Reinforcement Learning for Trading in a Synthetic Multi-Good Market

Mathieu Schneider

05.01.2026

Abstract

This paper studies whether a reinforcement learning agent can learn a stable and feasible trading strategy in a stochastic multi-good market with endogenous price effects. We develop a fully synthetic market environment featuring mean-reverting log-price dynamics, persistent and cross-good price impact, correlated shocks, and trading feasibility constraints. A single agent interacts with this environment under partial observability and inventory and budget constraints, and is trained using Proximal Policy Optimization. Results show rapid convergence toward a stable trading behavior across training horizons and hyperparameter settings, with consistent wealth outcomes and declining constraint violations. The framework provides a controlled testbed for analyzing learning dynamics in market-like environments.

1 Introduction

1.1 Motivation and Research Question

Decision-making in stochastic markets with multiple assets and endogenous price effects is a complex problem, particularly when prices are influenced by the agent's own actions and when the underlying dynamics are only partially observable. In such settings, reinforcement learning provides a natural framework, as it allows agents to learn effective strategies directly through interaction with the environment rather than relying on historical data.

This paper investigates whether a reinforcement learning agent can learn a stable and feasible trading

1.2 Contribution

We propose a fully synthetic market environment that combines mean-reverting price dynamics, endogenous price impact, cross-good interactions, and correlated shocks. This environment is embedded in a reinforcement learning framework using Proximal Policy Optimization (PPO). We analyze learning dynamics, convergence behavior, and robustness across training horizons and hyperparameter configurations.

1.3 Application Context: Resource Trading in a Colonization Game

The market environment was originally developed as part of a simulation game centered on planetary colonization and resource exploitation. In this setting, players extract and accumulate multiple resources and must decide when and how to sell them to a market under price uncertainty and liquidity constraints.

The reinforcement learning model is designed to support this gameplay by providing an automated trading policy that operates under the same constraints as the player. The objective is to assess whether such a model can generate coherent and stable trading behavior suitable for integration into a game environment.

1.4 Scope of the Paper

While motivated by a game application, the focus of this paper is on the design and evaluation of the market environment and the learning behavior of the agent. Gameplay mechanics and player experience are not analyzed. The remainder of the paper presents the market model, the reinforcement learning methodology, and the empirical results of the training experiments.

2 Data Construction

2.1 Overview

As previously discussed, our objective was to develop an automatic response system to commodity price movements on a trade hub using a machine learning model. However, we did not have access to sufficiently rich historical data to apply more traditional predictive approaches. For this reason, we chose to rely on a reinforcement learning framework.

Rather than learning from historical observations to forecast future prices, the model is trained by interacting with a synthetic environment generated by a custom price process. The agent learns autonomously through trial and error, gradually identifying strategies that perform well within this simulated market. In this section, we describe in detail the multi-good market process that generates the data used by the model

2.2 Price Formation Process

The price dynamics are generated by a mathematical process expressed in log-price terms. Each good is indexed by i, and throughout this study we focus on an economy composed of four goods: food, metal, industrial goods, and consumption goods. Conceptually, this formulation produces price trajectories subject to both stochastic and deterministic forces, while ensuring a tendency for prices to revert toward a long-run mean.

$$x_{t+1} = \mu + R(x_t - \mu) + Au_t + Bs_t - Cs_t + \varepsilon_{t+1} \tag{1}$$

The long-run mean of the log-price for each good is denoted by μ.The speed of mean reversion is controlled by a parameter R. Values close to one imply slow adjustment, allowing persistent deviations from the mean over time. The immediate effect of the agent's actions on the market

is captured through the variable

$$u_t = \frac{Q_t}{V},$$

where Q_t denotes the quantity bought or sold and V represents market size. Its influence on prices is governed by a linear coefficient A.Market size V is interpreted as the typical trading volume of each good per period. Delayed effects of past actions are modeled through the state variable

$$s_t = \lambda s_{t-1} + u_{t-1},$$

where λ controls persistence and coefficient B determines how this delayed impact affects prices. Cross-good effects of past actions are captured through the matrix C, allowing trades in one good to influence prices of other goods. This mechanism introduces interdependencies similar to supply-chain effects. Finally, ε_{t+1} denotes a stochastic shock with zero mean. Shocks differ across goods but are correlated through a covariance matrix Σ, allowing for joint movements in prices.

2.3 Justification of the Price Model

Our first motivation was to construct a sufficiently complex environment to meaningfully challenge the learning algorithm while remaining interpretable. Rather than developing a fully specified microeconomic model, we opted for a more econometric-style formulation that is easier to control, modify, and replicate, while still producing realistic price dynamics.

Modeling prices in log-space simplifies the design of the process and ensures positivity of prices. Importantly, although the underlying dynamics are linear in log-prices, the agent only observes prices in levels. As a result, the learning problem faced by the agent is inherently non-linear.

Finally, the inclusion of delayed effects, slow mean reversion, stochastic shocks, cross-good interactions, and endogenous price impact reflects key features of real markets. Large market participants influence prices through their actions, and prices across goods are interconnected through consumption and production relationships. These aspects are captured through the endogenous impact terms and the covariance structure of shocks.

2.4 Data Available to the Model

Although the price process depends on multiple latent time series variables, these quantities are not directly observable by the agent. The model only observes current and past prices in levels, not log-prices, as well as its own past actions.

In addition, the agent manages an inventory system: it can buy goods, hold them over time, and sell them later. Inventory levels also grow stochastically, representing exogenous inflows such as production or deliveries. The agent operates under a cash constraint, which limits its ability to purchase goods.

In summary, the agent's observable state consists solely of past and current prices, its own actions, inventory levels, and available cash, without access to the underlying parameters governing the price process.

2.5 Data Logging for Training and Evaluation

While training and evaluation are discussed in detail in the following section, it is important to note that every interaction between the agent and the environment is recorded. At each time step, the agent's actions and all relevant state variables are stored.

This comprehensive logging allows us to reconstruct the full history of each simulation and to analyze the behavior of prices, portfolios, rewards, and constraints over time. These recorded data form the basis for the evaluation of learning dynamics and for comparing different model configurations.

3 Methodology

3.1 Problem Formulation as a Markov Decision Process

3.1 Problem Formulation as a Markov Decision Process We formalize the trading problem as a finite-horizon Markov Decision Process (MDP) in which a single strategic agent interacts with a stochastic, multi-good market environment. Time is discrete, indexed by t=0,1,...,T, where T denotes the fixed episode length.

At each time step, the agent observes the current market and portfolio state, selects trading actions across multiple goods, and receives a reward based on the resulting change in wealth. Market prices evolve according to an exogenous stochastic process with mean reversion, persistent impact, cross-good interactions, and correlated shocks, as described in Section 2. The agent does not observe the underlying market parameters and cannot directly control price dynamics, except through the indirect impact of its trades.

Episodes have a fixed horizon T, after which the environment resets to initial conditions while preserving the learned policy. Training consists of repeated episodes under identical market specifications, allowing the agent to improve its behavior through experience rather than through changes in the data-generating process. This episodic structure enables a clear separation between learning dynamics and market dynamics and facilitates systematic evaluation of policy performance across training phases.

3.2 State Representation

At each decision time t, the agent observes a fixed-dimensional state vector summarizing recent market conditions and its own portfolio position. The state is constructed using a rolling observation window of length W, allowing the agent to condition its decisions on short-term dynamics while preserving the Markov property.

Formally, the observed state s given by:

$$s_t = (P_{t-W+1:t}, Q_{t-W:t-1}, I_{t-W+1:t}, C_{t-W+1:t}, t)$$

Where P is the price vector, Q is the vector of executed trades, I is the inventory holding and C is the cash holding

Prices, inventory, and cash are observed contemporaneously, while executed trades enter the state with a one-period lag. This reflects the economic assumption that trades influence future

prices through market impact, rather than current execution prices as they are already realized.

Before feeding the model, we windowed it. Meaning that we feed our model only with the past 20 periods data points.

By using a fixed observation window instead of recurrent architecture, the model maintains a fully feed-forward policy while still providing the agent with limited memory of past states and actions. This design choice simplifies training and improves stability without sacrificing essential temporal information.

3.3 Action Space and Trade Execution

At each time step t, the agent selects a vector of continuous actions representing intended trades across all goods. The action space is defined as a bounded, continuous space:

$$\mathcal{A} = [-100, 100]^N,$$

where each component $a_{t,i}$ corresponds to the trading intensity for good i.

Action Normalization and Scaling Actions are unitless and do not directly represent quantities traded. Instead, they are internally mapped to desired trade volumes using market-specific depth parameters:

$$Q_{t,i}^{\text{desired}} = \frac{a_{t,i}}{100} \cdot V_i,$$

where V_i denotes the market depth (or typical trading volume) of good i.

This normalization ensures that actions have a consistent interpretation across goods, that the policy output scale is independent of absolute market size, and that learning remains stable when goods differ significantly in liquidity.

Feasibility Constraints After the agent selects its actions, but before they are implemented, trades pass through feasibility constraints consisting of a budget constraint and inventory constraints.

For buy orders ($Q_{t,i}^{\text{desired}} > 0$), the total purchase cost is computed using current market prices. If the total cost exceeds available cash, all buy orders are scaled down proportionally so that the budget constraint is exactly satisfied.

For sell orders ($Q_{t,i}^{\text{desired}} < 0$), execution is limited by current inventory holdings. The agent cannot sell more units of a good than it holds, thereby ruling out short positions.

These constraints are applied deterministically and do not reject actions outright. Instead, infeasible components are smoothly adjusted to yield a feasible execution vector Q_t^{exec}.

3.4 Reward Function

The agent's objective is to maximize total portfolio wealth over the course of an episode. Accordingly, the reward function is defined as the net change in wealth between consecutive time steps, adjusted for feasibility penalties.

$$r_t = W_{t+1} - W_t - \pi_t^{\text{budget}} - \pi_t^{\text{resource}}$$

where pi "budget" and pi "resource" are non-negative penalty terms associated with budget and inventory feasibility constraints, respectively. They are computed when the feasibility constraints are. Their value is just the difference between the contraints the intended action of the agent. The goal is to make it learn to manage its cash and its inventory correctly.

3.5 Learning Algorithm and Training Protocol

The agent is trained using Proximal Policy Optimization (PPO), an on-policy actor–critic algorithm well suited to continuous action spaces and stochastic environments. PPO is chosen for its robustness to noisy reward signals and its ability to maintain stable learning dynamics in the presence of delayed and indirect action effects, such as market impact.

Both the policy and value function are parameterized with with identical architectures, each consisting of two hidden layers of 256 units with Tanh activation functions. The policy outputs a stochastic Gaussian distribution over actions, allowing for controlled exploration during training.

Training alternates between rollout collection and policy optimization. During each rollout phase, the agent interacts with the environment for a fixed number of time steps, after which the policy is updated over multiple optimization epochs using mini-batch stochastic gradient descent. PPO constrains policy updates through a clipped objective function, while generalized advantage estimation, entropy regularization, and gradient norm clipping are employed to improve training stability. All hyperparameters are held fixed within each experimental run and varied only across runs in a controlled manner.

3.6 Experimental Design and Evaluation Strategy

Training is conducted over a fixed number of environment interactions, corresponding to multiple episodes of equal length. Episodes are independent in terms of environment state, while the policy is continuously updated across episodes, allowing learning to accumulate under stationary market conditions.

To assess learning dynamics and robustness, multiple experimental runs are performed under identical market specifications. Experiments differ only in total training horizon or selected PPO hyperparameters. Baseline models trained for approximately 40,000, 60,000, and 120,000 environment steps are compared to variants that modify a single optimization parameter at a time, including the learning rate, PPO clipping range, or entropy coefficient.

Throughout training, all environment interactions are logged in detail. These logs are aggregated along three dimensions: within-episode time profiles, episode-level averages, and final-step outcomes. For cross-model comparison, performance is evaluated using the latest learning phase with sufficient data coverage, ensuring that comparisons reflect mature policies rather than transient learning behavior.

4 Results

4.1 Final Outcomes Across Models

Figure X reports final portfolio outcomes (in wealth divided by the initial wealth) for all trained models, evaluated at the last learning phase with sufficient data coverage. Final total wealth is very similar across all configurations, including models trained with different horizons (40k, 60k, 120k steps) and models differing by PPO hyperparameters.

By contrast, final cash holdings differ more substantially across models. These differences are not mirrored in total wealth, indicating that they are compensated by opposite inventory positions valued at prevailing market prices. Final outcomes therefore mainly differ in terms of portfolio composition rather than overall economic performance.

4.2 Within-Episode Portfolio Dynamics

To examine how portfolios evolve during an episode, Figures Y and Z display time-in-episode trajectories for wealth and cash. For each model, all learning phases are shown, with early and late phases highlighted.

Across models and learning phases, wealth trajectories exhibit similar shapes and dispersion. Cash trajectories display more variability but follow comparable patterns across configurations. No qualitative change in within-episode dynamics is observed as training progresses, suggesting that the model converge fast to an optimal strategy.

4.3 Learning Evolution Across Models

Figure A shows episode-mean rewards as a function of the learning phase for all models. The graphs seem strange because we don't see all the models. But they are all there, it's just that the mean of the rewards across all the episodes during the same learning phase is the same with all models. Once again it suggests that all the models we have converge to the same strategy really fast. The change in color at learning phase = 300 is because one of the variant of the model had the double amount of data, thus all the other models stops at 300.

The same pattern is observed for episode-mean budget penalties and resource penalties (Figures B and C). All models converge toward similar levels over learning phases, indicating that differences in training duration or hyperparameters do not lead to distinct long-run behaviors in terms of rewards or constraint usage.

4.4 Constraint Satisfaction Over Training

Time-in-episode plots of penalties provide additional insight into learning dynamics. Early learning phases are characterized by frequent and sometimes large penalty realizations, reflecting violations of budget and inventory feasibility constraints.

As training progresses, penalty magnitudes and frequencies decrease substantially. In later learning phases, penalties are close to zero for most time steps within episodes, indicating that the agent increasingly selects actions that are feasible given its cash and inventory constraints.

5 Discussion, Limitations, and Conclusion

5.1 Discussion

The results indicate that the proposed reinforcement learning framework is able to learn a stable trading strategy in a stochastic, multi-good price environment. Across all experimental configurations, the agent rapidly converges toward similar behaviors, both in terms of portfolio dynamics and long-run outcomes.

Although final wealth levels do not differ substantially across models, learning is reflected in the agent's ability to consistently operate within feasibility constraints and maintain stable portfolio trajectories. This suggests that the model effectively internalizes the structure of the price process and the trading constraints, allowing it to generate wealth in a systematic and repeatable manner within the simulated market.

5.2 Limitations

Several limitations should be acknowledged. First, the price formation process, while deliberately designed to be interpretable and controllable, may remain too simple to fully capture the complexity of real-world markets. The presence of a dominant strategy could partly stem from this controlled structure.

Second, the exploration of model configurations is limited. Only a small subset of PPO hyperparameters and training horizons were varied, and alternative optimization settings may lead to different learning dynamics.

Finally, the agent benefits from stochastic inventory inflows, which act as an exogenous source of tradable assets. It remains an open question whether the model would be able to generate comparable performance in the absence of such inventory shocks, relying solely on market timing and price movements.

5.3 Room for Future Work

Several extensions could be explored. Future work could test the robustness of the learned strategy under alternative price dynamics, including stronger non-linearities, regime shifts, or structural breaks. Exploring different reinforcement learning algorithms or network architectures, such as recurrent or attention-based models, may also provide insights into the role of memory and representation.

In addition, removing or modifying stochastic inventory inflows would allow for a clearer assessment of the agent's ability to generate wealth purely through trading decisions. Expanding the range of hyperparameters and training protocols would further improve understanding of convergence and robustness.

5.4 Final Conclusion

This study demonstrates that a reinforcement learning agent can successfully learn a stable and feasible trading strategy in a synthetic multi-good price environment. Despite the simplicity of the setting, the results highlight the ability of the model to rapidly converge toward consistent

behavior while respecting economic constraints. The framework provides a flexible foundation for further experimentation and offers a controlled setting in which to study learning dynamics in market-like environments.

ndix

Appendix

6 PPO Hyperparameters

Table 1 reports the hyperparameters used for the baseline PPO configuration. Unless stated otherwise, all experimental variants modify only a single parameter relative to this baseline.

Parameter	**Value**
Learning rate	1×10^{-4}
Discount factor (γ)	0.995
GAE parameter (λ)	0.95
PPO clip range	0.15
Entropy coefficient	0.02
Value function coefficient	0.7
Max gradient norm	0.3
Rollout length (n_{steps})	2000
Batch size	256
Optimization epochs	10
Policy architecture	$(256, 256)$ MLP
Activation function	Tanh

Table 1: Baseline PPO hyperparameters.

7 Figures

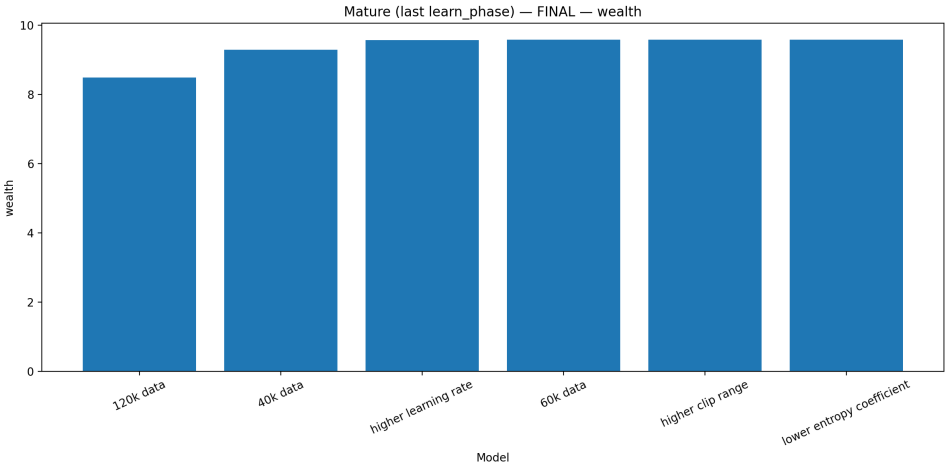

Figure 1: Final total wealth across models at the last learning phase.

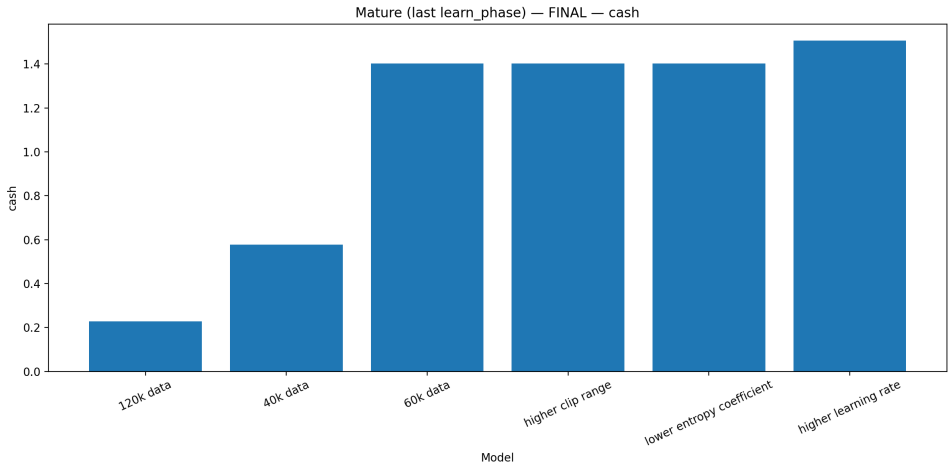

Figure 2: Final total wealth across models at the last learning phase.

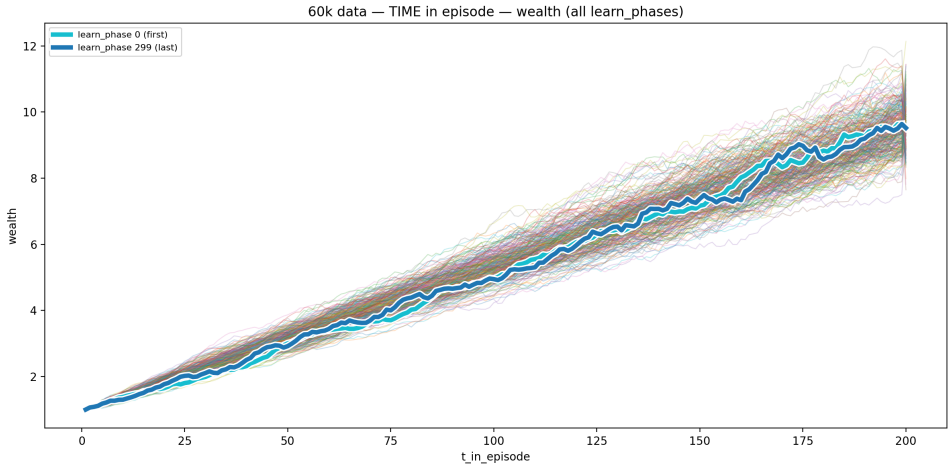

Figure 3: Final total wealth across models at the last learning phase.

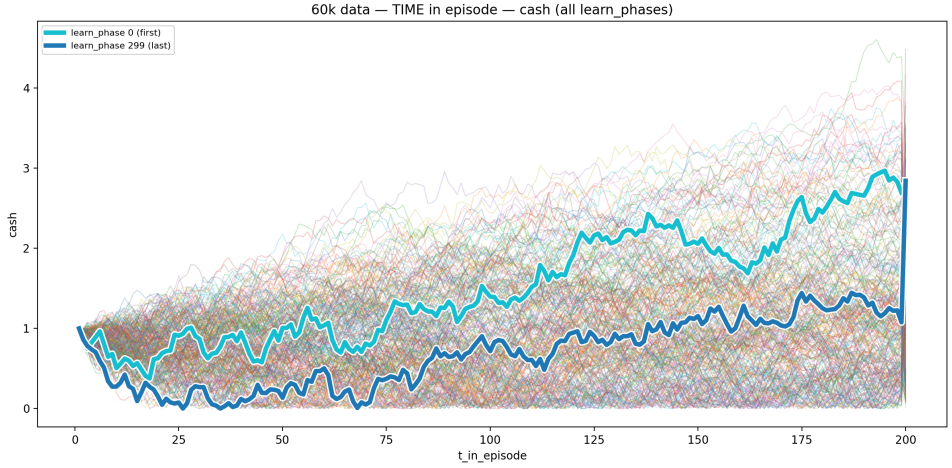

Figure 4: Final total wealth across models at the last learning phase.

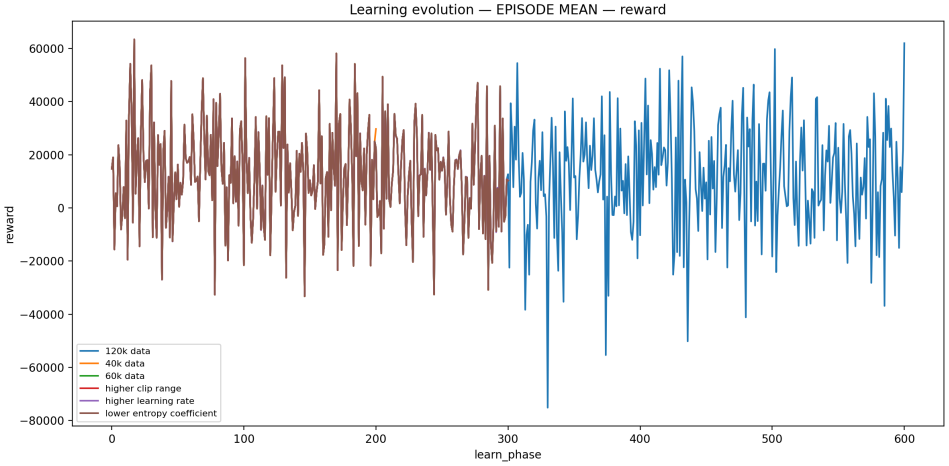

Figure 5: Final total wealth across models at the last learning phase.

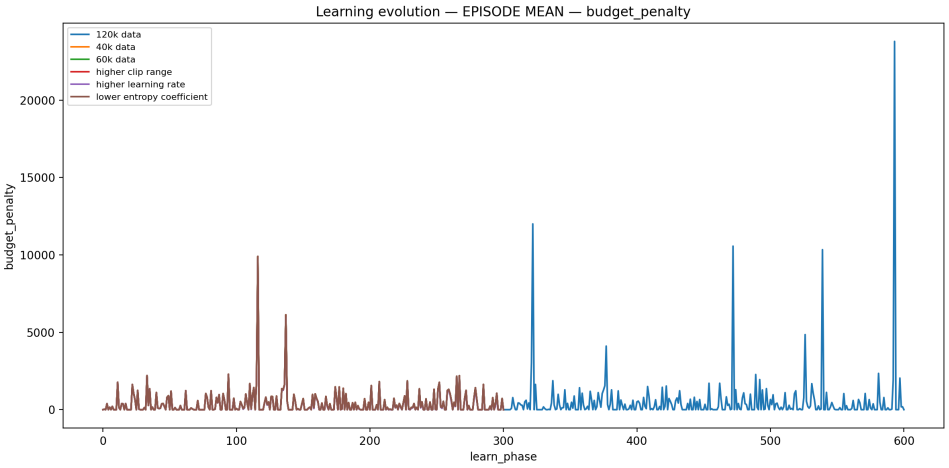

Figure 6: Final total wealth across models at the last learning phase.

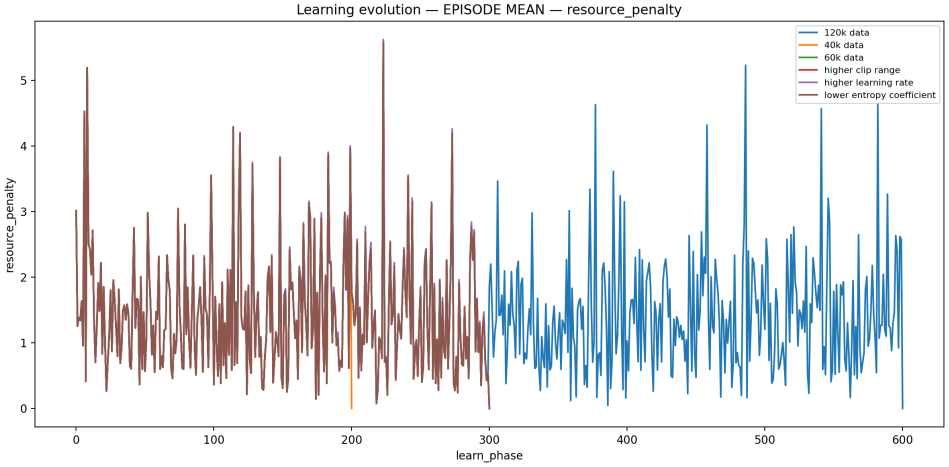

Figure 7: Final total wealth across models at the last learning phase.

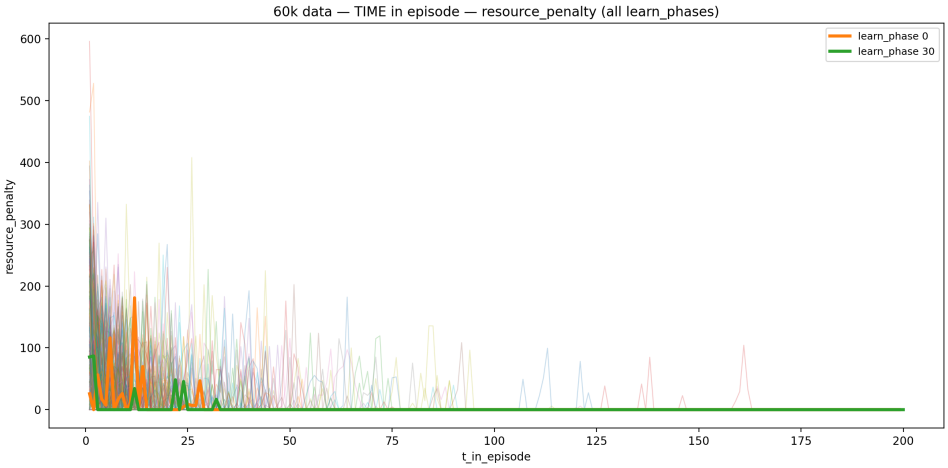

Figure 8: Final total wealth across models at the last learning phase.