Fast Global Solutions for Heterogeneous Firm Models Using Deep Learning

Yoshiya Yokomoto, Keio University

May 22, 2025

Abstract

This paper introduces a novel solution method for heterogeneous firm models with aggregate uncertainty that significantly reduces computational time while maintaining solution accuracy. The core innovation involves approximating the policy function with a neural network that includes the equilibrium price as a state variable. This strategy directly tackles the fundamental computational bottleneck of repeated market-clearing equilibrium price calculations during simulation, leveraging the neural network's ability to handle the resulting high-dimensional state space and overcome the curse of dimensionality. Applied to seminal models in the literature, including Khan and Thomas (2008) and Bloom et al. (2018), this approach achieves speed improvements of up to 50x. By maintaining the skeleton of established solution techniques while replacing key components with neural network approximations, my approach remains transparent and accessible to researchers already familiar with standard heterogeneous agent modeling techniques, opening new possibilities for analyzing complex firm dynamics with realistic computational resources.

1 Introduction

This paper introduces a novel global solution method that delivers dramatic computational speed improvements for solving heterogeneous firm models with aggregate uncertainty. By combining deep learning neural networks with traditional solution techniques, I address the curse of dimensionality that has been a persistent challenge in these models. My approach maintains the established distribution-tracking methods used in the literature while substantially accelerating the process of finding equilibrium prices in each simulation period. Applied to seminal heterogeneous firm models like Bloom et al. (2018), my method achieves speed improvements of up to 50x while maintaining solution accuracy. My main contributions are threefold:

First, I provide global solutions that capture potential non-linearities in general equilibrium models with stochastic business cycle simulations. Unlike approximation methods that may miss critical non-linear relationships, my approach preserves the rich dynamics of heterogeneous firm models while making them computationally tractable.

Second, my method delivers solutions that are orders of magnitude faster while maintaining the accuracy achieved by existing solution methods. This speed advantage proves particularly valuable when working with heterogeneous firm frameworks that previously required prohibitive computation times. For example, in applications to models such as Bloom et al. (2018), my approach reduces computation time from approximately 40 hours to around 50 minutes while preserving—and in some cases improving—solution accuracy. This efficiency gain is especially important for models with rich firm heterogeneity and aggregate uncertainty, where computational demands have historically limited their practical application.

Third, I preserve the skeleton of existing solution methods that are well-recognized and well-understood in the literature, particularly as surveyed in Terry (2017). By maintaining structural similarity to established techniques like the Krusell-Smith algorithm, my approach remains accessible to researchers already familiar with these methods. I replace key components of traditional methods with neural network approximations while keeping the overall solution structure intact, making my innovations immediately applicable to a wide range of heterogeneous firm models.

I achieve these improvements through state-of-the-art deep learning techniques utilizing neural networks. My approach overcomes the curse of dimensionality in models with large state spaces, which is particularly acute in heterogeneous firm models where both the distribution of firm-specific states and aggregate variables must be tracked simultaneously. Neural networks excel at handling these high-dimensional problems, allowing my method to efficiently navigate the expansive state spaces characteristic of heterogeneous firm models like Khan and Thomas (2008) and Bloom et al. (2018). Additionally, I leverage GPU acceleration to dramatically enhance computational performance for models with millions of potential state-space combinations.

The specific novelty of my approach lies in substantially speeding up the stochastic simulation process where finding equilibrium prices in each simulation period is required. In heterogeneous firm models with aggregate uncertainty, market-clearing conditions must be satisfied in each period, requiring iterations to find equilibrium prices that balance aggregate supply and demand. Traditional methods iterate repeatedly to find these market-clearing prices at each point in the simulation, creating a significant computational bottleneck. My neural network approach learns optimal responses across the state space, eliminating the need for costly repetitive optimization within the simulation loop. This particular computational challenge has been a persistent obstacle for researchers working with heterogeneous firm models, and my method specifically targets this

well-defined problem, providing a solution that maintains accuracy while delivering significant performance improvements.

To demonstrate the method's practicality and robustness, this paper applies my approach to two prominent heterogeneous firm models. First, it is applied to Khan and Thomas (2008), which features firm heterogeneity in terms of capital levels and idiosyncratic productivity shocks. Second, the methodology is extended to a more complex setting presented by Bloom et al. (2018), where firms are subjected to stochastic volatility shocks alongside two endogenous state variables, resulting in over three million state-space combinations. My results show that the neural network approach not only matches the key macroeconomic dynamics, microeconomic moments, and forecasting accuracy reported in these papers but does so with substantially reduced computational requirements.

The rest of the paper is organized as follows. The next section reviews the related literature. Section 2 introduces my methodology in the context of the Khan and Thomas (2008) model, contrasting this neural network approach with the traditional Krusell-Smith solution technique, and presents comprehensive results comparing the performance, accuracy, and computational efficiency of the method against benchmark approaches. Section 3 extends the application to the more complex Bloom et al. (2018) model to demonstrate scalability to larger state spaces.

1.1 Related Literature

This paper contributes to the literature on solution methods for heterogeneous agent models with aggregate uncertainty, with particular focus on heterogeneous firm models.

First, my work builds on global solution methods for heterogeneous agent models with aggregate uncertainty, particularly the Krusell-Smith framework and its extensions. Den Haan (2010) provides a survey of solution methods for the original Krusell and Smith (1998) model, while Terry (2017) offers a comprehensive survey specifically focused on solution methods for heterogeneous firm models such as Khan and Thomas (2008). A significant challenge in heterogeneous firm models is the high computational cost of simulation. As Terry (2017) and Bloom et al. (2018) emphasize, simulation is often crucial for estimation and calibration, making simulation speed improvements essential. Some methods, such as Algan et al. (2008) and Sunakawa (2020), have been developed to update forecasting rules without relying on simulation. In contrast, my approach does not aim to avoid simulation but rather to accelerate it by incorporating the equilibrium price into the policy function. For local solutions, notable contributions include Reiter (2009), Ahn et al. (2018), Boppart et al. (2018), Winberry (2018), and Auclert et al. (2021). For additional global methods beyond Krusell and Smith (1998), see also Den Haan and Rendahl (2010).

Second, my work leverages recent advances in deep learning techniques for solving macroeconomic models. Fernández-Villaverde et al. (2024) provides a comprehensive survey of this growing field. Several notable contributions include Fernández-Villaverde et al. (2020) and Maliar et al. (2021), who approximate value and policy functions with neural networks by jointly training these functions through minimizing a combined loss function based on the Bellman equation error and first-order conditions. Han et al. (2021) take a different approach by training value and policy functions separately using distinct loss functions. Azinovic et al. (2022) use a neural network for the policy function, approximating the policy function that satisfies equilibrium conditions by including equilibrium conditions in the loss function.

The key distinction of my approach is its specific application to heterogeneous firm models where finding equilibrium prices throughout the simulation process poses a significant computa-

tional challenge. While existing methods have made important contributions to both global solutions and deep learning applications in macroeconomics, none have effectively addressed the specific computational bottleneck of repeatedly calculating market-clearing equilibrium prices during simulation of heterogeneous firm models with aggregate uncertainty. By training neural networks to learn optimal responses across the state space, my method eliminates the need for costly repetitive optimization within the simulation loop while maintaining the essential structure of established solution techniques.

2 Method

This section explains my proposed solution method, using the model of Khan and Thomas (2008) as the benchmark framework. Subsection 2.1 first reviews the Khan and Thomas (2008) model. Next, Subsection 2.2 describes the standard Krusell-Smith (KS) method for solving such models. Subsequently, Subsection 2.3 details the novel deep learning-based solution approach proposed in this paper. Finally, Subsection 2.4 presents computational results from applying my method to the Khan and Thomas (2008) model and compares them against a benchmark derived from the Khan and Thomas (2008) model code provided by Terry (2017).

2.1 Khan and Thomas (2008)

The production function employed by firms follows a standard Cobb-Douglas form:

$$y = z\epsilon k^{\alpha} N^{\nu}, \tag{1}$$

where y denotes output, z represents aggregate productivity, ϵ captures idiosyncratic firm-level productivity, k stands for capital, and N denotes labor input.
Firms face productivity shocks at both aggregate and individual levels. Specifically, the idiosyncratic productivity evolves according to an AR(1) process:

$$\log(\epsilon') = \rho_{\epsilon} \log(\epsilon) + \eta'_{\epsilon}, \quad \eta'_{\epsilon} \sim N(0, \sigma^2_{\eta_{\epsilon}}), \tag{2}$$

while aggregate productivity evolves similarly:

$$\log(z') = \rho_{z} \log(z) + \eta'_{z}, \quad \eta'_{z} \sim N(0, \sigma^2_{\eta_{z}}). \tag{3}$$

Capital evolves following the conventional law of motion:

$$k' = (1 - \delta)k + i, \tag{4}$$

where δ is the depreciation rate, and i represents investment.
Firms face fixed adjustment costs when altering their capital stock, which are given by a random draw ξ scaled by the equilibrium wage rate w, represented as:

$$\psi(w) = w\xi, \tag{5}$$

where ξ is an independently and identically distributed random variable drawn from distribution G over $[0, \bar{\xi}]$.

On the household side, a representative agent makes consumption, labor supply, and portfolio decisions. The household maximizes its lifetime utility:

$$W(\lambda; z_i, \mu) = \max_{c, n_h, \lambda'} u(c, 1 - n_h) + \beta \sum_{j=1}^{N_z} \pi_{ij} W(\lambda'; z_j, \mu'), \tag{6}$$

$$u(c, 1 - n_h) = \log c + \phi(1 - n_h) \tag{7}$$

subject to a budget constraint involving consumption c, labor supply n_h, and portfolio holdings λ. Here, μ represents the distribution of idiosyncratic shocks and capital, evolving according to $\mu' = \Gamma(z, \mu)$.

The firm's optimization problem involves choosing employment n and investment k^*, given their current states and adjustment costs:

$$\begin{aligned} v^1(\varepsilon, k, \xi; z, \mu) = \max_{n, k^*} \Big[& z\varepsilon k^\alpha n^\nu - \omega(z, \mu)\, n + (1 - \delta)k \\ & + \max \Big\{ -\xi\, \omega(z, \mu) + R(\varepsilon, k^*; z, \mu'), R(\varepsilon, (1 - \delta)k; z, \mu') \Big\} \Big] \end{aligned} \tag{8}$$

$$R\left(\varepsilon_e, k'; z_i, \mu'\right) \equiv -\gamma k' + \sum_{j=1}^{N_z} \pi_{ij} d_j\left(z_i; \mu\right) \sum_{m=1}^{N_\varepsilon} \pi_{em}^{\varepsilon} v^0\left(\varepsilon_{\mathrm{m}}, k'; z_j, \mu'\right),$$

$$v^0(\varepsilon, k; z, \mu) \equiv \int_0^{\bar{\xi}} v^1(\varepsilon, k, \xi; z, \mu) G(d\xi)$$

Forecasting rule,

$$\mu' = \Gamma_\mu(z, \mu)$$

Using the aggregate quantities C and N to describe the market-clearing values of household consumption and hours, we can derive the following from the household's first-order condition:

$$\omega(z, \mu) = \frac{D_2 U(C, 1 - N)}{D_1 U(C, 1 - N)} \tag{9}$$

$$d_j(z, \mu) = \frac{\beta D_1 U\left(C_j', 1 - N_j'\right)}{D_1 U(C, 1 - N)} \tag{10}$$

Defining $p(z, \mu)$ as the price firms use to value current output (relative to the marginal utility of consumption), we have the two conditions:

$$p(z, \mu) = D_1 U(C, 1 - N) \tag{11}$$

$$\omega(z, \mu) = \frac{D_2 U(C, 1 - N)}{p(z, \mu)} \tag{12}$$

Using the definitions for p and w above, and denoting V as the value function measured in units

of household marginal utility, the Bellman equation can be rewritten as:

$$V^1(\varepsilon, k, \xi; z, \mu) = \max_{n \in \mathbf{R}_+} (z\varepsilon k^\alpha n^\nu - \omega n + (1-\delta)k)p + \max\left\{ -\xi\omega p + \max_{k^* \in \mathbf{R}_+} R\left(\varepsilon, k^*; z, \mu'\right), R\left(\varepsilon, (1-\delta)k; z, \mu'\right) \right\} \tag{13}$$

$$R\left(\varepsilon, k'; z, \mu'\right) \equiv -\gamma k' p + \beta \sum_{j=1}^{N_z} \pi_{ij} \sum_{m=1}^{N_\varepsilon} V^0\left(\varepsilon_m, k'; z_j, \mu'\right),$$

$$V^0(\varepsilon, k; z, \mu) \equiv \int_0^{\bar{\xi}} V^1(\varepsilon, k, \xi; z, \mu) G(d\xi)$$

Forecasting rule,

$$\mu' = \Gamma_\mu(z, \mu), \quad p = \Gamma_p(z, \mu)$$

Note that the second maximization in (13) reflects the firm's choice between investing to move to the new capital level k^* or remaining at the depreciated level $(1-\delta)k$. Importantly, the choice of k^* depends only on (z, ε, μ) and **not** on the firm's individual capital k.[1] The threshold level of the adjustment cost ξ that determines whether the firm invests is given by:

$$\xi^* = \frac{R(\varepsilon, k^*; z, \mu') - R(\varepsilon, (1-\delta)k; z, \mu')}{p\omega} \tag{14}$$

Equilibrium Conditions An equilibrium represents a set of firm value functions V^1, V^0, firm policies and adjustment thresholds k^*, n, ξ^*, prices $p(z, \mu), \omega(z, \mu)$, and mappings Γ_μ, Γ_p such that:

- Firm capital adjustment choices and policies conditional upon adjustment satisfy the Bellman equations defining V^1, V^0 above, and therefore firm capital transitions are given by

$$k'(\varepsilon, k, \xi; z, \mu) = \begin{cases} k^*(\varepsilon; z, \mu'), & \xi < \xi^*(\varepsilon, k; z, \mu) \\ (1-\delta)k, & \xi \geq \xi^*(\varepsilon, k; z, \mu) \end{cases} \tag{15}$$

- The distributional transition rule used in the calculation of expectations above by firms is consistent with the aggregate evolution of the distributional state

$$\begin{gathered} \mu'(\varepsilon', k') = \Gamma_\mu(z, \mu) = \iiint I_A(\varepsilon, k) d\mu(\varepsilon, k) dG(\xi) d\Phi(\eta'_\epsilon) \\ A(\varepsilon', k', \xi, \eta'_\epsilon; z, \mu) = \{(\varepsilon, k) \mid k'(\varepsilon, k, \xi; z, \mu) = k', \log(\varepsilon') = \rho_\epsilon \log(\varepsilon) + \eta'_\epsilon\}, \\ \Phi(x) = \mathbb{P}(\eta'_\epsilon \leq x) \end{gathered} \tag{16}$$

- Aggregate output, investment, and labor are consistent with the current distribution μ and firm policies:

$$Y(z, \mu) = \iint z\varepsilon k^\alpha n(\varepsilon, k, \xi; z, \mu)^\nu d\mu(\varepsilon, k) dG(\xi) \tag{17}$$

$$I(z, \mu) = \iint (k'(\varepsilon, k, \xi; z, \mu) - (1-\delta)k) d\mu(\varepsilon, k) dG(\xi) \tag{18}$$

[1] Since the adjustment cost $\xi\omega p$ is independent of k, the optimal choice of k^* is also independent of it.

$$N(z,\mu) = \iint n(\varepsilon,k,\xi;z,\mu)d\mu(\varepsilon,k)dG(\xi) + \int G(\xi^*(\varepsilon,k;z,\mu))d\mu(\varepsilon,k) \tag{19}$$

- Aggregate consumption satisfies the resource constraint

$$C(z,\mu) = Y(z,\mu) - I(z,\mu) \tag{20}$$

- The households are on their optimality schedules for savings and labor supply decisions, i.e. the first order conditions defining marginal utility and wages hold, and the price mapping is consistent

$$p(z,\mu) = \Gamma_p(z,\mu) = \frac{1}{C(z,\mu)} \tag{21}$$

$$\omega(z,\mu) = \frac{\phi}{p(z,\mu)} \tag{22}$$

- Aggregate productivity z follows the assumed AR(1) process in logs.

2.2 Krusell-Smith (KS) Method

The Bellman equation we actually solve is the one integrated over the distribution of ξ:

$$\begin{aligned} V^0(\varepsilon,k;z,\mu) = \max_{n\in\mathbf{R}_+}\Big[&\left(z\,\varepsilon\,k^{\alpha}\,n^{\nu} \;-\; \omega\,n \;+\; (1-\delta)\,k\right)p \\ &+\; G(\xi^*(\varepsilon,k;z,\mu))\Big(-\omega\,p\int_0^{\bar{\xi}}\xi\,G(\mathrm{d}\xi) \;+\; \max_{k^*\in\mathbf{R}_+} R\left(\varepsilon,k^*;z,\mu'\right)\Big) \\ &+\; \left(1-G(\xi^*(\varepsilon,k;z,\mu))\right)\,R\left(\varepsilon,(1-\delta)\,k;z,\mu'\right)\Big]. \end{aligned} \tag{23}$$

$$R\left(\varepsilon,k';z,\mu'\right) \equiv -\gamma k' p + \beta\sum_{j=1}^{N_z}\pi_{ij}\sum_{m=1}^{N_\varepsilon} V^0\left(\varepsilon_m,k';z_j,\mu'\right),$$

Forecasting rule,

$$\mu' = \Gamma_\mu(z,\mu), \quad p = \Gamma_p(z,\mu)$$

Here, $G(\xi^*(\varepsilon,k;z,\mu))$ denotes the probability of adjustment. Equation (23) has two maximization problems on the right-hand side. The first maximization is with respect to n, which is determined by the first-order condition. The second maximization is with respect to k^*. This involves finding the k^* that maximizes R, given the values from the forecasting rules, using a numerical optimization algorithm.

The most problematic aspect of solving this model is $\mu(\varepsilon,k)$, which is an infinite-dimensional object. This leads to an infinite number of state variable combinations, making the model unsolvable. Therefore, following Krusell and Smith (1998), we approximate $\mu(\varepsilon,k)$ with aggregate capital K. They interpret this as bounded rationality.[2] Consequently, $\mu(\varepsilon,k)$ in equation (23) and forecasting rules is replaced by K, resulting in the following equation:

[2] Han et al. (2021) construct generalized moments using a neural network, where $\mu(\varepsilon,k)$ is composed of a finite number of agents.

$$V^0(\varepsilon,k;z,K) = \max_{n\in\mathbf{R}_+}\Big[\big(z\,\varepsilon\,k^{\alpha}\,n^{\nu}\;-\;\omega\,n\;+\;(1-\delta)\,k\big)\,p \\ +\;G(\xi^*(\varepsilon,k;z,K))\Big(-\omega\,p\int_0^{\bar{\xi}}\xi\,G(\mathrm{d}\xi)\;+\;\max_{k^*\in\mathbf{R}_+}R\big(\varepsilon,k^*;z,K'\big)\Big) \\ +\;\big(1-G(\xi^*(\varepsilon,k;z,K))\big)\;R\big(\varepsilon,(1-\delta)\,k;z,K'\big)\Big] \tag{24}$$

$$R\left(\varepsilon,k';z,K'\right) \equiv -\gamma k' p + \beta\sum_{j=1}^{N_z}\pi_{ij}\sum_{m=1}^{N_\varepsilon}V^0\left(\varepsilon_m,k';z_j,K'\right) \tag{25}$$

$$\log(K') = a_K + b_K\log(K) \tag{26}$$

$$\log(p) = a_p + b_p\log(K) \tag{27}$$

Forecasting rules are assumed to take the log-linear form.

Algorithm 1: Krusell-Smith Method Procedure

1 Initialize: Set initial forecasting rule parameters and define the state grid;
2 **repeat**
3 **Step 1: Solve the Bellman Equation**;
4 Given forecasting rules, do value function iteration;
5 **Step 2: Simulate the Model**;
6 Run 2500 periods simulations using the obtained value function;
7 **Step 3: Update Forecasting Rules**;
8 Update parameters by regressing simulation outcomes;
9 **Step 4: Check Convergence**;
10 If $\|\log V' - \log V\| < \epsilon$, then terminate; otherwise, set iteration counter iter $\leftarrow$ iter $+ 1$;
11 **until** *convergence condition* $\|\log V' - \log V\| < \epsilon$ *is satisfied*;
12 **Output:** Value, policy function and converged forecasting rule parameters;

Algorithm 1 outlines the Krusell-Smith (KS) solution procedure. The process begins with an **Initialization** step, where the state space grid is defined. For instance, Terry (2017) employ 5 grid points for idiosyncratic productivity ε, 5 for aggregate productivity z, 10 for individual capital k, and 10 for aggregate capital K. Interpolation is used for the continuous state variables k and K. The algorithm then enters an iterative loop:

Step 1: Solve the Bellman Equation Given the current forecasting rules (equations (26) and (27)), the Bellman equation (24) is solved using Value Function Iteration (VFI). In this VFI process, the right-hand side of equation (24) is computed for all combinations of state variables (ε, z, k, K). The resulting values update V^0, and this process is repeated until the value function converges. Throughout this step, the future aggregate capital K' and current price p are determined by the existing forecasting rules.

Step 2: Simulate the Model Once the value function converges in Step 1, the next phase is to simulate the model's economy over a T-period horizon (e.g., 2500 periods) using the obtained value function. This simulation step is critical for generating data to update the forecasting rules but is also the most computationally demanding part of the KS method, especially in heterogeneous firm models. While the VFI for a model like Khan and Thomas (2008) might be relatively

quick (e.g., ∼2 seconds), the simulation can take significantly longer (e.g., ∼3 minutes per outer loop iteration). The primary reason for this computational expense is the need to determine the market-clearing equilibrium price p in each period of the simulation, as this price is implicitly defined by the entire distribution of firms. This contrasts with the original Krusell and Smith (1998) model where the interest rate could be directly inferred from aggregate capital K.

Algorithm 2: Simulation with Price Bisection (KS Method)

```
for t = 1, ..., 2500 periods do
    while price has not converged do
        Guess the equilibrium price: p_guess;
        for each state combination (ε, k) do
            for each candidate action (k′) do
                Compute the right-hand side of the Bellman equation;
            end
            Search for the combination that maximizes the right-hand side;
        end
        Aggregate and compute excess demand;
        Update the price interval based on excess demand;
    end
end
```

Inner Workings of the Simulation: The Price-Finding Bottleneck Algorithm 2 provides a schematic of the sub-procedure executed within each period of the simulation. The core challenge is to ensure the market-clearing condition, $p(z,\mu) = \frac{1}{C(z,\mu)}$ (equation (21)), holds in every simulation period t. Finding this equilibrium price $p(z,\mu)$ typically relies on an iterative numerical search, such as the bisection method. This search itself is a multi-step process. For **each** candidate price p_{guess} evaluated during this iterative search, the following sequence of operations must be performed:

(i) **Recompute Optimal Firm Investment (k^*):** For the current p_{guess}, firms' optimal investment choices (k^*) must be recalculated. This step requires, for each firm (or each point in the histogram approximation of the firm distribution), finding the investment level k' that maximizes its continuation value $R(\varepsilon, k'; z, K')$ (from equation (25)), considering all feasible investment levels. This re-optimization is essential because the k^* determined during the VFI in Step 1 was based on prices predicted by the *forecasting rules*, which may differ from the current p_{guess} being tested to clear the market.

(ii) **Aggregate Actions and Check Market Clearing:** Once all firms have their new optimal actions based on p_{guess}, these actions (e.g., investment, labor demand) are aggregated across the distribution to compute aggregate supply and demand (e.g., aggregate consumption $C(z,\mu)$), and thereby the excess demand in the market.

(iii) **Update Price Guess:** Based on the calculated excess demand, the bisection algorithm updates the price guess (i.e., narrows the search interval for p). The process then returns to step (1) with the new p_{guess}, continuing until the price converges (i.e., excess demand is sufficiently close to zero).

This repetitive cycle—re-optimizing individual firm behavior (step 1), aggregating actions (step 2), and then refining the price guess (step 3)—is detailed in Algorithm 2. Its significant computational

demand stems from this iterative process, particularly the nested loops involved in searching for the equilibrium price and performing the firm-level re-optimization (step 1) for each price candidate. The **computational burden** arises because this entire three-step sequence must be executed for every iteration of the price bisection search, and this entire search is performed in every single period of the T-period simulation. The burden scales with the number of state variables and the density of their grid points. For example, with the 5×50 histogram points used by Terry (2017) for (ε, k), the process is already intensive. This burden is exacerbated in models with larger state spaces; for instance, the histogram in Bloom et al. (2018) involves 16,835 points, significantly increasing the number of calculations required for each price guess within each simulation period.

Step 3: Update Forecasting Rules After completing the T-period simulation, the simulated time series for aggregate capital K and price p are used to update the parameters of the forecasting rules (equations (26) and (27)), typically via ordinary least squares regression.

Step 4: Check Convergence The algorithm then checks if the value function has converged (e.g., $\| \log V' - \log V \| < \epsilon$). If convergence is achieved, the process terminates, yielding the converged value function, policy functions, and forecasting rule parameters. Otherwise, the iteration counter is incremented, and the algorithm returns to Step 1 with the updated forecasting rules.

2.3 My Method

As seen in the previous section, the problem with the KS method lies in the need to recompute optimal actions every time the price is updated. Therefore, this issue can be resolved by constructing a policy function that includes the equilibrium price itself as an input (state variable). In my method, both the policy function dictating firm actions and the value function are approximated using neural networks, with the crucial feature being that the policy function explicitly takes the price as an argument: $k^* \approx g_{nn}(z, \varepsilon, p, K)$. This price-conditional policy function, once trained, can directly output optimal actions for any given price encountered during the simulation's bisection search, thereby eliminating the costly re-optimization bottleneck. The adoption of deep neural networks is particularly advantageous here due to their proven ability to approximate highly complex, non-linear functions in high-dimensional settings. Heterogeneous firm models, especially when augmenting the state space with the equilibrium price as a policy input (as is done here), naturally lead to such high dimensionality. For instance, the model by Bloom et al. (2018) already involves millions of state-space combinations even before considering the price as a continuous input to the policy. Neural networks, supported by theoretical insights such as Barron's theorem which highlights their potential to overcome the curse of dimensionality for certain classes of functions, provide a robust framework for accurately learning these high-dimensional policy and value functions. Furthermore, to serve as a more general solution method, I also approximate the forecasting rules with neural networks, following Fernández-Villaverde et al. (2023). There are pros and cons to using log-linear regression versus neural networks for the forecasting rules in terms of speed and accuracy. Therefore, I will discuss this in later. In the following, I will describe the version where the forecasting rules are also approximated by neural networks.

Overall Solution Algorithm My proposed solution method, detailed in Algorithm 3, is an iterative procedure. It starts by **initializing** the neural networks for the value function (V^0_{nn}), policy function (g_{nn}), and forecasting rules ($\Gamma^{nn}_p, \Gamma^{nn}_\mu$), along with optimizers and an initial training dataset $\mathcal{D}_0$. The algorithm then enters a **Main Loop**, which iterates until the forecasting rules for

Algorithm 3: Solution Algorithm of the Proposed method

Data: Models $V_{nn}^0, g_{nn}, \Gamma_p^{nn}, \Gamma_\mu^{nn}$, initial dataset $\mathcal{D}_0$, optimizers, schedulers
Result: Trained models $V_{nn}^0, g_{nn}, \Gamma_p^{nn}, \Gamma_\mu^{nn}$

Initialization:
Initialize $V_{nn}^0, g_{nn}, \Gamma_p^{nn}, \Gamma_\mu^{nn}$, optimizers, schedulers
Set $\mathcal{D} \leftarrow \mathcal{D}_0$.
Main Loop (repeat until forecasting rules converge):
while $\|\Gamma_p^{nn,new} - \Gamma_p^{nn,old}\|_\infty > \delta_1$ *or* $\|\Gamma_\mu^{nn,new} - \Gamma_\mu^{nn,old}\|_\infty > \delta_1$ **do**
    1. Value & Policy Iteration (repeat until RHS change $\leq \epsilon$):
    Initialize rhs_{prev} and rhs.
    while $\|rhs - rhs_{prev}\|_\infty > \epsilon$ **do**
        $rhs_{prev} \leftarrow rhs$.
        $rhs \leftarrow$ TrainPolicyNetwork$(g_{nn}, V_{nn}^0, \mathcal{D})$
        $loss_V \leftarrow$ TrainValueNetwork$(V_{nn}^0, g_{nn}, \mathcal{D})$
    end
    2. Simulation and Forecast Update:
    $\mathcal{D}_{new} \leftarrow$ SimulateModel$(g_{nn}, \Gamma_p^{nn}, \Gamma_\mu^{nn})$
    $\Gamma_p^{nn,old} \leftarrow \Gamma_p^{nn}$
    $\Gamma_\mu^{nn,old} \leftarrow \Gamma_\mu^{nn}$
    $\Gamma_p^{nn} \leftarrow$ UpdatePriceForecast$(\Gamma_p^{nn}, \mathcal{D}_{new})$
    $\Gamma_\mu^{nn} \leftarrow$ UpdateCapitalForecast$(\Gamma_\mu^{nn}, \mathcal{D}_{new})$
    $\mathcal{D} \leftarrow$ UpdateTrainingData$(\mathcal{D}, \mathcal{D}_{new})$
end
return $V_{nn}^0, g_{nn}, \Gamma_p^{nn}, \Gamma_\mu^{nn}$

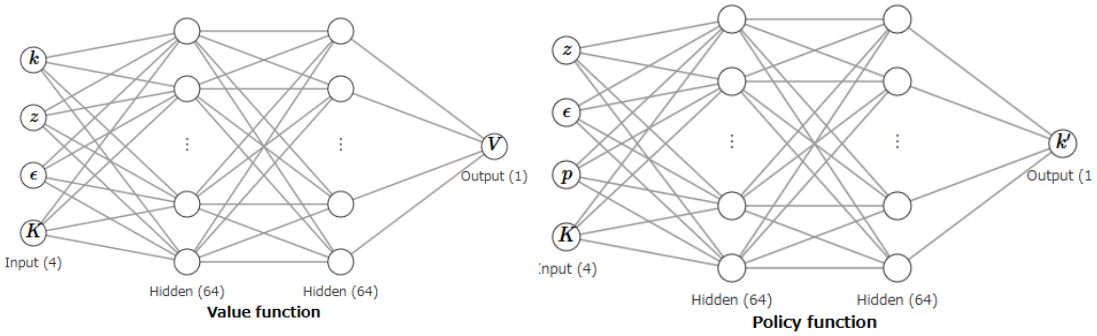
the aggregate price and capital achieve convergence. This outer loop structure is analogous to the Krusell-Smith algorithm. Within each iteration of this main loop, two key phases are executed sequentially:

First, a **Value & Policy Iteration** phase (Step 1 in Algorithm 3) solves for the optimal policy and value functions consistent with the current forecasting rules. This phase itself is an inner loop that continues until the change in the right-hand side of the Bellman equation (denoted 'rhs' in the algorithm) is below a threshold ϵ. Inside this inner loop, the policy network g_{nn} is trained by the **TrainPolicyNetwork** procedure, and subsequently, the value network V_{nn}^0 is trained to minimize the Bellman error via the **TrainValueNetwork** procedure, both using the current training data $\mathcal{D}$. When training the policy network with **TrainPolicyNetwork**, the parameters of the value network V_{nn}^0 are held fixed, and conversely, when training the value network with **TrainValueNetwork**, the policy network g_{nn}'s parameters are kept constant. In essence, this alternating update process is analogous to traditional Policy Iteration.

Second, once the policy and value functions have stabilized for the current forecasting rules, a **Simulation and Forecast Update** phase (Step 2) is performed. The economic model is simulated using the **SimulateModel** function with the converged policy g_{nn} and current forecasting rules $\Gamma_p^{nn}, \Gamma_\mu^{nn}$ to generate a new dataset of time series data, $\mathcal{D}_{new}$. This new data is then used to update the forecasting rule networks Γ_p^{nn} and Γ_μ^{nn} (through **UpdatePriceForecast** and

UpdateCapitalForecast). Finally, the main training dataset $\mathcal{D}$ is augmented with this new simulation data using **UpdateTrainingData**.

The main loop then repeats, taking the newly updated forecasting rules into the next **Value & Policy Iteration** phase, until the convergence criterion for the forecasting rules (e.g., $\|\Gamma_p^{nn,new} - \Gamma_p^{nn,old}\|_\infty \leq \delta_1$) is met. The algorithm then returns the converged neural network models.



(a) Value Function Approximation

(b) Policy Function Approximation

Figure 1: Neural Network Approximations

Neural Network Approximations and Loss Functions Specifically, the value and policy functions in equation (24) are approximated with neural networks as follows:

$$V^0(\varepsilon, k; z, \mu) \approx V_{nn}^0(\varepsilon, k; z, K), \tag{28}$$

$$k^* \approx g_{nn}(z, \varepsilon, p, K). \tag{29}$$

Figures 1a and 1b show snapshots of the value and policy functions, respectively. The neural networks have 2 hidden layers with 64 neurons per layer. The value function takes the state variables (ε, k, z, K) directly as input and outputs the value. The policy function g_{nn} takes (z, ε, p, K) as its inputs. As noted in Section 2.1, the optimal capital choice k^* is fundamentally a function of (z, ε, K) (where K proxies for μ as in the KS method). My method incorporates the *current* price p as an additional, direct input to this policy function. This explicit conditioning on p is crucial: it allows the neural network to learn the optimal k^* for any given p encountered during the simulation's price bisection search, thereby avoiding the need for repeated re-optimization.

The policy function $g_{nn}(z, \varepsilon, p, K)$ is trained to output the capital stock k' that maximizes the firm's expected continuation value(represented by R, as in equation (25)). This is achieved by minimizing the **Policy Function Loss**:

$$\mathcal{L}_{policy} = -\frac{1}{N}\sum_{i=1}^{N} R\Big(\varepsilon_i, k_i'; z_i, K_i\Big). \tag{30}$$

The expected continuation value $R(\varepsilon_i, k_i'; z_i, K_i)$, using future values from the value network V_{nn}^0, is:

$$R(\varepsilon_i, k_i'; z_i, K_i) = -\gamma\, k_i' p_{error,i} + \beta \sum_{j=1}^{N_z} \pi_{ij} \sum_{m=1}^{N_\varepsilon} \pi_{im} V_{nn}^0(\varepsilon_m, k_i'; z_j, K_i'). \tag{31}$$

To calculate $R(\cdot)$, the key inputs are the perturbed price $p_{error,i}$ and the resulting action k_i'. The perturbed price $p_{error,i}$ is constructed by adding a uniform noise term $error_i \in [-0.15, 0.15]$ to the price forecast $\Gamma_p^{nn}(z_i, K_i)$:

$$p_{error,i} = \Gamma_p^{nn}(z_i, K_i) + error_i. \tag{32}$$

This noise injection during training aims to make the policy function robust to the price variations encountered during the bisection search for the equilibrium price within the simulation. The range $[-0.15, 0.15]$ for the noise is chosen to sufficiently cover the typical fluctuations in price guesses observed during this bisection process. The action k_i' is then the output of the policy network, conditioned on the state $(z_i, \varepsilon_i, K_i)$ and this $p_{error,i}$:

$$k_i' = g_{nn}(z_i, \varepsilon_i, p_{error,i}, K_i). \tag{33}$$

The parameters of g_{nn} are optimized using the ADAM optimizer to minimize $\mathcal{L}_{policy}$.

The value function V_{nn}^0 is trained to satisfy the Bellman equation across the state space. The **Value Function Loss**, $\mathcal{L}_V$, quantifies the deviation from this ideal:

$$\mathcal{L}_V = \frac{1}{N} \sum_{i=1}^{N} \Big(V_{nn}^0(\varepsilon_i, k_i; z_i, K_i) - \text{RHS}_i \Big)^2. \tag{34}$$

This loss measures the mean squared error between the current network's output $V_{nn}^0(\varepsilon_i, k_i; z_i, K_i)$ and a target value, RHS_i. The term RHS_i is constructed from the right-hand side of the Bellman equation (24), representing the expected discounted value if the firm is at state (ε_i, k_i) in aggregate conditions (z_i, K_i) and follows the policy g_{nn}:

$$\begin{aligned} \text{RHS}_i = {} & \left(z_i \varepsilon_i k_i^\alpha n_i^\nu - \omega_i n_i + (1-\delta) k_i\right) p_i \\ & + G(\xi^*(\varepsilon_i, k_i; z_i, K_i)) \left[-\omega_i p_i \int_0^{\bar{\xi}} \xi G(d\xi) + R\left(\varepsilon_i, k_i^*; z_i, K_i'\right) \right] \\ & + \left(1 - G(\xi^*(\varepsilon_i, k_i; z_i, K_i))\right) R\left(\varepsilon_i, (1-\delta) k_i; z_i, K_i'\right). \end{aligned} \tag{35}$$

The components $R(\cdot)$ (the continuation value if investment occurs) and k_i^* (the optimal capital choice from the policy network) are given by:

$$R(\varepsilon_i, k_i^*; z_i, K_i') = -\gamma k_i^* p_i + \beta \sum_{j=1}^{N_z} \pi_{ij} \sum_{m=1}^{N_\varepsilon} V_{nn}^0\left(\varepsilon_m, k_i^*; z_j, K_i'\right) \tag{36}$$

$$k_i^* = g_{nn}(z_i, \varepsilon_i, p_i, K_i) \tag{37}$$

Unlike standard Value Function Iteration (VFI), where the Bellman equation's right-hand side

directly becomes the next iteration's value function, here the neural network V^0_{nn} is trained via gradient descent to minimize this squared difference $\mathcal{L}_V$. This process iteratively adjusts the network's parameters so that its output $V^0_{nn}(\text{state}_i)$ more closely aligns with the target RHS_i, effectively pushing the network to learn a representation that satisfies the Bellman optimality condition.

Forecasting rules are similarly approximated:

$$\Gamma_\mu(z,\mu) \approx \Gamma^{nn}_\mu(z,K), \tag{38}$$

$$\Gamma_p(z,\mu) \approx \Gamma^{nn}_p(z,K). \tag{39}$$

These neural networks, Γ^{nn}_μ (for next period's aggregate capital K') and Γ^{nn}_p (for the current price p), are trained using data from the most recent simulation block, $\mathcal{D}_{new}$. Specifically, they learn from the recorded time series of K' and p generated in $\mathcal{D}_{new}$, using the corresponding series of (z,K) states as inputs. The ADAM optimizer is employed, chosen primarily for its straightforward implementation and its ability to ensure stability (when used with a relatively low learning rate) by mitigating large fluctuations in the forecasting rules between iterations of the main loop.[3]

2.4 Results

In this section, I compare the performance of my proposed method with the benchmark Krusell-Smith (KS) method. For the KS method, I use the Fortran code provided by Terry (2017). My method is implemented in Python using the PyTorch library. The parameter settings and histogram grid specifications are identical across all methods. Computations were performed on a system with a Core(TM) i7-12700F 2.10 GHz CPU and a GeForce RTX 3080 GPU. I focus on several important dimensions: (i) computation time, (ii) the Bellman equation error, (iii) the dynamics of macroeconomic variables in an unconditional simulation, (iv) key microeconomic moments of the investment rate, and (v) the accuracy of the forecast rules. I present results labeled "My method (reg)" when using log-linear regression for the forecasting rule, and "My method (NN)" when using neural networks for the forecasting rule.

Computational Speed Comparison First, I show computation speed. Table 1 compares computation time. The VFI and simulation times reported in the table are measured during the first iteration of the outer loop. For the simulations, the KS method and My method (reg) use a simulation length of 2,500 periods, while My method (NN) uses a longer simulation of 8,000 periods for the stability of forecasting rules. It is worth noting that, from the second iteration onward, the VFI step in My method benefits from using the previous iteration's results as a warm start, significantly reducing the computation time to approximately 30 seconds per iteration. All reported total times are the cumulative results of six iterations of the outer loop.

Bellman Equation Error Table 2 reports the results of a Bellman error comparison, where the error is measured as $\log V' - \log V$ for a set of state points. Both methods calculate this error across the same 250 grid points, constructed by taking 5 points for the aggregate shock z, 5 for the idiosyncratic shock ε, 10 for individual capital k, and 10 for the aggregate capital K. The

[3] Fernández-Villaverde et al. (2023), for example, describe updating their forecasting rules by using the entire block of newly simulated data for a single batch update, as an alternative to mini-batch SGD (Stochastic Gradient Descent).

Method	VFI Time (sec)	Simulation Time (sec)	Total Time (min)
KS method (Terry (2017))	3	193	24
My method (reg)	123	68	14
My method (NN)	130	219	37

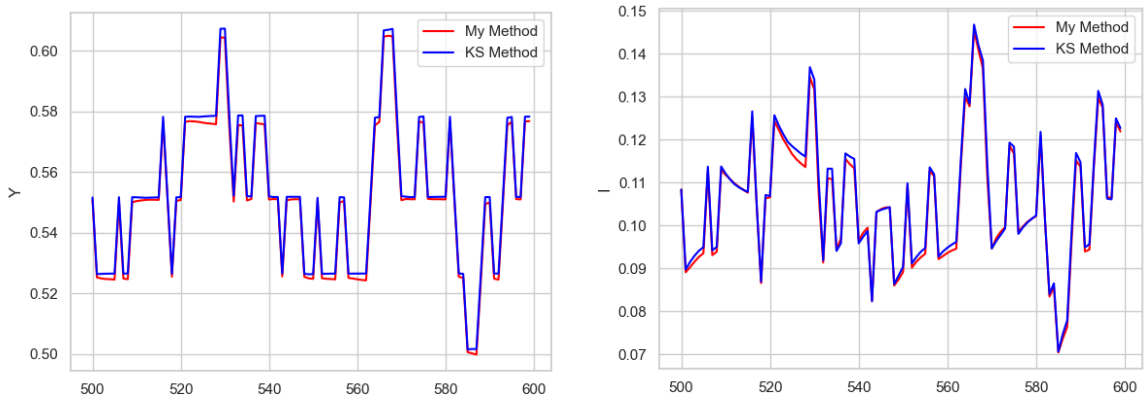
Table 1: Computation time comparison

table shows the **average** Bellman error over these 250 points. As shown in Table 2, my NN-based method achieves a lower average Bellman error (0.0015) compared to the KS method (0.0085), indicating that my neural network approximation yields a value function that satisfies the Bellman equation more accurately on average across the sampled state space.

Method	**Average Bellman Error** $(\log V' - \log V)$
KS method (Terry (2017))	0.0085
My method (NN)	0.0015

Table 2: Average Bellman equation error: $\log V' - \log V$.

Unconditional Simulation Comparison Next, I assess how the approximate policies perform in a long-run (unconditional) simulation. Figure 2 compares the time paths for aggregate output and investment between the KS method and my NN method for the same realization of shocks. The aggregate time paths under my NN method track those from the KS method closely. Despite relying on neural networks for approximating the policy function and value function, the NN method captures the dynamics of aggregate output and investment with high fidelity.



(a) Aggregate output (b) Aggregate investment

Figure 2: Comparison of unconditional simulation paths between KS and NN methods.

Microeconomic Investment-Rate Moments Table 3 reports key microeconomic investment-rate moments, comparing the benchmark KS method with my proposed methods (My method (reg)

and My method (NN)). My methods generate micro-level investment statistics that are quantitatively close to those produced by the KS benchmark.[4]

	KS	My method (reg)	My method (NN)
$\frac{i}{k}$	0.0947	0.1042	0.1019
$\sigma\left(\frac{i}{k}\right)$	0.2597	0.3362	0.3214
$\mathbb{P}\left(\frac{i}{k}=0\right)$	0.7693	0.7635	0.7414
$\mathbb{P}\left(\frac{i}{k}\geq 0.2\right)$	0.1724	0.1606	0.1714
$\mathbb{P}\left(\frac{i}{k}\leq -0.2\right)$	0.0280	0.0311	0.0331
$\mathbb{P}\left(\frac{i}{k}>0\right)$	0.1890	0.1961	0.2152
$\mathbb{P}\left(\frac{i}{k}<0\right)$	0.0417	0.0403	0.0528

Table 3: Microeconomic investment-rate moments. For KS, we reproduce the results from Terry (2017).

Forecast-System Accuracy Table 4 reports measures for the forecast of the aggregate price p and next-period aggregate capital K', comparing my NN method to KS.

	p (%)			K' (%)		
	My (reg)	My (NN)	KS	My (reg)	My (NN)	KS
Den Haan Statistics						
Max	0.53	0.75	0.11	0.89	2.08	0.39
Mean	0.07	0.12	0.06	0.16	0.33	0.25
Root Mean Squared Error (RMSE)						
RMSE	0.09	0.14	0.05	0.08	0.14	0.05
Forecast Regression R^2						
R^2	0.9994	0.9983	0.9998	0.9948	0.9995	0.9995

Table 4: Internal accuracy of forecasting rules for the aggregate price p and next-period capital K'. Values for KS are reproduced from Terry (2017).

As can be seen from the table, the KS method exhibits the best overall accuracy. My method (reg), which uses log-linear regression for the forecasting rule, achieves comparable or slightly lower accuracy, but is still quite accurate. On the other hand, the forecasting rule using neural networks (My method (NN)) shows relatively lower accuracy. Two potential reasons can be considered for this. First, the overall error levels are very small. The RMSE for p in Table 4 is presented in percentage terms; however, when measured on the original scale, the RMSE is 0.0008 (and the MSE is 6.4×10^{-7}). Second, the number of training samples for the forecasting rule is limited.

[4]Slight differences in the moments reported for My method (NN) may arise partly because these statistics were computed using a longer simulation (8,000 periods) than that used for the KS method and My method (reg) (2,500 periods), leading to variation from different realized shock histories.

The training samples for the forecasting rule are drawn from the preceding simulation. If there are differences in the realizations of shocks, some shocks may not be sufficiently learned. Unlike linear regression, neural networks share parameters across different shock realizations, making them potentially more susceptible to this issue.

Therefore, the choice between using log-linear regression or neural networks for the forecasting rule should depend on the desired trade-off between speed, accuracy, and the ability to capture non-linearities. However, even when using neural networks for forecasting, coefficients from a log-linear regression are needed for initialization. Consequently, beginning with log-linear regression is generally advisable as a practical starting point. A key advantage of my method remains the ease with which these forecasting rule components can be interchanged.

3 Application to the Model of Bloom et al. (2018)

In this section, I demonstrate that the efficiency of my method becomes even more pronounced in larger models. I apply my method to the model of Bloom et al. (2018). The basic structure of this model is similar to that of Khan and Thomas (2008), but Bloom et al. (2018) additionally incorporates both aggregate and idiosyncratic stochastic volatility shocks for firm productivity, as well as nontrivial capital and labor adjustment costs. Below, I provide a more detailed overview of the production environment and demonstrate how my neural-network-based solution can handle the large state space more efficiently than traditional methods.

3.1 Model

Each firm produces output according to a Cobb-Douglas production function:

$$y = z\varepsilon k^{\alpha} n^{\nu}, \tag{40}$$

where z is an aggregate productivity process, ε is idiosyncratic productivity, and k and n are firm-specific capital and labor, respectively.

Productivity Processes. Both aggregate and idiosyncratic productivity follow persistent stochastic processes with time-varying volatility. Specifically:

- **Aggregate Productivity:**

$$\log(z') = \rho_z \log(z) + \sigma_z u'_z, \quad u'_z \sim N(0,1) \tag{41}$$

- **Idiosyncratic Productivity:**

$$\log(\varepsilon') = \rho_\varepsilon \log(\varepsilon) + \sigma_\varepsilon u'_\varepsilon, \quad u'_\varepsilon \sim N(0,1) \tag{42}$$

The time-varying volatilities, σ_z and σ_ε capture switches between "low-uncertainty" and "high-uncertainty" regimes, typically modeled as states governed by a two-state Markov chain. These aggregate uncertainty states $(\sigma_z, \sigma_\varepsilon)$ become part of the aggregate state space.

Capital Dynamics and Adjustment Costs. Capital evolves according to the standard law of motion:

$$k' = (1 - \delta_k)\, k + i, \tag{43}$$

where δ_k is the depreciation rate and i is investment. Investment is subject to adjustment costs AC^k, which include a fixed cost proportional to output and a term capturing partial irreversibility:

$$\text{AC}^k = \mathbf{1}_{|i|>0}\, y\, F^K + S|i|\, \mathbf{1}_{i<0} \tag{44}$$

Here, $\mathbf{1}_{(\cdot)}$ is an indicator function, F^K is the fixed disruption cost parameter, and S is the resale loss fraction for disinvested capital.

Labor Dynamics and Adjustment Costs. Labor (hours worked) also faces adjustment frictions and evolves as:

$$n = (1 - \delta_n)\, n_{-1} + s, \tag{45}$$

where n_{-1} is the previous period's labor, δ_n is an exogenous separation rate and s represents net hiring (or firing). Labor adjustment costs AC^n include a fixed cost proportional to output and a variable cost related to hiring/firing flow:

$$\text{AC}^n = \mathbf{1}_{|s|>0}\, y\, F^L + |s|\, H\, w \tag{46}$$

where F^L is the fixed disruption cost for labor adjustment, second term is a linear hiring/firing cost, which is expressed as a fraction of the aggregate wage w. Crucially, because labor adjustment depends on the previous period's labor input, n_{-1} becomes an endogenous state variable for the firm.

Firm Problem and State Space. The firm's objective is to maximize the present discounted value of profits net of adjustment costs. Following the approach in Khan and Thomas (2008), we define the intertemporal price of consumption goods $p(z, \sigma_z, \sigma_\varepsilon, \mu)$ as the household's marginal utility of current consumption. The wage $w(z, \sigma_z, \sigma_\varepsilon, \mu)$ is also derived from the household's first-order conditions. This definition of p allows us to redefine the firm's value function in terms of these marginal utility units, $\tilde{V} \equiv pV$. The Bellman equation for $\tilde{V}$ can then be expressed as follows:

$$\begin{aligned}\tilde{V}(k, n_{-1}, \varepsilon; z, \sigma_z, \sigma_\varepsilon, \mu) = \max_{i,s} \Big\{ & p(z, \sigma_z, \sigma_\varepsilon, \mu) \Big[y - w(z, \sigma_z, \sigma_\varepsilon, \mu) n - i - \text{AC}^k - \text{AC}^n \Big] \\ & + \beta \mathbb{E} \Big[\tilde{V}(k', n, \varepsilon'; z', \sigma_z', \sigma_\varepsilon', \mu') \Big] \Big\}.\end{aligned} \tag{47}$$

This value function $\tilde{V}$ depends on the firm's individual state (k, n_{-1}, ε) and the aggregate state $(z, \sigma_z, \sigma_\varepsilon, \mu)$, where μ represents the distribution of firms over their idiosyncratic states and $\sigma_z, \sigma_\varepsilon$ represent the current volatility regime. Solving the model requires tracking the evolution of the

distribution μ and the equilibrium price p via forecasting rules:

$$\begin{aligned} \mu' &= \Gamma_\mu(z, \sigma_z, \sigma_\varepsilon, \mu), \\ p &= \Gamma_p(z, \sigma_z, \sigma_\varepsilon, \mu). \end{aligned} \tag{48}$$

Bloom et al. (2018) solve this model using the Krusell-Smith method, approximating the infinite-dimensional distribution μ with aggregate capital K. The forecasting rules (48) are thus typically approximated as functions of the aggregate state variables $(z, \sigma_z, \sigma_\varepsilon)$ and aggregate capital K, i.e., $K' = \Gamma_K(z, \sigma_z, \sigma_\varepsilon, K)$ and $p = \Gamma_p(z, \sigma_z, \sigma_\varepsilon, K)$.

In the original code, the firm's problem is solved by discretizing the state space. The number of grid points for each state variable, corresponding to the order of individual states (k, n_{-1}, ε) and aggregate states relevant for the firm's decision and forecasting $(z, \sigma_z, \sigma_\varepsilon, K)$, is shown below:

$$\underbrace{91}_{\substack{\text{grid points} \\ \text{for } k}} \times \underbrace{37}_{\substack{\text{grid points} \\ \text{for } n_{-1}}} \times \underbrace{5}_{\substack{\text{grid points} \\ \text{for } \varepsilon}} \times \underbrace{5}_{\substack{\text{grid points} \\ \text{for } z}} \times \underbrace{2}_{\substack{\text{states} \\ \text{for } \sigma_z}} \times \underbrace{2}_{\substack{\text{states} \\ \text{for } \sigma_\varepsilon}} \times \underbrace{10}_{\substack{\text{grid points} \\ \text{for } K}} = 3{,}367{,}000 \quad \text{points.}$$

Here, σ_z and σ_ε represent the two possible volatility regimes (e.g., high/low), and the 10 points for K represent the discretization of aggregate capital used to approximate the distribution μ in the forecasting rules. This large state space poses a significant computational challenge.

3.2 Result

In this section, I compare the performance of my method against results generated using the code provided by Bloom et al. (2018). It is important to note that the benchmark Bloom et al. (2018) results reported here are based on a modified version of their original code, specifically concerning the discretization grid. Consequently, these results may differ from those reported in the published version of Bloom et al. (2018).[5]

Computation and Simulation Speed. Table 5 presents the total solution time, VFI time, and simulation time for each method. As the table shows, my method achieves a significant reduction in computation time compared to the KS method. Simulations were run for 5,000 periods for the KS method and my method (reg), and for 15,000 periods for my method (NN) to ensure stability of the forecasting rules. All reported total times are the cumulative results of 16 iterations of the outer loop. Unlike the results for the Khan and Thomas (2008) model in the previous section, I observe a substantial difference in the Value Function Iteration (VFI) time as well.

Method	Total Time (min)	VFI Time (min)	Simulation Time (min)
KS method	2380	29	133
My Method (reg)	50	3.3	1.7
My Method (NN)	95	3.3	4.5

Table 5: Speed comparison for the Bloom et al. (2018) model. VFI time refers to the time required to solve the bellman equation. Simulation time is measured for 5,000 periods for the first simulation.

[5]The benchmark results use a modified version of the original Bloom et al. (2018) code regarding the state space grid to ensure a consistent treatment of adjustment costs, necessary for comparability with my continuous-action method. A detailed explanation of the modification and its rationale is provided in Appendix C.

Sources of Speed Improvement. The dramatic reduction in simulation time, particularly evident in Table 5, stems from two key features of my method. Firstly, as discussed previously, the use of a policy function $g_{nn}(z, \varepsilon, p, K)$ that includes the equilibrium price p as a state variable eliminates the need for repeated optimization of firm actions within the price bisection loop required in each simulation period. Once trained, the policy network provides an instantaneous mapping from the state (including the guessed price) to the optimal action, bypassing the computationally intensive maximization step inherent in the standard KS simulation.

Secondly, I leverage GPU acceleration for the remaining computations within the simulation. Specifically, finding the equilibrium price in each period involves evaluating the policy function, computing firm-level variables (like output and investment) based on the chosen actions, and then aggregating these across all points in the distribution grid to check the market clearing condition (**??**). In the Bloom et al. (2018) model, this distribution grid consists of 16,835 points, making efficient computation crucial. While the final aggregation step is inherently sequential, the preceding steps—evaluating the neural network policy function and computing resulting variables for each of the 16,835 grid points—are highly parallelizable. Executing these steps on a GPU allows for substantial speed gains compared to a CPU implementation. This parallelization translates into a significant performance difference: while a CPU implementation of my method in the simulation part might process approximately 2 simulation periods per second, the GPU implementation achieves around 50 periods per second. Consequently, the synergy between the pre-trained, price-conditional policy function and GPU-accelerated parallelization explains the drastic improvement in simulation speed observed in Table 5.

Business-Cycle Properties. Table 6 reports the HP-filtered standard deviation, ratio to output, and correlation with output for key macroeconomic variables. I compare the results from the KS method, my method using log-linear regression for forecasting (My method (reg)), and my method using neural networks for forecasting (My method (NN)).

	KS Method			My Method (Reg)			My Method (NN)		
	Std	Ratio	Corr	Std	Ratio	Corr	Std	Ratio	Corr
Output	1.832	1.000	1.000	1.532	1.000	1.000	1.586	1.000	1.000
Investment	9.634	5.257	0.925	7.508	4.901	0.854	7.908	4.985	0.877
Consumption	0.910	0.501	0.617	1.033	0.674	0.531	0.968	0.610	0.497
Hours	1.644	0.897	0.821	1.260	0.823	0.721	1.217	0.767	0.784

Table 6: HP-filtered standard deviations, ratios to output, and correlations with output

Internal Accuracy of Forecast Systems. Table 7 presents the internal accuracy of the forecasting rules for the aggregate price p and next-period aggregate capital K'. I report the Den Haan statistics (maximum and mean), the root mean squared error (RMSE), and the R^2 from forecast regressions. Both of my methods achieve high accuracy in forecasting the aggregate price and next-period capital. The Den Haan statistics, RMSE, and R^2 values are comparable to, or better than, those of the KS method. The neural network forecasting rule (My method (NN)) shows particularly strong performance, with slightly better accuracy metrics than the linear regression forecasting rule (My method (reg)) in several cases. Appendix A.2 provides detailed forecast accuracy statistics broken down by aggregate shock state for the Bloom et al. (2018) model application

(see Table 9). Furthermore, Appendix D explores the relationship between the length of the simulation used for training and the resulting forecast accuracy. For the log-linear forecasting rules (My method (reg)), increasing the simulation length yielded only marginal improvements in accuracy (Table 10 and Table 11). Notably, even compared to log-linear rules trained on longer simulations, the neural network approach (My method (NN)) consistently achieves higher accuracy for the price forecast (p). This suggests the presence of significant non-linearities in the determination of the equilibrium price, which the neural network is better able to capture compared to the log-linear specification. For detailed comparisons across different simulation lengths, refer to Appendix D.

Table 7: Internal accuracy for forecast performance

	p (%)			K' (%)		
	My (reg)	My (NN)	KS	My (reg)	My (NN)	KS
Den Haan Statistics						
Maximum	4.02	3.56	3.67	6.03	6.04	6.87
Mean	0.87	0.84	0.86	1.91	1.72	1.97
Root Mean Squared Error						
RMSE	0.55	0.46	0.47	0.22	0.26	0.42
Forecast Regression R^2						
R^2	0.9682	0.9799	0.9786	0.9958	0.9959	0.9901

4 Conclusion

This paper addressed the significant computational challenge posed by heterogeneous firm models with aggregate uncertainty, particularly those where equilibrium prices are implicitly determined by the state distribution. I introduced a novel global solution method that integrates deep learning techniques into the widely-used Krusell-Smith framework. The core innovations are the approximation of the value and policy functions using neural networks and, crucially, the inclusion of the equilibrium price as an explicit state variable in the policy function approximation $g_{nn}(z, \varepsilon, p, K)$.

This approach directly tackles the primary computational bottleneck inherent in simulating such models: the iterative search for market-clearing prices within each period. By pre-training a policy function conditioned on the price, my method bypasses the need for repeated optimization during simulation, leveraging the capacity of neural networks to efficiently represent high-dimensional functions. Combined with GPU acceleration for parallel computation across the state distribution, this strategy yields dramatic reductions in computation time.

Applying the method to the models of Khan and Thomas (2008) and Bloom et al. (2018), I demonstrated its practical benefits. The results show speed improvements of up to 50-fold compared to benchmark implementations, particularly in the complex Bloom et al. (2018) environment. This efficiency is achieved while maintaining high standards of accuracy, as measured by

Bellman equation errors, the replication of key micro- and macroeconomic moments, and the internal consistency of forecasting rules.

The computational gains offered by this method have significant implications for macroeconomic research. It makes the analysis of richer, more realistic heterogeneous firm models computationally feasible, allowing for the inclusion of features like multiple shocks, non-convexities, and complex adjustment cost structures that were previously prohibitive. This facilitates more rapid policy experiments, extensive sensitivity analysis, and robust model calibration. Furthermore, by preserving the overall structure of the Krusell-Smith algorithm, the method remains accessible and relatively straightforward to implement for researchers familiar with standard techniques.

References

Ahn, S., G. Kaplan, B. Moll, T. Winberry, and C. Wolf (2018). When inequality matters for macro and macro matters for inequality. *NBER macroeconomics annual 32*(1), 1–75.

Algan, Y., O. Allais, and W. J. Den Haan (2008). Solving heterogeneous-agent models with parameterized cross-sectional distributions. *Journal of Economic Dynamics and Control 32*(3), 875–908.

Auclert, A., B. Bardóczy, M. Rognlie, and L. Straub (2021). Using the sequence-space jacobian to solve and estimate heterogeneous-agent models. *Econometrica 89*(5), 2375–2408.

Azinovic, M., L. Gaegauf, and S. Scheidegger (2022). Deep equilibrium nets. *International Economic Review 63*(4), 1471–1525.

Bloom, N., M. Floetotto, N. Jaimovich, I. Saporta-Eksten, and S. J. Terry (2018). Really uncertain business cycles. *Econometrica 86*(3), 1031–1065.

Boppart, T., P. Krusell, and K. Mitman (2018). Exploiting mit shocks in heterogeneous-agent economies: the impulse response as a numerical derivative. *Journal of Economic Dynamics and Control 89*, 68–92.

Den Haan, W. J. (2010). Comparison of solutions to the incomplete markets model with aggregate uncertainty. *Journal of Economic Dynamics and Control 34*(1), 4–27.

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

Fernández-Villaverde, J., S. Hurtado, and G. Nuno (2023). Financial frictions and the wealth distribution. *Econometrica 91*(3), 869–901.

Fernández-Villaverde, J., G. Nuño, and J. Perla (2024). Taming the curse of dimensionality: quantitative economics with deep learning. Technical report, National Bureau of Economic Research.

Fernández-Villaverde, J., G. Nuno, G. Sorg-Langhans, and M. Vogler (2020). Solving high-dimensional dynamic programming problems using deep learning. *Unpublished working paper*.

Han, J., Y. Yang, et al. (2021). Deepham: A global solution method for heterogeneous agent models with aggregate shocks. *arXiv preprint arXiv:2112.14377*.

Khan, A. and J. K. Thomas (2008). Idiosyncratic shocks and the role of nonconvexities in plant and aggregate investment dynamics. *Econometrica 76*(2), 395–436.

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

Maliar, L., S. Maliar, and P. Winant (2021). Deep learning for solving dynamic economic models. *Journal of Monetary Economics 122*, 76–101.

Reiter, M. (2009). Solving heterogeneous-agent models by projection and perturbation. *Journal of Economic Dynamics and Control 33*(3), 649–665.

Sunakawa, T. (2020). Applying the explicit aggregation algorithm to heterogeneous macro models. *Computational Economics 55*(3), 845–874.

Terry, S. J. (2017). Alternative methods for solving heterogeneous firm models. *Journal of Money, Credit and Banking 49*(6), 1081–1111.

Winberry, T. (2018). A method for solving and estimating heterogeneous agent macro models. *Quantitative Economics 9*(3), 1123–1151.

A Detailed Forecast Accuracy

A.1 Detailed Forecast Accuracy Statistics: Khan-Thomas Model

This section provides detailed internal accuracy statistics for the forecasting rules used in the Khan-Thomas model application, broken down by aggregate productivity state (A_t). Table 8 compares the benchmark KS method with the log-linear regression (My method (reg)) and neural network (My method (NN)) versions of my proposed method.

Table 8: Internal Accuracy of Forecasting Rules (Khan-Thomas Model)

	Price p			Capital K'		
Statistic	KS	My(reg)	My(NN)	KS	My(reg)	My(NN)
Den Haan Statistics (%)						
Maximum	0.11	0.57	0.76	0.38	1.00	2.08
Mean	0.05	0.09	0.12	0.23	0.23	0.34
Root Mean Squared Error (RMSE) (%)						
$A = A_1$	0.06	0.15	0.08	0.06	0.14	0.08
$A = A_2$	0.05	0.11	0.10	0.05	0.11	0.11
$A = A_3$	0.05	0.04	0.16	0.05	0.06	0.09
$A = A_4$	0.04	0.08	0.11	0.05	0.12	0.08
$A = A_5$	0.04	0.03	0.23	0.05	0.04	0.25
Forecast Regression R^2						
$A = A_1$	1.0000	0.9898	0.9973	1.0000	0.9983	0.9992
$A = A_2$	1.0000	0.9943	0.9968	1.0000	0.9990	0.9990
$A = A_3$	1.0000	0.9992	0.9901	1.0000	0.9996	0.9994
$A = A_4$	1.0000	0.9970	0.9953	1.0000	0.9984	0.9994
$A = A_5$	1.0000	0.9996	0.9749	1.0000	0.9999	0.9933

Notes: Compares internal accuracy statistics for different forecasting methods in the Khan-Thomas model. Den Haan statistics and RMSE are reported as percentage points of log deviation (original values multiplied by 100).

A.2 Detailed Forecast Accuracy Statistics: Bloom et al. Model

This section provides detailed internal accuracy statistics for the approximate equilibrium forecast mappings used in the Bloom et al. (2018) model application, broken down by the aggregate state (A, S, S_{-1}), representing discretized grid points for aggregate productivity, current uncertainty, and lagged uncertainty. Table 9 compares the benchmark Krusell-Smith (KS) method with my proposed methods using log-linear regression forecasting (My method (reg)) and neural network forecasting (My method (NN)). RMSE values are multiplied by 100.

Table 9: Internal Accuracy Statistics for Forecast Mappings by State (% RMSE) - Bloom et al. Model

Aggregate State	Capital $\log(K_{t+1})$						Price $\log(p_t)$					
(A, S, S_{-1})	KS		My(reg)		My(NN)		KS		My(reg)		My(NN)	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
(1,0,0)	0.43	0.96	0.29	0.99	0.24	0.99	0.60	0.72	0.76	0.75	0.65	0.83
(1,0,1)	0.32	0.98	0.04	1.00	0.70	0.95	0.35	0.91	0.22	0.96	0.33	0.94
(1,1,0)	0.59	0.93	0.45	0.97	0.66	0.98	0.18	0.96	0.19	0.98	0.31	0.97
(1,1,1)	0.49	0.96	0.07	1.00	0.21	1.00	0.26	0.95	0.21	0.97	0.31	0.97
(2,0,0)	0.40	0.98	0.30	0.99	0.25	0.99	0.61	0.84	0.71	0.75	0.55	0.90
(2,0,1)	0.54	0.97	0.07	1.00	0.80	0.94	0.39	0.95	0.20	0.97	0.51	0.88
(2,1,0)	0.32	0.98	0.22	0.97	0.60	0.96	0.28	0.93	0.44	0.66	0.27	0.97
(2,1,1)	0.47	0.98	0.08	1.00	0.24	1.00	0.29	0.96	0.24	0.97	0.34	0.96
(3,0,0)	0.33	0.98	0.24	0.99	0.23	0.99	0.52	0.87	0.63	0.76	0.46	0.91
(3,0,1)	0.35	0.99	0.06	1.00	0.70	0.95	0.40	0.96	0.17	0.98	0.22	0.98
(3,1,0)	0.57	0.95	0.36	0.98	0.43	0.98	0.27	0.95	0.22	0.98	0.53	0.89
(3,1,1)	0.47	0.98	0.08	1.00	0.21	1.00	0.28	0.97	0.25	0.96	0.36	0.95
(4,0,0)	0.39	0.98	0.29	0.99	0.22	0.99	0.58	0.84	0.74	0.70	0.49	0.86
(4,0,1)	0.43	0.98	0.09	1.00	0.73	0.93	0.41	0.90	0.15	0.98	0.16	0.99
(4,1,0)	0.60	0.94	0.33	0.99	0.36	0.99	0.26	0.95	0.22	0.98	0.86	0.78
(4,1,1)	0.47	0.98	0.10	1.00	0.25	0.99	0.28	0.97	0.28	0.93	0.34	0.95
(5,0,0)	0.43	0.97	0.28	0.99	0.25	0.99	0.63	0.77	0.69	0.70	0.44	0.90
(5,0,1)	0.34	0.98	0.13	1.00	0.67	0.89	0.42	0.86	0.14	0.98	0.15	0.98
(5,1,0)	0.27	0.97	0.09	1.00	0.35	0.99	0.26	0.89	0.19	0.98	0.49	0.95
(5,1,1)	0.50	0.97	0.11	1.00	0.22	0.99	0.27	0.96	0.28	0.93	0.33	0.93

Notes: The table shows accuracy statistics for the approximate equilibrium forecast mappings conditional on the aggregate state (A_t, S_t, S_{t-1}) for the Bloom et al. (2018) model. The functional forms are log-linear for KS and My method (reg), and neural networks for My method (NN). Statistics are computed from an unconditional simulation (5000 quarters for KS/reg, 15000 for NN, after discarding initial burn-in).

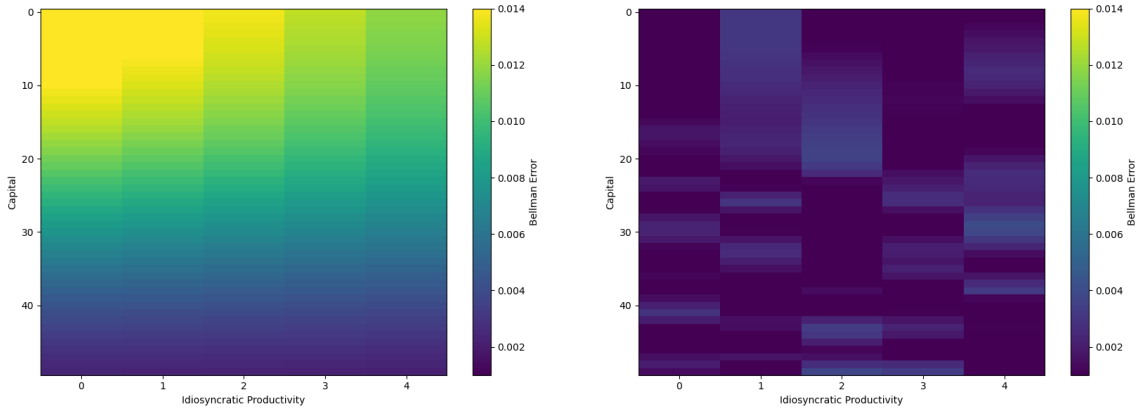
B Difference in Microeconomic Investment-Rate Moments

In the comparison of Microeconomic Investment-Rate Moments for the Khan and Thomas (2008) model (Table 3), the standard deviation of the investment rate produced by my method differed somewhat from the benchmark KS result. It is conjectured that this discrepancy stems from differences in the underlying Bellman error patterns across the state space.

Figure 3 presents heatmaps comparing the Bellman errors ($\log V' - \log V$) for the Terry (2017) KS implementation and My method, calculated while holding the aggregate shock and aggregate capital fixed at the same representative values for both methods. As can be visually inferred from

the figure, the Bellman errors for My method are uniformly small across the displayed (k, ε) state space. In contrast, the errors for the Terry (2017) implementation exhibit a tendency to increase as the firm's capital stock k increases.

Furthermore, an examination of the simulations reveals differences in the utilized portion of the state space. The effective upper and lower bounds of the capital grid k containing a non-zero mass of firms during the simulation were approximately (Upper: 41, Lower: 0) for My method, whereas they were (Upper: 41, Lower: 12) for the KS method. This suggests that the distribution dynamics differ slightly, potentially influenced by the error profiles, which could contribute to the observed variations in moments like the standard deviation of investment rates.



(a) Terry (2017) (b) My method

Figure 3: Bellman error ($\log V' - \log V$) heatmaps from the Terry (2017) code and My method. Aggregate shock and aggregate capital are fixed at the same values in both plots.

C Details on Benchmark Code Modification (Bloom et al. (2018))

As mentioned in Section 3.2, the benchmark results for the Bloom et al. (2018) model presented in this paper utilize a modified version of their original code concerning the state space discretization. This modification was necessary to ensure a meaningful comparison with my proposed method, addressing differences in how adjustment costs were effectively triggered. The original code provided by Bloom et al. (2018) used a specific grid structure for capital (k) and lagged labor (n_{-1}) where the spacing was related to depreciation amounts, and interpolation was not employed. A consequence of this design was that moving to the adjacent lower grid point effectively represented non-adjustment relative to depreciation and could bypass the fixed component of the adjustment cost (F^K or F^L). This led to minimal observed active disinvestment in their original simulations.

In contrast, my proposed method outputs continuous actions. This inherent difference in handling adjustments near the depreciation point significantly altered aggregate dynamics, making a direct comparison difficult. To facilitate a more meaningful comparison, I modified the grid structure in the benchmark code implementation used in this paper. Specifically, the grids for capital (k) and lagged labor (n_{-1}) were **set to be equally spaced on a logarithmic scale**. Under this log-equispaced grid, the adjacent lower grid point k_{n-1} no longer corresponds precisely to the

depreciated capital stock $(1 - \delta_k)k_n$. Consequently, any active choice to change the capital stock, including moving to k_{n-1}, now necessarily implies non-zero net investment ($i \neq 0$). This ensures that the fixed adjustment cost F^K (and similarly F^L for labor based on $s \neq 0$) is triggered whenever the firm actively adjusts its input, aligning the benchmark's cost mechanism more closely with the implications of the continuous action space in my method.

This revised implementation, using logarithmically spaced grids, leads to a more consistent application of fixed adjustment costs and allows for a fairer comparison of the model dynamics generated by the benchmark and my proposed approach. The benchmark results reported in Section 3.2 are based on this modified implementation.

D Forecasting Accuracy vs. Simulation Length (Bloom et al. Model)

This appendix examines the impact of the simulation length used for generating training data on the internal accuracy of the forecasting rules in the Bloom et al. (2018) model application. We compare results obtained using shorter versus longer simulations for both the log-linear (My method (reg)) and neural network (My method (NN)) forecasting rules. All accuracy statistics (RMSE, R^2) are calculated for the logarithms of the aggregate price ($\log(p_t)$) and next-period aggregate capital ($\log(K_{t+1})$). Reported Root Mean Squared Error (RMSE) values are multiplied by 100, representing percentage point deviations.

D.1 Log-Linear Forecasting Rules (My method (reg))

This subsection compares the accuracy of the log-linear forecasting rules when trained on data from 5,000 versus 10,000 simulation periods. Table 10 shows the overall accuracy metrics, while Table 11 provides a detailed breakdown by aggregate state (A, S, S_{-1}).

Table 10: Overall Forecasting Accuracy vs. Simulation Length: My method (reg)

Simulation	Price $\log(p_t)$		Capital $\log(K_{t+1})$	
Length (Periods)	RMSE (%)	R^2	RMSE (%)	R^2
5,000	0.55	0.9682	0.22	0.9958
10,000	0.54	0.9742	0.21	0.9970

Notes: Compares overall internal accuracy statistics for log-linear forecasting rules trained on simulations of different lengths. RMSE is multiplied by 100.

Increasing the simulation length from 5,000 to 10,000 periods for training the log-linear forecasting rules results in modest improvements in the overall RMSE and R^2. However, examining the state-by-state results (Table 11), the R^2 for the price forecast remains significantly below 1.00 in several states. This suggests potential non-linearities in the true law of motion for the price that the log-linear specification struggles to capture fully, even with more simulation data. This motivates the use of neural networks for the forecasting rule.

Table 11: State-by-State Forecasting Accuracy: My method (reg) (5k vs 10k periods)

Aggregate State	Capital $\log(K_{t+1})$				Price $\log(p_t)$			
(A, S, S_{-1})	5k Periods		10k Periods		5k Periods		10k Periods	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
(1,0,0)	0.29	0.9894	0.24	0.9936	0.76	0.7503	0.45	0.7700
(1,0,1)	0.04	0.9997	0.11	0.9973	0.22	0.9619	0.22	0.9200
(1,1,0)	0.45	0.9652	0.16	0.9972	0.19	0.9771	0.35	0.9329
(1,1,1)	0.07	0.9994	0.11	0.9986	0.21	0.9664	0.31	0.9247
(2,0,0)	0.30	0.9878	0.24	0.9942	0.71	0.7457	0.53	0.8023
(2,0,1)	0.07	0.9993	0.11	0.9983	0.20	0.9738	0.20	0.9529
(2,1,0)	0.22	0.9701	0.26	0.9953	0.44	0.6554	0.33	0.9607
(2,1,1)	0.08	0.9994	0.10	0.9989	0.24	0.9690	0.30	0.9396
(3,0,0)	0.24	0.9918	0.25	0.9928	0.63	0.7566	0.65	0.7616
(3,0,1)	0.06	0.9996	0.11	0.9986	0.17	0.9833	0.23	0.9500
(3,1,0)	0.36	0.9846	0.19	0.9948	0.22	0.9758	0.35	0.9193
(3,1,1)	0.08	0.9992	0.12	0.9986	0.25	0.9643	0.32	0.9338
(4,0,0)	0.29	0.9897	0.24	0.9931	0.74	0.7047	0.71	0.7371
(4,0,1)	0.09	0.9986	0.09	0.9991	0.15	0.9813	0.23	0.9619
(4,1,0)	0.33	0.9879	0.26	0.9904	0.22	0.9765	0.35	0.9227
(4,1,1)	0.10	0.9982	0.12	0.9983	0.28	0.9289	0.32	0.9279
(5,0,0)	0.28	0.9906	0.24	0.9917	0.69	0.7033	0.61	0.7385
(5,0,1)	0.13	0.9967	0.07	0.9964	0.14	0.9810	0.28	0.8039
(5,1,0)	0.09	0.9989	0.24	0.9934	0.19	0.9812	0.32	0.9460
(5,1,1)	0.11	0.9978	0.10	0.9983	0.28	0.9323	0.31	0.9052

Notes: Compares internal accuracy statistics for log-linear forecasting rules (My method (reg)) trained on 5,000 vs. 10,000 simulation periods. Statistics are conditional on the aggregate state (A_t, S_t, S_{t-1}). RMSE is multiplied by 100.

D.2 Neural Network Forecasting Rules (My method (NN))

This subsection compares the accuracy of the neural network forecasting rules when trained on data from 15,000 versus 25,000 simulation periods. Table 12 shows the overall accuracy metrics, while Table 13 provides the detailed breakdown by aggregate state.

Table 12: Overall Forecasting Accuracy vs. Simulation Length: My method (NN)

Simulation	Price $\log(p_t)$		Capital $\log(K_{t+1})$	
Length (Periods)	RMSE (%)	R^2	RMSE (%)	R^2
15,000	0.46	0.9799	0.26	0.9960
25,000	0.45	0.9806	0.29	0.9964

Notes: Compares overall internal accuracy statistics for neural network forecasting rules trained on simulations of different lengths. RMSE is multiplied by 100.

Table 13: State-by-State Forecasting Accuracy: My method (NN) (15k vs 25k periods)

Aggregate State	Capital $\log(K_{t+1})$				Price $\log(p_t)$			
(A, S, S_{-1})	15k Periods		25k Periods		15k Periods		25k Periods	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
(1,0,0)	0.24	0.9947	0.25	0.9947	0.65	0.8332	0.53	0.8862
(1,0,1)	0.70	0.9491	0.55	0.9682	0.33	0.9387	0.13	0.9934
(1,1,0)	0.66	0.9758	0.76	0.9571	0.31	0.9722	0.53	0.9035
(1,1,1)	0.21	0.9971	0.33	0.9917	0.31	0.9687	0.35	0.9571
(2,0,0)	0.25	0.9940	0.26	0.9950	0.55	0.8964	0.56	0.8959
(2,0,1)	0.80	0.9399	0.55	0.9689	0.51	0.8839	0.13	0.9935
(2,1,0)	0.60	0.9619	0.71	0.9680	0.27	0.9654	0.50	0.9359
(2,1,1)	0.24	0.9952	0.32	0.9927	0.34	0.9557	0.31	0.9712
(3,0,0)	0.23	0.9930	0.26	0.9949	0.46	0.9075	0.47	0.9180
(3,0,1)	0.70	0.9465	0.52	0.9746	0.22	0.9809	0.12	0.9950
(3,1,0)	0.43	0.9788	0.65	0.9658	0.53	0.8854	0.64	0.8776
(3,1,1)	0.21	1.0000	0.31	0.9935	0.36	0.9499	0.29	0.9741
(4,0,0)	0.22	0.9926	0.28	0.9941	0.49	0.8581	0.46	0.9262
(4,0,1)	0.73	0.9329	0.54	0.9747	0.16	0.9894	0.21	0.9848
(4,1,0)	0.36	0.9868	0.65	0.9658	0.86	0.7779	0.61	0.8794
(4,1,1)	0.25	0.9928	0.27	0.9935	0.34	0.9493	0.28	0.9717
(5,0,0)	0.25	0.9934	0.30	0.9892	0.44	0.9046	0.48	0.8930
(5,0,1)	0.67	0.8869	0.52	0.9626	0.15	0.9810	0.25	0.9677
(5,1,0)	0.35	0.9906	0.67	0.9567	0.49	0.9482	0.25	0.9747
(5,1,1)	0.22	0.9917	0.20	0.9953	0.33	0.9259	0.34	0.9455

Notes: Compares internal accuracy statistics for neural network forecasting rules (My method (NN)) trained on 15,000 vs. 25,000 simulation periods. Statistics are conditional on the aggregate state (A_t, S_t, S_{t-1}). RMSE is multiplied by 100.

Comparing the results for the neural network forecasting rules trained on 15,000 versus 25,000 periods (Table 12 and Table 13), we see very little difference in the overall accuracy metrics and only minor, non-systematic changes in the state-by-state performance. This suggests that 15,000 periods are likely sufficient for the neural network to learn the relevant dynamics for forecasting in this model. Interestingly, even with the neural network's flexibility and longer training data,

the R^2 does not consistently reach 1.00 across all states, particularly for the price forecast in certain low-productivity or high-uncertainty regimes. This may indicate inherent limits to the predictability of the equilibrium price in these specific states, possibly due to the complex interactions and non-linearities present in the Bloom et al. (2018) model, rather than limitations of the forecasting function itself.

E Deep Learning Hyperparameters

This section details the hyperparameters used for training the neural network approximations of the value function (V^0_{nn}) and policy function (g_{nn}) in the two main model applications presented in the paper. The specific settings for the Khan and Thomas (2008) and Bloom et al. (2018) models are summarized in Table 14.

Table 14: Neural Network Hyperparameters by Model Application

Hyperparameter	Khan-Thomas (2008)	Bloom et al. (2018)
Network Architecture (Value/Policy)	2 hidden layers, 64 neurons each	2 hidden layers, 128 neurons each
Optimizer	ADAM	ADAM
Learning Rate Schedule	Decay $1 \times 10^{-3} \rightarrow 1 \times 10^{-5}$	Decay $1 \times 10^{-3} \rightarrow 1 \times 10^{-5}$
Batch Size (Value Function)	256	256
Batch Size (Policy Function)	256	256
Outer Loop Iterations (Max)	6	16
Target Network Update Rate (τ)	0.001	0.001

Notes: The learning rate was gradually decayed over the training steps within each outer loop iteration, starting from 1×10^{-3} down to 1×10^{-5}. The target network update rate τ corresponds to the parameter in the soft update rule $\theta_{target} \leftarrow \tau\theta_{main} + (1 - \tau)\theta_{target}$.