (d_{target}) sticks to the bounds.
- Optional "whiteness" regularizer: penalize low-lag autocorrelation of FD residuals to avoid over-memory.

> **Everything backpropagates end-to-end** because kernel weights are differentiable functions of (d_{eff}).

---

4) Epstein–Zin Preferences (replace CRRA)

Let ($\beta \in (0, 1)$) be the subjective discount, ($\gamma > 0$) risk aversion, ($\psi > 0$) intertemporal elasticity (EIS). Define transforms:

- ($z(V) := V^{1-\frac{1}{\psi}}$) (EIS/consumption space)
- ($y(V) := V^{1-\gamma}$) (risk space)

4.1 EZ aggregator (Kreps–Porteus form)

For lifetime utility (V_t) and consumption (C_t):

$$([V_t = \left[(1-\beta) C_t^{1-\frac{1}{\psi}} + \beta \left(E_t[V_{t+1}^{1-\gamma}] \right)^{\frac{1-\frac{1}{\psi}}{1-\gamma}} \right]^{\frac{1}{1-\frac{1}{\psi}}}.])$$

4.2 Practical RL parameterization (stable targets)

We **train in** (z)**-space** with a two-head critic predicting ($\hat{z}_t \approx z(V_t)$) and ($\hat{y}_t \approx y(V_t)$).

- **External (shaped) reward:** ($r_t^{\text{ext}} := (1-\beta) C_t^{1-\frac{1}{\psi}}$).
- **One-step bootstrap target for** (z)**:**

$$([T_t^{(z)} := (1-\beta) C_t^{1-\frac{1}{\psi}} + \beta \left(\hat{y}_{t+1} \right)^{\frac{1-\frac{1}{\psi}}{1-\gamma}}.])$$

- **Value loss:** ($L_{\text{value}} := \frac{1}{2} \left(\hat{z}_t - T_t^{(z)} \right)^2$).

- Optional **consistency** regularizer: encourage ($\hat{y}_t \approx (\hat{z}_t)^{\frac{1-\gamma}{1-\frac{1}{\psi}}}$) with a small weight.

> **Degeneracies:** ($\psi \to 1$) approaches additive/separable (log-like); ($\gamma \to 1$) reduces risk curvature; recipe reduces toward CRRA smoothly.

---

5) ACTOR & CRITIC (Z-functions, distributions, exact log-probs)

We now have a **single action dimension**: the consumption rate ($c_t \in (0,1)$).

Dimensions and symbols used throughout this section

- State at time (t): ($s_t \in R^{1+d+n}$) is the concatenation ($s_t := \text{concat}\left(\tilde{W}_t, \tilde{x}_t\right)$), where ($\tilde{W}_t = W_t \,/\, M_t$) and ($\tilde{x}_t$) is the standardized feature vector.
- Consumption fraction (action component): ($c_t \in (0,1)$). Dollar consumption: ($C_t := c_t W_t$).
- Hyperparameters for heads: ($\sigma_{\min} > 0$) (std floor).

5.1 Actor (f_θ)

The actor takes ($s_t \in R^{1+d}$) and passes it through a shared backbone (e.g. an MLP) to produce parameters for a scalar Gaussian in a latent space:

- Pre–squash Normal parameters: ($[\mu_c(s_t) \in R, \quad \ell_c(s_t) \in R, \quad \sigma_c(s_t) := \text{softplus}(\ell_c) + \sigma_{\min} .]$)
- Sample pre–squash variable ($[y_c \sim N\left(\mu_c(s_t), \sigma_c(s_t)^2\right) .]$)
- Squash to the action space via the sigmoid: ($[c_t := \sigma(y_c) = \frac{1}{1+e^{-y_c}} \in (0,1) .]$)
- Deterministic (evaluation) action is given by ($[c_t^{\text{det}} := \sigma\left(\mu_c(s_t)\right) .]$)

There is no risky–weights head any more; (w_t) is implicitly equal to (1) on the single asset.

5.2 Exact log–probability of (c_t)

Let

$$([y_c = \text{logit}(c_t) = \log\frac{c_t}{1-c_t} .])$$

The log–probability under the squashed Gaussian is

$$([\log p(c_t \mid s_t) = \log N\left(y_c, \mu_c(s_t), \sigma_c(s_t)^2\right) - \log\left(c_t(1-c_t)\right),])$$

where the first term is the Gaussian log–density of (y_c) and the second term is the log–Jacobian of the sigmoid.

This ($\log p(c_t \mid s_t)$) is the **only** action log–probability used in PPO here.

5.3 Critic (g_ψ) (two heads for EZ)

The critic takes (s_t) and outputs two scalars:

- $(\hat{z}_t \approx z(V_t))$ with $(z(V) := V^{1-\frac{1}{\psi}})$.
- $(\hat{y}_t \approx y(V_t))$ with $(y(V) := V^{1-\gamma})$. These are used to build the EZ bootstrap target and TD residual below.

---

6) ENVIRONMENT STEP (FULL SEQUENCE)

At time (t), given state (s_t) and sampled consumption rate (c_t), the environment performs:

1. **Consumption and wealth evolution**

Dollar consumption:

$$([C_t = c_t W_t\,.])$$

Remaining wealth:

$$([W_t^{\text{after}} = (1 - c_t) W_t\,.])$$

Apply the risky asset gross return (R_{t+1}):

$$([W_{t+1} = W_t^{\text{after}} R_{t+1} = (1 - c_t) W_t R_{t+1}\,.])$$

Optionally clip $(W_{t+1} \geq \varepsilon_W)$ if needed for numerical stability.

2. **Running max and next state**

$$([M_{t+1} = \max\left(M_t, W_{t+1}\right), \quad \tilde{W}_{t+1} = \frac{W_{t+1}}{M_{t+1}}\,.])$$

The feature pipeline (including FracDiff) produces the next standardized feature vector $(\tilde{x}_{t+1})$ from market data up to time $(t+1)$.

The next state is

$$([s_{t+1} = \text{concat}\left(\tilde{W}_{t+1}, \tilde{x}_{t+1}\right)\,.])$$

3. **Termination**

   Episodes end when $(t = T - 1)$ (or when data runs out).

Wealth transition

- Gross growth factor:

$$([G_{t+1} := (1 - c_t)\left(R_f[t+1] + w_t^\top \tilde{R}[t+1]\right) - \kappa \|w_t - w_{t-1}\|_1\,.])$$

- Next wealth: ($W_{t+1} := W_t \cdot G_{t+1}$). Safety floor may clip ($G_{t+1} \geq \varepsilon_g > 0$).

---

7) REWARDS (EXTERNAL EZ FLOW, INTRINSIC ICM)

We use the Epstein–Zin flow term for external reward and the Intrinsic Curiosity Module (ICM) to supply an intrinsic shaping signal.

EZ parameters

- Discount ($\beta \in (0,1)$)
- Risk aversion ($\gamma > 0$)
- Elasticity of intertemporal substitution (EIS) ($\psi > 0$)
- Consumption ($C_t = c_t W_t$)

7.1 External reward (EZ flow term in (z)-space)

The **external reward** at time (t) is the EZ flow term in (z)–space, depending only on consumption:

$$([r_t^{\text{ext}} = (1-\beta) C_t^{1-\frac{1}{\psi}} = (1-\beta)(c_t W_t)^{1-\frac{1}{\psi}}.])$$

This is the main objective that encourages good consumption timing.

7.2 Intrinsic Curiosity Module (ICM) — complete specification

We define the ICM exactly and fully:

Network dimensions

Let:

- Feature dimension (d)
- State dimension ($1+d$) since state is ($\text{concat}(\tilde{W}_t, \tilde{x}_t)$)
- State-embedding dimension (m) (e.g., 64)
- Hidden widths for ICM networks:
  - Encoder hidden width (E) (e.g., 128)
  - Forward-model hidden width (F) (e.g., 128)

Define the state encoder:

$$([\phi_\omega : R^{1+d} \to R^m.])$$

State encoder network

Given state ($s_t \in R^{1+d}$):

$$([e1 = \text{GELU}(W_{e1}s_t + b_{e1}), \quad W_{e1} \in R^{E \times (1+d)}.])$$

$$([e2 = \text{GELU}(W_{e2}e1 + b_{e2}), \quad W_{e2} \in R^{E \times E}.])$$

$$([\phi(s_t) = W_{eo}e2 + b_{eo}, \quad W_{eo} \in R^{m \times E}.])$$

Define:

$$([\phi_t := \phi(s_t), \qquad \phi_{t+1} := \phi(s_{t+1}).])$$

Action embedding (scalar action)

Because the action is **only** consumption ($c_t \in (0,1)$), we embed it as:

$$([y_c = \text{logit}(c_t) = \log \frac{c_t}{1 - c_t}.])$$

Then define:

$$([\psi(a_t) := \psi(c_t) := y_c \in R^1.])$$

Dimensions: action embedding is 1-dimensional.

Forward dynamics model

Maps (($\phi(s_t), \psi(a_t)$)) into a prediction of ($\phi(s_{t+1})$).

Input dimension to forward model:

$$([m + 1.])$$

Forward model layers:

$$([u1 = \text{GELU}\left(W_{f1}, \text{concat}(\phi_t, \psi(c_t)) + b_{f1} \right), \quad W_{f1} \in R^{F \times (m+1)}.])$$

$$([u2 = \text{GELU}(W_{f2}u1 + b_{f2}), \quad W_{f2} \in R^{F \times F}.])$$

$$([\hat{\phi}_{t+1} := W_{fo}u2 + b_{fo}, \quad W_{fo} \in R^{m \times F}.])$$

There is **no inverse model** in the consumption-only version.

Intrinsic reward

Given:

- Encoded next state (ϕ_{t+1})
- Predicted next state ($\hat{\phi}_{t+1}$)

The intrinsic reward is:

$$([r_t^{\text{int}} := \eta \left|\phi_{t+1} - \hat{\phi}_{t+1}\right|_2^2,])$$

with a small scale factor $(\eta > 0)$ (e.g., (10^{-3})).

ICM losses

Forward loss:

$$([L_{\text{fwd}}(\omega) := \left|\phi_{t+1} - \hat{\phi}_{t+1}\right|_2^2 .])$$

Inverse loss:

$$([L_{\text{inv}} := 0])$$

since we removed portfolio weights and do not reconstruct (w_t).
The action is 1-dimensional and directly known, so inverse dynamics is unnecessary.

Total ICM loss:

$$([L_{\text{ICM}} = L_{\text{fwd}} .])$$

7.3 Total reward used by PPO

$(r_t := r_t^{\text{ext}} + r_t^{\text{int}})$.

---

8) Advantages, EZ targets, and losses (consumption-only)

All variables used below are defined here or earlier sections.

8.1 EZ one-step target in (z)-space

We have two critic heads:

- $(\hat{z}_t \approx z(V_t) := V_t^{1-\frac{1}{\psi}})$
- $(\hat{y}_t \approx y(V_t) := V_t^{1-\gamma})$

The one-step EZ bootstrap target is:

$$([T_t^{(z)} = (1-\beta) C_t^{1-\frac{1}{\psi}} + \beta \left(\hat{y}_{t+1}\right)^{\frac{1-\frac{1}{\psi}}{1-\gamma}} .])$$

All terms are fully defined:

- $(C_t = c_t W_t)$ is consumption
- $(\hat{y}_{t+1})$ comes from the critic applied to next state
- exponents come from EZ preference structure

8.2 Value loss (z-head)

$$([L_{\text{value}} = \frac{1}{2}\left(\hat{z}_t - T_t^{(z)}\right)^2 .])$$

8.3 TD residual in (z)-space and GAE

Define the **combined reward**:

$$([r_t = r_t^{\text{ext}} + r_t^{\text{int}}])$$

and the **EZ temporal-difference residual**:

$$([\delta_t^{\text{EZ}} := r_t + \beta\left(T_t^{(z)} - r_t^{\text{ext}}\right) - \hat{z}_t .])$$

This matches the structure of the general EZ TD residual while incorporating intrinsic reward.

Generalized Advantage Estimation (GAE)

Let $(\lambda \in [0,1])$ be the GAE parameter.
Compute the advantages by backward recursion:

$$([\tilde{A}_t = \delta_t^{\text{EZ}} + (\beta\lambda), \delta_{t+1}^{\text{EZ}} + (\beta\lambda)^2, \delta_{t+2}^{\text{EZ}} + \cdots])$$

Practical implementation uses backward iteration over a rollout.
We normalize $(\tilde{A}_t)$ to mean 0 and variance 1 in each minibatch.

8.4 PPO clipped policy loss (consumption-only)

Let:

- $(\log\pi_{\theta_{\text{old}}}(c_t \mid s_t))$ be the stored behavior log-prob.
- $(\log\pi_{\theta}(c_t \mid s_t))$ be recomputed with the current actor.
- Importance ratio:

$$([r_t(\theta) := \exp\left(\log\pi_\theta(c_t \mid s_t) - \log\pi_{\theta_{\text{old}}}(c_t \mid s_t)\right) .])$$

- Clipping parameter $(\varepsilon \in (0,1))$.

The PPO objective (to **minimize**) is:

$$([L_{\text{PPO}} = -E_t\left[\ \min\left(r_t(\theta)\tilde{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\tilde{A}_t\right)\right] .])$$

8.5 Entropy term (Gaussian action only)

The actor samples

$$([y_c \sim N(\mu_c(s_t), \sigma_c(s_t)^2)])$$

before applying the sigmoid.

Entropy of a Normal:

$$([H_c = \frac{1}{2}\log\left(2\pi e \sigma_c^2\right).])$$

We encourage exploration by adding the entropy term:

$$([L_{\text{ent}} := -H_c.])$$

There is no Dirichlet entropy here since we removed risky-weight allocations.

8.6 ICM loss

As defined in §7.2:

$$([L_{\text{ICM}} = L_{\text{fwd}} = \left|\phi_{t+1} - \hat{\phi}_{t+1}\right|_2^2.])$$

The inverse loss is zero in consumption-only and can be enabled later if desired.

8.7 Final training loss

Define scalar weighting hyperparameters:

- $(c_v > 0)$: value loss weight
- $(\beta_{\text{ent}} > 0)$: entropy loss weight
- $(c_{\text{icm}} > 0)$: curiosity loss weight

The full objective is:

$$([L_{\text{total}} = L_{\text{PPO}} + c_v L_{\text{value}} + \beta_{\text{ent}} L_{\text{ent}} + c_{\text{icm}} L_{\text{ICM}}.])$$

---

9) TRAINING PROCEDURE (COLLECT → TARGETS → PPO)

9.1 Hyperparameters (additions/changes)

- **EZ:** choose $(\gamma \in 5, 10)$, $(\psi \in 0.5, 1.0, 1.5)$, $(\beta \in [0.95, 0.999])$.
- **FracDiff:** (d_{target}) init (0.3) – (0.5) within $([0, 1])$, tolerance (10^{-4}), $(K_{\max} \in [1024, 4096])$ (match horizon).

- **RL:** keep PPO (λ), clip, epochs, minibatch, lrs same initially.

9.2 Rollout collection (unchanged mechanics)

- Collect tuples ($\left(s_t, a_t = (c_t, w_t), r_t, s_{t+1}, \log\pi_{\theta_{\text{old}}} \right)$) where (r_t) includes EZ flow + curiosity.
- Align time by dropping first (K) steps due to FracDiff.

9.3 Target building & PPO update

- For each step, compute ($T_t^{(z)}$), (δ_t^{EZ}), GAE, and (z)-value loss.
- Recompute current ($\log\pi_\theta$) exactly (§5.2); perform clipped PPO with entropy and ICM losses.
- After epochs, set ($\theta_{\text{old}} \leftarrow \theta$).

9.4 Evaluation (deterministic)

- Use ($c_t := \sigma(\mu_c)$), ($w_t := \alpha \ / \ \sum_i \alpha_i$).
- Recover EZ value via inverse transform for reporting: ($\hat{V}_t = \hat{z}_t^{1/(1-\frac{1}{\psi})}$).
- Report PnL, CAGR, MDD, Calmar, turnover, and ($\hat{V}_0$).

---

10) DIAGNOSTICS, CHECKS, & ABALATIONS

- **EZ sanity:** as ($\psi \to 1$) or ($\gamma \to 1$), curves and training behavior should smoothly approach separable/CRRA.
- **Scale hygiene:** track ($\hat{z}_t, \hat{y}_t$) magnitudes; clamp/normalize if exploding.
- **FracDiff:** monitor learned (d_{target}) trajectory; inspect ACF/PACF of FD outputs; avoid non-stationary drift.
- **Alignment:** verify all post-FD tensors drop the first (K) steps; shapes of policy/value/ICM batches match.
- **Ablations:** (i) turn off FracDiff (identity) to test EZ alone; (ii) (ψ) grid with fixed (γ); (iii) compare CRRA vs EZ at matched (γ) with ($\psi \approx 1$).

---

11) MINIMAL MIGRATION CHECKLIST

- [ ] Expose (γ, ψ, β) in config; leave PPO hypers unchanged initially.
- [ ] Critic: switch to **two heads** (($\hat{z}$, $\hat{y}$)); keep shared backbone.
- [ ] Reward pipe: compute ($r_t^{\text{ext}} = (1-\beta)C^{1-1/\psi}$); add curiosity as before → (r_t).
- [ ] Targets: build ($T^{(z)}$) with next-state ($\hat{y}$); compute (δ^{EZ}) and GAE in (z)-space.
- [ ] Insert **Learnable FracDiff** before feature builder; shift by (K).
- [ ] Log (d_{target}, K), ACF diagnostics, and ($\hat{z}$, $\hat{y}$) summaries.

Model Evaluation

Hypotheses and experimental setup

We evaluate this set of hypotheses.

1. Performance hypothesis
   We train the agent with a fixed set of hyperparameters and measure
   - average external reward per episode,
   - the sequence of realized gross returns under the learned policy,
   - simple risk measures such as mean, standard deviation, and Sharpe ratio of per step rewards.
2. Path dependence hypothesis
   We keep data, architecture, and hyperparameters fixed, and only let the random seed that controls network initialization and the order of environment interactions vary. For each seed, we record
   - per episode average external reward,
   - per episode average curiosity reward,
   - terminal wealth,
   - summary statistics of the consumption fraction c_t.

Metrics

Since this is an RL problem rather than a supervised prediction task, classical accuracy does not apply. Instead we print:
```
(venv) james@MacBook-Pro-502 EZ_Optimization % python -m src.eval
Non-finite values detected in train split during cleaning
  keeping 363 rows out of 391
Warning: no fully finite rows in test split.
  Falling back to column-wise imputation instead of dropping everything.
  non-finite feature counts per column (test): [ 0  0  0  0  0 98 98]
  non-finite returns in test: 0
/Users/james/Desktop/GitHub/EZ_Optimization/src/train.py:68: RuntimeWarning: Mean of empty slice
  col_means = np.nanmean(
wealth: [ 10000.         10086.27717377  10001.62868886   9854.72473591
  10387.37248522  10513.40313031  10462.42804041  10336.92568461
   9977.97571284  10402.11748941  11078.88449752  11578.0997685
  11605.48823183  12435.14442234  13091.63060983  12906.39579346
  13554.66160685  13745.89725453  12939.23576564  13860.40002845
  12676.17405194  13369.91935279  13657.0544277   13270.7070526
  15019.33216571  15573.88510977  15984.86141853  15801.81525975
  15897.15154576  16658.8390342   17462.17299272  18072.6122216
```

17635.84882459	13797.73843349	12620.10478097	13539.61016067
14087.03249909	14676.10127167	14186.73809018	14320.42932773
14365.89797861	14987.29491194	14906.17796054	14330.78386115
14900.12178577	15286.98068883	14998.18151383	15218.45265848
16300.69324797	15828.86795782	16158.18651299	16967.53238609
17563.71207087	17424.51001361	18964.2985533	19258.54372537
19132.48991185	18650.80217516	18959.29288266	19365.32058513
18032.63429182	18186.59822321	18627.6962139	18126.83300567
19794.28534547	19618.36703128	19515.87774311	17675.23532837
16770.52407017	16658.1752358	17656.5529593	18094.90679508
18846.14748553	20114.11741338	20560.68757217	20567.24310487
21361.2196443	20338.15431521	21250.4882297	21668.01294992
21253.1844527	21504.67873944	20560.52303304	22854.80539714
22399.42964669	22614.20754752	22120.4749859	22737.45221525
22759.34668441	22364.24847365	23244.78701315	22686.94649819
22893.501029	22941.69887309	23635.93130748	23874.81365112
24043.01116948	24295.0447285	24749.27079118	24120.19961699
24668.12500213	24686.72945765	24555.19756105	25400.65511091
25146.92948248	25634.57742712	25303.59423341	25558.91193946
26389.5692003	25596.66971223	24425.68710088	24707.30614309
25013.58180563	24343.42365201	25109.96927917	26054.04615858
25353.75259377	25881.94298792	24859.46264311	25165.18552093
25776.112244	26705.93717473	27435.76180988	28202.84575016
29226.91108453	29848.7860795	30797.23241587	30787.33863754
32021.80115863	31862.32274406	33170.20617488	33748.79003098
34849.55062154	35091.15148024	35368.91765818	35843.93695106
36663.05520199	36745.75784491	35064.67802019	35724.34801566
37660.68690266	38643.6295901	41479.10998297	40587.04390364
43075.68009836	43330.97056256	41484.42382849	43907.30415116
46479.21325816	48498.80837444	52288.73725389	49283.8836365
51903.44924943	50113.8870267	52348.25739233	53171.72465863
53711.37427674	57495.24190557	60366.82233742	60914.67944268
59767.82491867	62124.90107618	61403.23807656	52450.79514929
55723.43682573	60197.64827572	63756.8318433	67351.07279063
70113.03152103	67849.53336045	70481.62852435	73155.91581005
71329.10674382	75212.07059415	72801.74578852	72346.36572126
70280.67525015	74675.92430959	76099.30602931	80501.10796971
76403.23868894	74866.83957147	82107.86778267	79579.21458761
77835.15204453	79697.94228164	78395.4983712	83153.95182492
78706.55449268	78316.91728948	72046.12023929	72338.08493833
74843.55661601	67936.12002699	63574.23873357	68457.58811065
68805.98462952	67083.33348401	66362.78132302	62108.30540472
57032.54698247	58064.72495472	62429.72940082	62902.49946742
61922.80287359	60636.84101217	62864.50732005	59003.45868398

```
  58467.56131603  54231.21966566  49946.67083708  50190.42942602
  44668.21623716  48529.68089198  51299.20194392  48204.13969823
  46882.59177388  46085.37304652  46470.48622745  50236.59369543
  52793.51676407  53391.20529081  54257.31112332  55227.00593274
  54567.30047933  57566.34048502  57976.642848    60919.79602399
  61972.20919882  62728.76136692  61702.52537782  60666.4331788
  61399.42923846  62503.88348752  60360.53417681  60498.54163058
  61064.98223602  61920.69988103  64310.46257781  66397.79714566
  64718.4996557   65941.831761    64681.11399763  63380.40522374
  65278.71296405  65269.33284534  67616.89241318  66858.0203972
  67322.55394554  66128.13278973  68454.85227513  68389.59001603
  70131.18387774  70162.89117177  70941.33098978  71803.59732131
  69583.58099974  69589.53358686  69943.38389709  71431.30263448
  73186.03174132  75491.89946156  76734.9189657   77702.91364602
  78795.26530204  77073.81305792  77842.92904083  81212.72538557
  83856.05580954  82361.96836763  79727.79760336  80753.29671449
  83643.69816407  84883.41060881  81144.77361133  80444.53649486
  75524.19756018  72898.81364901  72464.29122695  75909.65644644
  76719.85748147  70124.76314891  69433.30681041  70279.66056048
  63898.80340031  53072.72534824  49100.23532344  49484.22627115
  44552.83953646  39655.05140685  43041.72880834  47084.38120037
  48812.07737891  48821.53272876  52441.1937125   54200.71818619
  55791.82980812  54689.21811745  57826.35840593  58701.71398524
  56531.1841286   58143.03751263  61561.57757858  62470.11369334
  57349.01769367  54258.85957318  57990.60655311  54948.95549123
  59759.73815956  61962.17728532  61820.20665849  65819.12741891
  67309.57325998  69460.48577562  69387.66950332  71364.82968328
  70401.26779194  69115.85181417  67631.56128123  63790.62492301
  59212.82506912  65591.34249439  65259.46992971  65816.24767298
  68684.65084503  71472.45169668  73711.77102469  73159.04903336
  68575.51188741  71287.9417885   72185.92378938  73612.48588888
  75396.4975877   73904.37240605  74114.68318627  74638.47161418
  78402.27340973  79269.36997372  82122.01291557  83607.16488823
  85342.99848614  84062.826622    88220.66065681  85459.44131268
  88001.73004317  91926.13848724  94504.52335387  96731.21676242
  93289.14625567  97311.40138781  97985.87960113  98593.37090975
 100666.71908319 102585.15773624 101038.08729997 104842.60735617
 103215.99298351 105610.64968402 108201.55402424 107748.23354943
 104403.53910347 110134.40441304 108218.38791732]
Deterministic evaluation on train split:
  split: train
  initial_wealth: 10000.0
  final_wealth: 108218.3879173154
  pnl: 9.82183879173154
```

```
  cagr: 0.0819115834643418
  max_drawdown: -0.5328291933320188
  calmar: 0.15372953375942502
  V0_hat: 7.40093871508181
  n_steps: 363
wealth: [10000.         10193.01944908 10419.17364791 10711.76051764
 10817.06416238 11424.74122113 10979.76691689 10684.57073022
 10713.60929446 10945.10179725 10998.09136612 11394.2487124
 11739.06323575 11789.46222658 10971.22307796 11167.15071083
 10142.25401136 10940.28074156 11265.51187292 11467.42691882
 11918.24031674 11134.27451968 11901.74982044 12057.98585539
 11839.82552937 12043.23540001 12289.28756958 12707.68832867
 13070.98609353 13049.69215997 11952.06409598 10456.62011789
 11782.96836429 12316.50968576 12542.97793298 13234.09878084
 14161.32730242 13605.79368033 13229.36536721 14652.11093954
 15196.00283636 15026.75522881 15418.80973907 16073.14677025
 16915.7688383  17008.55999825 17386.37170018 17781.86106081
 18297.34581347 17426.93909475 18631.8868459  18476.59613036
 19282.3958405  18268.40922777 17695.48495659 18328.49211808
 16716.36123238 16717.23327965 15314.30771533 16709.6760277
 16000.49943147 14506.10787443 15664.59972712 16506.60087018
 15533.16690755 16492.36875683 16061.69576786 16624.66702548
 16868.07448072 16909.92910833 18004.4472817  18565.06734502
 18236.14280539 17347.67249712 16966.359074   18479.38823727
 19296.69877436 19603.41417448 20617.29427673 21256.79538644
 20372.17200938 21350.44675079 22090.63945846 22340.71201182
 22850.83169315 23312.32467871 23081.58522618 24404.16526642
 23794.27793156 24437.08641898 24089.02659264 22702.80788246
 22529.67816392 23915.76555178 25102.1250501  25645.9752691
 26134.92691574 27058.08319614]
Deterministic evaluation on test split:
  split: test
  initial_wealth: 10000.0
  final_wealth: 27058.0831961391
  pnl: 1.7058083196139102
  cagr: 0.12962509011033352
  max_drawdown: -0.24770199748896937
  calmar: 0.5233106370735099
  V0_hat: 0.9194045989322054
  n_steps: 98
```

Overfitting, underfitting, and complexity

To study model complexity we vary

- the PPO learning rate,
- the curiosity weight,
- the number of PPO epochs per update.

If the model underfits, all seeds produce low reward and the consumption policy reacts weakly to the state. If the model overfits or becomes numerically unstable, we observe high variance value estimates, occasional NaNs, and poor out of sample behavior.

Empirically we find that

- Very large curiosity weight pushes the agent toward chasing intrinsic reward at the expense of external utility.
- Very small curiosity weight reduces exploration and makes it more likely that the agent stagnates.
- Large learning rates tend to create exploding gradients and numerical issues.

Path dependence results

When we train multiple agents with different random seeds, the most striking pattern is qualitative path dependence.

- For some seeds, the agent's early episodes include sequences of months where market returns and curiosity bonuses are aligned. The critic receives coherent value targets, and PPO updates gradually increase nontrivial consumption and investment behavior. These runs show improving external reward and relatively stable critic outputs.
- For other seeds, early episodes are unlucky. Either the market returns are unfavorable, or the curiosity module generates large bonuses on transitions that do not coincide with good consumption events. In these runs the critic's value targets become noisy and occasionally large in magnitude. PPO updates then shrink the consumption fraction c_t toward very small values, which effectively freezes wealth dynamics and stops exploration. Once this happens, later episodes rarely recover, even though the same architecture and hyperparameters are used.

These observations support the gateway hypothesis: the first few episodes act as a gate that determines which basin of attraction the learning trajectory enters.

Class imbalance and normalization

There is no class label, so class imbalance does not apply in a strict sense. However, there is an imbalance between many small return months and a few extreme shocks. Expanding normalization and clipping of advantages help prevent these rare events from dominating the gradient updates.

We experimented with removing normalization and found that training quickly became unstable, with value estimates diverging. With expanding normalization in place, the training dynamics are stable enough to study path dependence itself rather than numerical artifacts.

Model Deployment

Possible deployment scenario

In a real application, a model of this type could be deployed as an automated decision rule or a decision support tool for consumption and investment. A hypothetical deployment pipeline would be

1. Continuously collect new monthly S&P 500 and macro data.

2. Update the feature pipeline and recompute expanding normalized features using a rolling window so that statistics remain current.

3. Periodically retrain or fine tune the RL agent on the most recent history.

4. At each decision date, feed the current state into the policy network and implement the recommended consumption and investment decisions subject to human risk constraints.

Risks and limitations

Several limitations make immediate deployment inappropriate.

- The model is trained and evaluated on one historical sample of one index with no transaction costs.

- The strong path dependence we observe means performance is sensitive to initialization and early data segments.

- Curiosity driven exploration can suggest actions that are useful for learning but unacceptable from a risk management perspective.

- Automated financial decision systems can affect market liquidity and volatility, so any real world deployment would require careful regulation and oversight.

For the DATA110 project we restrict ourselves to offline simulations and do not connect the model to any live trading system.

Project Meetings

2025/11/10, Zoom

2025/11/17, Davis Library Room 301

2025/12/03, Morrison Lounge

2025/12/04, Facetime, full group, final review of results

~2025/12/04, individual coding session(s) recorded in GitHub