Operator Learning in Macroeconomics

Yaolang Zhong*

Abstract

This paper proposes a novel solution framework for the class of dynamic macroeconomic models with a continuum of heterogeneous agents and aggregate uncertainty. In these models, an agent's state variables include her individual state vector and a distribution function representing all agents' states, an infinite-dimensional object. Unlike the prevalent benchmark method that approximates the distribution function with a high-dimensional vector of simulated agents, this paper suggests the formulation of the policy function as an operator that maps between function spaces. The operator is parameterized by a cutting-edge neural network architecture known as the neural operator. This proposed framework offers significant computational advantages due to its three defining properties: discretization-invariance, permutation-invariance, and aggregation-sharing. The effectiveness of this approach is demonstrated by solving the Bewley-Huggett-Aiyagari model with aggregate uncertainty, a benchmark in computational economics literature. The proposed framework not only demonstrates computational efficiency as it manages tens of thousands of agents during simulations to precisely approximate the distribution function but also showcases its superior performance, achieving solutions with less than a one percent relative error in a shorter computational time compared to the benchmark method.

1 Introduction

Dynamic models that incorporate both heterogeneity and aggregate uncertainty have become key areas of focus in macroeconomics. The reasons for their popularity include the significant implications of heterogeneity for aggregate consumption, savings, and labor supply

*Department of Economics, University of Warwick, Coventry CV4 7AL, UK. Email: yaolang.zhong@warwick.ac.uk. Acknowledgments: I am grateful to my supervisors, Dr. Mingli Chen, Prof. Herakles Polemarchakis, and Prof. Eric Renault, for their guidance. Special thanks to Prof. James Fenske for his invaluable support during the job market application process. I am indebted to Dr. Xiangyu Feng, Prof. Wiji Arulampalam, Dr. Claudia Rei, Dr. Ao Wang, Dr. Kenichi Nagasawa, Dr. Luis Candelaria, Zoe Zhang, Jiaqi Li, Amedeo Andriollo, Johannes Böken and Gokul Gopalan Ramachandran for their constructive feedback and comments.

behaviors [Blundell, Pistaferri, and Preston, 2008, Krueger and Perri, 2006], and the pivotal role of heterogeneity in the propagation of aggregate shocks, as illustrated by Kaplan, Moll, and Violante [2018]. These models also provide more realistic representations of economies, allowing for richer dynamics and more complex interactions [Cagetti and De Nardi, 2008].

Another important reason for the prevalence of these models is the development in the computational literature of numerical methods. Models with heterogeneity and aggregate uncertainty have been known to be difficult to solve. Yet, swift advancements in computational techniques have enabled economists to numerically solve these types of models. In particular, there has been a burgeoning interest in the application of deep learning techniques to address the computational challenges, as evidenced by Maliar, Maliar, and Winant [2021], Maliar and Maliar [2022], Han and Yang [2021], Azinovic, Gaegauf, and Scheidegger [2022]. These studies capitalize on the intrinsic properties of the feed-forward neural network architecture to mitigate the curse of dimensionality and leverage simulation-based data generation algorithms, which have shown significant computational advantages over traditional grid-based methods.

However, computational challenges remain. As we will see, the challenges stem from two sources: the limitations of the feed-forward neural network architecture as the function approximator, and the associated computational strategy not fully utilizing all the useful information of the data structure stemming from the economic model. I regard this as the domain knowledge of the economic problems, which is essential to facilitate optimization.

This paper addresses these challenges by resolving the two issues discussed. It specifically focuses on finding the global solution in discrete-time and continuum-agent settings. A key feature of this type of model is that an agent's state variables encompass her individual state, represented as a vector, and a distribution function of all agents' individual states, an infinite-dimensional object. This paper introduces a novel solution framework by redefining the agents' policy function as an operator, a mapping from functions to functions. In this reformulation, when processing state variables for the policy, it initially takes only the distribution function as input and outputs a conditional policy function, which then requires only the individual vector as input. This approach translates solving the agents' policy function into learning an operator. The paper employs a state-of-the-art technique from the machine learning literature, known as the neural operator, for operator parameterization [Li et al., 2020a, Kovachki et al., 2021]. The neural operator, a neural network architecture designed for approximating mappings between function spaces, draws its approximation capabilities from the Universal Approximation Theorem for Operators, a concept dating back to the seminal work of Chen and Chen [1995].

To understand the benefits of reformulating the problem as an operator and using neural operator parameterization, and to contextualize this paper's contributions within the existing literature, it is essential to review current prevalent methods. Regarding the models of interest, there are two primary solution frameworks. The first, the Krusell and Smith framework [Krusell and Smith, 1998, Den Haan and Rendahl, 2010, Maliar and Maliar, 2014]—hereafter KS framework—characterizes the distribution by tracking a select number of moments. While tractable and intuitive, the KS framework has faced criticism for relying on additional assumptions about agents' rationality and the arbitrary selection of moments.

The second framework, the Deep Learning approach using feed-forward neural networks [Maliar, Maliar, and Winant, 2021, Han and Yang, 2021, Azinovic, Gaegauf, and Scheidegger, 2022, Kahou, Fernández-Villaverde, Perla, and Sood, 2021]—hereafter NN framework—models the distribution across a finite, potentially large, number of simulated agents. It uses the feed-forward neural network to parameterize the policy function, with aggregate motion characterized by the collective transitions of all simulated agents following a specific policy. The NN framework addresses the KS framework's limitations but introduces a new set of computational challenges. It is these challenges that this paper's proposed operator learning framework aims to address, with the NN framework serving as the benchmark method for comparison throughout most of the paper.

To understand the limitations of the NN framework, it is important to first recognize that the dimensionality of the state variables increases linearly with the number of simulated agents. Each state variable corresponds to a node in the input layer of the neural network; thus, when the number of simulated agents is sufficiently large, the input layer already dominates, resulting in a neural network that expands linearly in size with the number of agents. This presents researchers with a significant trade-off: simulating a sufficient number of agents to accurately approximate the continuous distribution requires training an extensively large neural network, which leads to considerable computational time and memory demands.

Additionally, the feed-forward neural network parameterization may be problematic due to its architecture's dependence on a specific number of agents. Consequently, a new network must be trained from scratch whenever an alternative number of simulated agents is considered. This lack of flexibility, or non-generalizability, can cause issues in two scenarios: first, when researchers wish to train the neural network with a smaller population, or equivalently, a coarser discretization of the distribution, and then aim to test its performance with a larger population, or a finer discretization; second, when the economic environment under study is an overlapping-generation (OLG) model involving dynamics such as birth and death probabilities, leading to a fluctuating number of agents.

The operator learning framework resolves this issue, as the neural operator architecture it utilizes possesses the advantageous property of *discretization invariance*. This means the size of the neural operator remains constant regardless of the number of simulated agents representing the distribution function, a point that is elaborated upon in the introduction of neural operators in Section 4. Specifically, Kovachki et al. [2021] proved that the neural operator is the only known class of models for operator approximation that ensure both discretization invariance and universal approximation. This eliminates the aforementioned trade-off, allowing researchers to handle a large number of simulated agents without increasing network size. Furthermore, the number of simulated agents can vary between training and evaluation phases and during simulations, whether for reasons of computational efficiency or due to the specifications of the model.

The second limitation of the NN framework is that it fails to exploit two critical properties of the data structure arising from the economic model: *permutation-invariance*[1] and *aggregation-sharing*. Permutation-invariance dictates that the policy function parameterized by the feed-forward neural network should, in theory, produce the same output irrespective of the ordering of agents in the input. However, without this property being explicitly incorporated into the data, considerable computational effort is expended to enable the neural network to recognize this invariance. Aggregation-sharing means that all simulated agents share a common distribution function and are interdependent, rendering it inefficient for the neural network to process the identical aggregate component of the state variables for each agent repeatedly. Both properties, when not properly integrated, lead to the computational inefficiencies that plague the NN framework.

The proposed operator learning framework address this limitation immediately by the way it formulates the problem. Like the NN framework, it manages a population of agents, but instead of processing individual cross-sectional agents, it uses the constructed empirical distribution function as the input for the operator, inherently encoding the permutation-invariance property. Moreover, since the processing of agents' states is decomposed into two stages, the operator framework processes the distribution function just once during the simulation of state transitions, rather than redundantly for each agent. Subsequently, only the individual states of each agent are processed through the conditional policy function.

Given that the KS framework deals with the moments of distributions, it implicitly satisfies the discretization-invariance, permutation-invariance, and aggregation-sharing properties discussed above. Table 1 presents a comparison of the three frameworks.

To validate the proposed framework, I implemented it to solve the Bewley-Huggett-Aiyagari model incorporating aggregate uncertainty, as discussed by Den Haan, Judd, and Juillard [2010]. This model is a standard benchmark within the computational suite project and is commonly used to assess new methodologies in computational economics research.

A specialized optimization strategy was developed to enhance convergence. Specifically, for the data-generating process, I employed a hybrid sampling approach: the grid method for the low-dimensional individual state, and the stochastic simulation method for the infinite-dimensional aggregate state, namely, the distribution function. The rationale is that establishing grid points in an infinite-dimensional space is unfeasible, and remains impractical even in high-dimensional but finite cases. Therefore, I simulate the economic model by ap-

[1]Permutation-invariance was discussed and satisfied in some methods belonging to the NN framework, see e.g., Han and Yang [2021], Kahou, Fernández-Villaverde, Perla, and Sood [2021].

Table 1: Comparison of Three Numerical Frameworks for the Desirable Properties

Property	Framework		
	KS[1]	NN[2]	Operator[3]
Full Information of Distribution	×	✓	✓
Discretization-Invariance	✓	×	✓
Permutation-Invariance	✓	×	✓
Sharing-Aggregation	✓	×	✓

[1] Krusell-Smith
[2] Deep Learning with feed-forward neural network
[3] Deep Learning with neural operator (This Paper)

proximating the solution within the ergodic set, as suggested by Judd, Maliar, and Maliar [2011]. For the low-dimensional part, the grid point method yields higher accuracy than stochastic simulation.

Considering that a portion of the data, the distribution function, is endogenously generated, the initialization of the neural operator is critical for convergence. In the initial stages, when the policy is suboptimal, the corresponding ergodic set can differ substantially from the convergent set, which in turn, supplies low-quality data detrimental to the optimization process. To counteract this, I designed a specific initialization scheme: I first addressed a version of the Bewley-Huggett-Aiyagari model without aggregate uncertainty. This simpler model, characterized by static aggregation, can be solved swiftly. The resulting policy function serves a dual purpose: it acts as an initial training step for the neural operator—reflecting the concept of "transfer learning" in deep learning literature—and assists in simulating an approximate ergodic set—akin to "off-policy learning" in reinforcement learning. The initialization of the neural operator is then performed through a supervised learning task, with the output of the policy function from the simpler problem serving as the target.

Training was conducted on a 2018 MacBook Pro equipped with an Intel Core i5 processor and 16 GB of 2133 MHz LPDDR3 Memory. The optimization was programmed in Python, utilizing the PyTorch deep learning framework. Within a reasonable computational timeframe, the neural operator approach successfully delivered a solution, achieving an error measured by the Euler residuals within a fraction of one percentage point—an impressive level of precision for approximating a high-dimensional policy function with kink points [Maliar, Maliar, and Winant, 2021]. Notably, in my simulations, I managed 10,000 agents while retaining a network size comparable to the NN framework that handles only 1,000 agents. In a direct comparison within the same setting of 1,000 agents, the operator learning framework achieved convergence rapidly, whereas the NN framework continued to yield a suboptimal policy.

Contributions to the Literature: As previously stated, this paper contributes to the computational economics and macroeconomics literature by introducing a new machine learning-powered solution framework for a prevalent class of models: the heterogeneous agent models with aggregate uncertainty. The principal computational challenge lies in learning the agents' policy function, which includes a distribution function as part of the input. The current literature's approaches typically reduce the infinite-dimensional distribution function to a low-dimensional vector [Krusell and Smith, 1998, Den Haan and Rendahl, 2010, Maliar and Maliar, 2014] or a high-dimensional one [Maliar, Maliar, and Winant, 2021, Han and Yang, 2021], thereby simplifying it to the case of a conventional policy function and parameterizing it with a feed-forward neural network, which introduces computational inefficiencies. The innovations of the proposed framework in this paper are threefold: first, a reformulation of the policy function with a distribution function component into an operator; second, the parameterization of this operator using the neural operator, an advanced neural network architecture from the machine learning literature; and third, a specially designed optimization strategy. Each of these aspects contributes to a significant gain in computational efficiency.

While this paper utilizes the neural operator, a tool developed within the machine learning literature [Lu et al., 2019, Kovachki et al., 2021, Li et al., 2020a,c, 2021, Goswami et al., 2023], it also contributes to the machine learning field by demonstrating its application within the economics discipline, and more broadly, in the social sciences and strategic game settings. Until now, computer scientists have primarily focused on applying neural operators to solve partial differential equations (PDEs) that characterize physical or engineering systems. To the best of my knowledge, this work represents the first application of neural operators in discrete time economic models. Furthermore, its adoption is not a mere application to a well-defined task; this paper contributes by formulating an economic problem with domain knowledge, making it suitable for the application of neural operators.

Furthermore, this paper extends its contributions to an emergent body of literature that examines the strategic interactions among artificial-intelligence agents governed by reinforcement learning-based policies in an economic environment [Chen et al., 2021, Zheng et al., 2020, 2021]. Despite the distinct nature of model-based learning in computational economics and model-free learning in reinforcement learning, the framework proposed in this paper exhibits commonalities with the literature on neural network-parameterized policies, simulation-based data generation, and optimization techniques such as off-policy learning. The operator formulation and the adoption of the neural operator architecture within the proposed framework offer promising avenues for the design of policy optimization algorithms in this field. The operator formulation and the implementation of the neural operator architecture within the proposed framework provide insightful directions for the development of policy optimization algorithms in this burgeoning field.

The remainder of the paper is structured as follows: Section 2 introduces the economic environment targeted for solution with the proposed method; Section 3 contrasts my computational strategy with the two principal frameworks in the literature; Section 4 delves into the neural operator architecture and its mathematical properties; Section 5 covers the implementation details; Section 6 details the results; and Section 7 provides the conclusion.

2 The Economic Environment

This section provides a brief overview of the economic environment from Den Haan, Judd, and Juillard [2010], which I intend to address using my method. This model serves as a benchmark, encompassing features such as a continuum of agents, aggregate uncertainty, idiosyncratic shocks, and an incomplete market. This is the first model in the computational suite project for the comparison of the properties of numerical algorithms. The structure is closely aligned with that of Krusell and Smith [1998] except the introduction of income taxation and unemployment benefit, only by which the agents' budget constraint would be occasionally binding.

From this point forward, I will use lowercase letters for individual variables, uppercase letters for aggregate variables and bold uppercase letters for operations.

Agents' Problem

There is a continuum of infinitely lived and ex-ante identical agents. Each period the agents receive the time endowment $\bar{l}$ and face a binary income shock ϵ: agents are employed when $\epsilon = 1$, and earn the after-tax wage $(1-\tau_t)\bar{l}W_t$; agents are unemployed when $\epsilon = 0$, and earn the unemployment benefit μW_t. Here W_t is the per unit of time wage rate, τ_t is the tax rate, and μ is a model parameter denoting the fraction of wage.

Market is incomplete featured by the capital borrowing constraint $k_t \geq 0$; the net rate of return for capital accumulation is $R_t - \delta$, where R_t is market-determining interest rate and δ is the fixed depreciation rate. The maximization problem of agent i is as follows:

$$\mathbb{E}\sum_{t=0}^{\infty}\beta^t\frac{(c_t)^{1-\gamma}-1}{1-\gamma}$$

subject to

$$c_t + k_{t+1} = R_t k_t + \left[(1-\tau_t)\,\bar{l}\epsilon_t + \mu\,(1-\epsilon_t)\right]W_t + (1-\delta)k_t$$

Here c_t is the consumption level chosen by agent i and k_t is the beginning-of-period capital.

Firms and the Government

The good market is competitive and the production follows a Cobb-Douglas form:

$$Y_t = Z_t K_t^{\alpha} \left(\bar{l} L_t\right)^{1-\alpha}$$

where K is the per capita capital, L is the employment rate, and $\alpha \in [0,1]$ is the capital sharing. Z_t is a binary aggregate productivity shock: $Z_t \in \{Z_b, Z_g\}$.

The government keep her budget balanced in each period by redistributing all taxation from the employed to the unemployed.

The firms' first-order optimality together with the government's budget constraint determine the prices and tax rate

$$R_t = \alpha Z_t \left(\frac{K_t}{\bar{l} L_t}\right)^{\alpha-1}, \quad W_t = (1-\alpha) Z_t \left(\frac{K_t}{\bar{l} L_t}\right)^{\alpha}, \quad \tau_t = \frac{\mu(1-L_t)}{\bar{l} L_t} \tag{1}$$

Shocks

The aggregate shock Z_t follows a first-order Markov structure. The idiosyncratic shock ϵ_t is correlated to the aggregate shock by following a first-order Markov structure conditional on the transition of Z_t, and confront to the law of the large number.

To be more concrete, the transition of (ϵ_t, Z_t) is characterized by a 4×4 transition matrix $\mathbf{\Pi}$, whose element $\pi_{\epsilon\epsilon' Z Z'}$ denotes the probability to state (ϵ', Z') if the current state is (ϵ, Z). The calibration is such that the unemployment rate $u_t = 1 - L_t$ is a function of the aggregate shock: $u(Z_b) = u_b > u(Z_g) = u_g$.

The Recursive Equilibrium

To analyze the recursive equilibrium, agents' state variables for decision-making are divided into individual states (k, ϵ) and aggregate states $(Z, \mathbf{\Gamma})$; the latter are used for computing current prices and predicting future prices. In a rational expectations equilibrium, $\mathbf{\Gamma}$ represents the joint distribution of agents over capital holdings k and employment status ϵ. Since ϵ is binary, the joint distribution function $\mathbf{\Gamma}(k, \epsilon)$ comprises two one-dimensional continuous distribution functions on k: $\mathbf{\Gamma}(k, 0)$ and $\mathbf{\Gamma}(k, 1)$. For simplicity, I assume that the agents observe only the marginal distribution of k. This assumption facilitates focusing on the continuous distribution function of k without the need to additionally consider ϵ.

As discussed, the motion of Z is characterized by the exogenous matrix $\mathbf{\Pi}$; the law of motion of $\mathbf{\Gamma}$ is denoted by $\mathbf{H} : \mathbf{\Gamma}' = \mathbf{H}(\mathbf{\Gamma}, Z, Z')$. The agents' problem can be therefore express recursively as

$$\mathbf{V}(k, \epsilon; Z, \mathbf{\Gamma}) = \max_{k'} \{\mathbf{U}(c) + \beta \mathbb{E}[\mathbf{V}(k', \epsilon'; Z, \mathbf{\Gamma}') \mid \epsilon, Z]\} \tag{2}$$

subject to

$$c + k' = Rk + [(1-\tau)\bar{l}\epsilon + \mu(1-\epsilon)]W + (1-\delta)k, \tag{3}$$

$$\epsilon', Z' \sim \mathbf{\Pi}(\epsilon, Z), \tag{4}$$

$$\mathbf{\Gamma}' = \mathbf{H}(\mathbf{\Gamma}, Z, Z'), \tag{5}$$

$$k' \geq 0 \tag{6}$$

For a particular value function $\mathbf{V}(k, \epsilon, Z, \mathbf{\Gamma})$, the corresponding policy function is denoted by $k' = \mathbf{g}(k, \epsilon, Z, \mathbf{\Gamma})$. The solutions to 2 subject to 3, 4, 5, 6 are denoted as $\mathbf{V}^*(\cdot)$ and $\mathbf{g}^*(\cdot)$.

A recursive competitive equilibrium is then defined by a law of motion $\mathbf{H}(\cdot)$, a pair of individual functions $\mathbf{V}^*(\cdot)$ and $\mathbf{g}^*(\cdot)$, and the prices system(R, W, τ) such that (i) $\mathbf{V}^*(\cdot)$ and $\mathbf{g}^*(\cdot)$ solve the agents' optimality problem, (ii) R, W and τ are competitive and (iii) $\mathbf{H}(\cdot)$ is generated by $\mathbf{g}^*(\cdot)$.

The main computational challenge for this problem lies in the fact that the endogenous state variable $\mathbf{\Gamma}$ is a function, thus constituting an infinite-dimensional object. Additionally, the law of motion for the aggregates, $\mathbf{H}$, is a nontrivial function, compounded by the fact that the policy functions $\mathbf{g}$ are nonlinear and exhibit a kink due to the borrowing constraint.

3 Comparison of Computational Strategies

Below, I briefly discuss two primary frameworks from the literature that address these challenges that mentioned in the introduction: the KS framework that characterizes the distribution using a select number of moments and tracks only the motion of these moments and the NN framework that represents the distribution through a finite (and potentially large) number of simulated agents, with aggregate motion being the cumulative transition of all simulated agents adhering to a specific policy. Subsequently, I introduce my computational strategy, the operator learning framework, proposed in this paper.

KS Framework

The KS framework proposes approximating $\mathbf{\Gamma}$ by a finite set of moments $\mathbf{m} = (m_1, \ldots, m_I)$. The aggregate motion is then approximated by the motions of these moments: $\mathbf{H}_m : \mathbf{m}' = \mathbf{H}_m(\mathbf{m}, Z, Z')$. For instance, in Krusell and Smith [1998], I is chosen to be 1, so $\mathbf{m} = (K)$, meaning that only the mean of capital is used to characterize the distribution. In the economic model studied in this paper, denote the policy function implementing the KS framework as $\mathbf{g}_{\text{KS}}(k, \epsilon, Z, K)$, which is the solution to the following:

$$\mathbf{V}(k, \epsilon; Z, K) = \max_{k'} \{\mathbf{U}(c) + \beta \mathbb{E}[\mathbf{V}(k', \epsilon'; Z', K') \mid \epsilon, Z]\} \tag{7}$$

subject to

$$\begin{aligned} \log K' &= a_0 + a_1 \log K, && \text{if} \quad Z = Z_b, \\ \log K' &= b_0 + b_1 \log K, && \text{if} \quad Z = Z_g \end{aligned} \tag{8}$$

along with the budget constraint (3), the transitions of exogenous shocks (4), and the borrowing constraint (6). The KS framework implements a nested fixed-point optimization scheme that alternates between solving the agents' problem given the belief about the law of motion of moments and updating the belief given the agents' policy.

Conceptually, the KS algorithm approximates a computationally intractable infinite-dimensional problem by a computationally tractable low-dimensional problem. It is elegant, intuitively understandable, and fast. However, it has two main limitations: Theoretically, it implicitly imposes the bounded rationality assumption that agents observe only the set of moments and perceive only the motion of those moments. Practically, the moment selection is arbitrary; the sufficiency for the first I moments to characterize the distribution is case-dependent, and if I is large enough, the dimensionality of the problem becomes too high, making the grid-based projection method computationally inefficient. The discussion is summarized in the Table 2 below.

Pros	**Cons**
• Makes the problem tractable • Intuitive • Fast	• Assumes bounded rationality • Arbitrary moment selection

Table 2: Summary of KS Framework

NN Framework

The NN framework, instead, incorporates the full information of the distribution into the state variables using the individual states of all simulated agents. This transforms the infinite-dimensional original problem into a possibly high but finite-dimensional problem, where the dimensionality depends on the number of agents simulated. The aggregate law of motion is then approximated by the transitions of simulated agents based on a policy function. For the model in Section 2, denote the policy function implementing the NN framework as $\mathbf{g}_{\text{NN}}(k, \epsilon, Z, K)$, which is the solution to the following recursive form of the agents' problem:

$$\mathbf{V}(k, \epsilon; Z, \{k\}_{i=1}^{N}) = \max_{k'} \left\{ \mathbf{U}(c) + \beta \mathbb{E}\left[\mathbf{V}(k', \epsilon'; Z', \{k'\}_{i=1}^{N}) \mid Z \right] \right\} \tag{9}$$

subject to

$$k'_i = \mathbf{g}(k_i, \epsilon_i, Z, \{k\}_{i=1}^{N}) \quad \text{for} \quad i = 1, \ldots, N \tag{10}$$

and the budget constraint (3), the transitions of exogenous shocks (4), and the borrowing constraint (6).

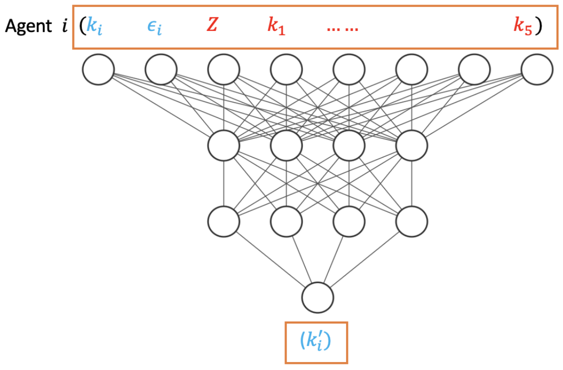
Since the number of agents is assumed to be large enough to approximate the continuous distribution function, to overcome the curse of dimensionality, the literature proposes using a feed-forward neural network as the function approximator for $\mathbf{g}_{\text{NN}}$. The feed-forward neural network has been shown to perform well in model reduction by condensing information and extracting features from the input to the hidden layers; see Goodfellow et al. [2016] for a review.

The NN framework addresses the issues in the KS framework, keeping the consequential high-dimensional problem tractable by adopting deep learning techniques. However, there are a few limitations that hinder computational efficiency. First, there is a trade-off in the choice of N: an increase in N leads to a better approximation of the continuous distribution function but also to a proportionally larger size of the network. More specifically, the size of the input layer grows linearly with N. If N is sufficiently large and the size of the hidden layers remains fixed, the network size grows roughly linearly with N. Figure 1 demonstrates this.

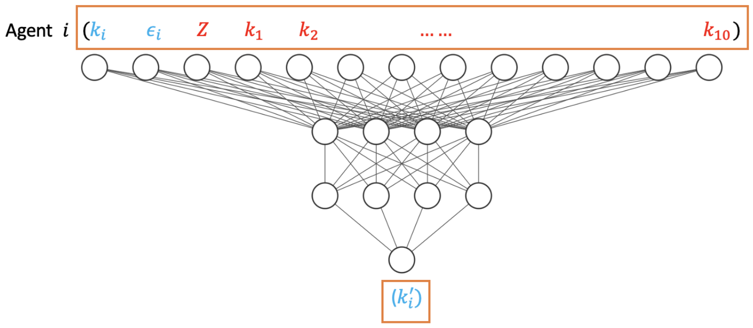
Moreover, in simulations where all agents transition to the next states based on the policy parameterized by the neural network (and in training, if all agents' states are used as training data), the computational cost is quadratic in N since there are more agents (rows) and a higher-dimensional state (columns) for each agent. Figure 2 demonstrates this.

The scaling from the two aspects discussed above makes the computational costs heavily dependent on N. The second limitation is that the NN framework solution is tied to a chosen N; the neural network architecture, especially the input layer dimension, is fixed once chosen. This inflexibility can cause issues when, for example, a larger number of simulated agents is required for evaluating the solution, or if the model setup includes overlapping-generation features so the population size may vary over time.

Finally, the discussed NN framework does not consider two features of the data structure: permutation-invariance and aggregation-sharing. Permutation-invariance means that the policy function $\mathbf{g}_{\text{NN}}$ should be invariant under permutations of the agents' ordering. However, the plug-in vector $\{k\}_{i=1}^{N}$ does not encode this information, so it would take a significant amount of computational time for the neural network to learn this pattern. Aggregation-sharing means that all simulated agents share the same economic environment, and they

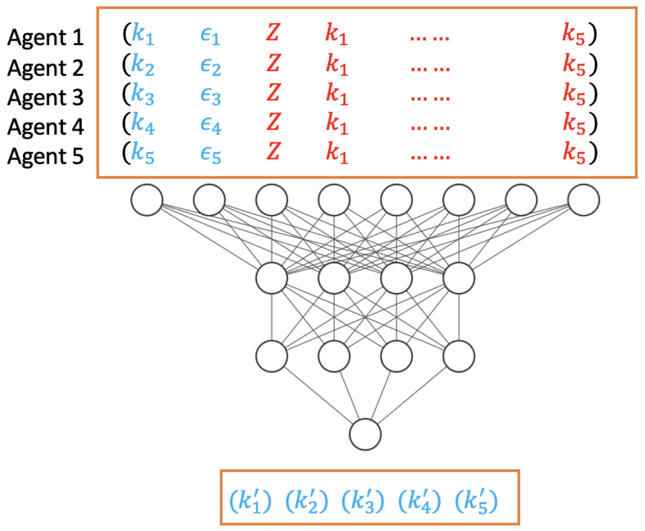


(a) 5 simulated agents

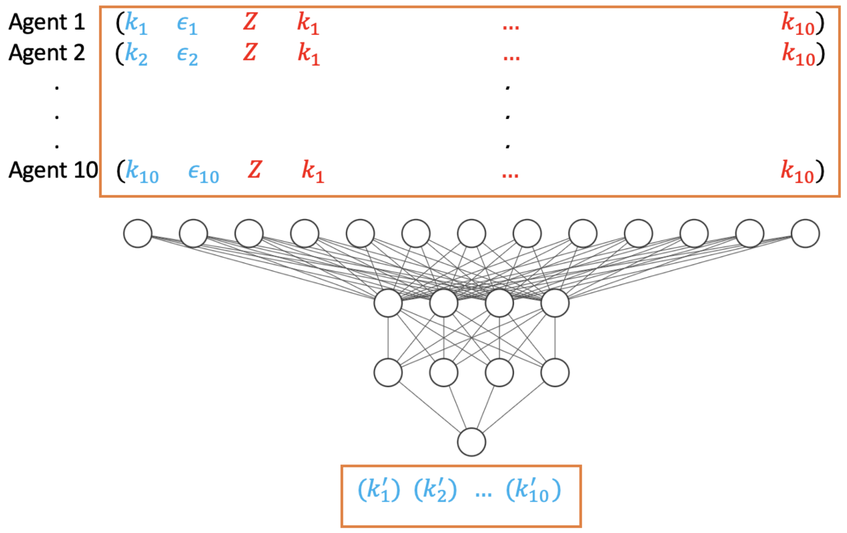


(b) 10 simulated agents

Figure 1: An illustration of the neural network size variation with the number of simulated agents. The above figure shows a network with 5 simulated agents, while the figure below depicts a larger network with 10 simulated agents, demonstrating the increase in the size of the input layer corresponding to the number of simulated agents, as discussed in the main text.


Agent 1 $(k_1 \quad \epsilon_1 \quad Z \quad k_1 \quad \ldots\ldots \quad k_5)$
Agent 2 $(k_2 \quad \epsilon_2 \quad Z \quad k_1 \quad \ldots\ldots \quad k_5)$
Agent 3 $(k_3 \quad \epsilon_3 \quad Z \quad k_1 \quad \ldots\ldots \quad k_5)$
Agent 4 $(k_4 \quad \epsilon_4 \quad Z \quad k_1 \quad \ldots\ldots \quad k_5)$
Agent 5 $(k_5 \quad \epsilon_5 \quad Z \quad k_1 \quad \ldots\ldots \quad k_5)$
$(k_1') \; (k_2') \; (k_3') \; (k_4') \; (k_5')$

(a) With 5 simulated agents.


Agent 1 $(k_1 \quad \epsilon_1 \quad Z \quad k_1 \quad \ldots \quad k_{10})$
Agent 2 $(k_2 \quad \epsilon_2 \quad Z \quad k_1 \quad \ldots \quad k_{10})$
Agent 10 $(k_{10} \quad \epsilon_{10} \quad Z \quad k_1 \quad \ldots \quad k_{10})$
$(k_1') \; (k_2') \; \ldots \; (k_{10}')$

(b) With 10 simulated agents.

Figure 2: Illustration of the quadratic computational cost in neural network simulations. The subfigure above demonstrates the network with 5 simulated agents, and the subfigure below with 10 simulated agents, emphasizing the increased computational burden as the number of agents, and thus the dimensionality, grows.

have a large portion of state variables the same as each other. However, the NN framework implicitly considers each agent independent when processing her state vector. The discussion is summarized in the Table 3 below.

Pros	**Cons**
• Addresses the issues in the KS	• Computational costs heavily depend on N • Solution is fixed to a given N • Does not use data structure features like permutation-invariance and aggregation-sharing

Table 3: Summary of the NN Framework

This Paper: Operator Learning Framework

The two frameworks discussed so far share a common intuition: they approximate the distribution function with a "distribution vector". This can be achieved either parsimoniously in parametric form (KS) or in a non-parametric form with sufficient information (NN), aiming to reduce the policy function back to a case where the arguments include only a finite-dimensional vector.

This paper proposes an alternative view by reformulating the policy function into an operator form:

$$\mathbf{g}(k, \epsilon, Z, \mathbf{\Gamma}) := \mathbf{G}(\mathbf{\Gamma})(k, \epsilon, Z) \tag{11}$$

Here, processing the state $(k, \epsilon, Z, \mathbf{\Gamma})$ into the policy function $\mathbf{g}$ is decomposed into two steps: firstly, an operator $\mathbf{G}$ takes the function $\mathbf{\Gamma}$ as its sole input and outputs another function $\mathbf{G}(\mathbf{\Gamma})$, which we call the "conditional policy function" henceforth; secondly, the vectors (k, ϵ, Z) serve as inputs to the function $\mathbf{G}(\mathbf{\Gamma})$.

While the input and output functions of the operator $\mathbf{G}$ do not necessarily need to be defined on the same domain, for simplicity, in the case studied, this alignment is effortlessly achieved. Given that both ϵ and Z are binary, $\mathbf{G}(\mathbf{\Gamma})(k, \epsilon, Z)$ can be interpreted as four separate 1-dimensional continuous functions of k, each defined on the same domain as the distribution function $\mathbf{\Gamma}(k)$.[2]

With this formulation, finding the solution for the policy function $\mathbf{g}$ translates to finding the solution for the operator $\mathbf{G}$ that satisfies Equation (11) for all states $(k, \epsilon, Z, \mathbf{\Gamma})$.

The question then arises: what are the benefits of this operator formulation? First,

[2]In other cases, e.g., when both ϵ and Z are continuous, we can consider $\mathbf{\Gamma}(k) := \mathbf{\Gamma}(k, \epsilon, Z)$ as a 3-dimensional continuous function, thereby defining $\mathbf{\Gamma}$ and $\mathbf{G}(\mathbf{\Gamma})$ on the same domain once again.

consider the parameterization of $\mathbf{G}$: if $\mathbf{G}$ is parameterized by a feed-forward neural network (or another functional form), there would be no distinction from the existing framework. To explore an alternative parameterization, we note the role of the distribution function $\mathbf{\Gamma}$ in agents' decision-making: it integrates from the individual level to the aggregate level. Ideally, the operator $\mathbf{G}$ should involve an integral operation to transform the input distribution function into the output conditional policy function.

This paper adopts the advanced machine learning technique known as the neural operator to parameterize $\mathbf{G}$, a generalization of neural networks that maps between infinite-dimensional function spaces. Intuitively, the neural operator is composed of linear integral operators and nonlinear activation functions. It boasts a critical property—discretization-invariance—meaning that the size of the neural operator remains constant regardless of the discretization of the input and output functions. This property addresses the first two disadvantages discussed in the NN framework context. Section 4 provides an introduction.

Furthermore, the operator formulation inherently includes the properties of permutation-invariance and aggregation-sharing. To illustrate, let's consider $\mathbf{\Gamma}$ explicitly as a cumulative distribution function (CDF), with an empirical representation consisting of two vectors: the input vector $\{\bar{k}_1, \dots, \bar{k}_J\}$ in ascending order, where $k_j \in [k_{\min}, k_{\max}]$ for $j = 1, \dots, J$, and the output vector $\{\frac{1}{J}, \dots, \frac{J}{J}\}$. This vector is constructed from the cross-sectional simulated agents $\{k_1, \dots, k_N\}$ with $J \leq N$. The transition from the non-ordered set of simulated agents $\{k_1, \dots, k_N\}$ to the ordered vector $\{\bar{k}_1, \dots, \bar{k}_J\}$ implicitly utilizes permutation-invariance.

The operator formulation's two-step processing of state variables directly exhibits aggregation-sharing: since only the distribution function is processed in the first step, which is shared by all simulated agents, this processing is done only once for all agents, rather than being redundantly repeated for each one. Figure 3 provides a clear illustration.

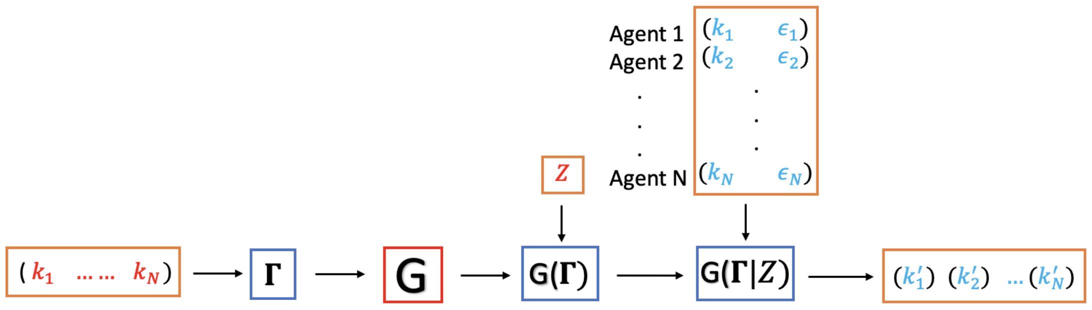


Figure 3: The processing flow of the neural operator, illustrating the transformation from the distribution function to the conditional policy function. This figure also demonstrates the permutation-invariance and aggregation-sharing properties discussed in the main text.

To conclude this section, the proposed neural operator formulation addresses the issues of the NN framework. The gains in computational efficiency resulting from this approach are demonstrated in the experimental results presented in Section 6.

4 Neural Operator

This section introduces the neural operator, a concept recently proposed in the machine learning literature [Li et al., 2020b,c,a]. The objective is to approximate a mapping between function spaces. In this discussion, I use notions that match the studied economic model, where functions have a 1-dimensional input $k \in [k_{\min}, k_{\max}]$ and produce a 1-dimensional vector output; for a comprehensive discussion, see Kovachki et al. [2021].

Consider a set of distribution functions $\mathcal{T} : \{\mathbf{\Gamma} \in \mathcal{T}, \mathbf{\Gamma} : [k_{\min}, k_{\max}] \to \mathbb{R}\}$, a set of conditional policy functions $\bar{\mathcal{G}}_{\mathbf{\Gamma}} : \{\bar{\mathbf{g}}_{\mathbf{\Gamma}} \in \bar{\mathcal{G}}_{\mathbf{\Gamma}}, \bar{\mathbf{g}}_{\mathbf{\Gamma}} : [k_{\min}, k_{\max}] \to [k_{\min}, k_{\max}]\}$, and a mapping $\mathbf{G} : \mathcal{T} \to \bar{\mathcal{G}}_{\mathbf{\Gamma}}$ such that $\bar{\mathbf{g}}_{\mathbf{\Gamma}}(k) = \mathbf{G}(\mathbf{\Gamma})(k)$. Our aim is to parameterize $\mathbf{G}$ with $\mathbf{G}_\theta$ to satisfy:

$$\mathbf{G}_\theta(\mathbf{\Gamma})(k) \approx \mathbf{G}(\mathbf{\Gamma})(k), \quad \text{for} \quad \mathbf{\Gamma} \in \mathcal{T} \quad \text{and} \quad k \in [k_{\min}, k_{\max}], \tag{12}$$

according to a particular measure.

A neural operator with L layers is a sequence of operations in function space, described by:

$$\bar{\mathbf{g}}_{\mathbf{\Gamma}}(k) := \mathbf{G}_\theta(\mathbf{\Gamma})(k) = (\mathcal{Q} \circ \sigma_L \circ \Phi_L \circ \ldots \circ \sigma_1 \circ \Phi_1 \circ \mathcal{P}(\mathbf{\Gamma}))(k). \tag{13}$$

In layer ℓ, $v_{\ell+1}(k) = \sigma_\ell \circ \Phi_\ell(v_\ell)(k)$. Here,

$$(\Phi_\ell v_\ell)(k) = \int_{q \in Q} \phi_\ell(k, q) v_\ell(q)\, dq + (W_\ell v_\ell)(k), \tag{14}$$

is an integral kernel operation plus a linear transformation W_ℓ. The function σ_ℓ is an element-wise nonlinear transformation, e.g., $\mathrm{ReLU}(x) = \max(x, 0)$ and $\mathrm{Sigmoid}(x) = \frac{1}{1+\exp(-x)}$. The mappings $\mathcal{P} : \mathbb{R} \to \mathbb{R}^{v_0}$ and $\mathcal{Q} : \mathbb{R}^{v_L} \to \mathbb{R}$ are element-wise lifting and projection mappings, respectively.

In practice, both the input function $\mathbf{\Gamma}(k)$ and the output function $\mathbf{G}(\mathbf{\Gamma})(k)$ are discretely represented at sufficiently many but finite locations $\{\bar{k}_1, \ldots, \bar{k}_J\} \in [k_{\min}, k_{\max}]$, referred to as "sensors" in the machine learning literature. The theoretical foundation of the neural operator's approximation capabilities is underpinned by the universal approximation theorem for operators, dating back to the seminal work by Chen and Chen [1995]. I adopt a version of the theorem as presented by Lu, Jin, and Karniadakis [2019]:

Theorem 1 (Universal Approximation Theorem for Operators [Lu, Jin, and Karniadakis, 2019])**.** *Let σ be a continuous non-polynomial activation function, and let X be a Banach space. Let $K_1 \subset X$ and $K_2 \subset \mathbb{R}^d$ be compact subsets of X and $\mathbb{R}^d$, respectively. Suppose*

V is a compact subset of $C(K_1)$, and G is a nonlinear continuous operator mapping V into $C(K_2)$. Then for any $\epsilon > 0$, there exist positive integers n, p, m, constants $c_i^p, \xi_{ij}^p, \theta_i^p, \zeta_p \in \mathbb{R}$, vectors $w_p \in \mathbb{R}^d$, and points $x_j \in K_1$ for $i = 1, \dots, I$, $p = 1, \dots, P$, $j = 1, \dots, J$, such that

$$\left| \mathbf{G}(\mathbf{\Gamma})(y) - \sum_{p=1}^{P} \left(\sum_{i=1}^{I} c_i^p \sigma \left(\sum_{j=1}^{J} \xi_{ij}^p \mathbf{\Gamma}(k_j) + \theta_i^p \right) \right) \sigma(w_p \cdot y + \zeta_p) \right| < \epsilon,$$

for all $\mathbf{\Gamma} \in V$ and $y \in K_2$.

Here the integral operator $\phi_\ell(k, q)$ is what aimed to learn from data. Various methods have been proposed in the literature to parameterize the integral operator $\phi_\ell(k, q)$ [Li et al., 2020b,c]. In this paper, I adopt the Fourier Neural Operator (FNO) framework [Li et al., 2020a], which parameterizes the kernel ϕ_ℓ in the Fourier domain rather than the spatial domain and uses the Fast Fourier Transform (FFT) to compute Equation 14. Specifically, let $\phi_\ell(k, q) = \phi_\ell(k - q)$, signifying a convolution, thereby enabling the use of the Convolution Theorem:

Theorem 2 (Convolution Theorem)**.** *Let f and g be functions with Fourier transforms $\mathcal{F}\{f\}$ and $\mathcal{F}\{g\}$, respectively. The Fourier transform of the convolution $f * g$ is the point-wise product of $\mathcal{F}\{f\}$ and $\mathcal{F}\{g\}$. Mathematically, this relationship is expressed as:*

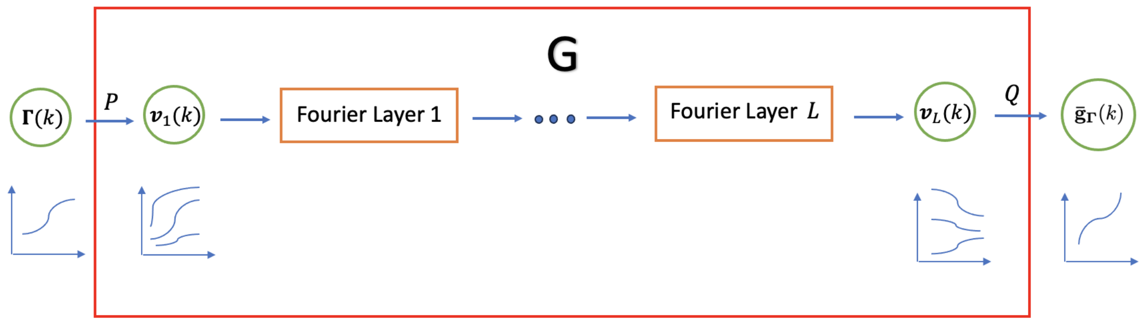
$$\mathcal{F}\{f * g\}(k) = \mathcal{F}\{f\}(k) \cdot \mathcal{F}\{g\}(k).$$

Denote $\mathcal{F}$ and $\mathcal{F}^{-1}$ as the Fourier transform and inverse Fourier transform, respectively. Li et al. [2020a] suggest parameterizing $\mathcal{F}(\phi_\ell)$ with a function R_θ^ℓ in the Fourier domain, parameterized by θ:

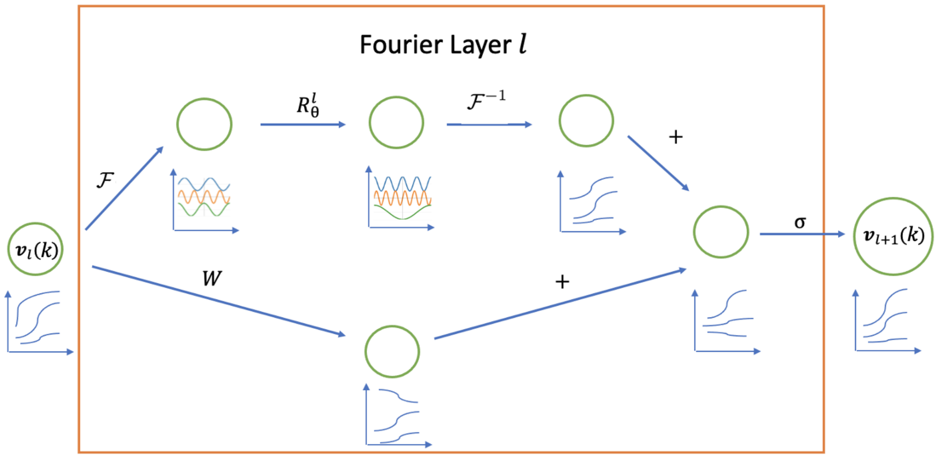
$$\int_{q \in Q} \phi_\ell(k, q) v_\ell(q)\, dq = \mathcal{F}^{-1} \left(\mathcal{F}(\phi_\ell) \cdot (\mathcal{F}(v_\ell)) \right)(k) \approx \mathcal{F}^{-1} \left(R_\theta^\ell \cdot (\mathcal{F}(v_\ell)) \right)(k).$$

Figure 4 illustrates the architecture of the neural operator with each hidden layer represented as a Fourier layer.

Since the kernel function ϕ_ℓ is implicitly assumed to be periodic when it is transformed into the Fourier domain, it admits a Fourier series expansion and the frequency modes κ are discrete. In practice, the Fourier series is truncated at a maximal number of modes $\kappa_{\max}$ for a finite-dimensional parameterization. Therefore, R_θ^ℓ is a $(\kappa_{\max} \times d_{v_\ell} \times d_{v_{\ell+1}})$ complex-valued tensor. For computational complexity, the bulk of the computational cost lies in computing the Fourier transform $\mathcal{F}(v_\ell)$ and its inverse. General Fourier transforms have a complexity of $\mathcal{O}(J^2)$; however, since the Fourier series is truncated, the complexity reduces to $\mathcal{O}(J\kappa_{\max})$. The FFT has a complexity of $\mathcal{O}(J \log J)$, but it requires a uniform discretization of the function.



(a) The overall architecture of the Fourier Neural Operator.



(b) A detailed view of a single Fourier layer within the architecture.

Figure 4: Illustration of the Fourier Neural Operator (FNO) architecture. Subfigure (a) shows the overall architecture, and Subfigure (b) provides a detailed view of the components within a single Fourier layer.

5 Implementation Details

Before delving into the implementation details, let me to reiterate the economic problem I intend to address, with a slight reformulation, as follows:

$$\mathbf{V}(k, \epsilon; Z, \mathbf{\Gamma}) = \max_{c,k'} \{\mathbf{U}(c) + \beta \mathbb{E}[\mathbf{V}(k', \epsilon'; Z, \mathbf{\Gamma}') \mid \epsilon, Z]\}$$

subject to

$$\begin{aligned} m &= \mathbf{M}(k, \epsilon, Z, \mathbf{\Gamma}) \\ c &= \mathbf{C}(k, \epsilon, Z, \mathbf{\Gamma}) \\ k' &= m - c \geq 0 \\ \epsilon', Z' &\sim \mathbf{\Pi}(\epsilon, Z) \\ \mathbf{\Gamma}' &= \mathbf{H}(\mathbf{\Gamma}, Z, Z') \end{aligned}$$

and the solution is a policy function $k' = \mathbf{g}(k, \epsilon; Z, \mathbf{\Gamma})$ such that

$$\mathbf{g}(k, \epsilon; Z, \mathbf{\Gamma}) = \arg\max_{k'} \{\mathbf{U}(c) + \beta \mathbb{E}[\mathbf{V}(k', \epsilon'; Z, \mathbf{\Gamma}') \mid \epsilon, Z]\} \tag{15}$$

The modification here is to represent the computation of the prices (R, W, τ) and, consequently, the agents' wealth m, as a function denoted by $\mathbf{M}$. The agents' policy is expressed in terms of the consumption policy $\mathbf{C}$. The capital in next period k' is then determined as the wealth minus consumption.

Approximation

Suppose there is a set of J grid points $\bar{\mathbf{k}} = (\bar{k}_1, ..., \bar{k}_J) \in [0, k_{max}]$, which is called "sensors" in the literature. Also there is cumulative-distribution function (CDF) of $k \in [0, k_{max}]$ denoted as $\tilde{\mathbf{F}}$. The construction of $\bar{\mathbf{k}}$ and $\tilde{\mathbf{F}}$ are discussed soon. I use the upper bar symbol to denote those variables that are fixed during the training and the upper tilde symbol for those that are amendable. $\mathbf{\Gamma}$ can be represented as a $2 \times J$ vectors $\hat{\mathbf{\Gamma}} \equiv (\bar{\mathbf{k}}, \tilde{\mathbf{F}}(\bar{\mathbf{k}}))$.

Define an alternative policy function as the fraction of wealth to consume, and rewrite it into an operator form:

$$\zeta = \frac{c}{m} = \frac{\mathbf{C}(k, \epsilon, Z, \hat{\mathbf{\Gamma}})}{\mathbf{M}(k, \epsilon, Z, \hat{\mathbf{\Gamma}})} \equiv \mathbf{g}(k, \epsilon, Z, \hat{\mathbf{\Gamma}}) = \mathbf{G}(\hat{\mathbf{\Gamma}})(k, \epsilon, Z) \tag{16}$$

so that $k' = (1-\zeta)m$. I use the neural operator $\mathbf{G}_\theta$, which parameters denoted as θ to approximate the operator $\mathbf{G}$:

$$\mathbf{g}_\theta(k,\epsilon,Z,\hat{\mathbf{\Gamma}}) \equiv \mathbf{G}_\theta(\hat{\mathbf{\Gamma}})(k,\epsilon,Z) \approx \mathbf{G}(\hat{\mathbf{\Gamma}})(k,\epsilon,Z)$$

That is, the neural operator $\mathbf{G}_\theta$ takes as input the $2\times J$ vectors $\hat{\mathbf{\Gamma}}$ and output the $4\times J$ vectors

$$\boldsymbol{\zeta}_J \equiv \begin{bmatrix} (\hat{\zeta}_j,\ldots,\hat{\zeta}_J)_{\epsilon=0,Z=Z_b} \\ (\hat{\zeta}_j,\ldots,\hat{\zeta}_J)_{\epsilon=1,Z=Z_b} \\ (\hat{\zeta}_j,\ldots,\hat{\zeta}_J)_{\epsilon=0,Z=Z_g} \\ (\hat{\zeta}_j,\ldots,\hat{\zeta}_J)_{\epsilon=1,Z=Z_g} \end{bmatrix}$$

The Objective Function

Since the outputs of true $\mathbf{G}^*$ are unobserved, the training of neural operator cannot be formulated as a supervised learning task with a norm of the difference $||\mathbf{G}_\theta(\hat{\mathbf{\Gamma}})(k,\epsilon,Z) - \mathbf{G}^*(\hat{\mathbf{\Gamma}})(k,\epsilon,Z)||$. To construct an objective function, recall that the unique solution that solves the Bellman equation must satisfy the derived Euler equation in the absence of borrowing constraint:

$$\frac{\mathrm{d}u}{\mathrm{d}c}(c) = \beta\mathbb{E}[(1-\delta+R')\frac{\mathrm{d}u}{\mathrm{d}c}(c')] \tag{17}$$

A standard approach in the literature is to define the relative Euler residual in, e.g., L_2 norm. For a given state $\omega \equiv (k,\epsilon,Z,\hat{\mathbf{\Gamma}})$ and a neural operator derived policy $\zeta = \mathbf{g}_\theta(\omega)$, define the unit-free Lagrange multiplier

$$h \equiv 1 - \frac{\beta\mathbb{E}[(1-\delta+r'\frac{\mathrm{d}u}{\mathrm{d}c}(\zeta' m'))}{\frac{\mathrm{d}u}{\mathrm{d}c}(\zeta m)} \tag{18}$$

with $m = \mathbf{M}(\omega), m' = \mathbf{M}(\omega'), \zeta' = \mathbf{g}_\theta(\omega')$ and R' the function of ω' according to 1. Then agents' optimality can be expressed in terms of the Kuhn-Tucker conditions:

$$h \geq 0, \quad k' \geq 0, \quad hk' = 0 \tag{19}$$

To get rid of the inequalities in 19 and construct an differentiable objective function, I rephrase the Kuhn-Tucker conditions in terms of the Fischer-Burmeister (FB) transformation with equality,

$$\Psi^{FB}(a,b) = a+b-\sqrt{a^2+b^2} = 0 \tag{20}$$

with $a = 1 - \zeta$ and $b = h$. In this case the objective function is

$$\xi(\omega, \theta) \equiv ||\Psi^{FB}(1 - \zeta, h)||^2 \tag{21}$$

Regarding the transition from ω to ω': The expectation operator $\mathbb{E}$ is taken over all possible (ϵ', Z') according to the transition matrix $\mathbf{\Pi}$. $k' = (1 - \zeta)m$. $\mathbf{\Gamma}' \equiv (\bar{\mathbf{k}}, \tilde{\mathbf{F}}'(\bar{\mathbf{k}}))$ with the $\tilde{\mathbf{F}}'$ the updated empirical CDF while keeping the grids $\bar{\mathbf{k}}$ fixed.

In iteration ℓ, suppose there is a set of observations $\{\omega : \omega \in \Omega^\ell\}$, then the objective function is

$$\Xi^\ell(\theta) = \frac{1}{|\Omega^\ell|} \sum_{\omega \in \Omega^\ell} \xi(\omega, \theta) \tag{22}$$

and the parameters in the neural operator are updated used the gradient descent method:

$$\theta^{\ell+1} \leftarrow \theta^\ell - \lambda^\ell \nabla_\theta \Xi(\theta^\ell) \tag{23}$$

What remains is the construction of $\bar{\mathbf{k}}, \tilde{\mathbf{F}}$ and Ω^ℓ.

Data Generating Process

Denote $\{\mathcal{T} : \mathbf{\Gamma} \in \mathcal{T}\}$ the set of all possible $\mathbf{\Gamma}$. A natural candidate for Ω_l is the set $\mathbf{\Omega} \equiv [0, k_{max}] \times \{0, 1\} \times \{Z_b, Z_g\} \times \mathcal{T}$. However, while grids can be taken on $[0, k_{max}] \times \{0, 1\} \times \{Z_b, Z_g\}$, it is not possible on the $\mathcal{T}$. Therefore I adopt stochastic simulation and concentrate only on those $\mathbf{\Gamma}$ in the ergodic set $\hat{\mathbf{\Gamma}}_{\tilde{g}}$ [Judd et al., 2011] , given a policy $\tilde{\mathbf{g}}$.

To be concrete, a cross section of N agents is simulated. In each period, the agents are characterized by the cross sectional capital $\tilde{\mathbf{k}} = (\tilde{k}^1, ..., \tilde{k}^N)$ in ascending order, the corresponding idiosyncratic risk $\tilde{\epsilon} = (\tilde{\epsilon}^1, ..., \tilde{\epsilon}^N)$, the aggregate shock $\tilde{Z}$, and a shared policy for transition $\tilde{\mathbf{g}}$.

The empirical CDF $\tilde{\mathbf{F}}(k)$ is the interpolation of mapping $\tilde{\mathbf{k}}$ to $(\frac{1}{N}, ..., \frac{N}{N})$:

$$\tilde{\mathbf{F}}(k) = \begin{cases} 0 & \text{if } k < \tilde{k}^1 \\ \frac{i}{N} + \left(\frac{k - \tilde{k}}{\tilde{k}^{i+1} - \tilde{k}}\right)\left(\frac{i+1}{N} - \frac{i}{N}\right) & \text{if } \tilde{k}^i \leq k < \tilde{k}^{i+1},\ i = 1, 2, \ldots, N-1 \\ 1 & \text{if } k \geq \tilde{k}^N \end{cases} \tag{24}$$

In each iteration ℓ, I run the stochastic simulation for T periods, discarding the beginning T_d periods for avoiding the impact of initial distribution, and select the time frame every t_{skip} periods to reduce the correlation between samples, so to obtain a set of $B = (T - T_d)/t_{skip}$

distributions $\{\tilde{\mathbf{F}}_b(\bar{\mathbf{k}})\}_{b=1}^B$.

Now turn to the discussion for the sampling of (k, ϵ, Z). In principle we can use the simulated data $(k^i, \epsilon^i, Z)_{i=1,b=1}^{N,B}$, which is in line with the pure stochastic simulation strategy discussed in Maliar et al. [2021]. Instead, I adopt the grids method with the rationale that the infinite-dimensional object Γ is already isolated using simulated simulation, with only three-dimension remains and two of them are just binary. In the absence of the concerns on the curse-of-dimensionality, the grids method gives higher accuracy comparing to the stochastic simulation method[Judd et al., 2011].

For the discretization of $k \in [0, k_{max}]$, since the policy function is highly non-linear around the borrowing constraint due to the presence of a kink-point[Aiyagari, 1994], I choose J grid points $\bar{\mathbf{k}} = (\bar{k}_1, ..., \bar{k}_J)$, following the polynomial rule[Maliar and Maliar, 2014]:

$$k_j = (\frac{j}{J})^d k_{max} \quad \text{for} \quad j = 1, ..., J \tag{25}$$

which enable more grids at the low capital levels and help in accurate approximation.

Note that the choice of grids are the same for both discretization of the agents' individual state and censoring of the distribution function. The rationals are two-folds: for implementation, the output function of the operator $\mathbf{G}_\theta$ are implicitly censoring on the same grids $\bar{\mathbf{k}}$ as the input function, so no extra effort is needed to query for new points; from the perspective of the economic theory, the fraction of hand-to-mouth agents is significant in aggregation due to their non-linear behaviors, therefore it is reasonable to have more censoring grids in the lower capital level when approximating the distribution function.

Pre-Training

A poor initialization of the policy operator is detrimental to the training, since the data (the distribution) is endogenously generated following the policy. To facilitate the convergence, I pre-train the operator using the solution of the following problem,

$$\mathbf{V}(k, \epsilon) = \max_{c,k'} \{\mathbf{U}(c) + \beta \mathbb{E}[\mathbf{V}(k', \epsilon') \mid \epsilon]\} \tag{26}$$

subject to

$$\begin{aligned} m &= \mathbf{M}(k, \epsilon) \\ c &= \mathbf{C}(k, \epsilon) \\ k' &= m - c \\ \epsilon' &\sim \mathbf{\Pi}(\epsilon) \\ k' &\geq 0 \end{aligned}$$

i.e. the version without aggregate shock. This model has a static aggregation, and hence all the prices are fixed and agents' state includes only the individual variables. Using policy function iteration this problem can be solved in seconds.

Denote the solution policy function as $\mathbf{g}_{static}$, that is, the economy in static in aggregate level. The pre-training is then simply a supervised learning task with the objective function

$$\xi_{SL}(\omega, \theta) = ||\frac{g_\theta(\omega)}{\mathbf{g}_{static}(\omega)} - 1||^2 \quad \text{for} \quad \omega \sim \mathbf{g}_{static} \tag{27}$$

$$\Xi^{\ell}_{SL}(\theta) = \frac{1}{|\Omega^\ell|} \sum_{\omega \in \Omega^\ell} \xi_{SL}(\omega, \theta) \tag{28}$$

That is, the ergodic set is generated by $\mathbf{g}_{static}$.

The Algorithm 1 below is the pseudo-code that fully describes the neural operator training algorithm. Note that the formal training process closely resembles the pre-training process, with two exceptions: 1) the objective function for optimization, and 2) the policy followed by the agents in the simulation.

6 Results

In this section, we present the training results. For benchmarking purposes, I set $N = 1,000$, as this is the maximum number of simulated agents in the experiments conducted by [Maliar, Maliar, and Winant, 2021] for the NN framework. To demonstrate that the proposed method can efficiently manage such a number of agents without the need for extremely high discretization, I set $J = 100$. Regarding the neural operator architecture, the hyperparameters are chosen with $n\_modes = 64$, indicating the number of modes retained in the Fourier layer for each dimension, and $hidden\_channels = 16$, which represents the width of the Fourier neural operator. It is important to note that these parameters were not extensively tuned; rather, they were selected to yield a comparable number of parameters to those in the NN framework with a feed-forward neural network architecture—specifically when managing 1,000 agents followed by a 64×64 hidden layer as per the experiments of [Maliar, Maliar, and Winant, 2021]. For comparison, the neural operator has 72,052 parameters, while the latter case encompasses 68,481 parameters.

Benchmark Results

In Figure 5, the training losses are plotted against computational time. As observed from the figure, the loss reduced to a level of 10^{-5} within approximately 5 minutes, aided by less than 1 minute of pre-training. This reduction corresponds to a fractional Euler error of less

Algorithm 1 Neural Operator Training Algorithm

```
Step 0: Initialization:
1. Fourier neural operator $\mathbf{G}_\theta$
2. Solution $\mathbf{g}_{\text{static}}$ of the static problem 26 such that $\zeta_{\text{static}} = \mathbf{g}_{\text{static}}(k, \epsilon)$ and $k'_{\text{static}} = \zeta_{\text{static}} m$

3. Set of $N$ agents with: aggregate shock sequence $(Z_0, \dots, Z_{T-1})$, panel of idiosyncratic shocks $(\epsilon_0^1, ..., \epsilon_0^N), ..., (\epsilon_{T-1}^1, ..., \epsilon_{T-1}^N)$, initial cross sectional capital holding $(\tilde{k}_0^1, \dots, \tilde{k}_0^N)$
4. Set of $J$ grids $\bar{\mathbf{k}} = (\bar{k}_1, ..., \bar{k}_J)$ as per the polynomial rule 25

Step 1: Pre-training:
for $\ell$ from 1 to $L_{\text{pre}}$ do
    1. Sampling the Distributions:
    for $t$ from 0 to $T-1$ do
        1-1. Construct $\tilde{\mathbf{F}}_t$ using $(\tilde{k}_t^1, \dots, \tilde{k}_t^N)$ as per 24 and $\tilde{\mathbf{\Gamma}}_t \equiv (\bar{\mathbf{k}}, \tilde{\mathbf{F}}_t(\bar{\mathbf{k}}))$
        1-2. Compute cross sectional wealth $(\tilde{m}_t^1, \dots, \tilde{m}_t^N)$ as per the model specification $m = \mathbf{M}(k, \epsilon, Z, \hat{\mathbf{\Gamma}})$
        1-3. Agents transit from $(k_t^1, \dots, k_t^N)$ to $(k_{t+1}^1, \dots, k_{t+1}^N)$ using $\mathbf{g}_{\text{static}}$
    end for
    2. Data Processing:
    2-1. Discard initial $T_d$ periods; select every $t_{\text{skip}}$ to obtain $\{\tilde{\mathbf{\Gamma}}_b, Z_b\}_{b=1}^B$
    2-2. Set initial capitals for next iteration: $(\tilde{k}_0^1, \dots, \tilde{k}_0^N) = (\tilde{k}_T^1, \dots, \tilde{k}_T^N)$
    3. Parameters Update:
    3-1 Set $\mathbf{G}_\theta$ to the train mode
    for $b$ from 1 to B do
        3-2 construct the data set $\{\omega : \omega \in \Omega_\ell\}$, where $\Omega_\ell = \bar{\mathbf{k}} \times \{0, 1\} \times \{Z_b\} \times \{\tilde{\mathbf{\Gamma}}_b\}$
        3-3.Update using objective functions 27 and 28
    end for
end for

Step 2: Training:
for $\ell$ from 1 to $L_{\text{train}}$ do
    1. Sampling the Distributions:
    Set $\mathbf{G}_\theta$ to the evaluation mode
    for $t$ from 0 to $T-1$ do
        Same as in Step 1 but agents transit using $g_{\theta^\ell}$
    end for
    2. Data Processing:
    Same as in Step 1
    3. Parameters Update:
    Set $\mathbf{G}_\theta$ to the train mode
    for $b$ from 1 to B do
        Same as in Step 1 but update using objective functions 21 and 22
    end for
end for
```

than one percent, which is commendable, especially for high-dimensional policy functions that exhibit a kink point, as highlighted by Maliar, Maliar, and Winant [2021].

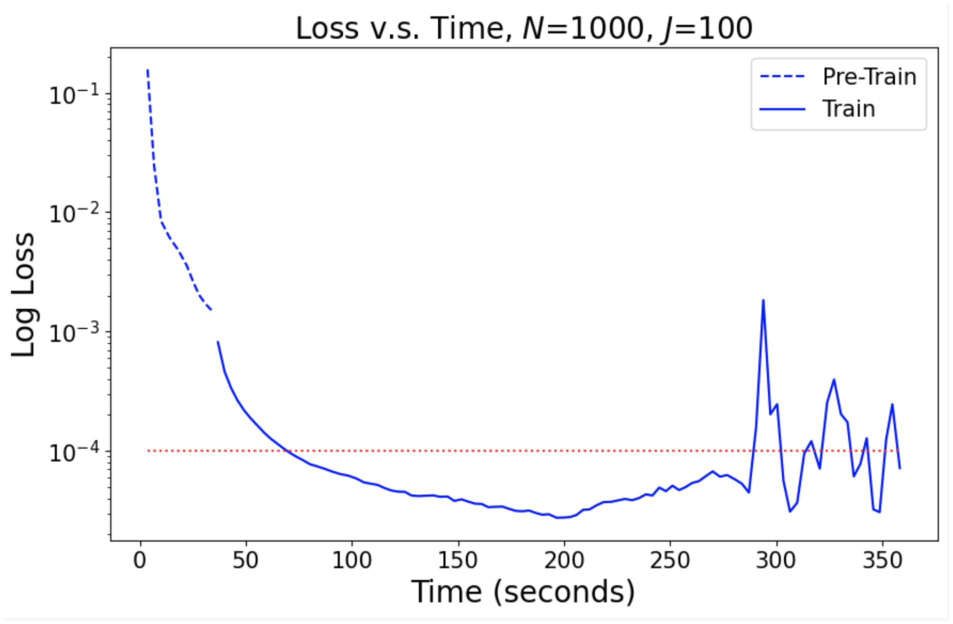


Figure 5: Training Loss vs. Time (seconds) for benchmark cases. The dotted line corresponds to the pre-training, and the solid line corresponds to the formal training.

Figure 6 offers a direct comparison between the proposed neural operator framework and the NN framework, with both frameworks undergoing 50 iterations for pre-training and 100 iterations for formal training. The neural operator framework reached a reasonable precision in just over 6 minutes, including less than 1 minute of pre-training. Conversely, for the NN framework, pre-training alone took approximately 6 minutes, and the subsequent formal training phase took an additional 10 minutes, ultimately resulting in lower precision.

Given that the policy function incorporates a distribution function as one of its inputs, providing a comprehensive view of the entire policy function is challenging. Instead, I have selected a random period cross-section from the simulated data to depict the distribution function $\mathbf{\Gamma}(k)$ and the conditional policy function $\mathbf{g}(k, \epsilon, Z; \mathbf{\Gamma})$. On the left side of Figure 7, the empirical cumulative distribution function (CDF) is plotted over a pre-determined interval for capital $k \in [k_{\min}, k_{\max}]$. The upper bound $k_{\max}$ is chosen to be sufficiently large, ensuring agents rarely approach this threshold during simulations. On the right, we illustrate four 1-dimensional functions of k associated with different combinations of individual and aggregate shocks, ϵ and Z. Since these four lines are nearly identical when plotted over the full interval $[0, k_{\max}]$, I have focused on a narrower range around the lower bounds. This zoomed-in view clearly showcases the kink points of the operator framework solution resulting from borrowing constraints when agents are unemployed, which is clear evidence

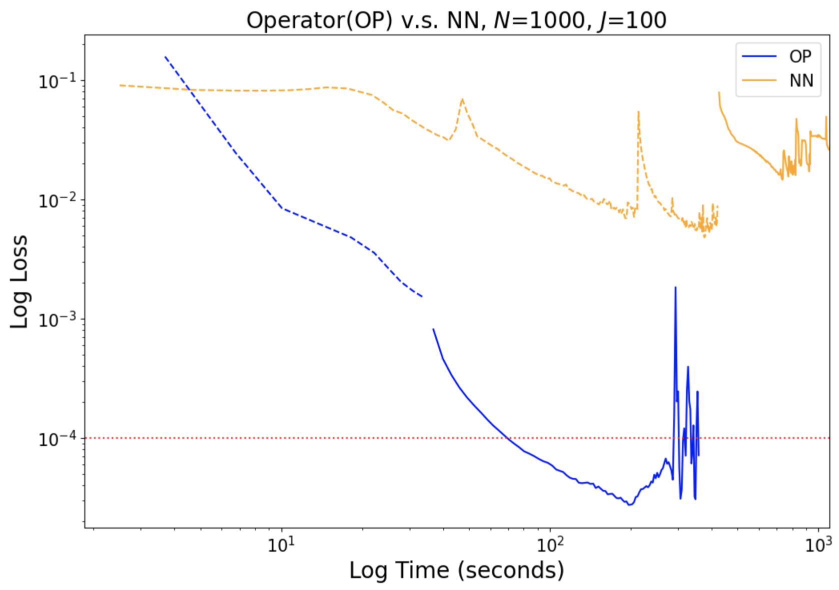


Figure 6: Comparison of Training Efficiency: Operator vs. NN. With an equal number of iterations (50 for pre-training and 100 for formal training), the neural operator framework achieves lower loss levels more quickly than the NN framework.

of successful training.

For reference, I have also plotted an alternative policy function, $g(k, \epsilon, Z, K)$, derived from the KS framework. Here, K is deduced from the empirical CDF shown on the left. Both policy functions closely resemble each other, underscoring the success of our training approach. It is worth noting that any discrepancies between the two do not necessarily reflect a failure in approximating the true policy function, as the solution from Krusell and Smith [1998] is itself an approximation.

To further illustrate that our policy function closely aligns with that of Krusell and Smith [1998], and to showcase the consistency in the ergodic sets of the economies, Figure 8 presents the simulated aggregate capital based on both policies. These simulations share the same initial distribution and identical sequences for both aggregate and idiosyncratic shocks.

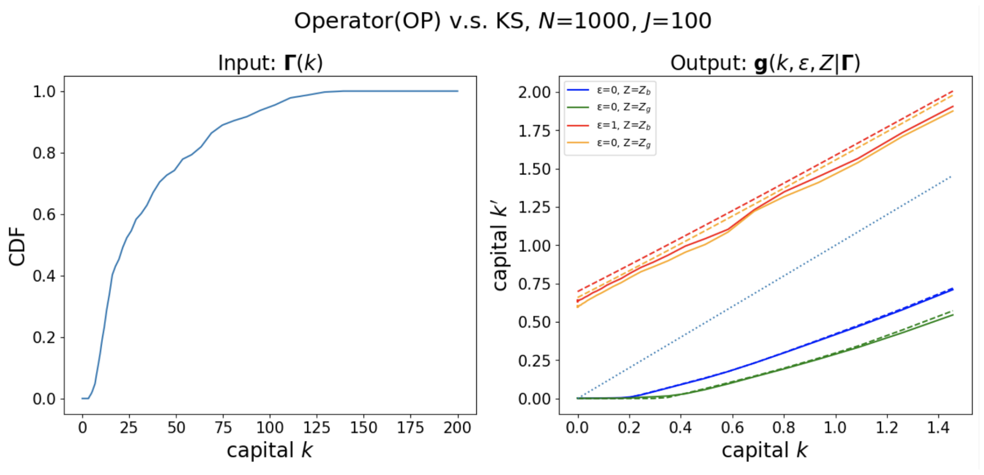


Figure 7: Comparison of the Operator and KS frameworks for an instance of distribution with $N = 1,000$. Left: Empirical CDF of capital $\mathbf{\Gamma}(k)$ from simulation. Right: Conditional policy function $\mathbf{g}(k, \epsilon, Z; \mathbf{\Gamma})$ for 4 cases of ϵ and Z, with kink points indicative of borrowing constraints. The dotted lines represent KS framework solutions for reference, demonstrating the operator framework's alignment with known benchmarks.

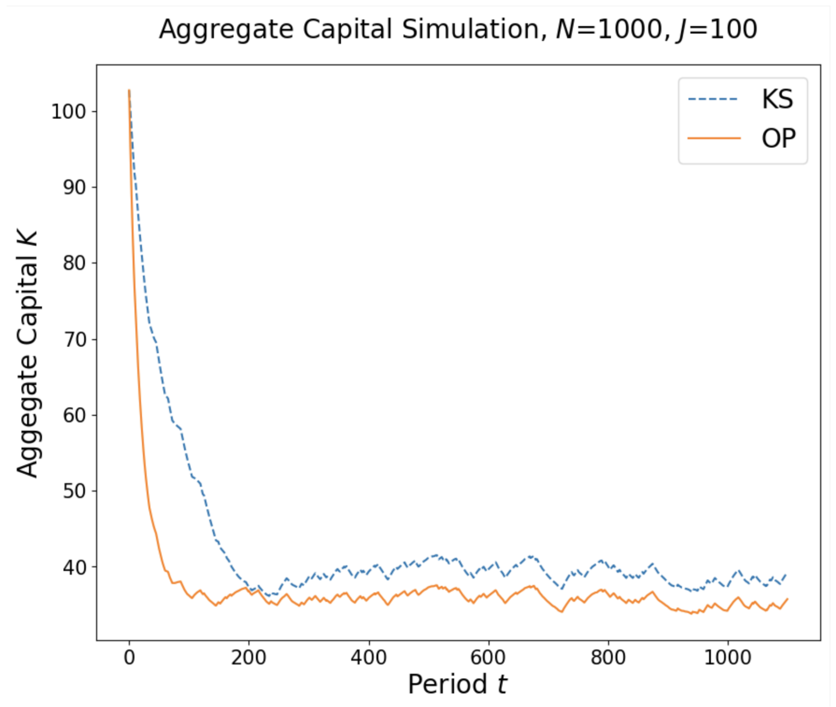


Figure 8: Simulated Aggregate Capitals: Two agent populations with identical initial capital holdings and shock sequences. Blue line: Simulation following policy from KS framework. Orange line: Simulation following policy from operator learning framework.

Robustness Check

Here, I compare experiments on cases with different N and J to assess how the variances contribute to the precision and computational time. Figure 9 indicates that, while keeping the discretization of functions at $J = 100$, increasing the number of simulated agents from $N = 1,000$ to $N = 10,000$ has an negligible effect on performance. More importantly, the computational time increases only marginally: this significantly demonstrates the simulation efficiency of the proposed framework, as it can handle an order of magnitude more agents with minimal cost. On the other hand, maintaining the same number of simulated agents at $N = 1,000$ and increasing the level of discretization of functions from $J = 100$ to $J = 1,000$ again shows a negligible effect on performance, which suggests that a discretization level of $J = 100$ is adequate for function approximation. The computational time increases roughly tenfold, displaying the linear scaling of cost with discretization level. Figure 10 presents another instance of the input distribution function and output conditional policy function, but for $N = 10,000$; comparing the left-hand side with that in Figure 7 demonstrates that increasing the number of simulated agents significantly impacts the construction of the distribution function.

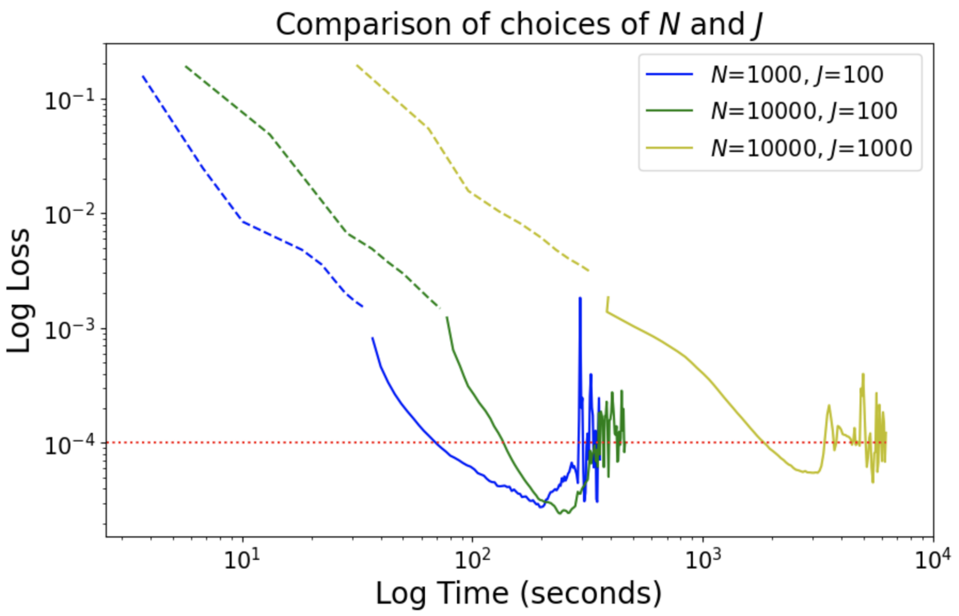


Figure 9: Training Loss vs. Time (seconds) for various N and J. The graph illustrates that increasing N from 1,000 to 10,000, with J fixed at 100, results in negligible performance changes and a slight computational time increase.

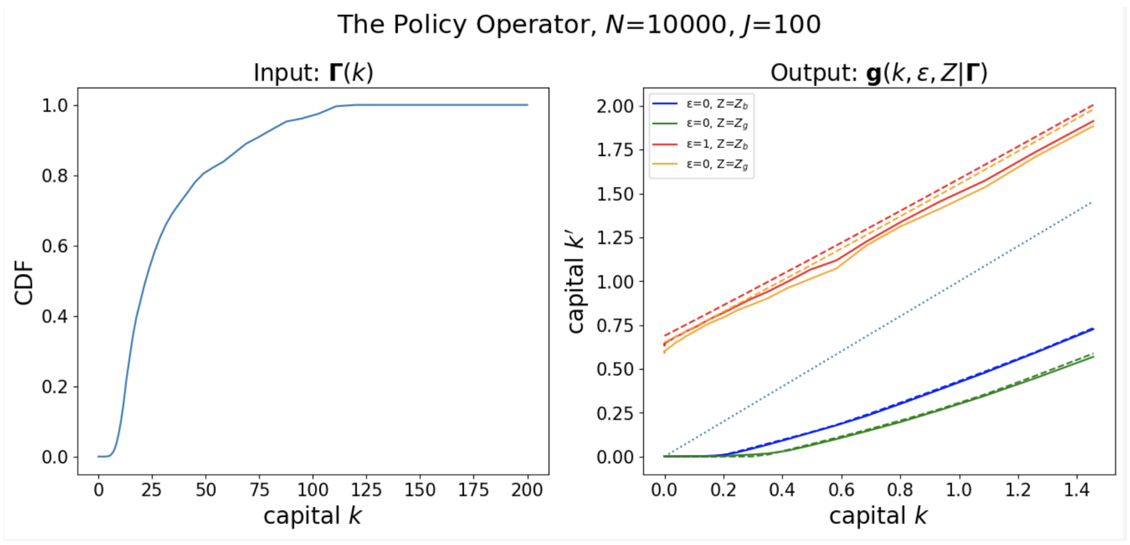


Figure 10: Operator vs. KS framework comparison for $N = 10,000$. The figure highlights that a larger N significantly refines the distribution function.

7 Conclusion

This paper's proposed solution framework marks a significant advancement in the field of dynamic macroeconomic modeling, particularly for models encompassing a continuum of heterogeneous agents and aggregate uncertainty. This framework, distinct from traditional methods, innovatively applies the neural operator—a cutting-edge architecture in machine learning—for the parameterization of the policy function. This method eschews the usual approximation of the distribution function with high-dimensional vectors of simulated agents, opting instead for an operator that adeptly maps between function spaces. Key to its success are its discretization-invariance, permutation-invariance, and aggregation-sharing properties. When tested against the Bewley-Huggett-Aiyagari model, which incorporates aggregate uncertainty and serves as a benchmark in computational economics, the framework not only efficiently managed simulations with tens of thousands of agents to precisely approximate the distribution function but also achieved solutions with less than one percent relative error. This was accomplished in a notably shorter computational time compared to traditional methods, underscoring the framework's potential to significantly enhance computational efficiency and accuracy in economic modeling.

References

S Rao Aiyagari. Uninsured idiosyncratic risk and aggregate saving. *The Quarterly Journal of Economics*, 109(3):659–684, 1994.

Marlon Azinovic, Luca Gaegauf, and Simon Scheidegger. Deep equilibrium nets. *International Economic Review*, 63(4):1471–1525, 2022.

Richard Blundell, Luigi Pistaferri, and Ian Preston. Consumption inequality and partial insurance. *American Economic Review*, 98(5):1887–1921, 2008.

Marco Cagetti and Mariacristina De Nardi. Wealth inequality: Data and models. *Macroeconomic dynamics*, 12(S2):285–313, 2008.

Mingli Chen, Andreas Joseph, Michael Kumhof, Xinlei Pan, Rui Shi, and Xuan Zhou. Deep reinforcement learning in a monetary model. *arXiv preprint arXiv:2104.09368*, 2021.

Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE transactions on neural networks*, 6(4):911–917, 1995.

Wouter J Den Haan and Pontus Rendahl. Solving the incomplete markets model with aggregate uncertainty using explicit aggregation. *Journal of Economic Dynamics and Control*, 34(1):69–78, 2010.

Wouter J Den Haan, Kenneth L Judd, and Michel Juillard. Computational suite of models with heterogeneous agents: Incomplete markets and aggregate uncertainty. *Journal of Economic Dynamics and Control*, 34(1):1–3, 2010.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis. Physics-informed deep neural operator networks. In *Machine Learning in Modeling and Simulation: Methods and Applications*, pages 219–254. Springer, 2023.

Jiequn Han and Yucheng Yang. Deepham: A global solution method for heterogeneous agent models with aggregate shocks. *arXiv preprint arXiv:2112.14377*, 2021.

Kenneth L Judd, Lilia Maliar, and Serguei Maliar. Numerically stable and accurate stochastic simulation approaches for solving dynamic economic models. *Quantitative Economics*, 2 (2):173–210, 2011.

Mahdi Ebrahimi Kahou, Jesús Fernández-Villaverde, Jesse Perla, and Arnav Sood. Exploiting symmetry in high-dimensional dynamic programming. Technical report, National Bureau of Economic Research, 2021.

Greg Kaplan, Benjamin Moll, and Giovanni L Violante. Monetary policy according to hank. *American Economic Review*, 108(3):697–743, 2018.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Dirk Krueger and Fabrizio Perri. Does income inequality lead to consumption inequality? evidence and theory. *The Review of Economic Studies*, 73(1):163–193, 2006.

Per Krusell and Anthony A Smith, Jr. Income and wealth heterogeneity in the macroeconomy. *Journal of political Economy*, 106(5):867–896, 1998.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33: 6755–6766, 2020c.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Lilia Maliar and Serguei Maliar. Numerical methods for large-scale dynamic economic models. In *Handbook of computational economics*, volume 3, pages 325–477. Elsevier, 2014.

Lilia Maliar and Serguei Maliar. Deep learning classification: Modeling discrete labor choice. *Journal of Economic Dynamics and Control*, 135:104295, 2022.

Lilia Maliar, Serguei Maliar, and Pablo Winant. Deep learning for solving dynamic economic models. *Journal of Monetary Economics*, 122:76–101, 2021.

Stephan Zheng, Alexander Trott, Sunil Srinivasa, Nikhil Naik, Melvin Gruesbeck, David C Parkes, and Richard Socher. The ai economist: Improving equality and productivity with ai-driven tax policies. *arXiv preprint arXiv:2004.13332*, 2020.

Stephan Zheng, Alexander Trott, Sunil Srinivasa, David C Parkes, and Richard Socher. The ai economist: Optimal economic policy design via two-level deep reinforcement learning. *arXiv preprint arXiv:2108.02755*, 2021.