ALIENs for Continuous Time Economies.[*]

Goutham Gopalakrishna[†], Yuntao Wu[‡]

December 13, 2024

Abstract

This paper extends the growing deep learning based literature on solving equilibrium economic models and introduces Active Learning Inspired Equilibrium Nets (ALIENs). The method is particularly tailored for continuous-time models involving high-dimensional state spaces, aggregate shocks, and nonlinear dynamics. ALIENs extend the deep learning methods to include time-stepping and active learning to transform non-linear problems into sequences of contraction mappings, ensuring stability and convergence. Active learning prioritizes computational efforts in economically significant regions of the state space, improving accuracy. The proposed methodology is validated across a few applications, from infinite-horizon heterogeneous agent models with free boundaries to high-dimensional asset pricing models. Additionally, the paper introduces *Deep-Macrofin+*, a numerical library that facilitates the implementation of these techniques for researchers.

Keywords: Deep learning, global dynamics, computational methods.

1 Introduction

The past decade has seen a surge in macroeconomic and asset pricing models to capture nonlinear global dynamics. Models in continuous time offer the ability to capture complex interactions due to their tractability and offer portfolio choices in closed form compared to the discrete time counterparts. While continuous-time

[*]We are very grateful to the comments and discussion from Markus Brunnermeier, Pierre Collin-Dufresne, Marlon Azinovic, Jonathan Payne, Moritz Lenel, Sebastian Merkel, Galo Nuno, Simon Scheidegger, and Yucheng Yang.

[†]University of Toronto, Rotman School of Management.
Email: goutham.gopalakrishna@rotman.utoronto.ca

[‡]University of Toronto, Electrical and Computer Engineering.
Email: winstonyt.wu@mail.utoronto.ca

models characterizing the global dynamics are certainly an improvement over models with linearized solutions, most of the papers resort to low-dimensional state spaces due to computational bottlenecks. The difficulty is amplified when the state variables are endogenous and correlated, and the equilibrium quantities exhibit stark nonlinearities. In problems with aggregate shocks and long-lived assets, the standard finite difference method breaks down even with two state variables, since it is difficult to preserve the monotonicity of finite difference schemes.[1] Therefore, it is not just the curse of dimensionality, but the combination of nonlinear dynamics and high dimensions that is problematic in these models. Recent advances have used neural networks that alleviate some of these problems.[2] Neural network methods work in theory but are often hard to implement in practice. The flexibility that the neural network methods provide is both a blessing and a curse since there are many hyperparameter choices to make that affect the neural network approximations to equilibrium quantities.

This paper adds to the existing neural networks based numerical literature by proposing a deep learning method with high stability and convergence properties through active learning. The methodology proposed is tailored to solve continuous-time equilibrium economic models that feature aggregate shocks, long-lived assets, and high nonlinearities in policy and value functions. The contribution of this paper is twofold. First, we show through examples how a time-stepping based method enhances the stability of continuous-time economic models when the value and policy functions are approximated using deep neural networks with multilayered perceptrons (MLPs) and recently developed Kolmogorov Arnold Networks (KANs). The enhanced properties arise because the time-stepping procedure transforms the numerical problem to resemble a contraction mapping. We formalize this using the contraction mapping principle and apply the technique to a class of low- and high-dimensional problems where policy functions are highly nonlinear. The results show that our solution technique performs better than both the traditional finite difference method and the basic neural network method. In particular, active learning plays a critical role in achieving convergence by strategically selecting training points that guide the algorithm to learn equilibrium relationships within the most economically relevant areas of the state space. Second, the paper offers a numerical library, *Deep-Macrofin+*, powered with active learning. Compared to packages with traditional numerical methods like finite difference, it has the potential to solve higher dimensional models, with less restrictions in the types of derivatives in HJB equations. Compared with existing neural network based packages, it is

[1]Preserving monotonocity is important when there are cross partial derivatives in the HJB equation. See, for example, D'avernas, Petersen and Vandeweyer (2023) and Merkel (2020) for details.

[2]See, for example, Duarte, Duarte and Silva (2024), Maliar, Maliar and Winant (2021), Azinovic, Gaegauf and Scheidegger (2022), Han, Yang and E (2021), Gu, Laurière, Merkel and Payne (2024), Huang (2023), among others.

more targeted at economic models and integrates various active learning techniques for better convergence. Lastly, it offers a more user-friendly approach with latex and string equation inputs.[3] To the best of our knowledge, this is the first paper to use neural networks with a time-stepping scheme and active learning, and offer a user-friendly toolbox tailored for high dimensional equilibrium economic models.

While our method follows the deep learning-based solution techniques in the literature in employing neural networks that are trained to solve a supervised learning problem, there are two main differences. First, we implement a time-stepping scheme with neural networks that ensures stability, especially in problems where policy functions are highly nonlinear. Second, the training points are sampled from the most important part of the subspace depending on the economic and computational complexity of the problem. We focus on two types of active sampling (i) "residual-based" and (ii) "loss weight-based". In the "residual-based" sampling, training points are drawn from regions where the training losses are large. That is, the subspace where equilibrium conditions are violated is given more importance. In the "loss weight-based" sampling, each training point is assigned dynamic weights over training iterations. The weights are assigned on the basis of the relative progression in the loss. While the residual-loss method focuses on the subspace with larger residual, the loss weight-based method focuses on the *change* in the loss over time. That is, individual losses in a multi-loss problem that decrease slower get a higher weight. The intuition is that the equilibrium relationships that are harder to learn relative to other relationships get a higher relative weight dynamically over the training iterations. Ultimately, both methods focus on drawing samples from the most relevant subspace for convergence dynamically without the need for manual intervention in the intermittent training procedure.

We demonstrate our technique using a few applications with varying degrees of complexity. In the first application, we solve an infinite-horizon heterogeneous agent macro-finance model with short-selling constraints that feature an endogenous free-boundary, rendering the policy functions non-smooth.[4] In theory, MLPs can be made infinitely deep or wide to minimize approximation errors. However, accurately approximating a function around a jump discontinuity is challenging, especially in equilibrium models that involve market clearing on long-lived assets. Our time-stepping-based method transforms the non-linear problem into a quasi-linear problem and solves the model as if there were a finite horizon. Convergence analysis shows that under certain conditions in the HJB equations, the neural network converges with sufficient accuracy.[5] The theoretical basis for the convergence comes from the fact

[3]Link to software, documentation and replication package can be found in Appendix E

[4]Approximations using deep neural networks often fail because the universal approximation theorem for MLPs with smooth activation functions assumes that the approximated functions and their partial derivatives are continuous. (Hornik, Stinchcombe and White (1989); Hornik (1991)). In contrast, the approximation theorem for non-smooth activation functions, such as ReLU, assumes that the approximated function is piecewise linear Petersen and Zech (2024).

[5]We present the conditions more precisely in the methodology section.

that the time-stepping scheme converts the problem into a sequence of contraction mappings.

In the second application, we solve the model without short-selling constraint but with contractible idiosyncratic and long-run risk.[6] The model features high non-linearities due to stochastic volatility and concentrated long-lived asset positions in one group of agents. This is a multi-loss problem with first-order conditions with respect to consumption and portfolio choices, goods market clearing, and capital market clearing conditions entering the loss function. We show how a loss weight-based active sampling leads to superior convergence by stabilizing the approximation of higher-order derivatives.

Lastly, we test our method in a high-dimensional model where we introduce asset heterogeneity a la Martin (2013). We show that our method solves up to 100 trees with an accuracy (MSE) of order 10^{-6}, where active learning leads to a better accuracy of policy functions. To understand why active learning helps in this high-dimensional model, we reduce the dimensionality of the same model and study a 3-tree version. Analyzing this model reveals that the residual-based active training is concentrated at the boundaries of the state space, where the equilibrium relationships are the most important and yet get violated when approximated with the basic neural network without active learning.

We build an active learning based numerical library called "Deep-Macrofin+" that facilitates researchers to solve such models with nonlinear policy functions without having to write deep learning training modules. The HJB equations and algebraic equations are treated as loss functions that the training algorithm optimizes to approximate the value and policy functions, following the core idea of deep learning-based solution methods in the literature. In addition to this standard feature, the tool incorporates two types of active sampling methods, a loss-weight-based method and a residual-based method. We demonstrate its usage in solving several macrofinance models in the GitHub repository[7], in addition to the ones in this paper.

Literature review: There have been many papers with deep learning-based techniques to solve discrete-time economic problems (Maliar and Maliar (2015), Azinovic et al. (2022), Maliar et al. (2021), Han et al. (2021), Bretscher, Fernández-Villaverde and Scheidegger (2022) etc.) Recently, few papers have emerged that solve continuous-time models by training neural networks on simulated points of discrete time approximations (Han, Jentzen and E (2018), Huang (2023), etc.). These techniques share the same spirit as the discrete-time methods in the sense that equilibrium functions are approximated on simulated paths after discretizing them. This paper fits within a growing yet nascent literature that solves continuous-time models by taking an analytic PDE approach. The papers closest in spirit are Duarte et al. (2024),

[6]This model builds on Di Tella (2017)

[7]The python library along with implementation details can be found in `https://github.com/rotmanfinhub/deep-macrofin`.

Gopalakrishna, Gu and Payne (2024), Gu et al. (2024), Sauzet (2021), and Payne, Rebei and Yang (2024). Duarte et al. (2024) takes a reinforcement learning-based approach by converting the equilibrium economic model into an optimal control problem and then solves it as a supervised learning problem. Our paper differs by introducing time-stepping and active learning, allowing for flexibility to tackle discontinuous policy functions and multi-loss objective functions leading to improved convergence properties. Since the time the paper has been made publicly available, the idea that smart sampling from the state space is required to improve convergence properties has proven to be useful for macro models. For example, Gu et al. (2024) finds that a sampling method similar to the residual-based active sampling in this paper leads to better convergence in a continuous-time version of the Krusell-Smith model. In Fernández-Villaverde, Hurtado and Nuño (2023), the law of motion of aggregate wealth is solved using neural network approximations, while value functions are solved using traditional finite difference schemes. Moreover, their model does not feature "long-lived" capital, which has proven to be tricky to handle in high dimensions with aggregate shocks (see Gopalakrishna et al. (2024), for example). The loss-weight based active sampling relates to Bretscher et al. (2022) where the weights in the loss function are normalized. The difference is that in this paper, the losses are computed based on the progression over epochs with a memory decay.

There is a substantial literature in computational physics and applied mathematics to approximate PDEs and HJBs using neural networks, starting from Sirignano and Spiliopoulos (2018) and Raissi, Perdikaris and Karniadakis (2019). Sirignano and Spiliopoulos (2018) proposes a deep Galerkin method to solve PDEs in high dimensions and incorporates Monte Carlo methods to compute second-order derivatives to speed up computation. In contrast to these papers, the framework that we propose is suited to solving problems in financial economics. For instance, many problems in macrofinance and asset pricing come with endogenous state variables that are often correlated - a feature that the models in applied mathematics do not typically deal with. Moreover, the non-linearity of the PDEs in the economic models comes from the fact that the advection, diffusion, linear, and cross-term coefficients of the PDE are endogenously dependent on the equilibrium policy functions. Lastly, the usage of active points in this paper relates to the literature on adaptive sparse grids that are concerned with a systematic way of generating the state-space grid. For example, Brumm and Scheidegger (2017) and Schaab and Zhang (2022), uses adaptive sparse grids to solve dynamic economic models, whereas Bungartz, Heinecke, Pflüger and Schraufstetter (2012) solves option pricing models using the finite element method.

While there have been many attempts at solving economic models using neural networks, the literature on providing a user-friendly toolbox to solve economic models has been sparse. Lu, Meng, Mao and Karniadakis (2021) offers a physics-informed neural network (PINN) library to solve both forward and inverse PDE problems using deep neural networks, but no active learning is implemented. D'avernas et al. (2023) provides a dedicated library to solve macro-finance equilibrium problems in continuous time with one or two state variables, but does not expand to higher dimensions or problems with more complicated equilibrium relations. Our toolbox fills this gap.

2 Methodology

In this section, we present the deep learning methodology. We start by briefly describing neural network approximations, and then formalize the time-stepping scheme. We then explain active sampling implementation and present algorithms.

2.1 Neural network approximations

Let $J(\boldsymbol{x})$ denote the function that takes as input $\boldsymbol{x}$ and needs approximation. Assume that $J : R^d \to R$ is continuous.[8] Let $\hat{J}(\boldsymbol{x})$ denote the neural network approximation represented as a parametric function built using a sequence of linear and non-linear transformations. That is, we have

$$\hat{J}(\boldsymbol{x}) = W^L(\sigma(W^{L-1}(\sigma(...(\sigma(W^0\boldsymbol{x} + b^0)) + b^{L-1})...))) + b^L,$$

where σ is the activation function, W^l is the weight matrix for the l-th layer, b^l is the bias vector for the l-th layer, and L is the number of layers. This is a feedforward neural network with d inputs, a single hidden layer of n hidden units, and a single output. There are other complex architectures that can be used, such as CNNs, RNNs, and Transformers, but we focus on the feedforward neural network for simplicity. The goal is to approximate any equilibrium function that is unknown in the economic model using the above architecture, and train it using the loss function defined as the residual of the equilibrium conditions. The loss function is model-specific and may contain multiple terms, such as the residual of the HJB equation, the residual of the boundary conditions, the residual of the market clearing conditions, etc. This is a supervised learning problem, where the training data is a set of points $(\boldsymbol{x}_i)$ that are drawn from the state space. This follows the large literature on using neural networks to solve equilibrium problems in economics. Our innovation is to incorporate time-stepping and active learning to improve the convergence properties of the training algorithm, as explained later in section 2.3.

[8]This is required to apply the universal approximation theorem of MLPs with continuous activation (*tanh*, *SiLu*, *SoftMax*, etc). To approximate the derivatives, then J needs to be continuous upto the order of derivatives we want to approximate (*e.g.* if we want to approximate J upto 2nd order derivative, then J needs to be C^2). When this condition is violated, MLPs will have poor approximations which we will get to in section 3.

2.2 Kolmogorov Arnold Networks

Kolmogorov Arnold Network (KAN) is a newly developed alternative of feedforward neural network proposed by Liu, Wang, Vaidya, Ruehle, Halverson, Soljačić, Hou and Tegmark (2024), based on the Kolmogorov–Arnold representation theorem, which posits that if $f : [0,1]^n \to \mathbb{R}$ is a multivariate continuous function, then f can be expressed as a finite composition of continuous functions of a single variable and the binary operation of addition:

$$f(x) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right),$$

where $\phi_{q,p} : [0,1] \to \mathbb{R}$, $\Phi_q : \mathbb{R} \to \mathbb{R}$. KAN is claimed to outperform MLP in terms of accuracy and interpretability. However, Shukla, Toscano, Wang, Zou and Karniadakis (2024) suggests that KAN requires further improvements to match MLP in solving PDEs due to its lack of robustness and computational parallelism. The network's implementation is complex and challenging to optimize efficiently. The functions $\phi(x)$ (subscripts omitted) are constructed using a base activation function $b(x)$[9] and a linear combination of B-spline functions $B_{i,k}(x)$, which are piecewise polynomial functions of order k:

$$\phi(x) = w_b b(x) + w_s \sum_i c_i B_{i,k}(x),$$

where w_b, w_s, and c_i are trainable parameters, and the values of k and the number of spline grids are user-defined. For our one-dimensional free boundary model, as shown in section 3, we use a one-input, one-output KAN with no hidden layer, $b(x) = SiLU(x)$, and $k = 3$ with 3 grids for the splines, which can be directly represented as $\phi(x)$.

2.3 Solution method

Let θ be the parameters of the neural network (weights of the matrices W and biases b), and $\mathcal{T}$ be the training data. The overall loss function is defined as[10]:

$$\mathcal{L}(\theta, \mathcal{T}) = \sum_i \lambda_i \mathcal{L}_i(\theta, \mathcal{T}),$$

where $\mathcal{L}_i(\theta, \mathcal{T})$ are the residuals for boundary/initial conditions, HJB equations, inequality constraints, and constraint-activated systems, and λ_is are corresponding weights. Under continuity assumptions, the neural network approximation converges to the equilibrium functions.

[9] $b(x)$ can be any common activation function, such as *SiLU*, *tanh*, etc.

[10] For simplicity, we assume all individual loss functions are L^2-loss, and $\|\cdot\| = \|\cdot\|_2$ represents the L^2-norm.

Neural network methods that are set up to minimize $\mathcal{L}(\theta, \mathcal{T})$ work with the information from the first order conditions of the model and HJB equations. Hence, by design, the approximations will not hold exactly at every point in the state space. The loss function $\mathcal{L}(\theta, \mathcal{T})$ measures how well the approximated solution Φ_θ satisfies the algebraic constraints, boundary conditions, initial conditions, HJB equations, and constraint-activated systems, but assumes no prior knowledge of the actual solution f. Assuming the following conditions hold: 1) the universal approximation theorem for neural networks; 2) smoothness of the neural network approximation with non-*ReLU* activations; 3) continuity properties of the PDEs; and 4) the existence of a unique solution to the PDE system, we can apply the approach outlined in Sirignano and Spiliopoulos (2018) to demonstrate that $\Phi_\theta \to f$. In practice, rather than a wide neural network with many neurons in a single layer, a deep neural network with multiple layers tends to achieve better convergence. The true solution f or its derivatives may not always be continuous or well-defined across the entire domain. They only need to be defined and continuous a.e. for the solution to exist. When using smooth activation functions, accurately approximating discontinuities becomes challenging.[11] We formalize this in Proposition 1. This situation is analogous to the Fourier approximation of functions with jump discontinuities, where the Gibbs phenomenon occurs, leading to oscillations around the discontinuity. In practice, since neural networks are neither infinitely wide nor infinitely deep, the approximation $\Phi(x)$ deviates from $f(x)$ within $B_\epsilon(x_0)$, a neighborhood around the discontinuity, much like how a low-order partial Fourier series fails to accurately approximate functions near jumps.

Proposition 1. *If $f : [0,1] \to \mathbb{R}$ has a jump discontinuity at $x_0 \in (0,1)$, then the neural network approximation $\Phi_\theta : [0,1] \to \mathbb{R}$ satisfies $\Phi_\theta(x_0) \to \dfrac{f(x_0^+) + f(x_0^-)}{2} := \lim_{h \to 0} \dfrac{f(x_0 + h) + f(x_0 - h)}{2}$.*

Proof. See Appendix D. $\square$

Our general approach to solving equilibrium models is to numerically solve them as if there was a finite horizon T, even for infinite horizon models. That is, in addition to the state variables of the model, a false transient time dimension t is added. In continuous time models, this amounts to modifying the HJB equations by adding a time derivative when computing the drifts using Ito's lemma. Terminal values at $t = T$ are set to constant at the beginning, and Φ is solved backward in time. Regardless of the terminal values chosen, we need to find a stationary point of the system such that the L^1-norm of time derivative $\|\nabla_t \Phi\|_1 = 0$. Since the algebraic constraints need to be satisfied for all time t, the only thing that needs to be modified is the way we solve HJB equations by incorporating time derivatives. Algorithm 1 outlines the

[11] This is demonstrated using a free boundary model as example in section 3.

Algorithm 1 Time Stepping Scheme

Input: X: state variables with time t,
$J_i : X \to \mathbb{R}$: agent value variables,
$E_j : X \to \mathbb{R}$: endogenous variables,
Output: Trained approximations $\hat{J}_i$, $\hat{E}_j$.

```
1: $\tau \leftarrow 0$, $\hat{J}_{i,\tau=0} = 1$, $\hat{E}_{i,\tau=0} = 1$ {Initialize as constant}
2: **while** True **do**
3:     Sample $X = (x_0, ..., x_n, t)$ from domain ($x_0, ..., x_n$ are defined by the problem domain, $t \in [0, 1]$)
4:     Embed boundary conditions: $J_{i,\tau+1}(t = 1) = \hat{J}_{i,\tau}$, $E_{i,\tau+1}(t = 1) = \hat{E}_{i,\tau}$ {At maximum time, the functions should satisfy the value from the previous step}
5:     **while** True **do**
6:         Update variables using neural networks, with $\frac{\partial J_i}{\partial t}$, and $\frac{\partial E_i}{\partial t}$ integrated.
7:         Compute loss on boundary conditions, endogenous equations, HJB equations and systems
8:         Compute total loss
9:         **if** inner_iter $\geq$ max_inner_loop OR inner loss converges **then**
10:            break
11:        **end if**
12:    **end while**
13:    $\hat{J}_{i,\tau+1} \leftarrow J_{i,\tau+1}(t = 0)$, $\hat{E}_{i,\tau+1} \leftarrow E_{i,\tau+1}(t = 0), \tau \leftarrow \tau + 1$
14:    **if** outer_iter $\geq$ max_outer_loop OR $\hat{J}_i$, $\hat{E}_i$ converge **then**
15:        break
16:    **end if**
17: **end while**
```

workflow of this time-stepping scheme with neural networks. The inner loop optimizes the neural networks based on the terminal values and the loss function $\mathcal{L}(\theta, \mathcal{T})$ for equilibrium, while the outer loop updates the time boundary condition at $t = T$ to ensure the time derivative vanishes. The underlying concept of this approach is the Value Function Iteration (VFI) algorithm. The Bellman operator B, defined such that $BU = \sup f + E_t(U)$, in the value function is a contraction map and possesses a unique fixed point U^* s.t. $BU^* = U^*$. The inner loop minimizes the L^2-norm of all loss functions over the entire domain of the state variables $X = (x_0, \ldots, x_n, t)$. Therefore, Φ_θ approximates the solution f, satisfying both the endogenous and HJB equations, with the time derivative $\frac{\partial \xi}{\partial t}$. To show that Φ_θ converges to the solution where the transient time derivative vanishes, we must demonstrate that $\|\nabla_t \Phi\|_1 \to 0$ through the outer loop iterations. We show this in the below proposition.

Proposition 2. *Define* $\mathfrak{C}^n = \left\{\Phi_\theta(x, t) : \mathbb{R}^{d+1} \to \mathbb{R}\right\}$ *as the class of neural networks with* $d+1$ *inputs and a single hidden layer of* n *hidden units. Under certain conditions,*

$\exists \Phi_\theta \in \mathfrak{C}^n$ s.t. as $n \to \infty$, after performing the training in Algorithm 1, Φ_θ converges and $\|\nabla_t \Phi\|_1 \to 0$.

Proof. See Appendix D. □

Note that the derivative in the auxiliary time dimension vanishes, yielding the exact solution to the original problem without time perturbation. With any reasonable initial guess for the value functions, the algorithm is guaranteed to converge, which is shown using the contraction mapping principle - for any initial condition/initial guess $x_0 \in X$, there exists a mapping P s.t. the sequence $x_0, x_1, ...,$ $x_{i+1} = P(x_i)$ converges to the fixed point x^*, as long as P is a contraction. Here, X can be any Banach space, including the L^1 space of functions.

2.4 Active learning

In principle, neural networks can be trained using points drawn uniformly from the state space in our time-stepping method. There are two potential bottlenecks with this kind of sampling. First, when policy functions are discontinuous and/or highly nonlinear, neural networks struggle to approximate them accurately. Second, when the number of state variables is large, it is computationally infeasible to sample points from the entire state space. Researchers in the past have tackled this problem by sampling from ergodic distribution of the state variables instead (Azinovic et al. (2022), Duarte et al. (2024), etc.) or sampling from the Kolmogorov forward equation (Gu et al. (2024)). In the first case, active learning improves the accuracy of the neural network approximation by focusing more on points that have violated equilibrium conditions as a result of discontinuity and/or nonlinearity of policy functions. In the second case, active learning helps reduce the computational burden by playing the same role *among* the training points drawn from the ergodic distribution. Thus, active learning is agnostic about the type of sampling but acts as a guiding mechanism for choosing the right training points. We implement two active learning methods (i) residual-based, and (ii) loss weight-based.

Residual-based: In uniform sampling, all positions in the domain have an equal likelihood of being sampled. However, equilibrium models may be more challenging to learn in certain regions, such as those with higher curvature or where regime shifts and discontinuities occur, as in the free boundary model. Residual-based sampling aims to improve the distribution of training points by focusing on areas where the residual of the system is large. These are the areas where equilibrium conditions are violated. After a fixed number of learning iterations, a dense set of points is sampled from the state space, and their losses are computed. Points with higher residuals are then added to the training set, allowing the model to focus on the regions where equilibrium conditions are violated. Algorithm 2 shows the process precisely.

Algorithm 2 Residual-based Learning

This only modifies the inner loop of Algorithm 1 (Line 5 to Line 12)
Additional Input: τ: number of times for residual-based sampling

```
 1: while True do
 2:     Update variables using neural networks, with $\frac{\partial J_i}{\partial t}$, and $\frac{\partial E_i}{\partial t}$ integrated.
 3:     Compute loss on boundary conditions, endogenous equations, HJB equations
        and systems
 4:     Compute total loss
 5:     if inner_iter % (max_inner_loop // $\tau$) == 0 then
 6:         Randomly sample 1000 points in the state space for loss computation
 7:         Retrieve $k$ = batch_size/$\tau$ points with highest loss, add to the training set
            for further inner loop iterations.
 8:     end if
 9:     if inner_iter $\geq$ max_inner_loop OR inner loss converges then
10:         break
11:     end if
12: end while
```

Loss weight-based: For equilibrium models involving multiple components in the loss function, some components may be more important than others. The loss weight-based method assigns weights to each loss item based on how well it progresses. While residual-based active sampling focuses on areas where equilibrium conditions have been violated in the current epoch, the loss weight-based method focuses on areas where the *change* in the loss has been low. Intuitively, this forces the neural network method to focus on equilibrium relationships that are harder to learn. The implementation is inspired by Bischof and Kraus (2021), where the weights are updated using the following equations:

$$\lambda_i^{bal}(t,t') = m\frac{\exp\left(\frac{\mathcal{L}_i(t)}{\mathcal{T}\mathcal{L}_i(t')}\right)}{\sum_{j=1}^{m}\exp\left(\frac{\mathcal{L}_j(t)}{\mathcal{T}\mathcal{L}_j(t')}\right)}$$
$$\lambda_i^{hist}(t) = \rho\lambda_i(t-1) + (1-\rho)\lambda_i^{bal}(t,0)$$
$$\lambda_i(t) = \alpha\lambda_i^{hist} + (1-\alpha)\lambda_i^{bal}(t,t-1)$$

m is the number of loss functions. $i \in \{1,...,m\}$ are indices for loss functions. $\mathcal{T}$ is softmax temperature. ρ is a Bernoulli random variable with $\mathbb{E}(\rho) \approx 1$. α is the exponential decay rate. As we can see, the weights λ_i^{bal} depend on the ratio $\left(\frac{\mathcal{L}_i(t)}{\mathcal{T}\mathcal{L}_i(t')}\right)$. The weight $\lambda_i^{bal}(t,t')$ represents the relative improvement of loss functions between epochs t' and t. The parameter ρ randomly decides whether to carry forward the loss weight $\lambda_i(t-1)$ from the previous epoch $t-1$, or to recompute the loss weights based on the relative improvement from the beginning, denoted as $\lambda_i^{bal}(t,0)$. The parameter α regulates the exponential decay of the weight, balancing between the previous epoch's weights and those calculated for the current epoch.

3 Applications

In this section, we apply our technique to solve three classes of equilibrium models that feature highly nonlinear policy functions. In each class of models, we demonstrate how our method leads to better convergence compared to either a traditional finite-difference method or a basic neural network method.

3.1 Free boundary model

Assume that time is continuous and the horizon is infinite (i.e. $t \in [0, \infty)$). There are two types of agents: households (h) and experts (e) who trade capital denoted by k_t that gets a stochastic depreciation shock driven by a standard Brownian motion Z_t. The output follows AK technology for simplicity. Agents have recursive preferences on consumption and choose optimal consumption, investment, and capital holdings by maximizing their lifetime utility subject to wealth constraints. Experts have a higher productivity rate of managing capital but cannot issue outside equity to households, reflecting the classic skin-in-the-game constraint found in macro-finance literature. In addition, agents cannot short-sell capital. These two assumptions mean that the market is incomplete, and there is an endogenous free-boundary in the state space at which the short-selling constraint binds. In the region where the short-selling constraint binds, the policy functions are linear. When the constraint does not bind, agents trade capital with each other and policy functions are highly nonlinear. These features are present in many heterogeneous agent macro-finance models with incomplete markets and are often notoriously difficult to solve using traditional methods. We first solve a simpler one-dimensional model since it allows us to compare against the finite difference solution. We then solve a variant of this model with more shocks with two state variables to demonstrate the performance of our model. We keep the number of state variables low in this section to demonstrate our technique, and then proceed to higher-dimensional models in Section 3.3. Note that even though the number of state variables is low, the difficulty in solving it numerically comes from the fact that the model has kinks in multiple policy functions, with an endogenous barrier to be pinned down in equilibrium. D'avernas et al. (2023) shows that a naive finite difference method does not work for this model even in two dimensions with correlated shocks, reflecting the difficulty in solving models with long-lived assets and aggregate shocks.

The model follows Brunnermeier and Sannikov (2014) and a detailed description is found in Appendix A.1. The goal is to demonstrate the difficulty in approximating policy functions with a kink in a model with an endogenous free boundary. In this simple 1D model, one remedy is to approximate functions with different neural networks in different regions in the state space. To reiterate, the purpose of using a simple 1D model is to demonstrate that our method provides a solution that is close to the finite-difference approximation. The method is scalable to high dimensions, which will be demonstrated in Section 3.3 where we solve up to 100 dimensions. The model and calibration details are relegated to the Appendix A. The two approaches we consider are as follows.

1. Basic Neural Network method: train one neural network for each value and policy function, trying to capture the discontinuity.
2. Our Method: Train two neural networks for each value and policy functions, one for each subdomain in the state space split by an endogenous free boundary.

3.1.1 Results

Table 1 shows the configurations used for the 1D free boundary model. The activation function for all models is $SiLU$ $(x\sigma(x))$. We use the Adam optimizer and train for 20,000 epochs for all MLP models, and we use a second-order LBFGS optimizer and train for 200 epochs for the KAN models. The MLP approximation converges in 10,000 epochs, whereas the KAN models converge in 20 epochs. The difference in the number of epochs is because the KANs have few parameters to optimize compared to the MLP method. The parsimony in the number of parameters is not only an advantage in terms of optimization speed but also in terms of interpretability. In the basic neural network method with KANs, q and ψ are approximated as hyperbolic cosine functions:

$$q = 1.0955 - 0.0009\cosh(8.6206\eta - 4.9742)$$
$$\psi = 1.4576 - 0.2371\cosh(4.1472\eta - 2.5626)$$

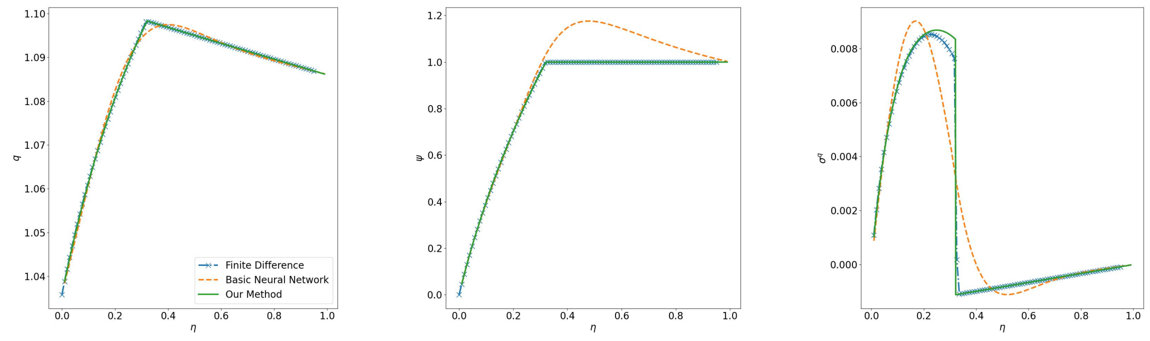
In our method with KANs instead of MLPs, the functions are approximated using a linear function on the region $\psi = 1$. That is, we have

$$q = \begin{cases} 1.0067 - 0.114\sin(1.9996\eta - 9.1262), & \psi < 1 \\ 1.1041 - 0.0182\eta, & \psi = 1 \end{cases}$$
$$\psi = -1.5368\sin(2.1795\eta - 9.3218) - 0.0984, \quad \psi < 1$$

Table 1 shows that the loss from our method and the basic neural network method are comparable. However, the basic neural network method has low accuracy, particularly around the free boundary. Figure 1 shows the equilibrium plots for q, ψ, σ^q using MLP models, together with finite-difference methods. Figure 9 in Appendix A shows the equilibrium plots using KAN instead of MLP. The return volatility function σ^q has a jump at the endogenous free boundary, and the price

Model Type	Hidden Units	Number of Parameters	Learning Rate	Total Loss
Basic Neural Network (MLP)	[30]*4	2881	1e-3	4.32e-4
Our Method (MLP)	[30]*4	2881	1e-3	1.48e-4 ($\psi < 1$) 5.61e-8 ($\psi = 1$)
Basic Neural Network (KAN)	[1,1]	19	1	4.89e-3
Our Method (KAN)	[1,1]	19	1	1.70e-3 ($\psi < 1$) 1.83e-9 ($\psi = 1$)

Table 1: 1D Free Boundary Model Configurations and Losses. The region $\psi < 1$ is where financial constraints bind and there is capital misallocation. The endogenous point where ψ becomes 1 is the free-boundary.



(a) Price **(b)** Capital Share: Experts **(c)** Price Return Diffusion

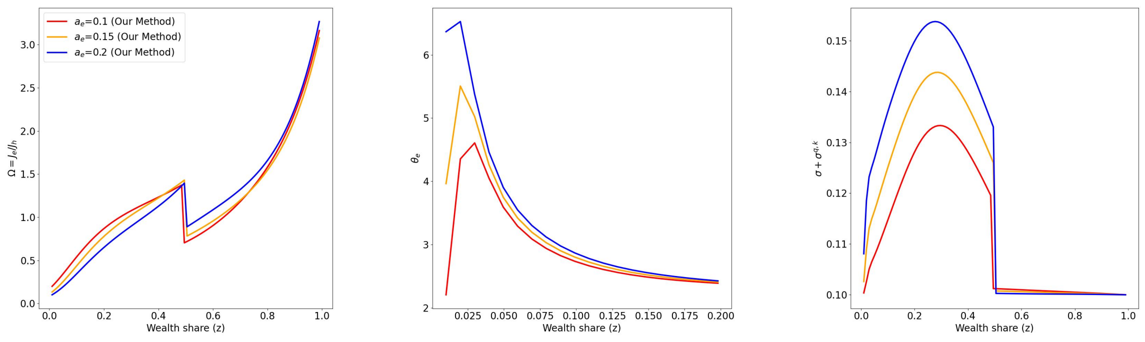
Figure 1: 1D Free Boundary Model Equilibrium Plots (MLP). The MSE between our method and finite difference solution is 1.99×10^{-8}, 1.60×10^{-6}, and 3.80×10^{-8} for the price, capital share, and return diffusion functions, respectively.

and capital share functions feature kinks. Compared with the basic neural network method, our method provides more accurate policy functions.

We next proceed to solving a slightly richer model with two state variables and an endogenous barrier. This model is an extension of the simple 1D model with an additional exogenous state variable, namely the productivity of experts $a_{e,t}$. We relegate the model details to the appendix and use this model as an apparatus to introduce our time-stepping method with active learning. The value functions are $J_{e,t}$ and $J_{h,t}$, and the policy functions are (q_t, χ_t, ψ_t). We parameterize each of these as a deep neural network object with MLPs. Each network contains 4 hidden layers with 64 neurons, activated by *SiLU*. At the end of the model, the outputs are restricted to be positive only with *SoftPlus*. The learning rates are set to 0.001 for all models. 500 points are randomly sampled in each training time step. For the basic neural network method, we train for 20,000 epochs in each region. For the time-stepping scheme, we train for 5,000 epochs in each time step and train for 50 time iterations. The 50 time iterations are unnecessary for regions 2 and 3, where the equilibrium conditions are simpler, but we use the same configuration for consistency. Table 2 presents the final loss of each model. Although the basic neural network method converges faster, the time-stepping scheme provides a better approximation, which can be seen in the capital share approximation in Figure 1 panel (b). The theoretical basis for the improved convergence properties comes from the fact that the time-stepping scheme transforms the non-linear problem into a sequence of contraction mappings. Our method, which augments time-stepping with active learning, has a slightly lower accuracy compared to the pure time-stepping scheme but converges much faster. This is because active learning enables the learning of equilibrium functions more accurately in economically interesting regions, which in this case is around the free boundary as seen in Figure 3. A comparison with the basic neural network method (without time stepping and/or active learning) in Figure 12 in the appendix reveals that our method has captured the endogenous jump in price return at the free boundary, which the basic neural network method fails to do.

Model Type	Financing Constraint	Shorting Constraint	No Constraint
Basic Neural Network	1.88e-4 (287)	2.48e-5 (186)	2.73e-5 (215)
Time-stepping	5.00e-6 (48398)	2.95e-6 (53793)	1.75e-7 (49364)
Our Method	2.58e-5 (9420)	1.48e-5 (8920)	2.12e-6 (11493)

Table 2: 2D Free Boundary Model Convergence Results. In the loss columns, the loss values are reported together with the number of epochs it takes for convergence in brackets.



(a) Value Functions **(b)** Portfolio Choice **(c)** Price Return Diffusion

Figure 2: 2D Free Boundary Model Equilibrium functions solved using our method.

3.2 Long-run risk model

Next, we solve a model that introduces complexity in terms of having a coupled system of PDEs and algebraic equations with a highly non-linear value function throughout the state space. The presence of idiosyncratic state and concentrated risk makes these functions nonlinear. Moreover, there are multiple components in the loss function, including a market clearing condition. Adding a risky capital market clearing condition in the loss is prone to instabilities (see, for example, Azinovic and Zemlicka (2023)) and in general makes it difficult for neural networks to converge

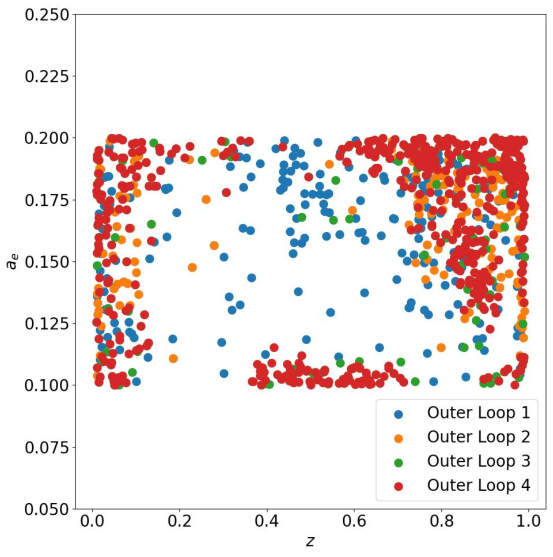
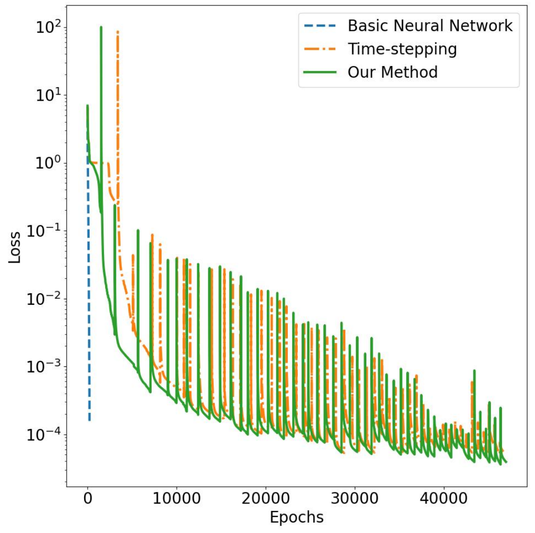


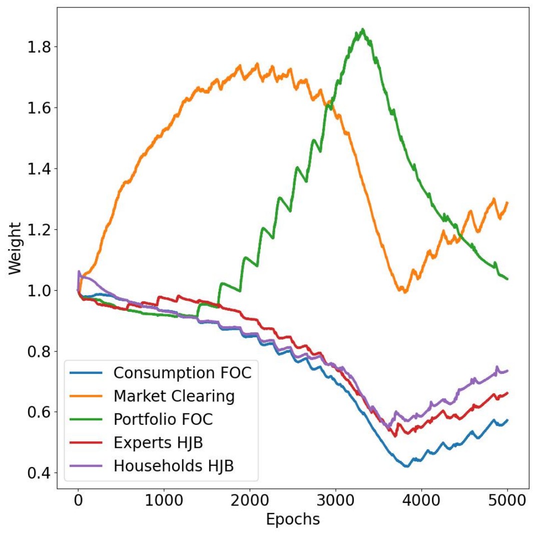
Figure 3: 2D Free Boundary Model Residual-based Sampled Points. In the first outer loop iteration, the residual-based sampling is mostly uniform across the state space. In the second iteration, the sampled points are mostly around the extreme wealth shares $z = 0$ and $z = 1$. In the third and fourth iterations, additional points are sampled around the free boundary $z \in [0.4, 0.5]$.

to the correct equilibrium function. To address this problem, we make a slight modification to active learning. Instead of sampling from regions where the loss is high as in the residual-based method, we turn to a weight-based method where the algorithm actively allocates different weights to the components of the loss function depending on the trajectory of the loss over previous iterations. For example, if the algorithm struggles to learn the market clearing condition consistently in prior iterations, then a loss weight-based method allocates higher weight to this particular loss component in the current iteration. Similarly, if the algorithm found it easier to learn the clearing condition, then it attaches a lower weight. This is active in the sense that the aggregate loss function dynamically adapts the weights and guides the learning process to reach the correct equilibrium solution. To show this concretely in action, we apply this to the long-run risk model that contains several components in the total loss - a) consumption first order condition, b) capital market clearing condition, c) portfolio first order condition, d) HJB equation of experts, and e) HJB equation of households. The parameters can be found in Appendix B, while we refer the readers to Di Tella (2017) for the detailed setup of the model. We parameterize the value functions ξ, ζ and endogenous variables q, r using neural networks. Each neural network contains 4 hidden layers with 30 neurons, activated by a *tanh* function. The final layer of ξ, ζ, q is restricted to be positive using a *SoftPlus* activation function. The learning rate is set to 0.001 for all models. A total of 500 points are randomly sampled in each time step for training. For the basic neural network method, we train for 20000 epochs. For our method with the time-stepping scheme, we train for 5000 epochs in each time step and train for 50 time iterations.

Figure 4a shows the progression of the loss over time. The basic neural network method appears to converge but in fact provides an inaccurate solution, as shown in Figure 13, especially when the level of stochastic volatility is low. For example, the price of risk is monotonic in the wealth share of experts (see Di Tella (2017)), but basic neural network training provides a non-monotonic approximation. As the wealth share of experts increases, the price of risk has to decrease as more wealth is concentrated in the expert, and the aggregate risk aversion in the economy decreases. This results in a lower endogenous risk ($\sigma + \sigma_p$) and a lower risk price (π), which is well captured by our method. The spikes in the progression of the loss in Figure 4a are due to time steps. Even within each time step, the active learning method has superior performance compared to other methods.



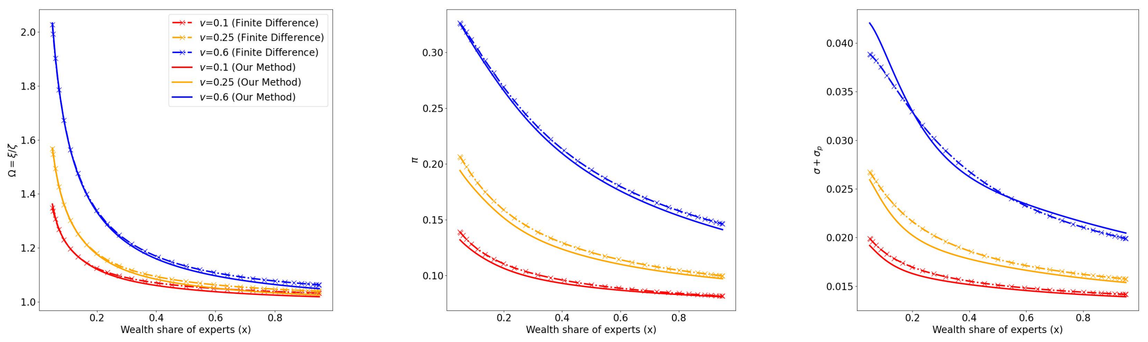
(a) Loss Convergence



(b) Loss Weights in the First Time Step

Figure 4: Long-run Risk Model Loss Progression. The loss for Basic Neural Network decreases to 1.57×10^{-4} in 222 epochs, but the approximated functions are not accurate. The loss for Time-stepping decreases to 5.23×10^{-5} in 46827 epochs. The loss for our method decreases to 3.92×10^{-5} in 46967 epochs.

Figure 4b presents the dynamic weights over epochs in the first time iteration. We can see that our method automatically assigns a higher weight to the market clearing condition at the beginning of the training process. As the market clearing condition is satisfied in the later stages, the portfolio first-order conditions are violated, thus receiving larger weights. After a sufficient number of iterations, the weights are close to each other. The loss weights typically do not add to one at convergence and are associated with the variance of the loss values.[12] At the end of each time step, we restart the weights and begin the dynamic updating again. The MSE with finite difference solution on the 961 referenced grids from Di Tella (2017) are 6.12×10^{-4} for $\hat{e}$ and 6.18×10^{-4} for $\hat{c}$. The fit for the other equilibrium quantities are provided in Figure 5.



(a) Value Functions　　**(b)** Price of Risk　　**(c)** Price Return Diffusion

Figure 5: Long-run risk Model Equilibrium Plots. The MSE between our method and finite difference solution is 1.08×10^{-4}, 2.34×10^{-5}, 8.80×10^{-7} for the ratio of value functions, price of risk, and price return diffusion, respectively.

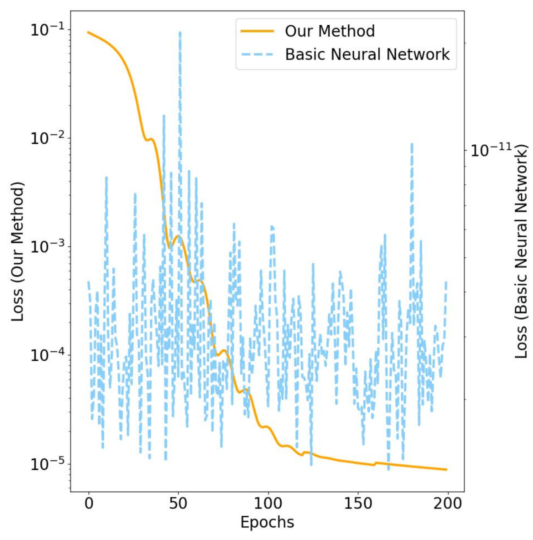
3.3 N trees model

In the final section, we solve a high-dimensional model with multiple risky assets, in the spirit of Martin (2013). The model details are provided in the Appendix C. In high dimensions, sampling from the entire state space can be computationally infeasible. Hence, we sample from the ergodic distribution of the state variables, similar to Azinovic et al. (2022) and Duarte et al. (2024). However, the idea of active sampling is still applicable. In particular, we use a residual-based active learning method to focus on the regions that are computationally more problematic. That is, we pad the training data that come from the ergodic distribution with points from the regions that have violated equilibrium conditions in previous iterations. Specifically, we do the following. We first start the model training without active learning. After a predefined number of epochs, we randomly sample a large set of test data points from the ergodic distribution of the state space and compute the residuals (loss) on each of the training data points. We select k points with the largest residuals to add to the training set. We iterate this procedure until we reach convergence. The procedure is given in Algorithm 2.

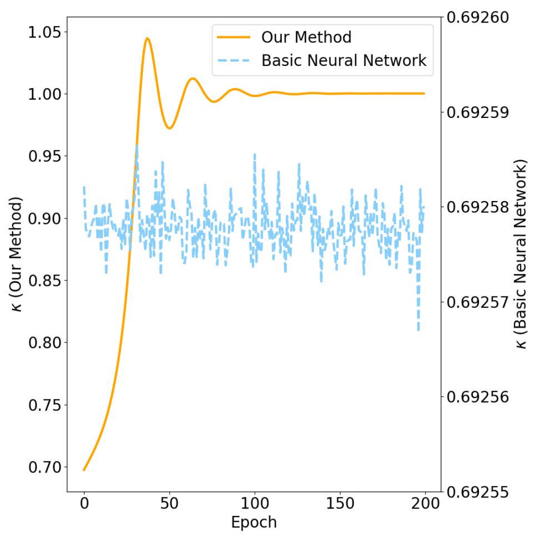
We solve a 100-tree model, where each tree is an exogenous stochastic process driven by uncorrelated Brownian shocks. We denote κ_i and q_i to be the dividend

[12]See, for example, Bischof and Kraus (2021) and Cipolla, Gal and Kendall (2018).

share of the tree. The average values of κ_i and q_i are 1 and 0.01 respectively. In the 100 trees, we also use ergodic sampling. Figure 6 shows the progression of loss and κ value over time. The dashed line in Figure 6b shows the approximation of the neural network to the dividend share κ that has a true solution equal to 1 in a 100-tree symmetric model. The basic neural network solution has trouble approximating the dividend share function, and the approximation error reduces when active learning is used in the solid line. The distribution plots can be found in Figure 20 in the appendix.



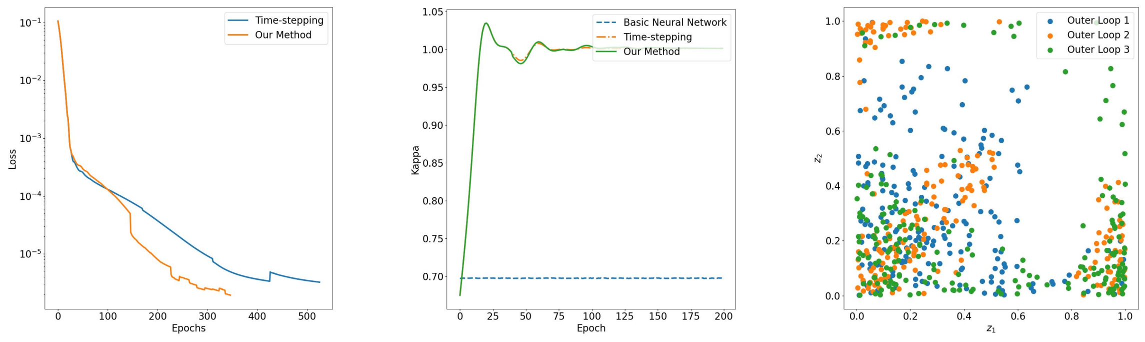
(a) Loss Convergence



(b) Convergence of κ

Figure 6: Progression of Time Stepping Scheme (100-Tree). In both plots, left axis is for our method and right axis is for basic neural network. We show the first 200 epochs for exposition. Basic neural network method gives a lower loss, but converges to the wrong solution ($\kappa \approx 0.6925$). Also both the loss and κ oscillates. Our method stabilizes faster in comparison.

Although the 100-dimensional example demonstrates that active learning helps converge faster and provides more accurate approximations, it does not tell us why it helps. We provide some insights into this using a 3-tree model. We employ a residual-based active sampling in which the training points are drawn uniformly from the state space. Figure 7a presents the progression of loss with and without active sampling. We see that active learning helps with faster convergence. More importantly, our method converges to the true solution compared to the basic neural network method. Figure 7c displays active training points over time iterations. The blue dots represent the training points at the first time iteration. They are uniformly distributed in the simplex of the state variables $z_1 \in [0, 1]$ and $z_2 \in [0, 1]$. The green dots are the points that are sampled from the regions that have violated the equilibrium conditions in



(a) Loss Convergence **(b)** Convergence of κ **(c)** Residual loss method

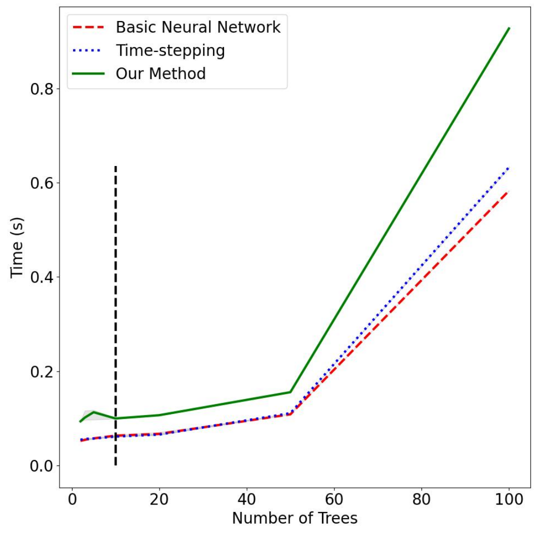
Figure 7: Progression of Time Stepping Scheme (3-Tree)

the first iteration. As the simulation progresses, the active points are concentrated near the boundary of the state space, where the approximation error is highest.

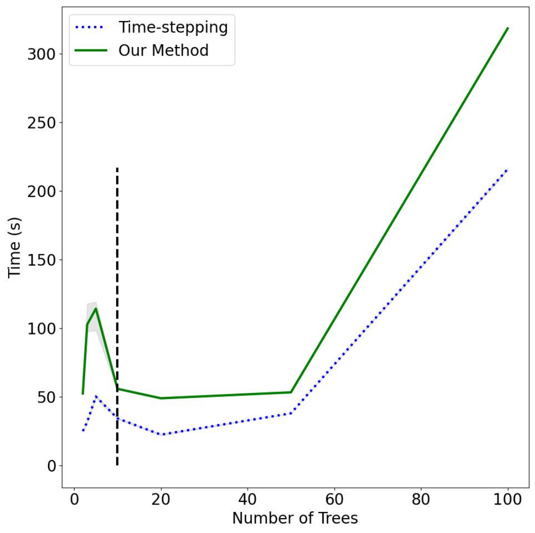
3.3.1 Training time and memory analysis

In this section, we analyze the training time and memory required for the N-tree model. We solve a total of 7 models for $N = 2, 3, 5, 10, 20, 50, 100$. We solve the model using the basic neural network method, the time-stepping method, and the time-stepping method with active learning (our method). The epoch training time includes the time for sampling, the forward pass of the neural network, loss computation, the backward propagation of the gradient, and the optimization step. The experiments are performed on Google Colab A100 GPU, and repeated 100 times[13]. The left panel of Figure 8 shows the average training time per epoch for each N-tree model, and the right panel shows the average training time to convergence for each N-tree model. The vertical dashed line is at $n = 10$, where we switch from uniform random sampling to ergodic log-normal sampling. The mean is plotted as lines, and the gray area shows the $[5\%, 95\%]$ region of the runtime. First, we observe that the training time does not explode as the number of trees grows large. This exemplifies the power of using neural networks as emphasized by many in the literature. The second thing to note is that the per epoch time taken for our method is similar to the basic neural network method, as seen in the left panel of Figure 8. The right panel shows the total training time. We see a hump-shaped pattern for our method because for low-dimensional models ($N < 6$), we use uniform sampling from the state space, which has a higher complexity. Beyond that, we use ergodic sampling which reduces the computational bottleneck and helps converge faster. The limitation of ergodic sampling is that we obtain a solution of the model where the economy lives in the long run, and not throughout the state space.

[13]We run 100 tree model on A100 GPU with 80GB VRAM on `www.runpod.io` and run the experiments one time only since the [5%,95%] for low dimensional models are tight with less variation.



(a) Training Time per Epoch



(b) Training Time (Total)

Figure 8: N-Tree Model Training Time.

	CUDA Memory (MB)						FLOPS ($\times 10^9$)					
	2-Tree	3-Tree	5-Tree	10-Tree	20-Tree	50-Tree	2-Tree	3-Tree	5-Tree	10-Tree	20-Tree	50-Tree
Basic	18.52	23.36	65.87	263.63	1142.16	8063.87	0.57	0.93	1.98	6.49	24.68	173.63
Time-stepping	14.84	30.38	78.06	289.02	1176.91	8129.29	0.73	1.15	2.30	7.10	25.93	177.53
Our Method	94.21	190.44	483.21	1812.23	7287.14		2.14	3.37	6.73	20.66	75.12	

Table 3: N-Tree Model Memory Consumption and FLOPs. Note that our method uses higher memory and FLOPs because we sample 5 times more points for residual computation. The memory and FLOPs used are about 6 times of time-stepping and basic methods. This aligns with the number of points we sample.

Table 3 shows the total memory usage and FLOPs for one full training iteration for each model for a batch size of 200 training points. The full iteration includes sampling stage, forward pass of the neural network, loss value computation, backward gradient propagation, and optimization. For residual-based sampling, we use 1000 random points and compute the residuals to select k points with the highest residual. The memory usage and FLOPs are computed using PyTorch's profiler, which has a tendency to overestimate the usage due to the additional computation from tracking done by the profiler.[14] One caveat is that due to the additional computation required by the profiler, the algorithm runs out of memory easily. Therefore, we report the memory usage and FLOPs for models up to $N = 20$ and $N = 50$ for with and without active learning, respectively.

[14]See `https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html`.

4 Conclusion

This paper introduces Active Learning Inspired Equilibrium Nets (ALIENs), a neural network-based method for solving equilibrium models in continuous-time economics. By combining time-stepping and active learning, ALIENs address computational challenges in high-dimensional, nonlinear models, transforming them into sequences of contraction mappings to ensure stability and convergence. We demonstrate our method using a few applications in macrofinance and asset pricing and show that it improves accuracy and convergence by focusing computational efforts on economically relevant regions. The accompanying *Deep-Macrofin+* library further supports the adoption of these techniques. Our method offers a robust framework for solving complex economic models, paving the way for future exploration of higher-dimensional problems and new neural network architectures.

Appendix

A Free Boundary Models

A.1 One-dimensional Benchmark

This section reports the parameters and system setup for the one-dimensional free boundary model. The model is based on Brunnermeier and Sannikov (2014).

State Variable: Experts' share of wealth: $z \in (0, 1)$, for implementation, use $[0.01, 0.99]$

Parameter	Definition	Value
σ	exogenous volatility of capital	0.1
δ_e	depreciation rate of capital for experts	0.05
δ_h	depreciation rate of capital for households	0.05
a	productivity of experts	0.11
a_h	productivity of households	0.07
ρ	discount rate of experts	0.06
r	discount rate of households	0.05
κ	adjustment cost parameter	2

Table 4: Free Boundary 1D Constant Parameters

We solve for the following system of PDEs when $\psi < 1$:

$$(r(1-\eta) + \rho\eta)q = \psi a_e + (1-\psi)a_h - \iota$$
$$\sigma_t^q + \sigma = \frac{\sigma}{1 - \frac{1}{q}\frac{\partial q}{\partial \eta}(\psi - \eta)}$$
$$(\sigma + \sigma_t^q)^2 \frac{q(\psi - \eta)}{\eta(1-\eta)} = a_e - a_h,$$

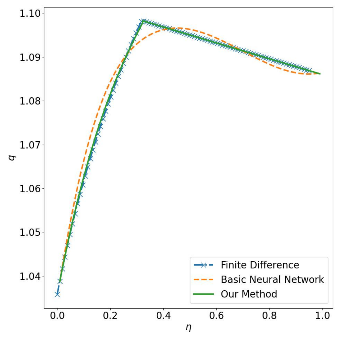
and the following PDE when $\psi = 1$:

$$(r(1-\eta) + \rho\eta)q = a_e - \iota$$
$$\sigma_t^q + \sigma = \frac{\sigma}{1 - \frac{1}{q}\frac{\partial q}{\partial \eta}(\psi - \eta)}.$$

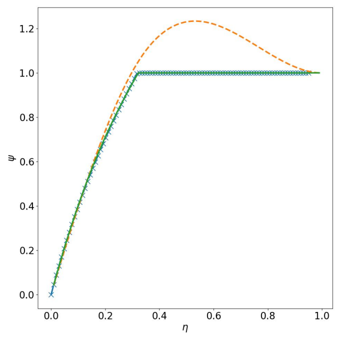
A.2 2D Free Boundary Model

State Variable:

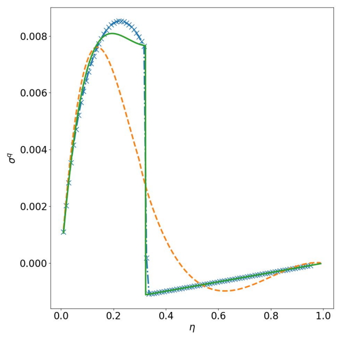
- Experts' share of wealth: $z \in (0, 1)$



(a) Price



(b) Capital Share: Experts



(c) Price Return Diffusion

Figure 9: 1D Free Boundary Model Equilibrium Plots (KAN)

- Productivity of experts: $a_e \in (fl, fu)$

For implementation, we use $[0.01, 0.99] \times [0.1, 0.2]$ (Therefore $\hat{a_e} = \frac{fl+fu}{2} = 0.15$).

Agents: Experts value function J_e and Households value function J_h.

Endogenous variables: Capital share of expert ψ, price q, and expert's inside equity share χ.

Parameter	Definition	Value
a_h	productivity of households	0.03
σ	exogenous volatility of capital	0.1
δ	depreciation rate of capital	0.05
p	persistent parameter for a_e	0.01
v	variation parameter for a_e	2.5
$\hat{a_e}$	mean of a_e	0.15
fl	min of a_e	0.1
fu	max of a_e	0.2
ϕ	correlation between dZ_t^k and dZ_t^a	0.5
γ	risk aversion	5
ρ	discount rate	0.05
κ	adjustment cost parameter	5
λ_d	birth/death rates of agents	0.03
$\bar{z}$	portion of experts born	0.1

Table 5: Free Boundary 2D Constant Parameters

Let $j \in \{e, h\}$ index the agents. The model follows from Brunnermeier and Sannikov (2016) with Epstein Zin utility and an additional state variable $a_{e,t}$ denoting productivity of experts.[15] Taking the additional state variable into account leads to the following growth rate equations:

$$
\begin{aligned}
\sigma^{q,k,1} &= \sigma + \sigma^{q,k} \\
\mu^{z} &= \frac{a_e - \iota}{q} - \hat{c}_e \\
&+ (\theta_e - 1)(\sigma^{q,k,1}(\zeta_e^1 - \sigma^{q,k,1}) + \sigma^{q,a}(\zeta_e^2 - \sigma^{q,a}) - 2\phi\sigma^{q,k,1}\sigma^{q,a}) \\
&+ (1 - \underline{\chi})(\sigma^{q,k,1}(\zeta_e^1 - \zeta_h^1) + \sigma^{q,a}(\zeta_e^2 - \zeta_h^2)) + \frac{\lambda_d}{z}(\bar{z} - z) \\
\mu^{q} &= \frac{1}{q}\left(\frac{\partial q}{\partial z}\mu^z z + \frac{\partial q}{\partial a_e}\mu_{ae} + \frac{1}{2}\frac{\partial^2 q}{\partial z^2} z^2((\sigma^{z,k})^2 + (\sigma^{z,a})^2 + 2\phi\sigma^{z,k}\sigma^{z,a})\right) \\
&+ \frac{1}{q}\left(\frac{1}{2}\frac{\partial^2 q}{\partial a_e^2}\sigma_{ae}^2 + \frac{\partial^2 q}{\partial z \partial a_e}(\phi\sigma^{z,k} + \sigma^{z,a})\sigma_{ae} z\right) \\
\mu_j^J &= \frac{\gamma}{2}((\sigma_j^{J,k})^2 + (\sigma_j^{J,a})^2 + 2\phi\sigma_j^{J,k}\sigma_j^{J,a} + \sigma^2) - (\Phi - \delta) \\
&+ (\gamma - 1)(\sigma_j^{J,k}\sigma + \phi\sigma\sigma_j^{J,a}) - \rho(\log(\rho) - \log(J_j) + \log(zq)) \\
\mu_j^R &= \frac{a_j - \iota}{q} + \Phi - \delta + \mu^q + \sigma\sigma^{q,k} + \phi\sigma\sigma^{q,a} \\
\mu_j^J J_j &= \frac{\partial J_j}{\partial t} + \frac{\partial J_j}{\partial z}\mu^z z + \frac{\partial J_j}{\partial a_e}\mu_{ae} + \frac{1}{2}\frac{\partial^2 J_j}{\partial z^2} z^2\left((\sigma^{z,k})^2 + (\sigma^{z,a})^2 + 2\phi\sigma^{z,k}\sigma^{z,a}\right) \\
&+ \frac{1}{2}\frac{\partial^2 J_j}{\partial a_e^2}\sigma_{ae}^2 + \frac{\partial^2 J_j}{\partial z \partial a_e}(\phi\sigma^{z,k} + \sigma^{z,a})\sigma_{ae} z
\end{aligned}
$$

[15]Details of the model are provided in Gopalakrishna (2020).

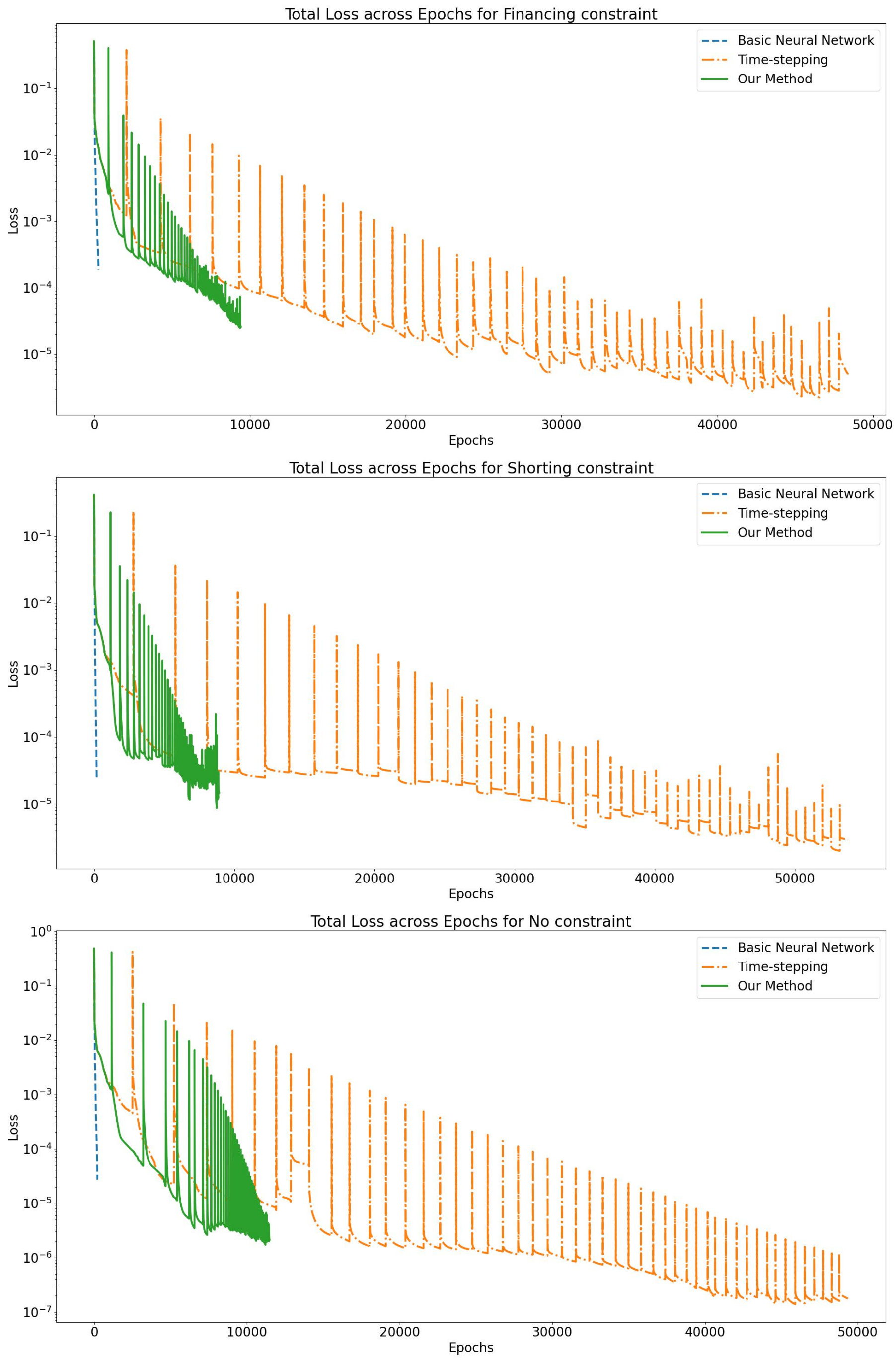
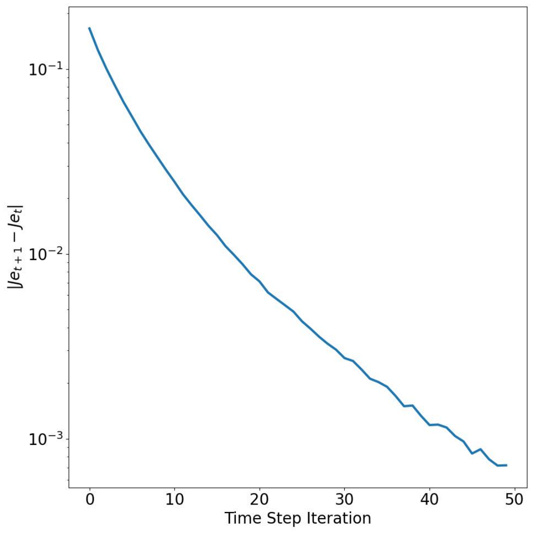
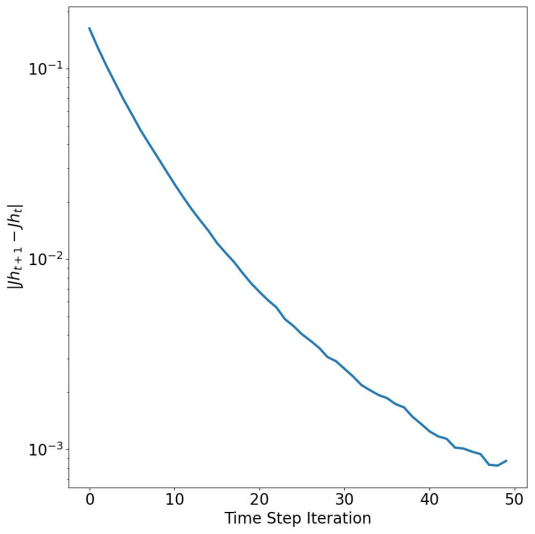


Figure 10: 2D Free Boundary Model Loss Progression

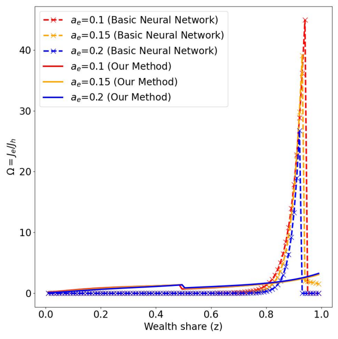


(a) Absolute Change of J_e per Time Step

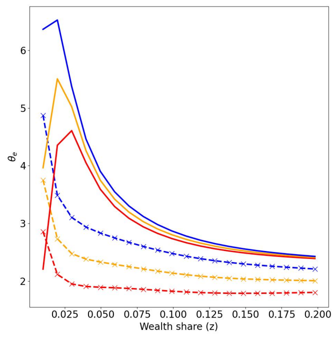


(b) Absolute Change of J_h per Time Step

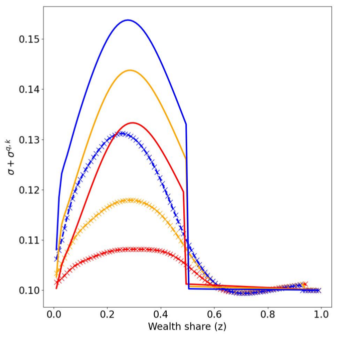
Figure 11: 2D Free Boundary Model Convergence of Value Functions



(a) Value Functions



(b) Portfolio Choice



(c) Price Return Diffusion

Figure 12: 2D Free Boundary Model Equilibrium Plots (Comparison - Appendix)

B Long-run Risk Model

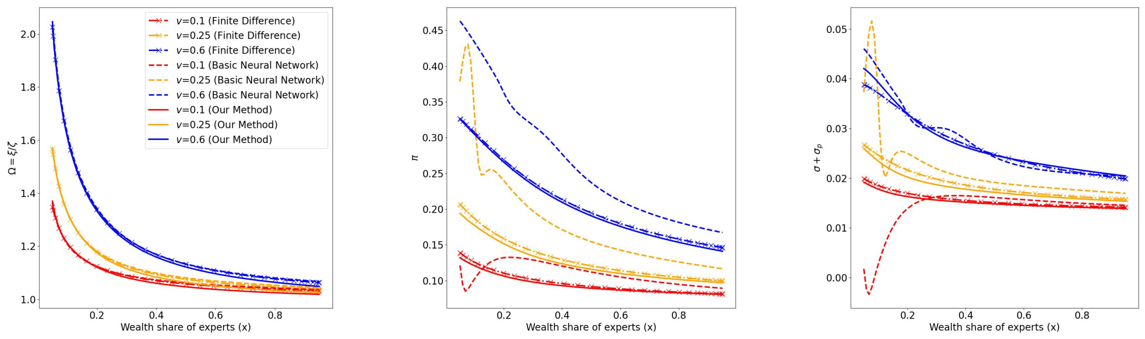
This model follows Di Tella (2017), and we refer the readers to the original paper for more details. Here, we only provide the set of parameters used.

Parameter	Definition	Value
a	relative risk aversion	1
σ	volatility of TFP shocks	0.0125
λ	mean reversion coefficient for idiosyncratic risk	1.38
$\bar{v}$	long-run mean of idiosyncratic risk	0.25
$\bar{\sigma_v}$	idiosyncratic volatility of capital on aggregate risk	-0.17
ρ	discount rate	0.0665
γ	risk aversion rate	5
ψ	inverse of elasticity of intertemporal substitution	$\psi = 0.5$ s.t. EIS=2
τ	Poisson retirement rate for experts	1.15
ϕ	moral hazzard	0.2
A	second order coefficient for investment function	53
B	first order coefficient for investment function	-0.87
δ	shift for investment function	0.05

Table 6: Long-run Risk Model Constant Parameters

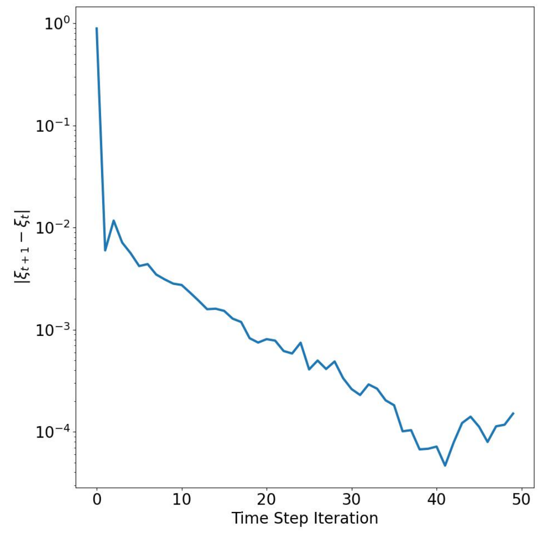
Type	Definition
State Variables	$(x, v) \in [0.05, 0.95] \times [0.05, 0.95]$
Agents	ξ (experts), ζ (households)
Endogenous Variables	p (price), r (risk-free rate)

Table 7: Long-run Risk Model Variables

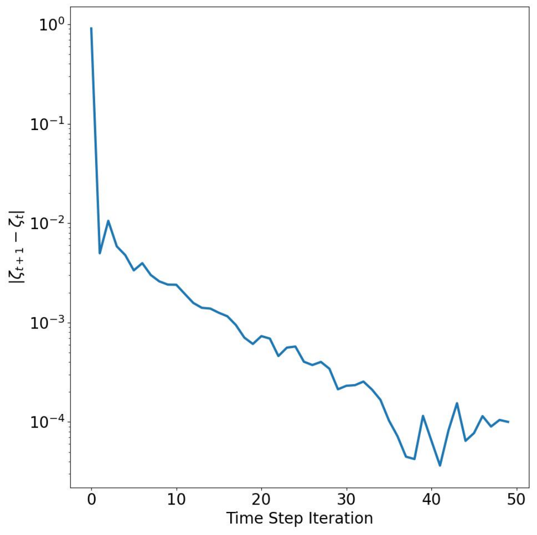


(a) Value Functions **(b)** Price of Risk **(c)** Price Return Diffusion

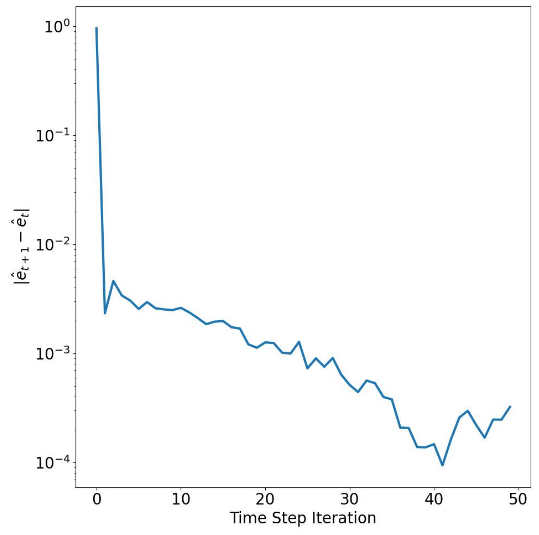
Figure 13: Long-run Risk Model Equilibrium Plots (Comparison - Appendix)



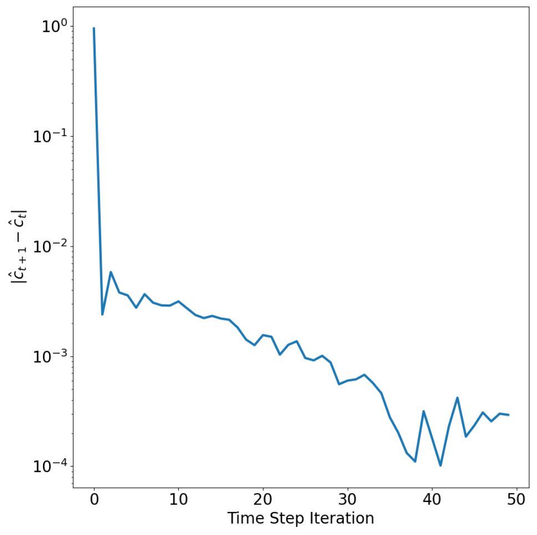
(a) Absolute Change of ξ per Time Step



(b) Absolute Change of ζ per Time Step



(c) Absolute Change of $\hat{e}$ per Time Step



(d) Absolute Change of $\hat{c}$ per Time Step

Figure 14: Long-run Risk Model Convergence of Value Functions and consumption

C N-Trees

State Variable:

- Wealth Share of agent $i \in [1, N-1]$: $z_i \in (0,1)$

Define $\mathbf{z} = (z_1, ..., z_{N-1})$, and $\tilde{\mathbf{z}} = \left(z_1, ..., z_{N-1}, 1 - \sum_{i=1}^{N-1} z_i\right)$. Similarly, we define the geometric drifts by $\mu^{\mathbf{z}} = (\mu^{z_1}, ..., \mu^{z_{N-1}})$, and $\mu^{\tilde{\mathbf{z}}} = (\mu^{z_1}, ..., \mu^{z_{N-1}}, \mu^{z_N})$. The geometric diffusions $\sigma^{\mathbf{z}}$ and $\sigma^{\tilde{\mathbf{z}}}$ and the arithmetic drifts and diffusions $\mu_a^{\mathbf{z}}, \mu_a^{\tilde{\mathbf{z}}}, \sigma_a^{\mathbf{z}}, \sigma_a^{\tilde{\mathbf{z}}}$ are defined similarly. We use $\cdot$ to denote the dot product and $\times$ to denote the Hadamard (element-wise) product.
Endogenous Variables: $\kappa : \mathbb{R}^{N-1} \to \mathbb{R}^N$, $z \mapsto \kappa(z)$.

Parameter	Definition	Value
γ	Household risk aversion	5
ρ	Fund discount rate	0.05
μ^{y_i}	Mean of the ith state variable	$0.01i$ for $i \geq 10$, $0.02 + 0.03i$ for $i < 10$
σ^{y_i}	Std of the ith state variable	$0.01i$ for $i \geq 10$, $0.02 + 0.03i$ for $i < 10$

Table 8: N-Tree Constant Parameters

Equations:

$$
\begin{aligned}
\mathbf{q} &= \frac{\tilde{\mathbf{z}}}{\kappa} \\
\mu^{z_i} &= \mu^{y_i} - \mu^{y} \cdot \tilde{\mathbf{z}} + \sigma^{y} \cdot \tilde{\mathbf{z}}(\sigma^{y} \cdot \tilde{\mathbf{z}} - \sigma^{y_i}) \\
\sigma^{z_i} &= \sigma^{y_i} - \sigma^{y} \cdot \tilde{\mathbf{z}} \\
\mu_a^{\mathbf{z}} &= \mu^{\mathbf{z}} \times \mathbf{z} \\
\sigma_a^{\mathbf{z}} &= \sigma^{\mathbf{z}} \times \mathbf{z} \\
\mu_a^{z_N} &= -\sum_{i=1}^{N-1} \mu_a^{z_i} \\
\sigma_a^{z_N} &= -\sum_{i=1}^{N-1} \sigma_a^{z_i} \\
\mu^{z_N} &= \frac{\mu_a^{z_N}}{z_N} \\
\sigma^{z_N} &= \frac{\sigma_a^{z_N}}{z_N} \\
\mu^{\mathbf{q}} &= \frac{1}{\mathbf{q}} \times \left(\nabla_{\mathbf{z}}\mathbf{q}\mu_a^{\mathbf{z}} + \frac{1}{2}(\sigma_a^{\mathbf{z}})^T \nabla_{\mathbf{z}}^2 \mathbf{q} \sigma_a^{\mathbf{z}}\right) \\
\sigma^{\mathbf{q}} &= \frac{1}{\mathbf{q}} \times (\nabla_{\mathbf{z}}\mathbf{q}\sigma_a^{\mathbf{z}})
\end{aligned}
$$

$$
\begin{aligned}
r &= \rho + \gamma \mu^{\mathbf{y}} \cdot \tilde{\mathbf{z}} - \frac{1}{2}\gamma(\gamma+1)(\tilde{\mathbf{z}})^2 \cdot (\sigma^{\mathbf{y}})^2 \\
\zeta &= \gamma \tilde{\mathbf{z}} \cdot \sigma^{\mathbf{y}} \\
\mu^{\kappa} &= \mu^{\tilde{\mathbf{z}}} - \mu^{\mathbf{q}} + \sigma^{\mathbf{q}} \times (\sigma^{\mathbf{q}} - \tilde{\mathbf{z}}) \\
\sigma^{\kappa} &= \sigma^{\tilde{\mathbf{z}}} - \sigma^{\mathbf{q}}
\end{aligned}
$$

HJB Equations:

$$
\begin{aligned}
\mu^{\kappa} \times \kappa =& \nabla_t \kappa + \nabla_{\mathbf{z}} \kappa \mu_a^{\mathbf{z}} + \frac{1}{2}(\sigma_a^{\mathbf{z}})^T \nabla_{\mathbf{z}}^2 \kappa \sigma_a^{\mathbf{z}} \\
\sigma^{\kappa} \times \kappa =& \nabla_{\mathbf{z}} \kappa \sigma_a^{\mathbf{z}}
\end{aligned}
$$

We parameterize κ as a multi-input, multi-output neural network with $N-1$ inputs and N outputs. Each neural network has 4 hidden layers with 80 neurons each, activated by *tanh*. Outputs are restricted to be positive using *SoftPlus*. Learning rates are set to 0.005 for all models. In each time step (or epoch without time-stepping), 200 points are randomly sampled for training. Non-time-stepping models are trained for 200 epochs, while time-stepping models are trained for 200 epochs in each time step and for upto 10 time iterations. Residual-based learning occurs 5 times at equal intervals within each time step or throughout the 200 epochs. For example, with 200 epochs, residual-based sampling occurs every 40 epochs.

For the 2-Tree, 3-Tree and 5-Tree models, we use uniform sampling, with μ^y and σ^y set to $[0.02, 0.05]$, $[0.02, 0.05, 0.08]$, $[0.02, 0.05, 0.08, 0.11, 0.14]$ respectively. Starting from the 10-tree model, we switch to ergodic sampling. This involves first sampling y_is values from LogNormal(μ^{y_i}, σ^{y_i}), then computing z_i by renormalization. Both μ^{y_i} and σ^{y_i} are set to $0.01i$ for all i. Table 9 shows the final loss of each model and the number of epochs required for convergence. While the basic non-time-stepping methods converge faster and yield lower loss, we show that time-stepping models achieve greater accuracy in the 2-Tree model.

Model Type	2-Tree Loss	3-Tree Loss	50-Tree Loss	100-Tree Loss
Basic Neural Network	2.82e-18 (7)	1.85e-17 (5)	6.60e-14 (3)	1.27e-12 (9)
Time-stepping	2.09e-6 (427)	3.24e-6 (526)	7.23e-6 (316)	3.69e-6 (311)
Our Method	1.24e-6 (243)	1.93e-6 (347)	8.32e-6 (166)	4.77e-6 (178)

Table 9: Tree Model Convergence Results

C.1 One-dimensional 2-Tree Model

We train a simple 2-Tree model on one-dimension using both a neural network approach and a finite difference method to validate our model setup. Figure 15 shows the loss convergence of these models, comparing scenarios with and without residual-based learning. It is evident that the neural network model achieves faster convergence with residual-based sampling. Figure 16 presents the equilibrium plots of the 2-Tree model using both finite difference and neural network approaches. The neural network approach, particularly with residual-based learning, provides the best approximation of the equilibrium distribution for κs and qs. Meanwhile, all models perform similarly for ζs and r.

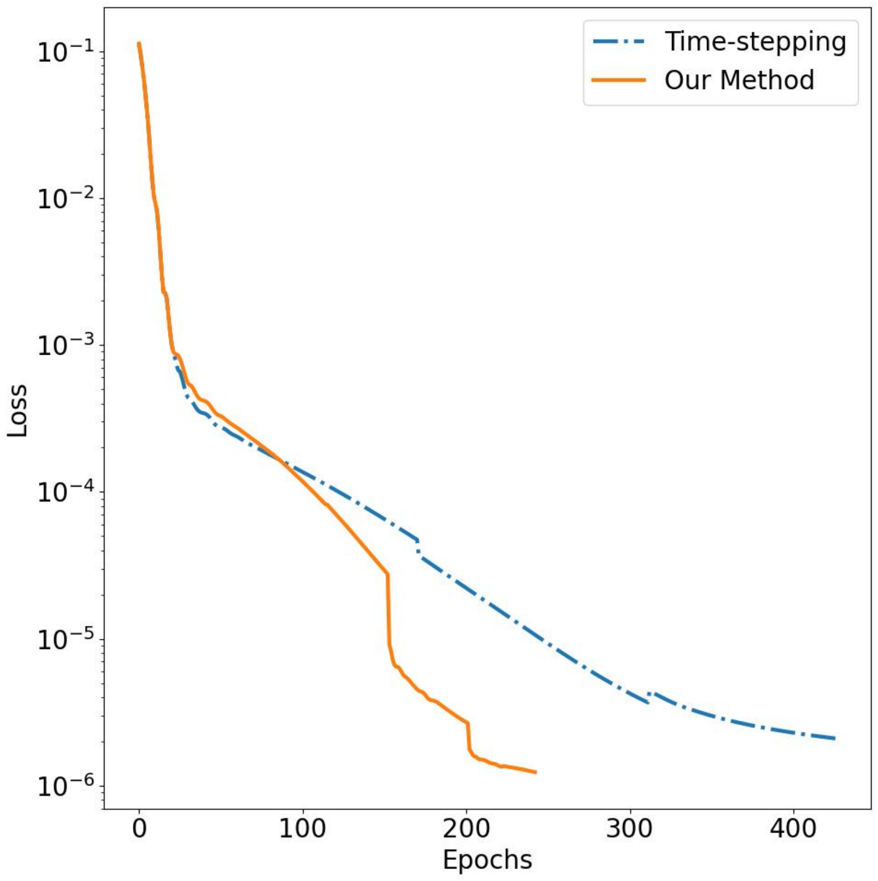
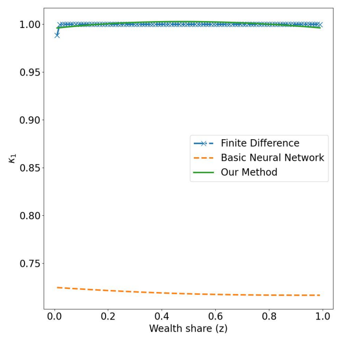
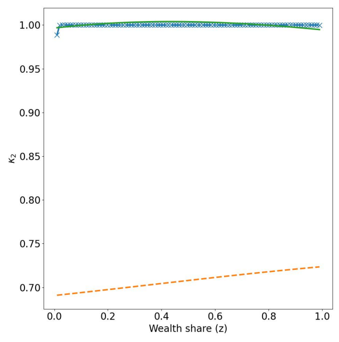


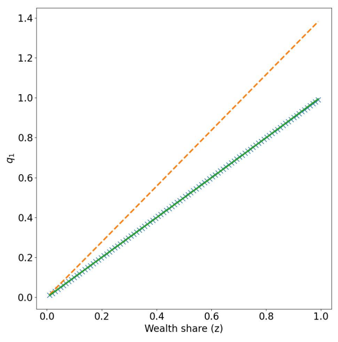
Figure 15: Loss Progression of Time Stepping Scheme (2-Tree)



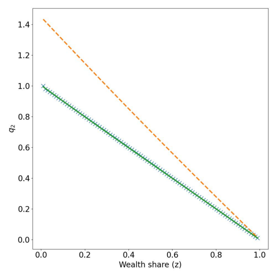
(a) κ_1



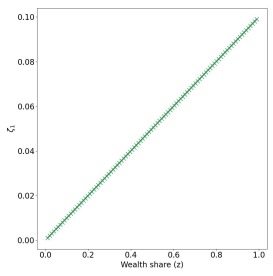
(b) κ_2



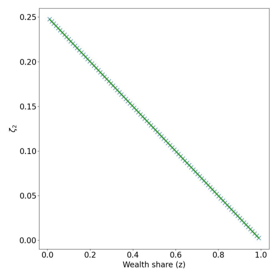
(c) q_1



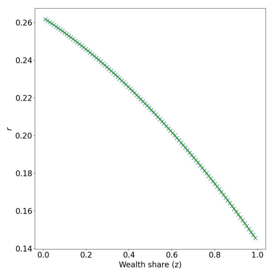
(d) q_2



(e) ζ_1



(f) ζ_2



(g) r

Figure 16: 2-Tree Model Equilibrium Plots. The MSE between our method and finite difference solution is 4.77×10^{-6}, 8.37×10^{-6}, 1.44×10^{-6}, 2.38×10^{-6}, 1.09×10^{-16}, 7.69×10^{-18}, 5.35×10^{-17} for κ_1, κ_2, q_1, q_2, r, ζ_1 and ζ_2, respectively.

C.2 Two-Dimensional 3-Tree Model

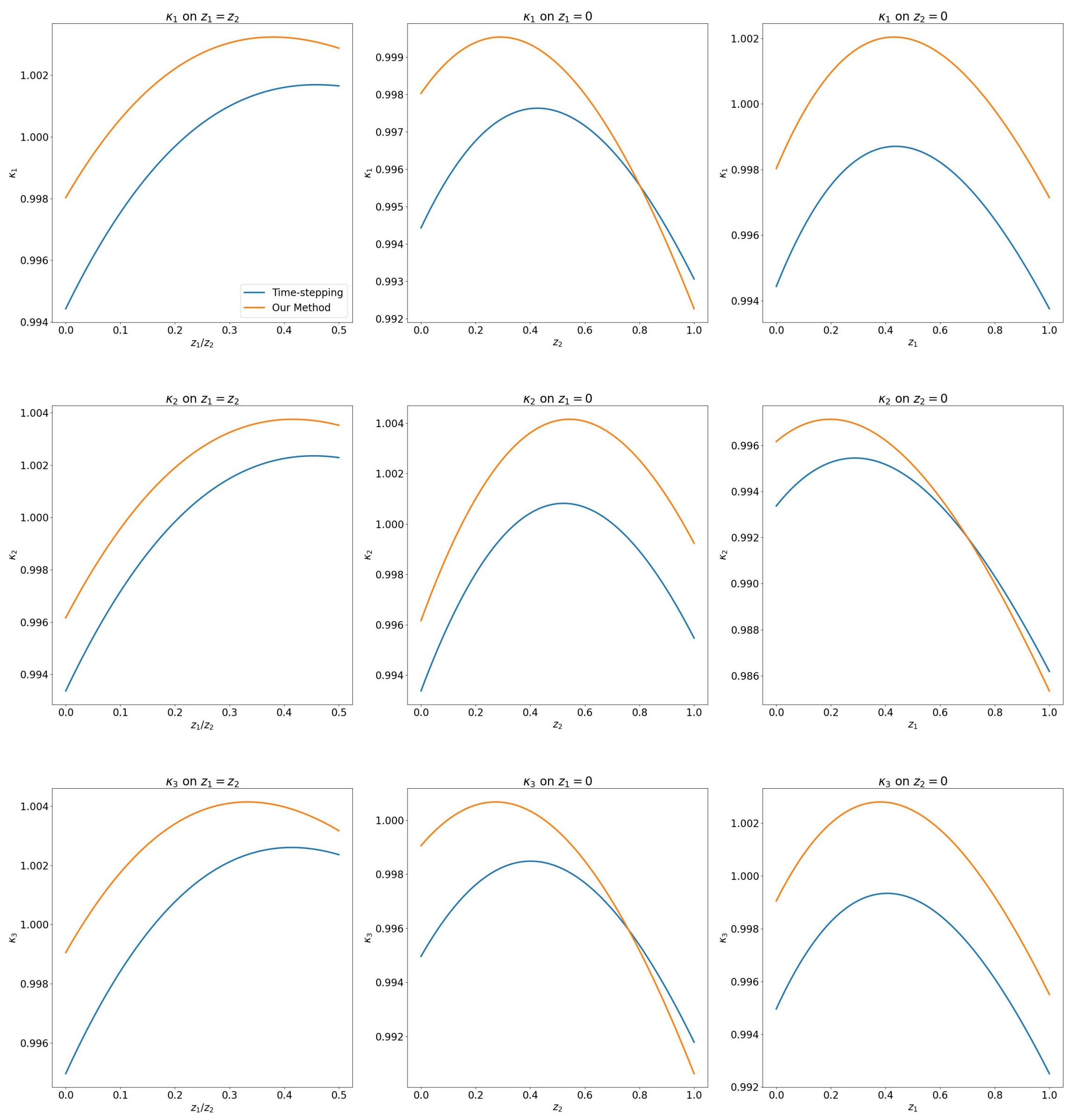
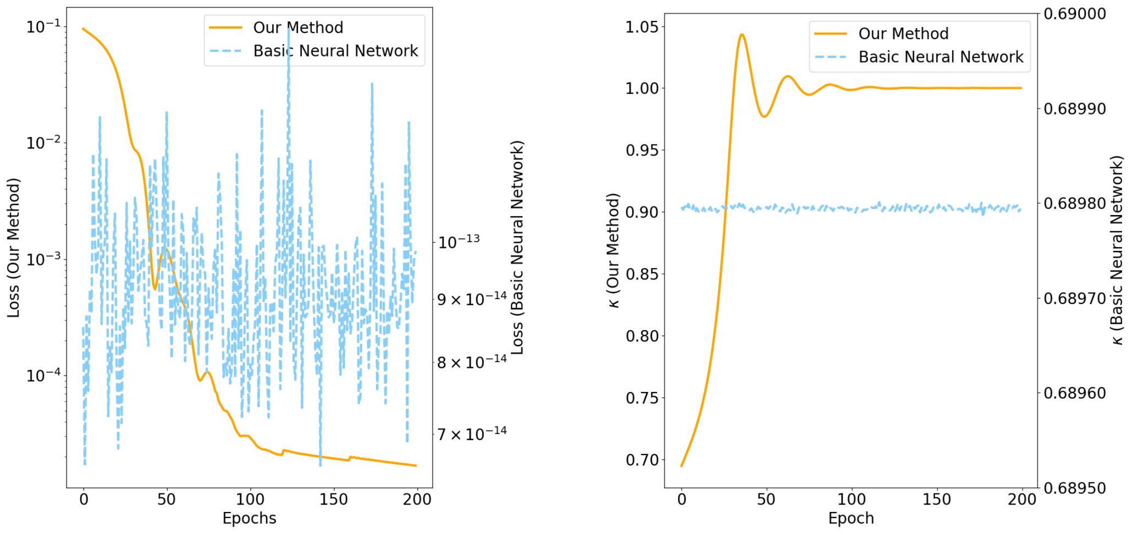


Figure 17: Equilibrium Plots in High Residual Domains (3-Tree)

C.3 50/100-Tree

At ergodic sampling, the κ values are expected to be 1, and the q values are anticipated to be around $1/50 = 0.02$. Figure 18 illustrates the progression of loss and κ convergence for the 50-Tree model. Figure 19 displays the distribution of equilibrium values for the 50-Tree model. To construct the distribution, we sample 1,000 points from the ergodic distribution and compute the κ and q values. We then plot the distributions of κ and q, focusing on the κ with the highest mean q value. Figure 20 shows the distribution of equilibrium for the 100-Tree model, based on 500 points from the ergodic distribution.



(a) Loss Convergence **(b)** Convergence of κ

Figure 18: Progression of Time Stepping Scheme (50-Tree) In both plots, left axis is for our method and right axis is for basic neural network. The first 200 epochs for time-stepping is shown. Basic Neural Network gives a lower loss, but converge to the wrong solution ($\kappa \approx 0.6898$). Also both the loss and κ values oscillates. Our method stabilizes faster.

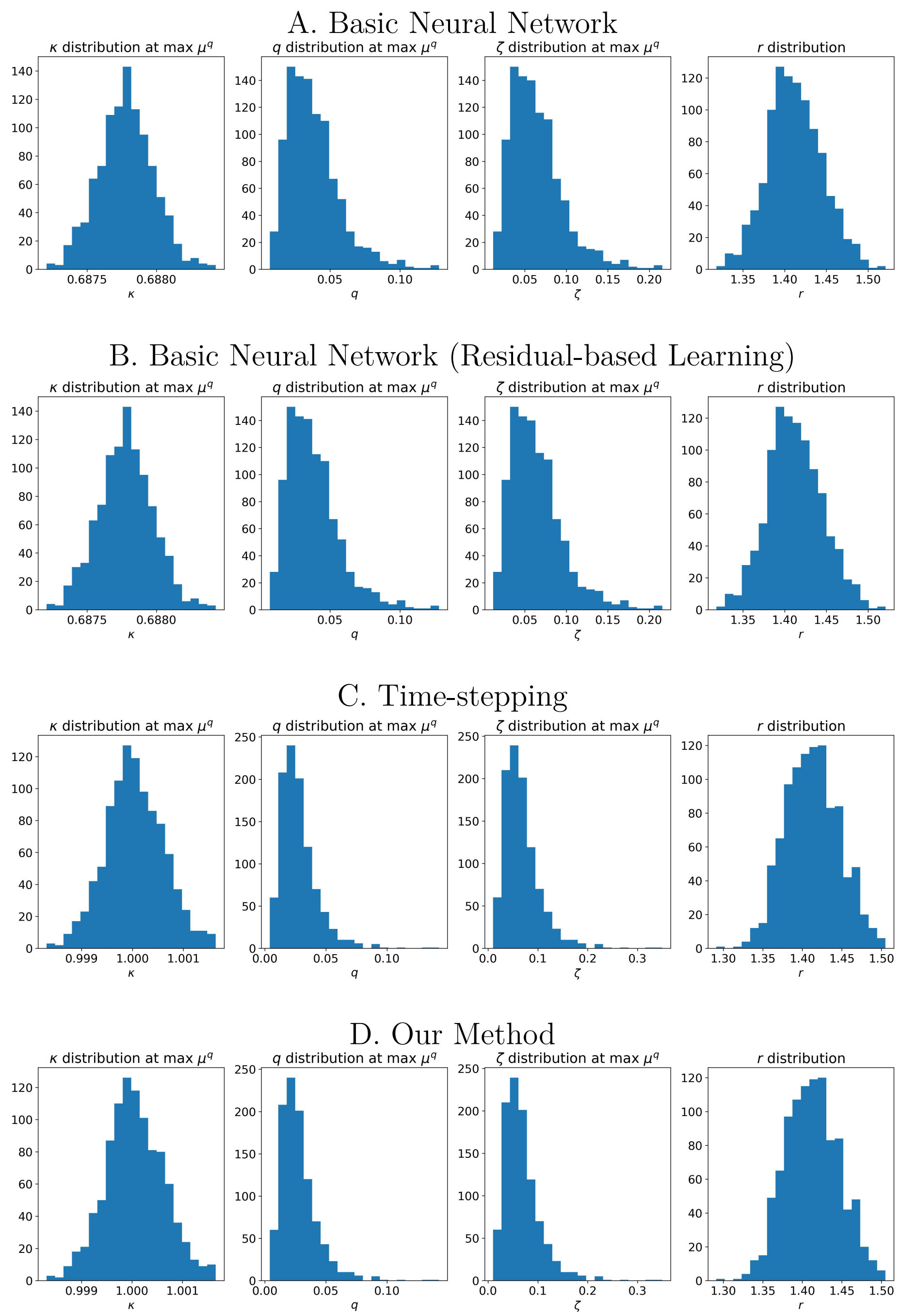


Figure 19: Distribution of Equilibrium Values (50-Tree)

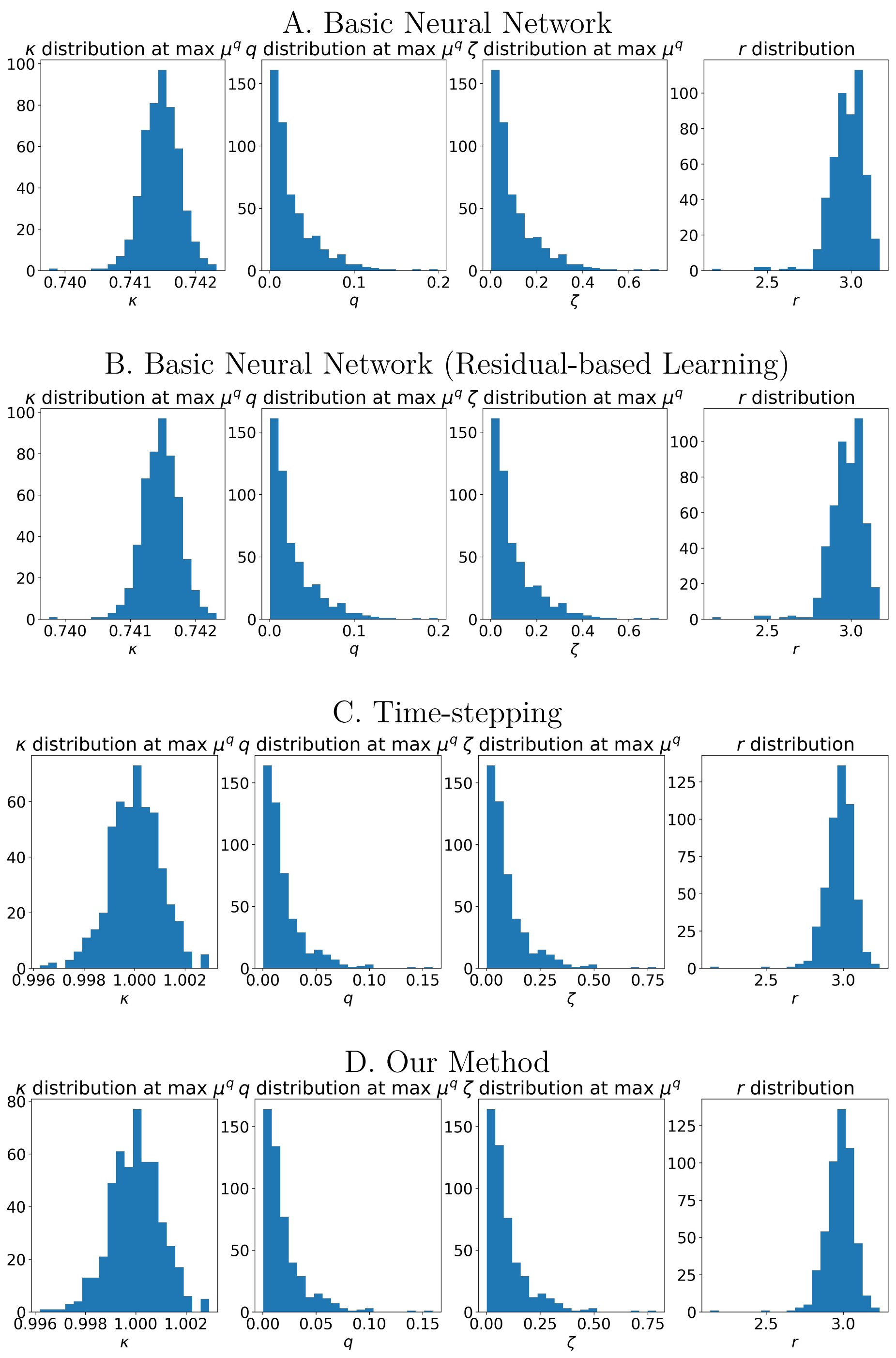


Figure 20: Distribution of Equilibrium Values (100-Tree)

D Proofs of Propositions

(i). Proposition 1

Proof. Let $\{x_i\}_i$ and $\{x_j\}_j$ be sequences of data samples in $\mathcal{T}$ s.t. $x_i < x_0$, $x_i \nearrow x_0^-$ and $x_j > x_0$, $x_j \searrow x_0^+$. Consider the objective function $\mathcal{L}$ on the set $\{x_i\}_i \cup \{x_j\}_j \cup \{x_0\}$:

$$\arg\min_{\theta,c} \sum_{x_i \nearrow x_0^-} |\Phi_\theta(x_i) - f(x_i)|^2 + \sum_{x_j \searrow x_0^+} |\Phi_\theta(x_j) - f(x_j)|^2 + |\Phi_\theta(x_0) - c|^2,$$

where $c \in \mathbb{R}$ can also be optimized relevant to the behavior of f around x_0, as $f(x_0)$ is not defined.
By the universal approximation theorem, $\Phi_\theta(x_i) \to f(x_i)$ as $x_i \nearrow x_0^-$ and $\Phi_\theta(x_j) \to f(x_j)$ as $x_j \searrow x_0^+$.
Since Φ_θ is continuous, but $f(x)$ has a jump discontinuity at x_0. The optimal c value should balance the values on either side of the jump, so $c = \dfrac{f(x_0^+) + f(x_0^-)}{2}$.
Then the optimal parameter θ^* will minimize $|\Phi_\theta(x_0) - c|^2$, and hence, $\Phi_\theta(x_0) \to \dfrac{f(x_0^+) + f(x_0^-)}{2}$. □

(ii). Proposition 2 In addition to the state variables determining the equilibrium, a fake transient time dimension t is added. The system is solved as if there was a finite horizon T. This requires modifying the HJB equations by adding a time derivative when computing the drifts using Ito's lemma:

$$\mu^J J = \frac{\partial J}{\partial t} + \sum_i \mu^{x_i} \frac{\partial J}{\partial x_i} + \sum_i \sum_j \frac{\sigma^{x_i}\sigma^{x_j}}{2} \frac{\partial^2 J}{\partial x_i \partial x_j}. \tag{D.1}$$

Proof. By rearranging (D.1) and replace J with a general Φ_θ approximator, we can rewrite the HJB equations involving time derivatives to be

$$\dot{\Phi}_\theta(x,t) := \nabla_t \Phi(x,t) = \mathcal{F}(x,t), \qquad \Phi_{\theta,0}(x,0) = \Phi_{\theta,1}(x,1) = 1.$$

Assume that $\mathcal{F}$ is locally Lipschitz, and we can apply Picard-Lindelof iteration. Let T be the number of outer loop iterations. The time interval $[0,T]$ can be broken down into T pieces, $[0,1] \cup [1,2] \cup \cdots \cup [T-1,T]$. The initial condition $\Phi_{\theta,1}(x,1) = 1$ and the algorithm iteration set the following recursive definition:

$$\Phi_{\theta,1}(x,T) = 1, \qquad \Phi_{\theta,k+1}(x,T-k) = \Phi_{\theta,k}(x,T-k)$$
$$\Phi_{\theta,1}(x,T-1) = \Phi_{\theta,1}(x,T) - \int_0^1 \dot{\Phi}_{\theta,1}(x,T-1+\tau)d\tau$$
$$\Phi_{\theta,2}(x,T-2) = \Phi_{\theta,2}(x,T-1) - \int_0^1 \dot{\Phi}_{\theta,2}(x,T-2+\tau)d\tau$$

$$\begin{aligned}
&\vdots \\
\Phi_{\theta,T}(x,0) &= \Phi_{\theta,T}(x,1) - \int_0^1 \dot{\Phi}_{\theta,T}(x,\tau)d\tau \\
\Rightarrow \Phi_{\theta,T}(x,0) &= 1 - \int_0^1 \dot{\Phi}_{\theta,1}(x,T-1+\tau)d\tau - \cdots - \int_0^1 \dot{\Phi}_{\theta,T}(x,\tau)d\tau \\
&= 1 - \sum_{i=1}^{T} \int_0^1 \dot{\Phi}_{\theta,i}(x,T-i+\tau)d\tau
\end{aligned}$$

As $T \to \infty$, we approximate infinite time horizon with finite time iterations, and $\Phi_{\theta,T}(x,0)$ is the approximaion.

Now we want to show that this iteration converges. Define a mapping $P : C^n[0,1] \to C^n[0,1]$ s.t. $(P\Phi_\theta)(x,t) = \Phi_\theta(x,t) - \int_0^t \mathcal{F}(x,\tau)d\tau$. Define $\{\phi_k\}_k = \left\{P^k\Phi_\theta\right\}_k$, and norm on $C^n[0,1]$ by $\|\cdot\|_C = \max_{t\in[0,T]} \|\phi(x,t)\|$.

$$\begin{aligned}
\|\phi_1 - \phi_0\| &= \left\|\int_0^t \mathcal{F}(x,\tau)d\tau\right\| \leq \int_0^t \|\mathcal{F}(x,\tau)\|\, d\tau \leq Mt \text{ (for some } M \in (0,1)) \\
\|\phi_2 - \phi_1\| &= \|P\phi_1 - P\phi_0\| \overset{\text{(By Lipschitz continuity)}}{\leq} L\int_0^t \|\phi_1 - \phi_0\|\, d\tau \leq LM\frac{t^2}{2} \\
&\vdots \\
\|\phi_{k+1} - \phi_k\| &\leq L\int_0^t \|\phi_k - \phi_{k-1}\|\, d\tau \leq L^{k-1}M\frac{t^k}{k!} \\
\Rightarrow \|\phi_{k+p} - \phi_k\| &\leq \sum_{i=0}^{p-1} \|\phi_{k+i+1} - \phi_{k+i}\| \leq \sum_{i=0}^{p-1} L^{i-1}M\frac{t^i}{i!} \\
\|\phi_{k+p} - \phi_k\|_C &= \max_{t\in[0,T]} \|\phi_{k+p}(x,t) - \phi_k(x,t)\| \leq \sum_{i=k+1}^{k+p} L^{i-1}M\frac{T^i}{i!} \\
&\leq \sum_{i=k+1}^{\infty} ML^{i-1}\frac{T^i}{i!} \to 0, \text{ as } k \to \infty.
\end{aligned}$$

The convergence $\sum_{i=k+1}^{\infty} ML^{i-1}\frac{T^i}{i!} \to 0$ comes from the fact that

$$\sum_{i=0}^{\infty} \frac{M}{L}\frac{(LT)^i}{i!} = \frac{M}{L}\exp(LT),$$

and we can take a large k for

$$\sum_{i=k+1}^{\infty} ML^{i-1}\frac{T^i}{i!} = \sum_{i=0}^{\infty} ML^{i-1}\frac{T^i}{i!} - \sum_{i=0}^{k} ML^{i-1}\frac{T^i}{i!} = 0.$$

Now we show that $\dot{\Phi}_\theta(x,t)$ vanishes.
The convergence of the outer loop iteration implies that $\|\Phi_{\theta,T}(x,0) - \Phi_{\theta,T-1}(x,0)\| =$

$\|\Phi_{\theta,T}(x,0) - \Phi_{\theta,T}(x,1)\| \to 0$ as $T \to \infty$ (with contraction mapping principle). The convergence of the inner loop iteration, based on Sirignano and Spiliopoulos (2018), ensures that both $\Phi_{\theta,T}(x,0)$ and $\Phi_{\theta,T}(x,1)$ converge to the same unique solution f augmented with $\nabla_t\Phi$. Therefore, we have $\nabla_t\Phi(x,0) = \nabla_t\Phi(x,1)$, creating a periodic boundary condition for both Φ and $\nabla_t\Phi$ along the time dimension. Given that Φ converges to a stable, unique solution as the outer iterations progress, and that the economic equilibrium problem guarantees a unique solution, we can conclude that $\nabla_t\Phi \to 0$. □

E Software and Replication Package

The python library along with implementation details can be found in `https://github.com/rotmanfinhub/deep-macrofin`. We provide a sample Jupyter notebook that demonstrates the parsing of equations at `https://github.com/rotmanfinhub/deep-macrofin/blob/main/examples/time_step/ditella.ipynb`. A comprehensive documentation of the library is available at `https://rotmanfinhub.github.io/deep-macrofin/`. Additionally, the replication package for the experiments conducted in this paper is available at `https://github.com/yuntaowu2000/active_learning_paper`.

References

Azinovic, Markon and Jan Zemlicka, "Intergenerational consequences of rare disasters," 2023.

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

Bischof, Rafael and Michael Kraus, "Multi-Objective Loss Balancing for Physics-Informed Deep Learning," 2021.

Bretscher, Lorenzo, Jesús Fernández-Villaverde, and Simon Scheidegger, "Ricardian Business Cycles," *SSRN Electronic Journal*, 11 2022.

Brumm, Johannes and Simon Scheidegger, "Using Adaptive Sparse Grids to Solve High-Dimensional Dynamic Models," *Econometrica*, 2017.

Brunnermeier, Markus K. and Yuliy Sannikov, "A Macroeconomic Model with a Financial Sector," *American Economic Review*, 2 2014, *104* (2), 379–421.

Brunnermeier, Markus K and Yuliy Sannikov, "Macro, Money and Finance: A Continuous Time Approach," Working Paper 22343, National Bureau of Economic Research 6 2016.

Bungartz, Hans Joachim, Alexander Heinecke, Dirk Pflüger, and Stefanie Schraufstetter, "Option pricing with a direct adaptive sparse grid approach," in "Journal of Computational and Applied Mathematics" 2012.

Cipolla, Roberto, Yarin Gal, and Alex Kendall, "Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics," in "2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition" 2018, pp. 7482–7491.

D'avernas, Adrien, Damon Petersen, and Quentin Vandeweyer, "A Solution Method for Continuous-Time Models," 2023.

Duarte, Victor, Diogo Duarte, and Dejanir H Silva, "Machine Learning for Continuous-Time Finance," 2024.

Fernández-Villaverde, Jesús, Samuel Hurtado, and Galo Nuño, "Financial Frictions and the Wealth Distribution," *Econometrica*, 2023, *91*, 869–901.

Gopalakrishna, Goutham, "A Macro-Finance model with Realistic Crisis Dynamics," in "Proceedings of Paris December 2021 Finance Meeting EUROFIDAI - ESSEC" 11 2020. Swiss Finance Institute Research Paper No. 20-96.

_ , Zhouzhou Gu, and Jonathan Payne, "Asset Pricing, Participation Constraints, and Inequality.," *Princeton Univeristy Working Paper*, 2024.

Gu, Zhouzhou, Mathieu Laurière, Sebastian Merkel, and Jonathan Payne, "Global Solutions to Master Equations for Continuous Time Heterogeneous Agent Macroeconomic Models," 6 2024.

Han, Jiequn, Arnulf Jentzen, and Weinan E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, 2018, *115* (34), 8505–8510.

_ , Yucheng Yang, and Weinan E, "DeepHAM: A Global Solution Method for Heterogeneous Agent Models with Aggregate Shocks," *SSRN Electronic Journal*, 12 2021.

Hornik, Kurt, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, 1991, *4* (2), 251–257.

_ , Maxwell Stinchcombe, and Halbert White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, 1989, *2* (5), 359–366.

Huang, Ji, "A Probabilistic Solution to High-Dimensional Continuous-Time Macro and Finance Models," 2023.

Liu, Ziming, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James

Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark, "KAN: Kolmogorov-Arnold Networks," *arXiv preprint arXiv:2404.19756*, 2024.

Lu, Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis, "DeepXDE: A Deep Learning Library for Solving Differential Equations," *SIAM Review*, 2021, *63* (1), 208–228.

Maliar, Lilia and Serguei Maliar, "Merging simulation and projection approaches to solve high-dimensional problems with an application to a new Keynesian model," *Quantitative Economics*, 3 2015, *6*, 1–47.

_ , _ , and Pablo Winant, "Deep learning for solving dynamic economic models.," *Journal of Monetary Economics*, 2021, *122*, 76–101.

Martin, Ian, "The Lucas Orchard," *Econometrica*, 1 2013, *81*, 55–111.

Merkel, Sebastian, "The Macro Implications of Narrow Banking: Financial Stability versus Growth," 2020.

Payne, Jonathan, Adam Rebei, and Yucheng Yang, "Deep Learning for Search and Matching Models," *SSRN Electronic Journal*, 2 2024.

Petersen, Philipp and Jakob Zech, "Mathematical theory of deep learning," 2024.

Raissi, M, P Perdikaris, and G E Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, 2019.

Sauzet, Maxime, "Projection Methods via Neural Networks for Continuous-Time Models," 2021.

Schaab, Andreas and Allen Zhang, "Dynamic Programming in Continuous Time with Adaptive Sparse Grids," *SSRN Electronic Journal*, 5 2022.

Shukla, Khemraj, Juan Diego Toscano, Zhicheng Wang, Zongren Zou, and George Em Karniadakis, "A comprehensive and FAIR comparison between MLP and KAN representations for differential equations and operator networks," *arXiv preprint arXiv:2406.02917*, 2024.

Sirignano, Justin and Konstantinos Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, 2018.

Tella, Sebastian Di, "Uncertainty Shocks and Balance Sheet Recessions," *Journal of Political Economy*, 2017, *125* (6), 2038–2081.