CHAPTER TEMPLATE

Fine-tuning Timeseries Predictors Using Reinforcement Learning

Hugo Cazaux[a,b] , Ralph Rudd[a], Hlynur Stefánsson[a], Sverrir Ólafsson[a] and Eyjólfur Ingi Ásgeirsson[a]

[a]Reykjavik University, Department of Engineering, Menntavegur 1, Reykjavik, 102, Iceland
[b]Corresponding author, email: hugot@ru.is

ARTICLE HISTORY
Compiled January 28, 2026

ABSTRACT
This chapter presents three major reinforcement learning algorithms used for fine-tuning financial forecasters. We propose a clear implementation plan for backpropagating the loss of a reinforcement learning task to a model trained using supervised learning, and compare the performance before and after the fine-tuning. We find an increase in performance after fine-tuning, and transfer learning properties to the models, indicating the benefits of fine-tuning. We also highlight the tuning process and empirical results for future implementation by practitioners.

KEYWORDS
Fine-tuning, Proximal Policy Optimization, Reinforcement Learning, Attention

1. Introduction

Timeseries predictors are generally trained using supervised learning on datasets. The standard setup divides the dataset into three segments: training, validation and testing. The model is initially fit on training data, then evaluated on the validation set to tune hyper-parameters and assess the predictive power. Finally, the test set is used by the final model to determine the accuracy on unseen data. These steps are well understood and constitute the backbone of supervised learning in timeseries prediction.

This methodology draws a strong parallel with large language models (LLMs), which are generally transformer-based models and use supervised learning for pre-training. The pre-training is a common step to all LLMs that starts with a large amount of raw data, compressed in the network. Once the pre-training is complete, alignment aims to tune the model to create an user-friendly experience. This step incentivizes answering and asking questions to contextualize requests, teaches the LLMs how to use external tools, or censors potentially harmful information that might lie within the embeddings. The novelty of this research lies in the extension of alignment to time-series prediction model.

Fine-tuning in large language models was initially based on human feedback. A standard setup consists of a human operator prompting a question and grading the answer. Later, practitioners aimed at removing subjectivity from the fine-tuning pipeline by instead proposing two answers to a prompt and have a human operator select the best one. These two methods fall under the umbrella of Reinforcement Learning with Human Feedback (RLHF), and although effective recent models have shown pure Re-

inforcement Learning (RL) approaches outperforming RLHF for a fraction of the cost.

The central idea of this chapter is to leverage the predictive power of supervised pre-training and to use RL algorithms to align the model with diverse constraints. These constraints can be domain specific, such as risk management or operational constraints, but can also be purely mathematical, such as incentivizing bolder out-of-sample predictions. The reward function is at the center of the tuning, and will determine which direction the model is pushed towards. This approach is more adapted to time-series prediction compared to RLHF, as it completely removes the human operator and the need to reduce subjectivity in the feedback. RL is also extremely cost effective, since it removes the need of a coordinated effort of human operators giving feedback on a large number of samples.

In the context of time-series prediction, RL for fine-tuning is novel. The standard implementations of reinforcement learning in time-series prediction consist of a completely untrained agent learning a policy over a simulated environment. In the case of finance, this environment might be a portfolio or an ensemble of assets. In this chapter, we used a pre-trained model from [1] that serves as a backbone for the RL implementation. The environment is set up to reflect the training data closely, with the main tuning tool available being the reward structure. The loss is back-propagated through the backbone, updating the weights according to the policy.

The research questions investigated in this chapter are: Can we fine-tune pre-trained models to enhance time-series predictions using reinforcement learning? What state-of-the-art reinforcement algorithms work best for fine-tuning?

This chapter is structured as follow: Section 2 presents a literature review, section 3 the data used to train/test the models, Section 4 the framework used to fine-tune and evaluate the models, Section 5 benchmarks the models on standard reinforcement learning tasks, Section 6 the results of the fine-tuning, Section 7 the tuning of the specific hyperparameters and finally Section 8 is the conclusion to the chapter.

2. Background

Fine-tuning has become an emerging trend since large pre-trained model became more widely available to the public [2]. Fine-tuning is a technique that intends to specialize a pre-trained backbone model, often to increase performance on selected benchmarks [3] or to benefit from previously acquired knowledge through transfer learning [4]. The democratization of open source models with available weights in natural language processing [5], [6] and image processing [7] enabled researchers and enthusiasts to propose their own fine-tuned version of an advanced model without the high computational cost of pre-training. Fine-tuning was leveraged to propose fine art classification [8], fine-tuning large language models for better medical care [9], biomedical tasks in different languages [10], and malware detection in images [11].

As the size of models and the parameter number grow exponentially, fine-tuning the entire model for each downstream tasks was replaced with a sparser approach called parameter-efficient fine-tuning [12], [13]. Methods such as Adapter [14], [15], LoRA [16] and Prefix-tuning [17] propose to modify the architecture of the original model to benefit from higher order patterns learned during supervised learning while also specializing in a downstream task. Supervised fine-tuning uses labeled data after pre-training to align the model towards a downstream task. This method has grown in popularity as large language models hit the public sphere and adapted for more intuitive or safer usage [18], [19], [20].

As the cost of computation carried over to efficient data labeling [21], alternative techniques for fine-tuning were explored. Reinforcement learning, one of the major paradigms in machine learning, has become one of the prime candidate for efficient fine-tuning. Adversarial networks had previously shown promising results [22], and policy learning has been employed in text-to-image [23] and multi-modal models [24]. Perhaps the most impressive implementation of reinforcement learning based fine-tuning comes from the DeepSeek-v3 report [25], which implements group proximal policy optimization to fine-tune a pre-trained model and implement chain-of-thoughts reasoning.

Within time-series prediction, fine-tuning has been focused on domain adaption. In a similar fashion to text and image generation, large pre-trained models are becoming available to researchers [26]. The models can then be fine-tuned for domain specific predictions and receive the same benefit as large language models [27], [28]. However, these methods involve supervised fine-tuning, which in the case of time-series prediction consists of adding data form the specific domain the model needs to be fine-tuned on. As large language models have proven in the past, this method of fine-tuning can quickly become unsustainable due to the increasing cost of data labeling. In this study, we follow the way paved by LLMs by proposing reinforcement learning to tune time-series predictors.

PPO is a policy gradient method developed by John Schulman et al. in 2017 [29]. The key innovation of this algorithm over older methods such as TRPO [30] or ACER [31] is the clip function that constrains policy updates of the agent. PPO has been used in a wide variety of applications: Atari games [32], track racing games [33], suspension monitoring for cars [34], and image captioning [35]. A number of articles have proposed innovations to the base algorithm, for instance an alternative minimization target [36], [37] introduced policy feedback; specifically improving early learning stages, which are recognized as a potential weak point of PPO [38]. Recently proposed improvements include a shift in learning to offline policy optimization [39] and including conservatism [40].

Multi-agent methods have gained significant attention in the field of reinforcement learning, particularly for their capability to simulate complex systems involving interactive agents. A notable early work in multi-agent systems is [41] which explored the dynamics of cooperative and competitive agents in a shared environment. Recent advancements have integrated PPO into multi-agent applications: [42] applied multi-agent PPO to competitive and cooperative tasks, [43] successfully employed multi-agent reinforcement learning in the complex environment of the Dota 2 game. The integration of PPO into multi-agent systems has also been explored in real-world scenarios such as traffic light control [44], and collaborative robotics [45]. Innovations specific to multi-agent PPO include [46] which introduced a meta-learning approach to enhance adaptability across different tasks and agent configurations and [47], which presented the concept of leniency in multi-agent learning, mitigating the non-stationary issue commonly faced in such environments.

Attention is a machine learning mechanism designed to imitate human awareness. Attention was brought to the forefront of the field with the transformer architecture, a self-attention-based architecture that enabled the recent breakthroughs in large language models [48]. It has since seen many implementations including in recurrent neural networks for search results customization [49], missing data imputation [50], and in computer vision [51]. In reinforcement learning, attention models have been developed within theoretical frameworks [52] and diverse applications such as source code summarizing [53], dynamic graph problems [54], and road networks management

[55].

The novelty of the framework presented lies in the combination of staple reinforcement learning models with time-series predictors. This chapter also creates an opportunity for further applications of the framework in simulated environment encompassing diverse fields.

3. Data

To contextualize the fine-tuning we detail the financial datasets used to train the backbone and to build the fine-tuning environment. We also present the MuJoCo framework, which we use to benchmark pure reinforcement learning performance between algorithms.

3.1. *Financial and ESG Data*

The financial and ESG data used in this chapter span from intraday market prices to annual sustainability ratings. Our primary sources are:

- **Refinitiv** [56]: a global leader in financial data and analytics, covering over 80% of global market capitalization with more than 450 ESG metrics. We extract daily price and volume data via Refinitiv Eikon, together with the three ESG pillar scores (Environmental, Social, Governance) and the combined ESG score.
- **Sustainalytics** [57]: provides ESG Risk Ratings for listed firms, widely used by asset managers and banks to construct sustainable portfolios. We incorporate their flagship ESG Risk Ratings into our dataset.
- **SASB Standards** [58], [59]: the Sustainability Accounting Standards Board identifies material sustainability issues by industry. Since August 2022, SASB standards underline IFRS S1 and S2 disclosures. We one-hot encode each firm's material SASB issue set based on the 2018 publication.

Table 1 shows a snippet of Apple's daily price data from 2005-12-05 to 2005-12-13. The full time span of the dataset is 2005-12-05 through 2024-08-07.

Date	Open	Low	High	Close	Volume
2005-12-05	2.17	2.15	2.19	2.16	5.84e8
2005-12-06	2.23	2.21	2.25	2.23	8.57e8
2005-12-07	2.24	2.20	2.24	2.23	6.79e8
2005-12-08	2.21	2.19	2.23	2.23	7.90e8
2005-12-09	2.24	2.21	2.25	2.24	5.55e8
2005-12-12	2.26	2.25	2.27	2.26	5.25e8
2005-12-13	2.25	2.24	2.27	2.26	4.94e8

Table 1.: Sample daily financial data for AAPL

To enrich the raw price and volume data, we compute:

- *Log returns*, controlling for market effects via the Fama–French 5 factors [60].
- Technical indicators from historical prices and volumes:
  - Relative Strength Index (RSI) [61],
  - Moving Average Convergence Divergence (MACD) [62],

- Bollinger Bands [63].

The target variable is the FF5-adjusted log return, following the methodology of [64]. Financial data are available at sub-daily frequency, whereas ESG scores refresh annually (Refinitiv) or "regularly" (Sustainalytics). We evaluated regression, interpolation, autoencoders and forward-fill strategies. To respect provider methodologies and avoid compounding model error, we adopt a forward-fill approach for ESG values between update dates.

3.2. *MuJoCo Benchmarking Environments*

Multi-Joint dynamics with Contact, commonly called MuJoCo [65], proposes several standard environments to train and benchmark models on. To evaluate pure reinforcement learning performance, we employ three standard MuJoCo tasks:

- **HalfCheetah-v4**,
- **Hopper-v4**,
- **Humanoid-v4**.

MuJoCo provides a high-fidelity physics simulator for continuous-control benchmarks, where:

- *State* $s_t \in \mathbb{R}^d$ consists of joint angles, velocities and (for Humanoid) contact forces.
- *Action* $a_t \in \mathbb{R}^m$ represents torque inputs to each joint.
- *Reward* combines forward progress, control costs, and (where applicable) healthy posture and contact penalties.

Environment	**Reward**
HalfCheetah-v4	$R = w_f F - w_{\text{ctrl}}\, C$
Hopper-v4	$R = w_f F + w_h H - w_{\text{ctrl}}\, C$
Humanoid-v4	$R = w_f F + w_h H - w_{\text{ctrl}}\, C - w_{\text{ctct}}\, C_{\text{tct}}$

Table 2.: MuJoCo environment reward functions (forward reward F, healthy reward H, control cost C, contact cost C_{tct})

Here, $w_f, w_h, w_{\text{ctrl}}, w_{\text{ctct}}$ are environment-specific weights. We use the default observation and action spaces as defined in OpenAI Gym's MuJoCo suite.

4. Framework Details

As mentioned in [66], implementation is key in deep policy gradient algorithms. As such, the framework below is implemented using the clean-rl library [67]. We evaluate three state-of-the-art algorithms for fine-tuning: Proximal Policy Optimization (PPO), Centralized Multi-Agent PPO (CMAPPO), and Group Relative Policy Optimization (GRPO). In this section, we also detail the environment used during training and the integration of the pre-trained transformer in the algorithms.

4.1. *Proximal Policy Optimization (PPO)*

- **Policy Function:** For an agent x, its policy at time t is a probability density function denoted as $\pi_\theta(a_t|o_t)$, where θ are the parameters of the policy, o_t is the observation for agent x at time t, and a_t are the actions that can be taken. The policy is then sampled to obtain the action taken $\alpha_t \sim \pi_\theta(a_t|o_t)$.
- **Objective Function:** The PPO objective function is defined as:

$$L^{PPO}(\theta) = \mathbb{E}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$

  where $r_t(\theta) = \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{\text{old}}}(a_t|o_t)}$ is the probability ratio, ϵ an hyperparameter and $\hat{A}_t$ is an estimator of the advantage at time t, typically computed using Generalized Advantage Estimation (GAE).
- Advantage Estimation: The advantage $\hat{A}_t$ is computed as:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{1}$$

  with $\delta_t = r_t + \gamma V(o_{t+1}) - V(o_t)$ and V a learned state-value function.
- **Training Process:** The agent is trained by iteratively updating its policy parameters. This involves:
  (1) Collecting trajectories by interacting with the environment using the current policy.
  (2) Estimating the advantages using GAE.
  (3) Calculating the surrogate objective function.
  (4) Optimizing the surrogate objective function using gradient ascent while ensuring the updates stay within a specified clipping range to maintain policy stability.

4.2. *Centralized Multi-Agent PPO (CMAPPO)*

- **Subagent Policy & Training:** Each subagent x_i observes its local state $o_{t,i}$, samples an action $\alpha_{t,i} \sim \pi_{\theta_i}(a_{t,i} \mid o_{t,i})$, and learns via its own reward $R_i(o_t, a_{t,i})$ using PPO:
  (1) *Collect trajectories:* Interact with environment to gather $\{(o_{t,i}, \alpha_{t,i}, r_{t,i})\}_{t=1}^{T}\}$.
  (2) *Advantage estimation:* Compute $\hat{A}_{t,i}$ via GAE: $\hat{A}_{t,i} = \delta_{t,i} + (\gamma\lambda)\delta_{t+1,i} + +(\gamma\lambda)^2\delta_{t+2,i} + \cdots$, with $\delta_{t,i} = r_{t,i} + \gamma V(o_{t+1,i}) - V(o_{t,i})$.
  (3) *Surrogate objective:*

$$L_i^{\text{PPO}}(\theta_i) = \mathbb{E}_t\big[\, \min(r_{t,i}\hat{A}_{t,i},\ \text{clip}(r_{t,i}, 1-\epsilon, 1+\epsilon)\hat{A}_{t,i})\big],$$

  where $r_{t,i} = \frac{\pi_{\theta_i}}{\pi_{\theta_i^{\text{old}}}}$.
  (4) *Policy update:* Perform gradient ascent on L_i^{PPO}, clipping updates to maintain stability.
- **Attention-Enhanced Aggregation:** Encode the global state e_t and subagent actions $\{\alpha_{t,i}\}$ via linear layers, compute attention weights $[w_{\text{env}}, w_1, \ldots, w_n] =$

$\mathrm{softmax}([f_{\mathrm{env}}(e_t),\ f_{\mathrm{sub}}(\{\alpha_{t,i}\})])$, then aggregate:

$$d_t = w_{\mathrm{env}}\, e_t \ + \ \sum_{i=1}^{n} w_i\, \alpha_{t,i}.$$

- **Superagent Decision:** The superagent samples its final action $\alpha_t^f \sim \pi_{\theta_f}(a_t^f \mid d_t)$, allowing coordinated, adaptive decisions across all agents.

4.3. *Group Relative Policy Optimization (GRPO)*

- **Policy Function:** As in PPO, we parameterize a stochastic policy $\pi_\theta(a \mid o)$ with parameters θ. At each step t, given observation o_t, we sample a group of G candidate actions

$$a_{t,i} \sim \pi_\theta(\cdot \mid o_t), \quad i = 1, \ldots, G.$$

- **Group Rewards and Relative Advantage:** Each candidate action $a_{t,i}$ is scored by a reward function $r(a_{t,i}, o_t)$, yielding

$$r_{t,i} = r(a_{t,i}, o_t).$$

We compute the group baseline (mean) and standard deviation:

$$\bar{r}_t = \frac{1}{G}\sum_{i=1}^{G} r_{t,i}, \quad \sigma_t = \sqrt{\frac{1}{G}\sum_{i=1}^{G}\big(r_{t,i} - \bar{r}_t\big)^2} + \epsilon.$$

The *relative advantage* of candidate i is then:

$$A_{t,i} \ = \ \frac{r_{t,i} - \bar{r}_t}{\sigma_t}.$$

- **Surrogate Objective:** Defining the probability ratio for each candidate,

$$\rho_{t,i}(\theta) = \frac{\pi_\theta(a_{t,i} \mid o_t)}{\pi_{\theta_{\mathrm{old}}}(a_{t,i} \mid o_t)},$$

the GRPO loss uses the same clipped surrogate as PPO but averages over the group:

$$L^{\mathrm{GRPO}}(\theta) = \mathbb{E}_t\left[\frac{1}{G}\sum_{i=1}^{G} \min\big(\rho_{t,i}(\theta)\, A_{t,i},\ \mathrm{clip}\big(\rho_{t,i}(\theta), 1-\epsilon, 1+\epsilon\big)\, A_{t,i}\big)\right].$$

Optionally, one may add a KL-penalty term $\beta\, D_{\mathrm{KL}}\big(\pi_\theta(\cdot \mid o_t) \| \pi_{\mathrm{ref}}(\cdot \mid o_t)\big)$ to constrain policy drift.

- **Training Process:** GRPO proceeds in iterative updates:
  (1) *Sample Groups:* For each observation o_t in a batch, sample G actions $\{a_{t,i}\}$.
  (2) *Evaluate Rewards:* Compute $r_{t,i} = r(a_{t,i}, o_t)$ for $i = 1 \ldots G$.

(3) *Compute Advantages:* Form relative advantages $A_{t,i} = (r_{t,i} - \bar{r}_t)/\sigma_t$.
(4) *Surrogate Update:* Optimize θ by ascending the clipped surrogate $L^{\mathrm{GRPO}}(\theta)$ (plus optional KL term), using minibatch gradient steps.
(5) *Repeat:* Collect new groups under the updated policy and continue until convergence.

4.4. *Design of the Reinforcement Learning Environment*

The RL environment is designed to facilitate the fine-tuning of forecasting policies:

- **State:** At time t, the state $s_t \in \mathbb{R}^{T \times N}$ is a matrix containing historical observations.
- **Agent Action:** The agent produces a forecast $a_{t,i} \in \mathbb{R}^{P \times 1}$ based on its local observation $o_{t,i}$.
- **Transition Dynamics:** Following the agents' actions, the true future $y_t \in \mathbb{R}^{P \times 1}$ is revealed, and the state is updated (via a sliding window mechanism).
- **Reward:** The reward r_t is computed based on the forecast error and any additional domain-specific criteria:

$$r_t = -\ell(a_t, y_t) - \psi(a_t), \tag{2}$$

  where $\ell(\cdot)$ is an error metric (e.g., absolute or squared error) and $\psi(\cdot)$ encapsulates further constraints or penalties.

In practice, the reward function used was $r_t = 2 \times e^{(-MSE(a_t, y_t)} - 1$. This implementation constrains the reward between $[-1, 1]$, and is driven up as the MSE converges towards 0.

4.5. *Latent Representation versus Actor Network*

In practice, the probability distribution each of the algorithms sample from is a neural network. In a classic reinforcement learning approach, a new network is created to learn the latent representation between observations and actions (the action network). In the case of PPO and CMAPPO, networks are also created to learn the value function (the critic network). To fine-tune a pre-trained backbone model, we need to integrate the trained network in the framework. There are two main paradigms for fine-tuning the network:

- **The backbone outputs a latent representation of the observation space.** The action network takes the latent representation as input and outputs a probability distribution over actions, which when sampled outputs the forecast. The critic network estimates the state value for advantage estimation and the gradients flow back through the action network, critic network, and the backbone, which leads to fine-tuning.
- **The backbone is connected to a projection layer that converts the latent representation to a forecast directly.** This is what commonly happens when the backbone is used independently as a predictor. In this paradigm, the backbone takes the place of the actor network. The critic network estimates the state value and the gradients flow back through the backbone and the critic network.

Using a separate action network can improve the flexibility since the actor network has the opportunity to learn from the latent features. Decoupling the backbone and the action network also allows us to adjust the hyperparameters for the action network individually. An actor network is also more likely to explore and better adapt to the reward structure of the environment, performing significantly better in the reinforcement learning environment. We can also delay the fine-tuning by temporarily freezing all the backbone layers. This can be beneficial to performance as it gives the opportunity for the action and value networks to learn about the environment before inducing changes in the backbone network. This process can help avoid catastrophic forgetting during the early stages of interacting with the environment.

By replacing the actor network with the backbone, we ensure that a new actor network will not corrupt the original predictor. This approach is simpler and more direct, as the actor network introduces new hyperparameters but directly using the backbones only involves a minor projection. With no actor network involved, there is also less risk of overfitting the reinforcement learning task, thus maintaining a good degree of generalization. However, without an intermediary network to adapt the learned features, the backbone might struggle to perform and learn in the reinforcement learning environment. This can lead to repeated poor performance which in turn can flow through the gradient and cause catastrophic forgetting. The environment also needs to be carefully designed to avoid a mismatch between the observations at each step of the training and the encoder size of the backbone.

Both methods are compared in Table 3 using standard PPO. The reference scores are the scores of the backbone without any fine-tuning. The latent paradigm performs significantly worse, with only a small improvement in the Financial sector and massive loss in Industrials and Technology. The Actor paradigm improves upon the reference on all datasets. As such, we implemented the actor paradigm when possible. The only latent representation used was in CMAPPO with the superagent, as the aggregation of the subagents action does not correspond to the encoder accepted size of the backbone.

Table 3.: Latent vs Actor paradigms comparison. The backbone is fine-tuned using PPO on Financial, Industrials and Technology. Reference is the base model without fine-tuning. Lower is better, in bold the best metric.

Dataset	Latent		Actor		Reference	
Metric	MSE	MAE	MSE	MAE	MSE	MAE
Financial	0.202	0.206	**0.200**	0.271	0.203	**0.118**
Industrials	0.274	0.251	**0.119**	**0.116**	0.128	0.121
Technology	0.341	0.264	**0.126**	**0.119**	0.131	0.119

5. Benchmarking

Three MuJoCo environments were selected as experimental settings. The three environments are: Hopper-v4, Half-Cheetah-v4 and Humanoid-v4. In this experiment, we use standard 64 hidden dimensions networks for the action and value heads. Table 4 presents the results of the three algorithms tested on each MuJoCo environment. CMAPPO wins out on all three environments, followed closely by default PPO. The GRPO algorithm, which does not use a critic network, underperforms slightly in the pure reinforcement learning task, especially in the Hopper-v4 environment.

Table 4.: Results of MuJoCo environment training. Higher is better, best value in bold.

Model Environment	PPO Reward	CMAPPO Reward	GRPO Reward
HalfCheetah-v4	-150.54	**-111.10**	-137.18
Hopper-v4	1185.06	**1960.75**	624.86
Humanoid-v4	2897.81	**3201.09**	2659.32

6. Results

Fine-tuning is by definition local and its performance is measurable on a case-by-case basis. To cover as many use cases as possible, we propose to examine the results through the use of two common techniques in fine-tuning: layers freezing and transfer learning.

6.1. *Fine-tuning and Frozen Layers*

In order to retain high level patterns learned during supervised training, we can freeze parts of the model to stop the loss propagation through the network. This technique is common in large language models alignment and is employed to build the results in Table 5. We fine-tune the model with no frozen layers, 25%, 50% and 75% frozen layers.

Table 5.: Results of fine-tuning models on Financial, Industrials and Technology dataset compared to the original model. In rows, the model's layers are progressively frozen. In columns, each sector represents the testing set of the model. Lower is better, best value in bold.

Frozen %	Model	Financial		Industrials		Technology	
Metric		MSE	MAE	MSE	MAE	MSE	MAE
0%	PPO	0.200	0.271	0.119	0.116	0.126	0.119
	CMAPPO	0.324	0.208	0.146	0.160	0.203	0.189
	GRPO	0.198	0.109	0.118	0.113	0.124	0.115
25%	PPO	0.199	0.114	0.120	0.116	0.125	0.118
	CMAPPO	0.300	0.204	0.202	0.211	0.525	0.341
	GRPO	0.198	0.108	**0.118**	**0.112**	0.124	0.113
50%	PPO	0.202	0.113	0.119	0.117	0.124	0.117
	CMAPPO	0.237	0.155	0.257	0.248	0.151	0.151
	GRPO	**0.195**	**0.108**	**0.118**	**0.112**	0.124	0.113
75%	PPO	0.200	0.114	0.119	0.117	0.124	0.117
	CMAPPO	0.270	0.183	0.289	0.272	0.137	0.135
	GRPO	**0.195**	**0.109**	0.118	0.113	**0.123**	**0.113**
Original	Backbone	0.202	0.111	0.120	0.115	0.124	0.116

GRPO performed the best overall, either improving or leaving the backbone model unchanged. Notably, freezing at least 50% of the encoder layers gave consistently the best performance when using GRPO. PPO proposed a minor improvement in some categories, for instance in Financial at 25%, but mostly left the model unchanged. CMAPPO performed the worst in the fine tuning, provoking large negative changes

to the model even with 75% of the encoder frozen. The source of the performance of GRPO in fine-tuning is the same reason it was the worst performer in the pure reinforcement learning task: the absence of a value function. While this is mostly a disadvantage learning control tasks, in the case of fine-tuning the difference of complexity between the value network and the backbone severely hinders the performance of PPO and CMAPPO. In the case of CMAPPO, the latent representation offered by the subagents are also reconciled using an action network. This design is coherent with the original implementation of CMAPPO but also adds another layer of abstraction the model needs to learn. A possible improvement for PPO and CMAPPO would be to run the model without propagating the loss back to the backbone to train the value network. By delaying the learning, the value network could learn a proper representation of the advantage in the task and nudge the backbone in the right direction.

6.2. *Transfer Learning*

Transfer learning is a machine learning technique through which a model learns general concepts applicable across multiple datasets. We experiment on transfer learning by fine-tuning and testing the model on the three datasets.

Table 6.: Reference values before fine-tuning.

Trained on	Financial		Industrials		Technology	
Tested on	MSE	MAE	MSE	MAE	MSE	MAE
Financial	**0.203**	0.118	0.207	0.113	0.203	**0.110**
Industrials	0.224	0.227	0.128	0.121	**0.122**	**0.114**
Technology	0.256	0.229	0.132	0.118	**0.131**	**0.117**

Table 6 presents the results of the model on the Finance, Industrials and Technology datasets before fine-tuning. Instead of training the backbone model on all three datasets and fine-tuning for one, we train the backbone on a single dataset and test the MSE/MAE on all three. The Financial appears as the most challenging dataset, performing quite worse than the baseline when tested on Industrials and Technology. The model trained on Industrials manages to nearly match the performance of the models trained on Financial and Technology. Finally, the Technology model is by far the best, outperforming Industrials even when tested on Industrials. This metric could be interpreted as the degree of high level patterns present in the dataset. These high level patterns can be applied to any similar dataset, and ultimately are more powerful predictive tools than the past history for a given example.

Table 7.: Results of fine-tuning models on Financial, Industrials and Technology dataset compared to the original model. The model is fine-tuned and tested on the specified sector for each row. In columns, each sector represents the original training set of the model. Lower is better, best value in bold.

Trained on →		Financial		Industrials		Technology	
Fine-tuned on ↓	Method	MSE	MAE	MSE	MAE	MSE	MAE
Financial	PPO	0.279	0.196	0.213	0.219	0.247	0.219
	CMAPPO	0.224	0.143	**0.145**	0.152	**0.151**	0.151
	GRPO	0.230	0.156	0.155	0.167	0.170	0.165
	Baseline	**0.203**	**0.118**	0.207	**0.113**	0.203	**0.110**
Industrials	PPO	0.201	0.115	0.123	0.119	0.131	0.121
	CMAPPO	0.204	0.117	0.127	0.121	0.137	0.125
	GRPO	**0.197**	**0.110**	**0.119**	**0.113**	0.125	**0.114**
	Baseline	0.224	0.227	0.128	0.121	**0.122**	**0.114**
Technology	PPO	0.198	0.114	0.125	0.117	0.127	0.120
	CMAPPO	0.202	0.115	0.128	0.119	0.129	0.120
	GRPO	**0.189**	**0.108**	**0.118**	**0.112**	**0.123**	**0.113**
	Baseline	0.256	0.229	0.132	0.118	0.131	0.117

Table 7 presents the results of the model on the Finance, Industrials and Technology datasets after fine-tuning. A first observation is the improvement in performance in all nearly all models from the baseline in Table 6. Some of the most substantial gains are found in the model trained on Financial, which improved its performance in MSE for both Industrials and Technology but moreover completely dominates the MAE benchmark. On the MSE front, the model trained on Industrials had the best results and beat out the best reference values for each sector.

Notable exceptions are the model trained and fine-tuned on Financial, and the model trained on Technology and fine-tuned on Industrials. In both cases, neither PPO, CMAPPO or GRPO managed to improve the performance, and testing on unseen data yielded a worse result. For the first case, the likely explanation is an overfitting to the train data: effectively, the model was trained twice on the same dataset, once with supervised learning, and again using reinforcement learning. The second case is different: the original Technology model already performed outstandingly well in Industrials, beating out even the models trained on the complete dataset. The fine-tuning failed to further improve that performance, marking the importance of establishing baselines before introducing fine-tuning to the pipeline.

The patterns noticed in Table 5 largely stand, with GRPO clearly distinguishing as the better option in nearly all cases. CMAPPO performed exceptionally well on Financial, outperforming both PPO and GRPO. The superagent managed to reconcile the actions of the subagents despite the added complexity of the actor and critic network. PPO nearly always improves on the baseline and constitute a valid choice for fine-tuning. The recommended algorithm stands out as GRPO, which uses fewer computational resources and yields the best performance. Committing to the actor paradigm and removing the critic network greatly simplifies the fine-tuning architecture, allowing for direct backpropagation through the backbone without the need for intermediary networks.

These results also clearly indicate the value of transfer learning for timeseries predictor. One of the best use case for fine-tuning appears to be adapting models from their supervised training dataset to another. This is in line with the current state of

fine-tuning in large language models, which often adapts model after pre-training to diverse specific tasks. This result also highlights two clear areas for improvement in timeseries predictors: firstly, large pre-trained models can be built, and later specialized to a given dataset. But the biggest challenge to generalize this method is to specify a model and a fine-tuning environment that allows for various observation space and exogenous features.

7. Key hyperparameters

PPO and its variants are known to be sensitive to hyperparameters. In order to compare each algorithm fairly, we show in this section specific and non-specific hyperparameters tuning. All models presented from this point onward use the backbone as the action network, and a value network with 2 layers and 256 hidden dimensions when relevant (PPO, CMAPPO).

7.1. *Training time*

Training time is common to PPO, CMAPPO and GRPO. A higher number of timesteps will lead to a better performance in the environment until the agent reaches a plateau, at the expense of a higher computational cost. We fine-tune the model on the Financial dataset using PPO at different timesteps and plot the MSE over time in Figure 1. We found 500 000 timesteps to be the best value as a balance between overfitting and underfitting.

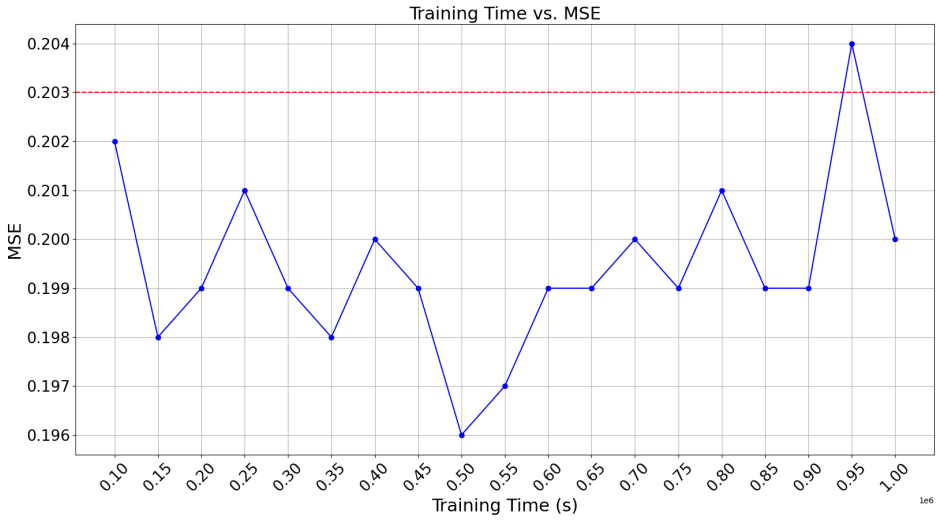

Figure 1.: Training time vs MSE. Dotted line is the original model performance before fine-tuning. Training time is scaled down from 1e6 for readability.

7.2. *Number of subagents (CMAPPO)*

Number of subagents is specific to CMAPPO and controls how many subagents are trained before the superagent. We fine-tune a predictive model on the Financial dataset

with an increasing number of subagents and test the MSE/MAE after fine-tuning. Table 8 presents the MSE and MAE with increasing numbers of subagents and compared to the backbone model. We found 10 subagents to be the best configuration, despite the backbone outperforming the fine-tuned model in all configurations.

Table 8.: The influence of the number of subagents when fine-tuning the backbone compared to the non fine-tuned backbone.

Number of Subagents	Financial	
Metric	MSE	MAE
2	0.925	0.401
4	0.461	0.275
6	0.394	0.256
8	0.337	0.211
10	**0.271**	**0.183**
12	0.283	0.191
Backbone	0.202	0.111

7.3. *Group size (GRPO)*

Group size is specific to GRPO and determines the size of the group used to calculate the advantage. Similarly to Subesection 7.2, we fine-tune a predictive model on the Financial dataset with an increasing group size and test the MSE/MAE after fine-tuning. Table 9 presents the results of the model at group sizes from 2 to 12. We found that a group size of 8 is optimal for both computational load and model performance.

Table 9.: The influence of the group size when fine-tuning the model on the Financial dataset.

Group Size	Financial	
Metric	MSE	MAE
2	0.200	0.204
4	0.198	0.199
6	0.197	0.199
8	**0.195**	0.196
10	0.199	0.201
12	0.201	0.203
Backbone	0.202	**0.111**

8. Conclusion

Fine-tuning timeseries predictors is emerging as an essential post-supervised training step to improve the performance of models. As the paradigm shifts from local models to larger, eclectic models harnessing the predictive power of many timeseries from diverse fields, fine-tuning becomes even more essential. At scale, is it far more cost effective to fine-tune a large model to a specific use case than retraining on large datasets. As the computational load for supervised learning gets higher and the models get larger, which has been the trend observed in LLMs and timeseries predictors, fine-tuning becomes even more attractive.

There are still several limitations, the most prominent being that pre-trained models use a fixed size input vector. This problem is not encountered in standard large language models, as the alphabet is tokenized to represent the entirety of the model output. But timeseries prediction is a continuous process, and further innovation is needed in foundational model to break out of fixed size vectors and scale up the models on large datasets, without relying on tricks such as projection layers. In the same spirit, architectural changes to foundational model allowing for variable output vector size would benefit the industry integration of timeseries predictors.

The environment definition and reward structure are key to the success of fine-tuning. Empirically, we noticed better results by bounding the reward to values between -1 and 1. The algorithm used is also a determining factor, and GRPO emerges as the clear winner in this chapter. This result is in line with the recent advances in LLMs, and further strengthens the conjecture that LLMs and timeseries predictors based on the same architecture share scaling features. If this conjecture reveals to be true, timeseries predictors are in a fantastic second mover position to implement even more innovations the thriving LLM community is building.

References

[1] H. Cazaux, R. Rudd, H. Stefánsson, S. Ólafsson, and E. I. Ásgeirsson, "Non-stationary inverted transformer with time2vec embedding," *IEEE Transactions on Artificial Intelligence*, vol. 5, no. 1, 2024.

[2] K. W. Church, Z. Chen, and Y. Ma, "Emerging trends: A gentle introduction to fine-tuning," *Natural Language Engineering*, vol. 27, no. 6, pp. 763–778, 2021.

[3] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, *et al.*, "Convolutional neural networks for medical image analysis: Full training or fine tuning?" *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1299–1312, 2016.

[4] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[5] Meta, *Llama 3.1*, `https://llama.meta.com/`, 2024.

[6] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[8] E. Cetinic, T. Lipic, and S. Grgic, "Fine-tuning convolutional neural networks for fine art classification," *Expert Systems with Applications*, vol. 114, pp. 107–118, 2018.

[9] H. Xiong, S. Wang, Y. Zhu, *et al.*, "Doctorglm: Fine-tuning your chinese doctor is not a herculean task," *arXiv preprint arXiv:2304.01097*, 2023.

[10] L. Luo, J. Ning, Y. Zhao, *et al.*, "Taiyi: A bilingual fine-tuned large language model for diverse biomedical tasks," *Journal of the American Medical Informatics Association*, ocae037, 2024.

[11] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture," *Computer Networks*, vol. 171, p. 107 138, 2020.

[12] L. Xu, H. Xie, S.-Z. J. Qin, X. Tao, and F. L. Wang, "Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment," *arXiv preprint arXiv:2312.12148*, 2023.

[13] Z. Fu, H. Yang, A. M.-C. So, W. Lam, L. Bing, and N. Collier, "On the effectiveness of parameter-efficient fine-tuning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, 2023, pp. 12 799–12 807.

[14] R. Zhang, J. Han, C. Liu, *et al.*, "Llama-adapter: Efficient fine-tuning of language models with zero-init attention," *arXiv preprint arXiv:2303.16199*, 2023.

[15] R. He, L. Liu, H. Ye, *et al.*, "On the effectiveness of adapter-based tuning for pretrained language model adaptation," *arXiv preprint arXiv:2106.03164*, 2021.

[16] E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[17] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," *arXiv preprint arXiv:2101.00190*, 2021.

[18] B. Gunel, J. Du, A. Conneau, and V. Stoyanov, "Supervised contrastive learning for pre-trained language model fine-tuning," *arXiv preprint arXiv:2011.01403*, 2020.

[19] Y. Zhou and V. Srikumar, "A closer look at how fine-tuning changes bert," *arXiv preprint arXiv:2106.14282*, 2021.

[20] T. Zhang, F. Wu, A. Katiyar, K. Q. Weinberger, and Y. Artzi, "Revisiting few-sample bert fine-tuning," *arXiv preprint arXiv:2006.05987*, 2020.

[21] T. Fredriksson, D. I. Mattos, J. Bosch, and H. H. Olsson, "Data labeling: An empirical investigation into industrial challenges and mitigation strategies," in *International Conference on Product-Focused Software Process Improvement*, Springer, 2020, pp. 202–216.

[22] T. Chen, S. Liu, S. Chang, Y. Cheng, L. Amini, and Z. Wang, "Adversarial robustness: From self-supervised pre-training to fine-tuning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 699–708.

[23] Y. Fan, O. Watkins, Y. Du, *et al.*, "Dpok: Reinforcement learning for fine-tuning text-to-image diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 79 858–79 885, 2023.

[24] S. Zhai, H. Bai, Z. Lin, *et al.*, "Fine-tuning large vision-language models as decision-making agents via reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 37, pp. 110 935–110 971, 2025.

[25] A. Liu, B. Feng, B. Xue, *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[26] Y. Liu, H. Zhang, C. Li, X. Huang, J. Wang, and M. Long, "Timer: Generative pre-trained transformers are large time series models," in *Forty-first International Conference on Machine Learning*, 2024.

[27] C. Chang, W.-C. Peng, and T.-F. Chen, "Llm4ts: Two-stage fine-tuning for time-series forecasting with pre-trained llms," *arXiv preprint arXiv:2308.08469*, 2023.

[28] Y. Liang, H. Wen, Y. Nie, *et al.*, "Foundation models for time series analysis: A tutorial and survey," in *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, 2024, pp. 6555–6565.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[30] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.

[31] Z. Wang, V. Bapst, N. Heess, *et al.*, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[32] L. Kaiser, M. Babaeizadeh, P. Milos, *et al.*, "Model-based reinforcement learning for atari," *arXiv preprint arXiv:1903.00374*, 2019.

[33] M. S. Holubar and M. A. Wiering, "Continuous-action reinforcement learning for playing racing games: Comparing spg to ppo," *arXiv preprint arXiv:2001.05270*, 2020.

[34] S.-Y. Han and T. Liang, "Reinforcement-learning-based vibration control for a vehicle semi-active suspension system via the ppo approach," *Applied Sciences*, vol. 12, no. 6, p. 3078, 2022.

[35] L. Zhang, Y. Zhang, X. Zhao, and Z. Zou, "Image captioning via proximal policy optimization," *Image and Vision Computing*, vol. 108, p. 104 126, 2021.

[36] T. Kobayashi, "Proximal policy optimization with relative pearson divergence," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 8416–8421.

[37] Y. Gu, Y. Cheng, C. P. Chen, and X. Wang, "Proximal policy optimization with policy feedback," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 7, pp. 4600–4610, 2021.

[38] C. C.-Y. Hsu, C. Mendler-Dünner, and M. Hardt, "Revisiting design choices in proximal policy optimization," *arXiv preprint arXiv:2009.10897*, 2020.

[39] Q. Cai, Z. Yang, C. Jin, and Z. Wang, "Provably efficient exploration in policy optimization," in *International Conference on Machine Learning*, PMLR, 2020, pp. 1283–1294.

[40] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn, "Combo: Conservative offline model-based policy optimization," *Advances in neural information processing systems*, vol. 34, pp. 28 954–28 967, 2021.

[41] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

[42] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, 2017.

[43] C. Berner, G. Brockman, B. Chan, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[44] X. Liang, X. Du, G. Wang, and Z. Han, "Deep reinforcement learning for traffic light control in vehicular networks," *arXiv preprint arXiv:1904.08117*, 2019.

[45] L. Matignon, G. Laurent, and N. Le Fort-Piat, "Coordinated multi-agent learning: The state of the art," *Artificial Intelligence Review*, vol. 37, no. 3, pp. 219–250, 2012.

[46] T. Yu, G. Qu, A. Singh, S. Levine, and C. Finn, "Meta-learning with latent embedding optimization in multi-agent systems," in *International Conference on Learning Representations*, 2020.

[47] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani, "Lenient multi-agent deep reinforcement learning," *arXiv preprint arXiv:1805.04566*, 2018.

[48] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[49] X. Guo, H. Zhang, H. Yang, L. Xu, and Z. Ye, "A single attention-based combination of cnn and rnn for relation classification," *IEEE Access*, vol. 7, pp. 12 467–12 475, 2019.

[50] R. Wu, A. Zhang, I. Ilyas, and T. Rekatsinas, "Attention-based learning for missing data imputation in holoclean," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 307–325, 2020.

[51] M. J. Er, Y. Zhang, N. Wang, and M. Pratama, "Attention pooling-based convolutional neural network for sentence modelling," *Information Sciences*, vol. 373, pp. 388–403, 2016, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2016.08.084. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025516306673.

[52] L. Bramlage and A. Cortese, "Generalized attention-weighted reinforcement learning," *Neural Networks*, vol. 145, pp. 10–21, 2022.

[53] W. Wang, Y. Zhang, Y. Sui, *et al.*, "Reinforcement-learning-guided source code summarization using hierarchical attention," *IEEE Transactions on software Engineering*, vol. 48, no. 1, pp. 102–119, 2020.

[54] U. Gunarathna, R. Borovica-Gajic, S. Karunasekara, and E. Tanin, "Solving dynamic graph problems with multi-attention deep reinforcement learning," *arXiv preprint arXiv:2201.04895*, 2022.

[55] C. Liu and G. Liu, "Jointppo: Diving deeper into the effectiveness of ppo in multi-agent reinforcement learning," *arXiv preprint arXiv:2404.11831*, 2024.

[56] Reuters, *Reuters*, https://www.reuters.com/, 2024.

[57] Sustainalytics, *Sustainalytics*, 2022. [Online]. Available: https://www.sustainalytics.com/.

[58] IFRS, *Ifrs s1*, https://www.ifrs.org/issued-standards/ifrs-sustainability-standards-navigator/ifrs-s1-general-requirements/, 2023.

[59] N. S. Soderstrom and K. J. Sun, "Ifrs adoption and accounting quality: A review," *European accounting review*, vol. 16, no. 4, pp. 675–702, 2007.

[60] E. F. Fama and K. R. French, "A five-factor asset pricing model," *Journal of Financial Economics*, vol. 116, no. 1, pp. 1–22, 2015.

[61] P. C. Belafsky, G. N. Postma, and J. A. Koufman, "Validity and reliability of the reflux symptom index (rsi)," *Journal of voice*, vol. 16, no. 2, pp. 274–277, 2002.

[62] T. T.-L. Chong and W.-K. Ng, "Technical analysis and the london stock exchange: Testing the macd and rsi rules using the ft30," *Applied Economics Letters*, vol. 15, no. 14, pp. 1111–1114, 2008.

[63] J. Bollinger, "Using bollinger bands," *Stocks & Commodities*, vol. 10, no. 2, pp. 47–51, 1992.

[64] H. Cazaux, R. Rudd, H. Stefánsson, S. Ólafsson, M. Raberto, and E. I. Ásgeirsson, "Correlation study between returns and esg ratings.," *Journal of Impact & ESG Investing*, vol. 5, no. 1, 2024.

[65] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[66] L. Engstrom, A. Ilyas, S. Santurkar, *et al.*, "Implementation matters in deep policy gradients: A case study on ppo and trpo," *arXiv preprint arXiv:2005.12729*, 2020.

[67] S. Huang, R. F. J. Dossa, C. Ye, *et al.*, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html.