Deep learning solution method for heterogeneous firm models

Yoshiya Yokomoto, Keio University

April 24, 2025

Abstract

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 our 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 byBloom et al. (2018), where firms are subjected to stochastic volatility shocks alongside two endogenous state variables, resulting in over three million state-space combinations. Our 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. In the next section, I will survey the related literature. Section 2 introduces our methodology in the context of the Khan and Thomas (2008) model, contrasting our neural network approach with the traditional Krusell-Smith solution technique, and presents comprehensive results comparing the performance, accuracy, and computational efficiency of our 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. Section 4 concludes with a discussion of the broader implications of our approach for macroeconomic modeling and potential future research directions.

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 computational 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

In this section, I will explain my method based on Khan and Thomas (2008). In their setup, firms differ in terms of their idiosyncratic productivity levels and their capital stocks, while being subject to both idiosyncratic and aggregate shocks.

2.1 Model

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:

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

while

$$R\left(\varepsilon_e, k'; z_i, \mu'\right) \equiv -\gamma k' + \sum_{j=1}^{N_z} \pi_{ij} d_j\left(z_i; \mu\right) \sum_{m=1}^{N_\varepsilon} \pi_{em}^{\varepsilon} v^0\left(\varepsilon_{\mathrm{m}}, k'; z_j, \mu'\right) \tag{8}$$

$$v^0(\varepsilon, k; z, \mu) \equiv \int_0^{\bar{\xi}} v^1(\varepsilon, k, \xi; z, \mu) G(d\xi) \tag{9}$$

Forecasting rule, (10)

$$\mu' = \Gamma_{\mu}(z, \mu) \tag{11}$$

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:

$$\begin{aligned} \omega(z, \mu) &= \frac{D_2 U(C, 1 - N)}{D_1 U(C, 1 - N)} \\ d_j(z, \mu) &= \frac{\beta D_1 U\left(C_j', 1 - N_j'\right)}{D_1 U(C, 1 - N)} \end{aligned} \tag{12}$$

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

$$p(z, \mu) = D_1 U(C, 1 - N) \tag{13}$$

$$\omega(z, \mu) = \frac{D_2 U(C, 1 - N)}{p(z, \mu)} \tag{14}$$

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:

$$V^1(\varepsilon, k, \xi; z, \mu) = \max_{n \in \mathbf{R}_+} (z\varepsilon k^\alpha n^\nu - \omega n + (1-\delta)k)p + \max\left\{ -\xi\omega p + \max_{k^* \in \mathbf{R}_+} R\left(\varepsilon, k^*; z, \mu'\right), R\left(\varepsilon, (1-\delta)k; z, \mu'\right) \right\} \tag{15}$$

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

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

Forecasting rule,

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

Note that the second maximization in (15) 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:

$$\xi^* = \frac{R(\varepsilon, k^*; z, \mu') - R(\varepsilon, (1-\delta)k; z, \mu')}{p\omega} \tag{16}$$

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

$$\begin{aligned} V^0(\varepsilon, k; z, \mu) = \max_{n \in \mathbf{R}_+} \Big[& \left(z\,\varepsilon\,k^\alpha\,n^\nu \;-\; \omega\,n \;+\; (1-\delta)\,k\right) p \\ & + \;\alpha(z, \varepsilon, \mu)\Big(-\omega\,p \int_0^{\bar{\xi}} \xi\, G(\mathrm{d}\xi) \;+\; \max_{k^* \in \mathbf{R}_+} R\big(\varepsilon, k^*; z, \mu'\big)\Big) \\ & + \;\big(1 - \alpha(z, \varepsilon, \mu)\big)\; R\big(\varepsilon, (1-\delta)\,k; z, \mu'\big)\Big]. \end{aligned} \tag{17}$$

Here, $\alpha(z, \varepsilon, \mu)$ denotes the probability of adjustment, which is given by $\alpha(z, \varepsilon, \mu) = G(\xi^*) = \xi^*/\bar{\xi}$.

2.2 Krusell-Smith (KS) Method

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 (17) and forecasting rules is replaced by K, resulting in the following equation:

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

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

$$V^0(\varepsilon,k;z,K) = \max_{n\in\mathbf{R}_+}\Big[\big(z\,\varepsilon\,k^{\alpha}\,n^{\nu} \;-\; \omega\,n \;+\; (1-\delta)\,k\big)\,p$$
$$+\;\alpha(z,\varepsilon,K)\Big(-\omega\,p\int_0^{\bar{\xi}}\xi\,G(\mathrm{d}\xi) \;+\; \max_{k^*\in\mathbf{R}_+} R\big(\varepsilon,k^*;z,K'\big)\Big) \tag{18}$$
$$+\;\big(1-\alpha(z,\varepsilon,K)\big)\;R\big(\varepsilon,(1-\delta)\,k;z,K'\big)\Big]$$

$$R\left(\varepsilon,k';z,K'\right) \equiv -\gamma k' p + \beta\sum_{j=1}^{N_z}\pi_{ij}\sum_{m=1}^{N_\varepsilon} V^0\left(\varepsilon_m,k';z_j,K'\right) \tag{19}$$

$$\log(K') = a_K + b_K\log(K) \tag{20}$$
$$\log(p) = a_p + b_p\log(K) \tag{21}$$

Forecasting rules are assumed to take the log-linear form. Equation (18) 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 in equation (19), given the values from the forecasting rules (20) and (21), using a numerical optimization algorithm.

Algorithm 1: Krusell-Smith Method Procedure

```
Initialize: Set initial forecasting rule parameters and define the state grid;
repeat
    Step 1: Solve the Bellman Equation;
    Given forecasting rules, do value function iteration;
    Step 2: Simulate the Model;
    Run 2500 periods simulations using the obtained value function;
    Step 3: Update Forecasting Rules;
    Update parameters by regressing simulation outcomes;
    Step 4: Check Convergence;
    If‖ log V' − log V‖ < ϵ, then terminate; otherwise, set n ← n + 1;
until convergence condition ‖ log V' − log V‖ < ϵ is satisfied;
Output: Value, policy function and converged forecasting rule parameters;
```

Algorithm 1 shows the solution procedure. In the KS algorithm, we first determine the range and number of grid points for each state variable. Terry (2017) use 5, 5, 10, and 10 grid points for ε, z, k, and K, respectively. Interpolation is performed for k and K. Then, in Step 1, solve the bellman equation. In the typical Value Function Iteration(VFI), the right-hand side of equation (18) is calculated for all combinations of ε, z, k, and K. This value is then used as the new V, and the right-hand side of equation (18) is calculated again. This process continues iteratively until convergence, at which point Step 1 is complete. During this process, K' and p in the equation are determined according to equations (20) and (21).

In Step 2, we perform a simulation using the value function obtained in Step 1. Since the simulation is a crucial aspect of my proposed solution method, let's examine its contents in detail. Algorithm 2 presents a simplified algorithm for the simulation.

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
```

While the VFI for the Khan and Thomas (2008) model takes only about 2 seconds, the subsequent simulation requires approximately 3 minutes per iteration, consuming the vast majority of the computational time. This is due to p being implicitly determined, depending on the distribution itself. While Krusell and Smith (1998) could explicitly derive the interest rate from K using the representative firm's first-order condition, in Khan and Thomas (2008), the price depends on the entire distribution. Although the distribution is approximated by a histogram in practice, we calculate optimal actions at each point in this histogram and compute the following value:

$$C\left(z,\mu\right)=\int\left[z_{i}\varepsilon k^{\alpha}n^{\nu}-\int_{0}^{\bar{\xi}}\left(k'\left(\varepsilon,k,\xi;z,\mu\right)-(1-\delta)k\right)G(d\xi)\right]\mu(d\varepsilon\times dk) \tag{22}$$

This equation is an equilibrium condition stating that output minus net of investment, is consumed in aggregate. From equation (13) and the household's optimization condition, we have:

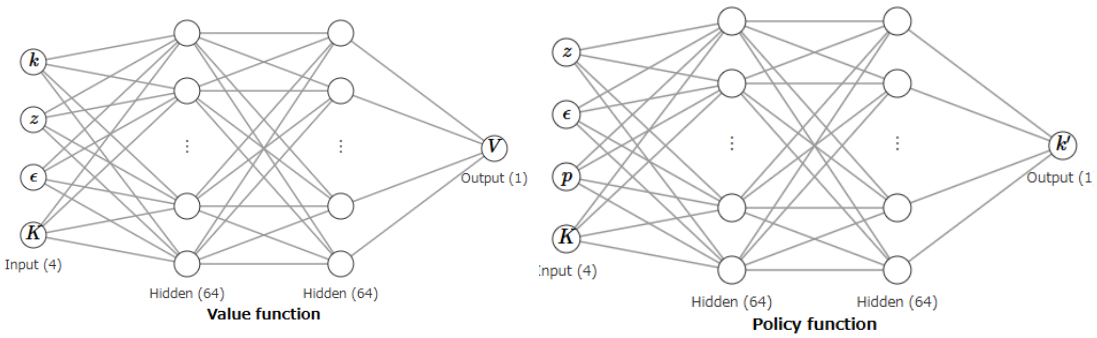
$$p(z,\mu)=\frac{1}{C(z,\mu)} \tag{23}$$

In the simulation, equilibrium condition requires that equation (23) holds in each period. In fact, computing equilibrium price $p(z,\mu)$ satisfying (23) is a fixed point problem, so we need to compute it by doing the bisection method. Equation (22) includes k', which is either k^* or $(1-\delta)k$. k^* is chosen to maximize (19). Since (19) includes p, it requires recomputing k^* for each new guess of p in the bisection. Why is recomputing $k*$ needed even though we have solved it in VFI? The answer is here. The choice of k^* computed during the VFI step is optimal under the perceived law of motion given by the forecasting rules (20) and (21). However, during the simulation's price bisection, we search for the *actual* market-clearing price that satisfies (23). The prices guessed within this bisection search, p_{guess}, may differ from the price implied by the forecasting rule (21) for the current aggregate state K. Thus we need to have re-optimization for every price determined by bisection. This re-optimization step corresponds to the computationally intensive innermost loop depicted in Algorithm 2. For each guess of p, the value of $R\left(\varepsilon,k^*;z,K'\right)$ is calculated for *all* grid points of k, given ε, z and K', and the k that maximizes it is chosen. This process is performed for every point in the histogram each time p is updated. In the code of Terry (2017), this calculation is done for 5×10 points in the (ε,k) histogram. As can be easily seen, the computational bur-

den increases as the number of state variables and their corresponding grid points that constitute the distribution increase. As we will see in the next chapter, the histogram in Bloom et al. (2018) consists of 16,835 points.

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 price as a state variable. In my method, both the policy function and the value function are approximated using neural networks and policy function include price as a state variable. 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.



(a) Value Function Approximation

(b) Policy Function Approximation

Figure 1: Neural Network Approximations

Specifically, the value and policy functions in equation (17) are approximated with neural networks as follows:

$$V^0(\varepsilon, k; z, \mu) \approx V^0_{nn}(\varepsilon, k; z, K), \tag{24}$$

$$k^* \approx g_{nn}(z, \varepsilon, p, K). \tag{25}$$

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 directly as input and outputs the value. The policy function's inputs are z, ε, p, and K. As mentioned in Section 2.1, the choice of k^* is a function of z, ε, and K. Therefore, I add the price as a state variable. By including the price, I avoid the need to re-optimize k^* for each price update during the bisection search in the simulation.

Forecasting rules are similarly approximated:

$$\Gamma_\mu(z, \mu) \approx \Gamma^{nn}_\mu(z, K), \tag{26}$$

$$\Gamma_p(z, \mu) \approx \Gamma^{nn}_p(z, K). \tag{27}$$

Since the policy function is a function that outputs the k that maximizes (19), it is trained by minimizing the following loss function:

Policy Function Loss:

$$\mathcal{L}_{policy} = -\frac{1}{N}\sum_{i=1}^{N} R\Big(\varepsilon_i, k_i'; z_i, K_i\Big). \tag{28}$$

Where,

$$k_i' = g_{nn}(z_i, \varepsilon_i, p_{error,i}, K_i) \tag{29}$$

$$\begin{aligned} R\big(\varepsilon_i, k_i'; z_i, K_i\big) = &-\gamma\, k_i' p_{error,i} \\ &+ \beta \sum_{j=1}^{N_z} \pi_{ij} \sum_{m=1}^{N_\varepsilon} \pi_{im} V_{nn}^0\big(\varepsilon_m, k_i'; z_j, K_i'\big). \end{aligned} \tag{30}$$

$$p_{error,i} = \Gamma_p^{nn}(z_i, K_i) + error_i, \quad error_i \in [-0.15, 0.15] \tag{31}$$

$p_{error,i}$ is a perturbed price, calculated as the price predicted by the forecasting rule $\Gamma_p^{nn}(z_i, K_i)$ plus a noise term $error_i$ drawn uniformly from the interval $[-0.15, 0.15]$. As explained previously, 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 policy network parameters are then trained by minimizing the loss function (29) (which is equivalent to maximizing the expected continuation value R) using gradient-based optimization. Specifically, the parameters are optimized using the ADAM optimizer.

Value Function Loss:

$$\mathcal{L}_V = \frac{1}{N}\sum_{i=1}^{N}\Big(V_{nn}^0(\varepsilon_i, k_i; z_i, K_i) - \mathrm{RHS}_i\Big)^2. \tag{32}$$

Where,

$$\begin{aligned} \mathrm{RHS}_i = &\left(z_i \varepsilon_i k_i^{\alpha} n_i^{\nu} - \omega_i n_i + (1-\delta) k_i\right) p_i \\ &+ \alpha(z_i, \varepsilon_i, K_i)\left[-\omega_i p_i \int_0^{\bar{\xi}} \xi_i G(d\xi_i) + R\left(\varepsilon_i, k_i^*; z_i, K_i'\right)\right] \\ &+ \left(1 - \alpha(z_i, \varepsilon_i, K_i)\right) R\left(\varepsilon_i, (1-\delta)k_i; z_i, K_i'\right). \end{aligned} \tag{33}$$

$$R(\varepsilon_i, k_i^*; z_i, K_i') = -\gamma k_i^* p + \beta \sum_{j=1}^{N_z} \pi_{ij} \sum_{m=1}^{N_\varepsilon} V_{nn}^0\left(\varepsilon_m, k_i^*; z_j, K_i'\right) \tag{34}$$

$$k^* = g_{nn}(z_i, \varepsilon_i, p_i, K_i) \tag{35}$$

The loss function for the value function is the squared difference between the left-hand side and the right-hand side of the Bellman equation. In standard VFI, the right-hand side is directly

used as the updated value function. However, in the neural network-based solution method, I train the network to minimize the difference between both sides. To stabilize the training of the value function, a target network is used. Details are provided in the Appendix.

Algorithm 3: Overall Solution Algorithm

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**
        $rhs_{prev} \leftarrow rhs$.
        $rhs \leftarrow \text{TrainPolicyNetwork}(g_{nn}, V_{nn}^0, \mathcal{D})$
        $loss_V \leftarrow \text{TrainValueNetwork}(V_{nn}^0, g_{nn}, \mathcal{D})$
    end
    2. Simulation and Forecast Update:
    $\mathcal{D}_{new} \leftarrow \text{SimulateModel}(g_{nn}, \Gamma_p^{nn}, \Gamma_\mu^{nn})$
    $\Gamma_p^{nn,old} \leftarrow \Gamma_p^{nn}$
    $\Gamma_\mu^{nn,old} \leftarrow \Gamma_\mu^{nn}$
    $\Gamma_p^{nn} \leftarrow \text{UpdatePriceForecast}(\Gamma_p^{nn}, \mathcal{D}_{new})$
    $\Gamma_\mu^{nn} \leftarrow \text{UpdateCapitalForecast}(\Gamma_\mu^{nn}, \mathcal{D}_{new})$
    $\mathcal{D} \leftarrow \text{UpdateTrainingData}(\mathcal{D}, \mathcal{D}_{new})$
end
return $V_{nn}^0, g_{nn}, \Gamma_p^{nn}, \Gamma_\mu^{nn}$

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.

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

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 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.

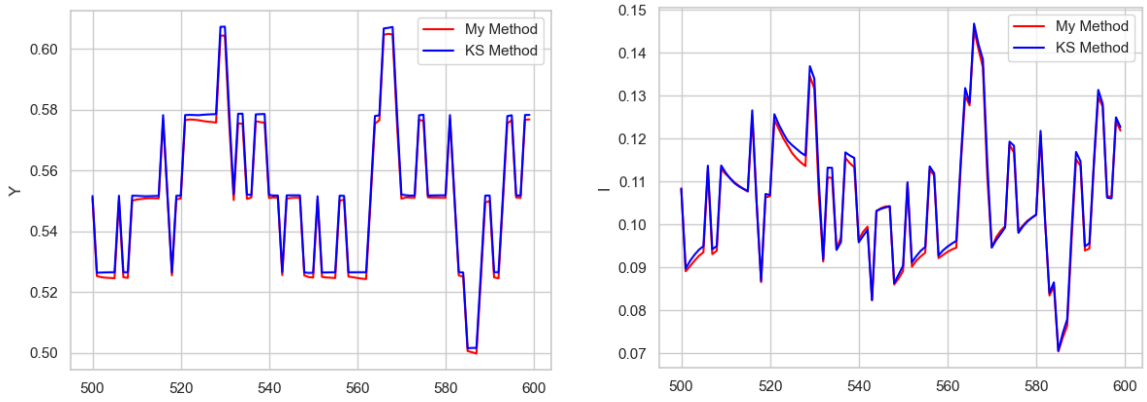
Method	**Average Bellman Error** ($\log V' - \log V$)
KS method (Terry (2017))	0.0085
My method (NN)	0.0015

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.

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.[3]

[3]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.



(a) Aggregate output (b) Aggregate investment

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

	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.
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. 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, it must be emphasized that for heterogeneous firm models like those of Khan and Thomas (2008) and Bloom et al. (2018), regardless of the solution method, the accuracy of the forecasting rule is critical; simulations can collapse if the forecasts are not sufficiently pre-

	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).

cise. Furthermore, even when using neural networks for forecasting, coefficients from an initial 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), which 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 j produces output at time t according to a Cobb-Douglas production function:

$$y_{j,t} = A_t z_{j,t} k_{j,t}^{\alpha} n_{j,t}^{\nu}, \tag{36}$$

where A_t is an aggregate productivity process, $z_{j,t}$ is idiosyncratic productivity, and $k_{j,t}$ and $n_{j,t}$ 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:**

$$\log(A_t) = \rho^A \log(A_{t-1}) + \sigma_{t-1}^A \varepsilon_t, \quad \varepsilon_t \sim N(0,1) \tag{37}$$

- **Idiosyncratic Productivity:**

$$\log(z_{j,t}) = \rho^Z \log(z_{j,t-1}) + \sigma^Z_{t-1}\varepsilon_{j,t}, \quad \varepsilon_{j,t} \sim N(0,1) \tag{38}$$

The time-varying volatilities, σ^A_{t-1} and σ^Z_{t-1}, capture switches between "low-uncertainty" and "high-uncertainty" regimes, typically modeled as states governed by a two-state Markov chain. These aggregate uncertainty states (σ^A, σ^Z) become part of the aggregate state space.

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

$$k_{j,t+1} = (1-\delta_k)\, k_{j,t} + i_{j,t}, \tag{39}$$

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

$$\text{AC}^k = \mathbb{I}(|i_{j,t}| > 0)\, y_{j,t}\, F^K + S|i_{j,t}|\, \mathbb{I}(i_{j,t} < 0) \tag{40}$$

Here, $\mathbb{I}(\cdot)$ 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:

$$n_{j,t} = (1-\delta_n)\, n_{j,t-1} + s_{j,t}, \tag{41}$$

where δ_n is an exogenous separation rate and $s_{j,t}$ 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:

$$\text{AC}^n = \mathbb{I}(|s_{j,t}| > 0)\, y_{j,t}\, F^L + |s_{j,t}|\, H\, w_t \tag{42}$$

where F^L is the fixed disruption cost for labor adjustment, H parameterizes the variable adjustment cost, and w_t is the aggregate wage rate. Crucially, because labor adjustment depends on the previous period's labor input, $n_{j,t-1}$ becomes an endogenous state variable for the firm.

Firm Problem and State Space. The Bellman equation, expressed in terms of marginal utility units $\tilde{V}$, depends on the firm's individual state (k, n_{-1}, z) and the aggregate state $(A, \sigma^A, \sigma^Z, \mu)$, where μ represents the distribution of firms over their idiosyncratic states:

$$\tilde{V}(k, n_{-1}, z; A, \sigma^A, \sigma^z, \mu) = \max_{i,n} \Bigg\{ p(A, \sigma^A, \sigma^z, \mu) \Big[y - w(A, \sigma^A, \sigma^z, \mu) n - i - \text{AC}^k - \text{AC}^n \Big] + \beta \mathbb{E}\Big[\tilde{V}(k', n, z'; A', \sigma^{A'}, \sigma^{z'}, \mu') \Big] \Bigg\}. \tag{43}$$

Solving the model requires tracking the evolution of the distribution μ and the equilibrium price p via forecasting rules, typically approximated as functions of the aggregate state variables and moments of the distribution μ:

$$\mu' = \Gamma_\mu(A, \sigma^A, \sigma^z, \mu), \quad p = \Gamma_p(A, \sigma^A, \sigma^z, \mu). \tag{44}$$

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 $(k, n_{-1}, z, A, \sigma^A, \sigma^Z, \mu)$, is shown below, resulting in a total grid size of:

$$\underbrace{91}_{\substack{\text{grid points} \\ \text{for } k}} \times \underbrace{37}_{\substack{\text{grid points} \\ \text{for } n_{-1}}} \times \underbrace{5}_{\substack{\text{grid points} \\ \text{for } z}} \times \underbrace{5}_{\substack{\text{grid points} \\ \text{for } A}} \times \underbrace{2}_{\substack{\text{states} \\ \text{for } \sigma^A}} \times \underbrace{2}_{\substack{\text{states} \\ \text{for } \sigma^Z}} \times \underbrace{10}_{\substack{\text{grid points} \\ \text{for } \mu}} = 3{,}367{,}000 \quad \text{points.}$$

Here, σ^A and σ^Z represent the two possible volatility regimes (e.g., high/low), and the 10 points for μ likely refer to a discretization of aggregate moments (like capital) used to approximate the distribution 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).[4]

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.

[4]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 (23). 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.

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).

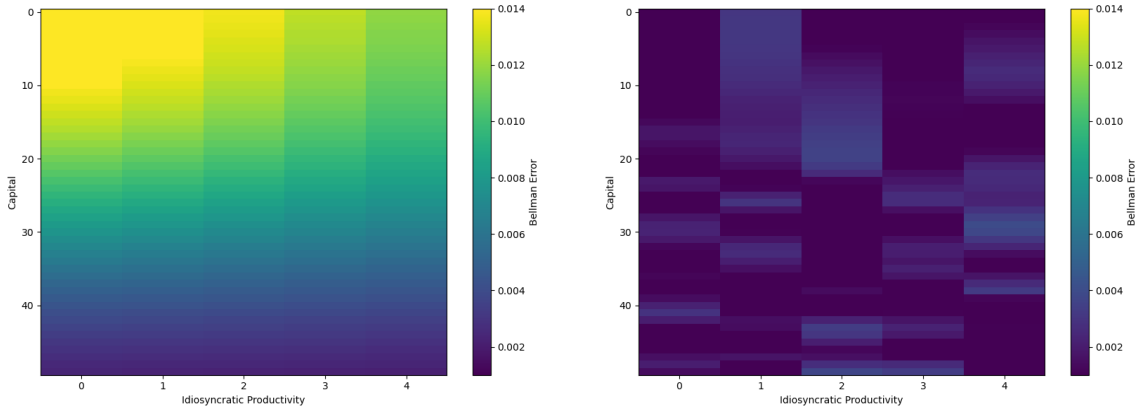
B Difference in Microeconomic Investment-Rate Moments

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. Model)

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 $\theta_{target} \leftarrow \tau\theta_{main} + (1 - \tau)\theta_{target}$.