The Blessings of Overparameterization: Applications in Solving Economic Models*

Mahdi Ebrahimi Kahou[†] Jesús Fernández-Villaverde[‡]

Abstract

We investigate the effects of overparameterization when using neural networks to solve dynamic models in economics. We document two findings. First, overparameterization does not lead to overfitting: as the number of parameters grows, test loss on Bellman and Euler residuals decreases and the resulting approximate policy and value functions converge toward their benchmark counterparts. Second, and more importantly, overparameterization dramatically improves algorithmic stability: with small networks, solutions across different random initializations of the network parameters (weights and biases) can disagree substantially, but as network width increases, this disagreement collapses and solutions concentrate tightly around the truth. We demonstrate these properties across three canonical economic models with known benchmark solutions: the Linear-Quadratic Regulator, the McCall job-search model, and the Real Business Cycle model. Our results suggest that economists solving economic models with neural networks need not be concerned about large networks, as overparameterization is a blessing rather than a curse.

1 Introduction

Deep neural networks have become the dominant tool for function approximation in modern machine learning. A deep neural network is a composition of nonlinear transformations with a large number of trainable parameters, often far exceeding the number of training points (data). This overparameterization is the defining feature of deep learning: it is what separates it from classical methods, and it drives its empirical success (He et al., 2016; Telgarsky, 2016).

*We thank Marlon Azinovic-Yang, Samira Ebrahimi Kahou, Farid Farrokhi, Denizalp Goktas, Amy Greenwald, William Jungerman, Vadim Marmer, Christian Matthes, Jan Rosa, and Hikaru Saijo for helpful comments and discussions.

†Bowdoin College. Email: `m.ebrahimikahou@bowdoin.edu`.

‡University of Pennsylvania. Email: `jesusfv@econ.upenn.edu`.

A growing literature in economics has adopted deep learning to solve dynamic programming problems, where the curse of dimensionality renders traditional grid-based methods infeasible (Maliar et al., 2021; Azinovic et al., 2022; Kahou et al., 2021). Applications now span heterogeneous-agent models with aggregate shocks (Han et al., 2022; Kase et al., 2022; Phan, 2025), life-cycle consumption-saving models with many durable goods and constraints (Druedahl and Røpke, 2026), financial frictions and the wealth distribution (Fernández-Villaverde et al., 2023), asset pricing with participation constraints (Gopalakrishna et al., 2024), continuous-time heterogeneous-agent macro via master equations (Gu et al., 2024), climate change and uncertainty (Barnett et al., 2023), search-and-matching labor markets (Payne et al., 2024), human capital accumulation and labor market learning (Jungerman, 2023; Adenbaum et al., 2024), overlapping-generations models with rare disasters (Azinovic and Žemlička, 2024), and sequence-space methods (Azinovic-Yang and Žemlička, 2025).

Yet the role of overparameterization in these economic applications is not well understood. Standard statistical wisdom warns that models with more parameters than data points overfit, producing approximations that underperform in unseen data. If this concern applies when neural networks solve Euler and Bellman equations, practitioners would need to carefully tune network size, adding significant complexity to an already demanding task. Moreover, even if overfitting is not a problem in principle, it is unclear whether overparameterization affects solution reliability. Neural network training minimizes a nonconvex objective: the composition of nonlinear transformations generates a loss landscape with many local minima, and gradient descent offers no guarantee of reaching a global optimum. Does a wider network consistently find a better answer, or does the nonconvex loss landscape mean solutions depend heavily on random initialization?

This paper provides a systematic answer. We document two blessings of overparameterization in the neural network solution of dynamic economic models. First, there is no overfitting: as the number of network parameters grows, out-of-sample Bellman and Euler residuals *decrease* with network width, and the resulting approximate policy and value functions converge toward their benchmark counterparts. Second, and more importantly, overparameterization dramatically improves *algorithmic stability*. With small, underparameterized networks, solutions obtained from different random initializations of the network weights can disagree substantially, a serious problem, since there is no way of knowing which initialization produced the better solution. As network width increases, this disagreement collapses and solutions concentrate tightly around the truth, making the algorithm reliable and initialization-independent.

These findings connect to a broader phenomenon recently documented in the machine learning literature. Belkin et al. (2019) showed that the classical bias-variance tradeoff breaks down for modern overparameterized models, and that test error can decrease again after an initial overfitting peak, a pattern they call *double descent*. Bartlett et al. (2020) formalized this for linear regression, showing that minimum-norm interpolants can generalize well even when the number of parameters vastly exceeds the number of observations. Our paper documents that this benign behavior of overparameterization extends to the function approximation problems that arise in computational economics, where the loss function is defined by equilibrium conditions (Euler equations, Bellman equations) rather than by labeled data.

Our findings stand in sharp contrast to a parallel literature on managing model complexity in computational economics. A central challenge in solving non-linear DSGE models by perturbation methods is that higher-order Taylor expansions around the steady state generate explosive sample paths, because the higher-order terms introduce unstable dynamics when the system is iterated forward in time. The standard fix is *pruning*: cross-products of higher-order terms are deliberately dropped from the recursion to restore stability. Originally proposed for second-order approximations by Kim et al. (2008), this approach was extended and fully developed by Andreasen et al. (2018), who established its theoretical foundations for approximations of any order. Pruning is thus an exercise in deliberate *under*-parameterization: parameters are removed to make the approximation well-behaved. Our paper delivers the opposite message. In the neural network approach, the approximation is not a fixed-order polynomial but a flexible function class whose width, measured by the number of trainable parameters, can be freely expanded. We show that increasing network width, rather than restricting it, is precisely what delivers accuracy and stability. The perturbation literature *prunes to survive*; our results suggest that neural network solvers *grow to thrive*.

We demonstrate these properties across three models with known analytical or numerical benchmark solutions. The first is the Linear-Quadratic (LQ) regulator, a cornerstone of dynamic programming with a closed-form solution, which allows us to assess approximation error exactly. The second is the McCall (1970) job-search model, a workhorse of labor economics in which an unemployed worker solves a stopping problem with a one-dimensional state. The third is the Real Business Cycle (RBC) model of Kydland and Prescott (1982), a two-dimensional dynamic problem that we benchmark against value function iteration. Across all three models, we find the same pattern: accuracy and stability improve with network width and depth, and there is no sign of overfitting.

The paper proceeds as follows. Section 2 defines the neural network architecture and notation. Section 3 presents the three economic applications and the main results. Section 4 concludes.

2 Neural Network Definition

Throughout the paper we approximate policy and value functions using feedforward neural networks. This section fixes the notation used across all three applications.

One hidden layer. Let $x \in \mathbb{R}^n$ be the input (the state of the economic model). A one-hidden-layer network with H hidden units computes

$$f_\theta(x) \;=\; W^{(2)}\,\sigma\big(W^{(1)}x + b^{(1)}\big) + b^{(2)}, \tag{1}$$

where $W^{(1)} \in \mathbb{R}^{H\times n}$ and $b^{(1)} \in \mathbb{R}^H$ are the weight matrix and bias vector of the hidden layer, $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function applied elementwise, and $W^{(2)} \in \mathbb{R}^{1\times H}$ and $b^{(2)} \in \mathbb{R}$ are the output-layer weights and bias. The full parameter vector is $\theta = \big(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\big)$, with $H(n+2)+1$ trainable parameters in total.

Two hidden layers. The extension to two hidden layers introduces an additional weight matrix $W^{(2)} \in \mathbb{R}^{H\times H}$ between the two hidden layers:

$$f_\theta(x) \;=\; W^{(3)}\,\sigma\big(W^{(2)}\,\sigma\big(W^{(1)}x + b^{(1)}\big) + b^{(2)}\big) + b^{(3)}, \tag{2}$$

with $\theta = \big(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, W^{(3)}, b^{(3)}\big)$ and $H(n+H+3)+1$ trainable parameters.

Activation functions. The baseline activation is the Rectified Linear Unit (ReLU), $\sigma(z) = \max\{0, z\}$. We also consider Leaky ReLU, $\sigma(z) = \max\{\alpha z, z\}$ with $\alpha = 0.01$, and the sigmoid, $\sigma(z) = \frac{1}{1+e^{-z}}$.

Training. We train all networks by minimizing a squared residual loss using the Adam optimizer. For policy functions the residual is the Euler equation; for value functions it is the Bellman equation. We use 50 independent random seeds per specification to assess stability across initializations. The hidden width H is varied systematically; larger H increases the parameter count and moves the network into the overparameterized regime.

3 Applications

We demonstrate the blessing of overparameterization across three canonical economic models. In each case, we train a one-hidden-layer neural network with ReLU activations using 50 random seeds, and measure both train and test loss alongside the absolute relative error against a known benchmark solution.

3.1 Linear-Quadratic Regulator

The Linear-Quadratic (LQ) regulator is a cornerstone of dynamic programming and optimal control. It provides our sharpest test of approximation accuracy because the true solution is known in closed form, so approximation error can be computed exactly at every point rather than relative to a numerical benchmark. We focus here on the deterministic case to keep the exposition simple; our third application, the RBC model, introduces aggregate uncertainty and illustrates how the same approach extends to stochastic environments.

The general deterministic LQ problem takes the form

$$\begin{aligned} v(x) &= \max_u \big\{ -x^\top R\, x - u^\top Q\, u + \beta\, v(x') \big\} \\ \text{s.t.} \quad & x' = Ax + Bu, \\ & x_0 \quad \text{given,} \end{aligned} \tag{3}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ is the control, $R \in \mathbb{R}^{n\times n}$ and $Q \in \mathbb{R}^{m\times m}$ are symmetric matrices penalizing the state and control respectively, and $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ govern the linear state transition. The closed-form solution is

$$v(x) = -x^\top P x, \qquad u(x) = -Fx, \tag{4}$$

where $P \in \mathbb{R}^{n\times n}$ and $F \in \mathbb{R}^{m\times n}$ are obtained from the discrete-time algebraic Riccati equation.

We consider a price-taking firm operating in a competitive industry and solving an investment problem with quadratic adjustment costs. The firm's own output is y and the industry-wide aggregate output is Y, which the firm takes as exogenous. The output price $(\alpha_0 - \alpha_1 Y)$ is decreasing in the aggregate, capturing a standard downward-sloping inverse demand curve. Each period the firm chooses investment u, which increases its output one-for-one next period ($y' = y + u$), but incurs a convex adjustment cost $\frac{\gamma}{2}u^2$. The aggregate evolves deterministically as $Y' = h_0 + h_1 Y$. The firm's problem is

$$\begin{aligned} v(y, Y) &= \max_u \left\{ (\alpha_0 - \alpha_1 Y)y - \frac{\gamma}{2}u^2 + \beta\, v(y', Y') \right\} \\ \text{s.t.} \quad & Y' = h_0 + h_1 Y, \\ & y' = y + u, \\ & y_0, Y_0 \quad \text{given.} \end{aligned}$$

The state is two-dimensional: the firm must track both its own output y and the industry aggregate Y, since Y affects the price it receives. With augmented state vector $x = (1, y, Y)^\top$, this is an instance of the general LQ framework with

$$R = \begin{bmatrix} 0 & -\frac{\alpha_0}{2} & 0 \\ -\frac{\alpha_0}{2} & 0 & \frac{\alpha_1}{2} \\ 0 & \frac{\alpha_1}{2} & 0 \end{bmatrix}, \qquad Q = \tfrac{\gamma}{2}, \qquad A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ h_0 & 0 & h_1 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

The closed-form solution $v(x) = -x^\top Px$ and $u(x) = -Fx$ follows directly, where P and F are obtained from the discrete-time algebraic Riccati equation. The optimal policy is linear in the state: the firm invests more when the aggregate is low (prices are high) and when its own output is low relative to where it wants to be.

The Euler equation, which equates the marginal cost of investment today to its discounted marginal benefit tomorrow, is:

$$\gamma\, u(y, Y) = \beta\big[\gamma\, u(y', Y') + (\alpha_0 - \alpha_1 Y')\big].$$

Because the firm's revenue $(\alpha_0 - \alpha_1 Y)y$ is linear in y, the Euler equation is independent of y: the state y drops out and the optimal policy depends only on the aggregate Y. The Euler equation therefore reduces to

$$\gamma\, u(Y) = \beta\big[\gamma\, u(Y') + (\alpha_0 - \alpha_1 Y')\big].$$

This is a linear equation in the policy function, which makes the LQ model the most tractable case for theoretical analysis of why overparameterization helps. The parameter values used in our experiments are reported in Table 1 (Appendix A).

3.1.1 Deep learning implementation

We train the policy and value functions separately. We begin with the policy network, as the value function network conditions on the estimated policy. The value function implementation involves additional details regarding the sequential training procedure and the construction of the Bellman residual. Appendix B describes the algorithm in full and reports the corresponding results: train and test Bellman loss as a function of network width, and the maximum and median absolute relative error of the NN value function against the closed-form solution. The patterns closely mirror those found for the policy function: approximation accuracy improves with width and initialization sensitivity collapses, so the value function approximation becomes both more accurate and more stable.

Network architecture. We use a one-hidden-layer network for the policy with ReLU activations and a linear output layer. We vary the number of hidden units across a grid and repeat each configuration with 50 random seeds. We denote the policy network by $u(\cdot\,; \theta_u)$, where θ_u is its parameter vector.

Training data. The training grid is $\mathcal{D} = \{Y_1, \ldots, Y_{n_Y}\}$ with $n_Y = 6$ points on $[0.1, 0.5]$. The policy network is trained by minimizing the mean squared Euler residual:

$$\min_{\theta_u} \frac{1}{n_Y} \sum_{i=1}^{n_Y} \Big[\gamma\, u(Y_i; \theta_u) - \beta\big(\gamma\, u(Y_i'; \theta_u) + \alpha_0 - \alpha_1 Y_i'\big)\Big]^2,$$

with $Y_i' = h_0 + h_1 Y_i$. The optimizer is Adam with a StepLR scheduler (step size 100, decay factor 0.99) and early stopping at loss below 10^{-8}.

Evaluation. Test data consists of a $T = 29$ period path of aggregate output simulated forward from Y_0 using the deterministic law of motion $Y_{t+1} = h_0 + h_1 Y_t$. This path is fixed across all seeds. To study the performance of the approximate solutions we focus on two measures: (1) the mean squared Euler residual evaluated at the test data, and (2) the absolute relative error between the approximate and closed-form solution at each test point,

$$\varepsilon_{u,t} \equiv \frac{|\hat{u}_t - u_t|}{|u_t|},$$

where $u_t = u^*(Y_t)$ is the closed-form policy and $\hat{u}_t = u(Y_t; \theta_u^*)$ is the neural network solution for seed s.

For each seed s we summarize path-level errors by two statistics: the maximum, $\bar{\varepsilon}_u^{(s)} \equiv \max_t\{\varepsilon_{u,t}^{(s)}\}$, and the median, $\tilde{\varepsilon}_u^{(s)} \equiv \text{median}_t\{\varepsilon_{u,t}^{(s)}\}$. For each statistics we report:

- **Median**: $\text{median}_s\{\bar{\varepsilon}_u^{(s)}\}$ and $\text{median}_s\{\tilde{\varepsilon}_u^{(s)}\}$, the error for a typical seed.
- **10th percentile**: $\text{quantile}_{0.1,s}\{\bar{\varepsilon}_u^{(s)}\}$ and $\text{quantile}_{0.1,s}\{\tilde{\varepsilon}_u^{(s)}\}$, a lower bound on what the method can achieve.
- **90th percentile**: $\text{quantile}_{0.9,s}\{\bar{\varepsilon}_u^{(s)}\}$ and $\text{quantile}_{0.9,s}\{\tilde{\varepsilon}_u^{(s)}\}$, a conservative upper bound on the error.

The gap between the 10th and 90th percentiles measures stability.

Results. Figure 1 plots train and test Euler-residual loss as functions of the number of parameters. We vary hidden units from $H = 1$ to $H = 7$ (4 to 22 parameters); beyond $H = 7$

both losses have reached the early-stopping floor of 10^{-8}, at which point the generalization gap is zero and the two curves are indistinguishable. This is a direct consequence of the linear structure of the LQ Euler equation, which makes the training problem a linear least-squares fit with a unique interpolant. The across-seed band collapses as width grows, documenting the stability blessing.

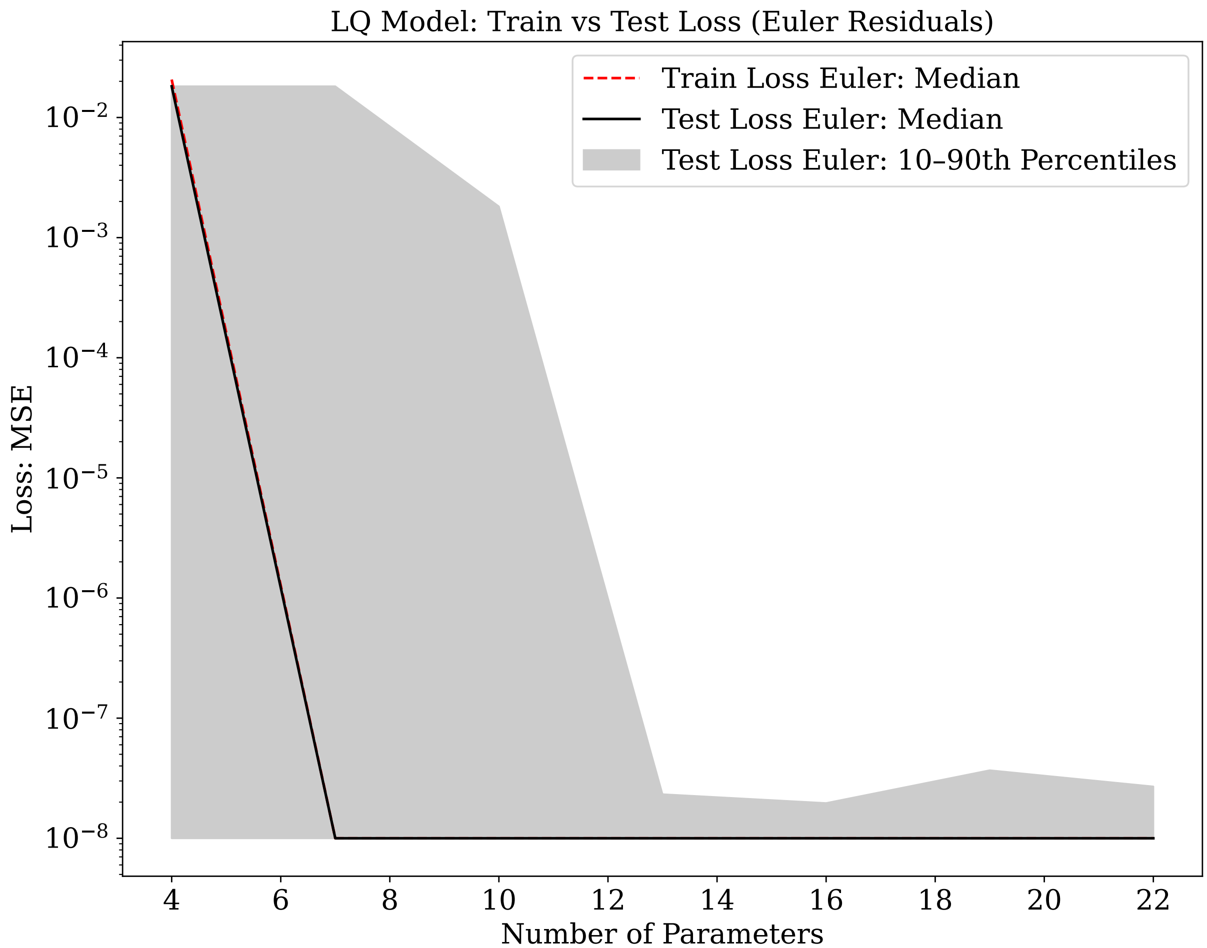

Figure 1: LQ model: Train (dashed red) and test (solid black) Euler-residual MSE as a function of the number of network parameters, aggregated over 50 seeds. The solid line is the median across seeds; the shaded band spans the 10th to 90th percentiles. Once both losses reach the early-stopping floor the generalization gap is zero, and the solutions become more stable as the network grows wider, with the across-seed band collapsing.

Figure 2 shows the median and maximum absolute relative error of the NN policy versus the closed-form solution. Both statistics and their across-seed dispersion fall with network width, confirming the accuracy and stability blessing.

Together, Figures 1 and 2 illustrate both blessings of overparameterization: as the number of parameters grows, the solution becomes more accurate and more stable, with across-seed disagreement collapsing as the network widens.

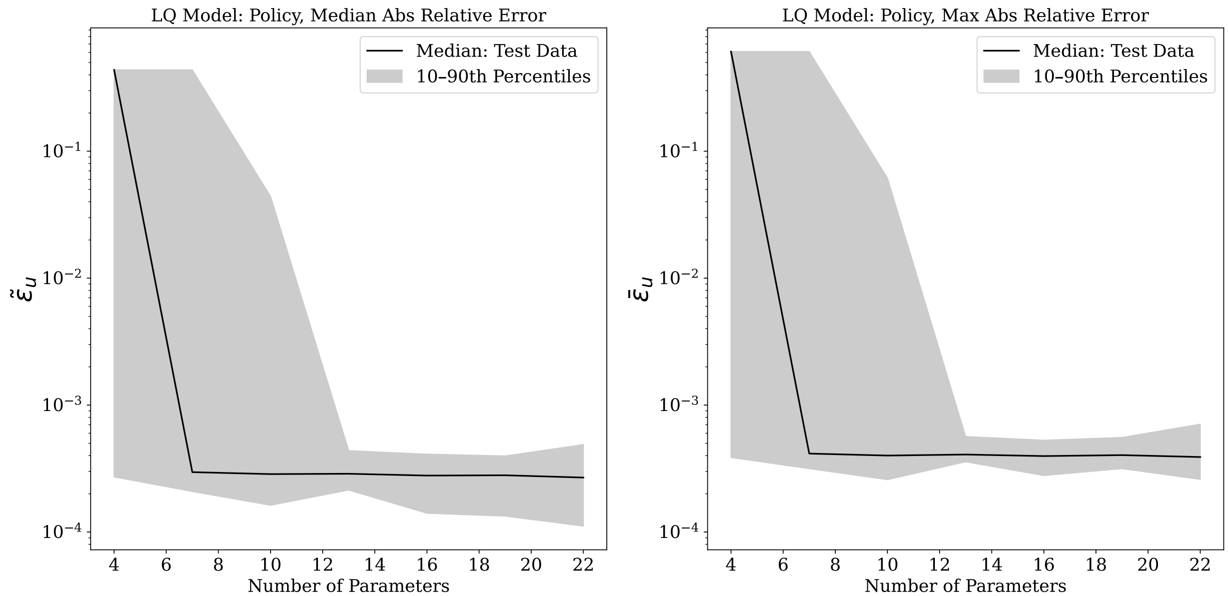

Figure 2: LQ model: Median (left) and maximum (right) absolute relative error of the NN policy versus the closed-form solution as a function of the number of parameters. The solid line is the median across 50 seeds; the shaded band spans the 10th to 90th percentiles. Both statistics fall and solutions become more stable, with cross-seed dispersion declining as the network grows wider.

Figure 3 contrasts the policy paths produced by an underparameterized network ($H = 3$) and an overparameterized network ($H = 32$) against the closed-form solution. With $H = 3$ the median path is already close to the theory, yet the across-seed band is wide, indicating high sensitivity to initialization and poor reliability. With $H = 32$ the median tracks the closed-form solution closely *and* the band collapses, reflecting both accuracy and stability across random seeds.

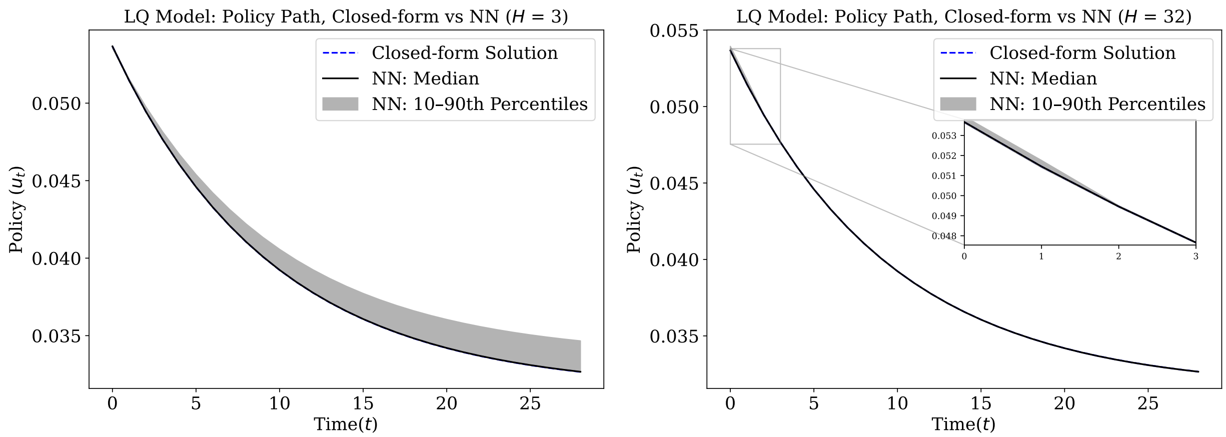

Figure 3: LQ model: Policy paths for an underparameterized ($H = 3$, left) and an overparameterized ($H = 32$, right) network (solid black: median across 50 seeds; shaded band: 10th–90th percentiles) versus the closed-form solution (dashed blue). Under-parameterization yields a median that is roughly accurate but a large across-seed dispersion, whereas overparameterization achieves both a close median and a narrow band, demonstrating the reliability gains from adding parameters beyond the interpolation threshold.

Robustness of these results to alternative activation functions is documented in Appendix C.1.

3.2 McCall Job Search Model

The McCall model (McCall, 1970) is the canonical model of job search in labor economics. An unemployed worker receives sequential wage offers and must decide each period whether to accept or keep searching. The model captures a fundamental tradeoff in labor markets: accepting a low offer provides immediate income but forecloses the option of finding a better offer in the future, while continued search is costly because the worker forgoes wages and only receives unemployment compensation c during the waiting period. The discount factor $\beta \in (0, 1)$ governs the worker's patience; a more patient worker is willing to search longer and will set a higher reservation wage.

Formally, each period the worker observes an i.i.d. wage draw w from a distribution f on $[0, B]$. Accepting yields a permanent income stream with present value $w/(1-\beta)$; rejecting yields unemployment compensation c this period and continuation value $\beta \int_0^B v(w')f(w')\,dw'$

next period. The value function satisfies the Bellman equation

$$v(w) = \max\left\{\frac{w}{1-\beta},\ c + \beta \int_0^B v(w')f(w')\,dw'\right\}.$$

The key economic insight of the model is that the optimal policy takes a simple threshold form: there exists a reservation wage $\bar{w}$ such that the worker accepts any offer $w > \bar{w}$ and rejects any offer $w \le \bar{w}$. This implies a closed-form solution for the value function:

$$v(w) = \begin{cases} \dfrac{\bar{w}}{1-\beta} & \text{if } w \le \bar{w}, \\ \dfrac{w}{1-\beta} & \text{if } w > \bar{w}, \end{cases}$$

where $\bar{w}$ solves

$$\bar{w} - c = \frac{\beta}{1-\beta}\int_{\bar{w}}^B (w' - \bar{w})f(w')\,dw'.$$

The left-hand side is the per-period gain from accepting the reservation wage over collecting unemployment compensation; the right-hand side is the option value of continued search.

The kink in the value function at $\bar{w}$ makes this a demanding test for neural network approximation, as the network must learn a non-smooth function from a small training grid. The closed-form solution serves as the exact benchmark. The parameter values used in our experiments are reported in Table 1 (Appendix A).

3.2.1 Deep learning implementation

Network architecture. We approximate $v(\cdot)$ with a one-hidden-layer neural network $v(w;\theta_v)$ with ReLU activations and a linear output layer. We vary the number of hidden units across a grid and repeat each configuration with 50 random seeds.

Training data. The training grid is a uniform grid $\mathcal{D} = \{w_1, \ldots, w_N\}$ with $N = 11$ equally spaced points on $[0, 1]$. We minimize the mean squared Bellman residual:

$$\min_{\theta_v} \frac{1}{N}\sum_{w_i \in \mathcal{D}} \left[v(w_i;\theta_v) - \max\left\{\frac{w_i}{1-\beta},\ c + \beta\int_0^B v(w';\theta_v)f(w')\,dw'\right\}\right]^2.$$

The integral is approximated by Gauss-Legendre quadrature with 50 nodes. The optimizer is Adam with a StepLR scheduler (step size 100, decay factor 0.99) and early stopping at loss below 10^{-8}.

Evaluation. Test loss is evaluated on a finer uniform grid of $N = 101$ points on $[0, 1]$, which is fixed across all seeds and hidden-unit configurations. This grid lies strictly outside the training set, so test loss measures genuine out-of-sample performance. The absolute relative error at each test point is

$$\varepsilon_v^{(s)}(w) \equiv \frac{|\hat{v}(w) - v(w)|}{|v(w)|},$$

where $v(w)$ is the closed-form value function and $\hat{v}(w) = v(w; \theta_v^*)$ is the neural network solution for seed s. We summarize each seed by two statistics: the maximum across the test grid, $\bar{\varepsilon}_v^{(s)} \equiv \max_w\{\varepsilon_v^{(s)}(w)\}$, and the median, $\tilde{\varepsilon}_v^{(s)} \equiv \text{median}_w\{\varepsilon_v^{(s)}(w)\}$. For each statistics we report:

- **Median**: $\text{median}_s\{\bar{\varepsilon}_v^{(s)}\}$ and $\text{median}_s\{\tilde{\varepsilon}_v^{(s)}\}$, the error for a typical seed.
- **10th percentile**: $\text{quantile}_{0.1,s}\{\bar{\varepsilon}_v^{(s)}\}$ and $\text{quantile}_{0.1,s}\{\tilde{\varepsilon}_v^{(s)}\}$, a lower bound on what the method can achieve.
- **90th percentile**: $\text{quantile}_{0.9,s}\{\bar{\varepsilon}_v^{(s)}\}$ and $\text{quantile}_{0.9,s}\{\tilde{\varepsilon}_v^{(s)}\}$, a conservative upper bound on the error.

The gap between the 10th and 90th percentiles measures stability.

Results. Figure 4 plots train and test Bellman-residual loss as functions of the number of parameters. Train loss reaches the early-stopping floor and test loss descends again beyond it, tracing out a double-descent pattern. The across-seed band collapses as width grows, confirming that overparameterization improves both accuracy and stability.

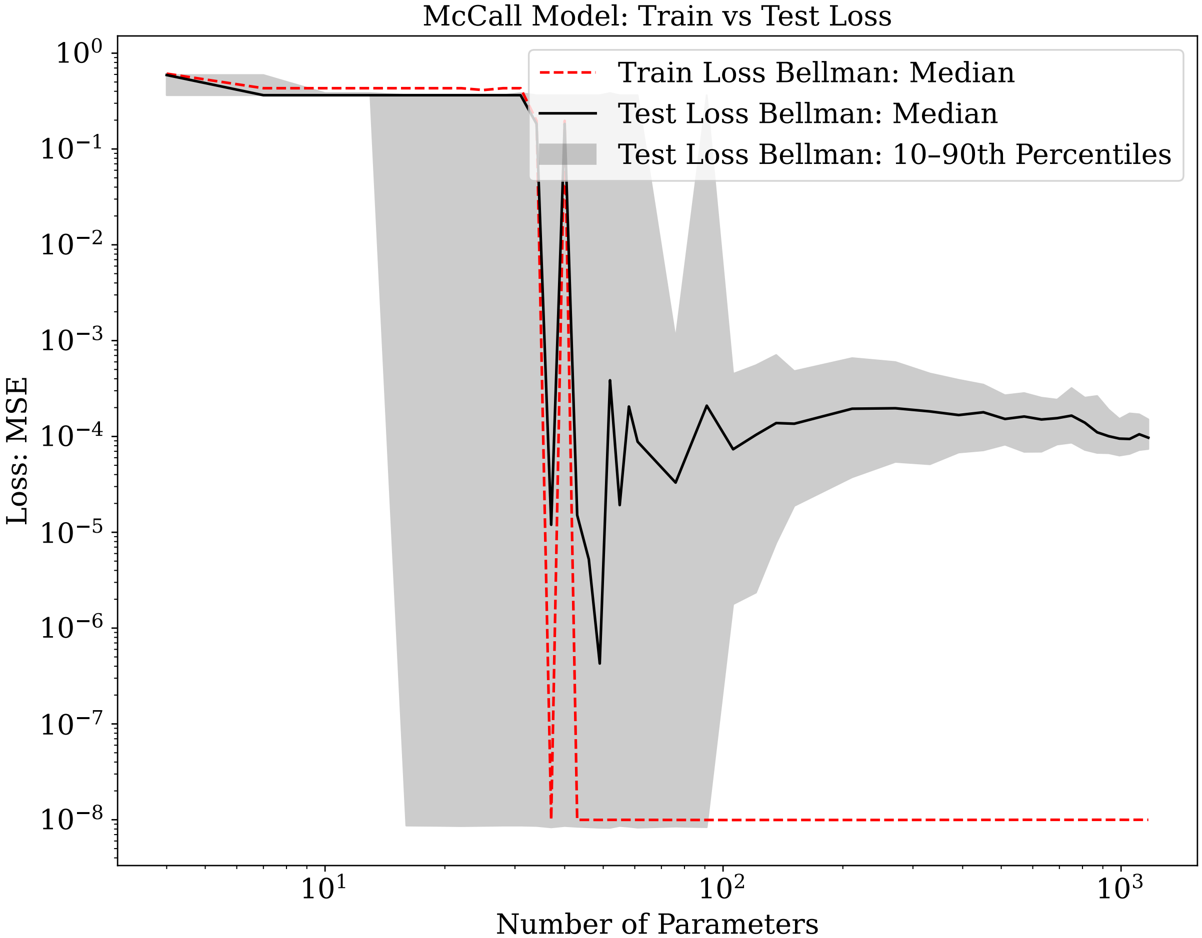

Figure 4: McCall model: Train (dashed red) and test (solid black) Bellman-residual MSE as a function of the number of network parameters, aggregated over 50 seeds. The solid line is the median across seeds; the shaded band spans the 10th to 90th percentiles. Train loss hits the early-stopping floor; test loss descends again in the overparameterized regime, tracing out a double-descent pattern; solutions become more stable, with the across-seed band collapsing.

Figure 5 shows the median and maximum absolute relative error versus the closed-form solution. Both statistics fall sharply with width: the median error drops from $\sim$30% for tiny networks to below 0.1%, and the maximum below 2%, in the overparameterized regime. The across-seed band narrows sharply, confirming the accuracy and stability gains.

Together, Figures 4 and 5 illustrate both blessings of overparameterization: as the number of parameters grows, the solution becomes more accurate and more stable, with across-seed disagreement collapsing as the network widens.

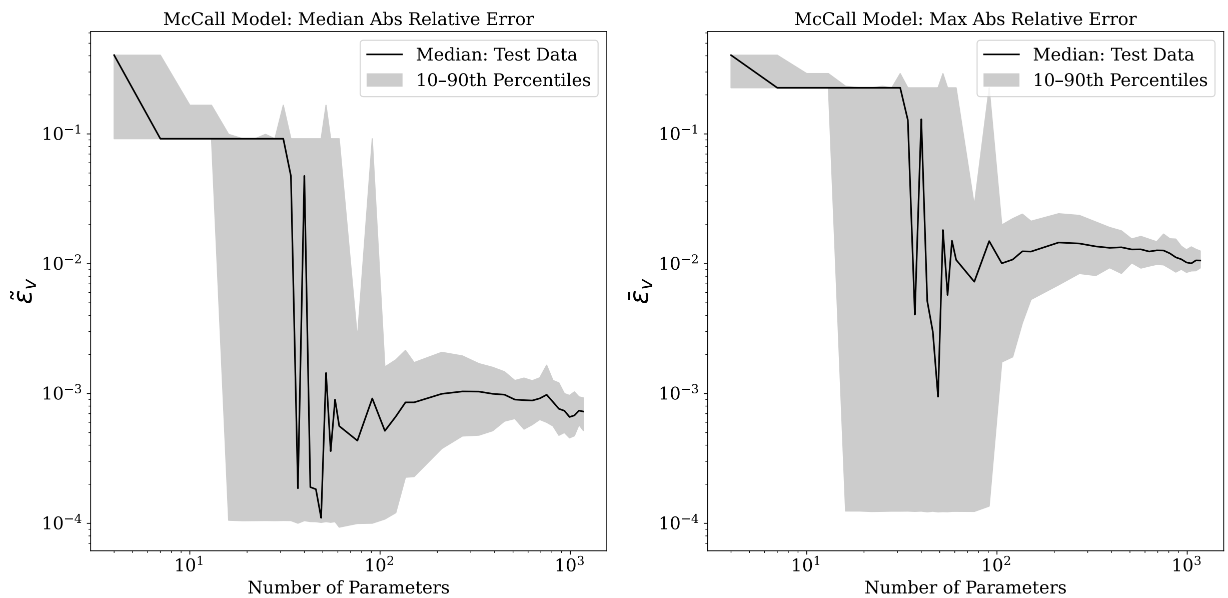

Figure 5: McCall model: Median (left) and maximum (right) absolute relative error of the NN value function versus the closed-form solution as a function of the number of parameters. The solid line is the median across 50 seeds; the band spans the 10th to 90th percentiles. Both statistics fall and solutions become more stable, with cross-seed dispersion declining as the network grows wider.

Figure 6 contrasts the value function learned by an underparameterized network ($H = 24$, left) with that of an overparameterized one ($H = 512$, right). Both networks track the closed-form solution (dashed blue) closely across most of the wage grid, but the underparameterized network struggles more around the kink at the reservation wage $\bar{w} \approx 0.63$: above $\bar{w}$ the worker accepts and $v(w) = w/(1-\beta)$; below it the worker rejects and v is flat. The key difference lies in the across-seed dispersion.

The underparameterized network is more volatile: the shaded band (10th–90th percentiles across 50 seeds) is wide, especially where it matters most, near the kink, indicating that the solution depends more on the random initialization. The overparameterized network, by contrast, produces a nearly invisible band; 50 random initializations converge to the same function. Overparameterization improves both accuracy and stability.

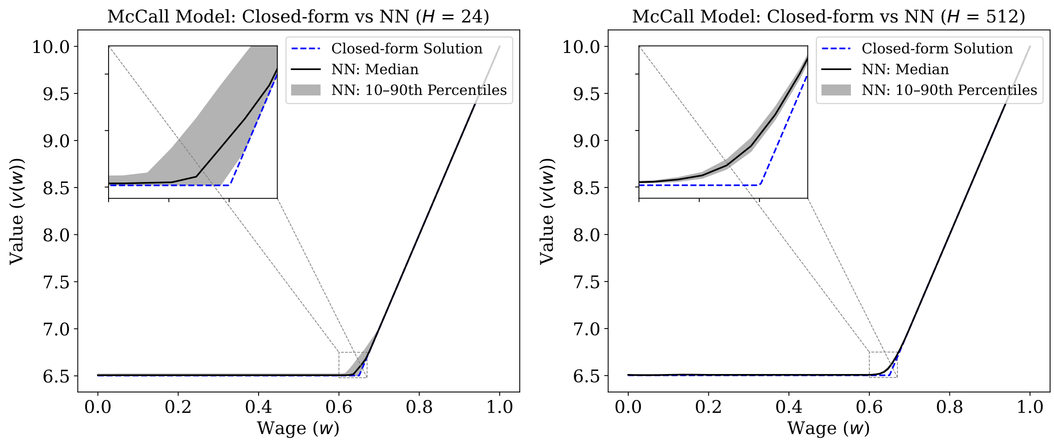

Figure 6: McCall model: Value function learned by under- ($H = 24$, left) and overparameterized ($H = 512$, right) networks against the closed-form solution (dashed blue). Solid black: median across 50 seeds; shaded band: 10th–90th percentiles. Both networks match the closed-form median closely and recover the kink at the reservation wage $\bar{w} \approx 0.63$, but the underparameterized network exhibits wider across-seed dispersion, reflecting sensitivity to initialization that disappears once the network is sufficiently overparameterized.

Robustness of these results to alternative activation functions and a two-hidden-layer architecture is documented in Appendix C.2.

3.3 Real Business Cycle Model

The Real Business Cycle (RBC) model of Kydland and Prescott (1982) is a model of aggregate fluctuations driven by productivity shocks. Unlike the McCall and LQ models, it does not admit a closed-form solution, making it the most demanding of our three applications.

The model features a representative household that makes an intertemporal tradeoff between consumption today and investment for tomorrow. Output is produced with a Cobb-Douglas technology $e^{z_t} k_t^{\alpha}$, where k_t is the capital stock and e^{z_t} is total factor productivity (TFP). z_t, the log of TFP, is stochastic and follows a persistent AR(1) process, so the household must forecast future productivity when making its savings decision.

The household's problem is

$$\max_{\{c_t, k_{t+1}\}} \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t u(c_t),$$

subject to the resource constraint

$$c_t + k_{t+1} = e^{z_t} k_t^{\alpha} + (1-\delta) k_t,$$

where $\delta \in (0,1)$ is the depreciation rate and z_t follows

$$z_t = \rho z_{t-1} + \sigma \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0,1).$$

The state space is two-dimensional: the household's decision depends on both the current capital stock k_t and the current productivity level z_t. The key object of interest is the policy function $k'(k,z)$, which maps the current state to the optimal next-period capital. The Euler equation, which is the first-order necessary condition for the household's optimal choice, is

$$u'(c_t) = \beta\, \mathbb{E}_t \left[u'(c_{t+1}) \left(\alpha e^{z_{t+1}} k_{t+1}^{\alpha-1} + 1 - \delta \right) \right].$$

The left-hand side is the marginal utility of saving one unit today; the right-hand side is the expected discounted marginal utility gain from the extra unit of capital tomorrow, which raises output by its marginal product $\alpha e^{z_{t+1}} k_{t+1}^{\alpha-1}$ and also returns undepreciated capital $(1-\delta)$.

This equation, in its general form, does not accept a closed-form solution. We benchmark the neural network approximation against value function iteration (VFI), which solves the model numerically on a fine grid to high precision. The parameter values used in our experiments are reported in Table 1 (Appendix A).

3.3.1 Deep learning implementation

Network architecture. We approximate the capital policy function $k'(k,z;\theta)$ with a one-hidden-layer neural network with ReLU activations. The output layer applies a Softplus activation, $\log(1+e^x)$, to ensure $k'(k,z;\theta) > 0$ at every state, which prevents numerical issues from the non-integer exponent $\alpha - 1$ in the Euler residual. We vary the number of hidden units across a grid and repeat each configuration with 50 random seeds.

Training data. The training grid is a Cartesian product

$$\mathcal{D} = \{k_1, \ldots, k_{n_k}\} \times \{z_1, \ldots, z_{n_z}\}$$

with $n_k = n_z = 20$ ($N = 400$ points total), covering $k \in [\frac{1}{2}k^*, \frac{3}{2}k^*]$ and $z \in \left[\frac{-3\sigma}{\sqrt{1-\rho^2}}, \frac{3\sigma}{\sqrt{1-\rho^2}}\right]$, where k^* is the deterministic steady-state capital. Given the resource constraint and log utility, the Euler residual at each grid point is

$$\ell_{\text{Euler}}(k, z; \theta) = \frac{1}{c(k, z; \theta)} - \beta \sum_{i=1}^{M} \frac{w_i}{\sqrt{\pi}} \left[\frac{\alpha e^{\rho z + \sqrt{2}\sigma\zeta_i} k'(k, z; \theta)^{\alpha-1} + 1 - \delta}{c\big(k'(k, z; \theta),\, \rho z + \sqrt{2}\sigma\zeta_i; \theta\big)} \right],$$

where $c(k, z; \theta) = e^z k^\alpha + (1-\delta)k - k'(k, z; \theta)$ and $\{(\zeta_i, w_i)\}_{i=1}^{M}$ are Gauss-Hermite quadrature nodes and weights with $M = 15$.

We minimize the following loss over $\mathcal{D}$ using Adam with learning rate 10^{-3} and a StepLR scheduler (step size 100, decay factor 0.99):

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{(k,z)\in\mathcal{D}} \ell_{\text{Euler}}(k, z; \theta)^2 + \lambda\big(k'(k^*, 0; \theta) - k^*\big)^2,$$

where $\lambda = 0.01$ is a penalty weight and k^* is the non-stochastic steady-state capital. The penalty term is necessary because the Euler equation is a necessary but not sufficient condition for optimality: both the stable and the explosive solution manifolds through the steady state satisfy it identically. What rules out the explosive solutions is the transversality condition (TVC), $\lim_{t\to\infty} \mathbb{E}_0 \left[\beta^t u'(c_t) k_{t+1}\right] = 0$, which the Euler residual loss does not enforce. The penalty nudges the solution toward the non-explosive manifold. Without this constraint, gradient-based training can converge to the explosive manifold, producing solutions with small Euler residuals but divergent simulated paths (Kahou et al., 2024). Training runs for up to 10,001 epochs with early stopping when the Euler loss falls below 10^{-8}.

Evaluation. Test loss is evaluated on a $T = 29$ period capital path simulated under the VFI policy with shocks drawn using a fixed seed, starting from $k_0 = 0.5\,k^*$. The path is constructed once and held fixed across all network configurations and seeds, so test losses are directly comparable. The test set is assembled as the sequence of (k_t, z_t) pairs along this path, stacked into a (29×2) tensor. Figure 7 shows the training grid (red dots) and the test path (blue crosses); the test path visits regions between but not on the training grid nodes, so test performance is a genuine measure of out-of-sample generalization.

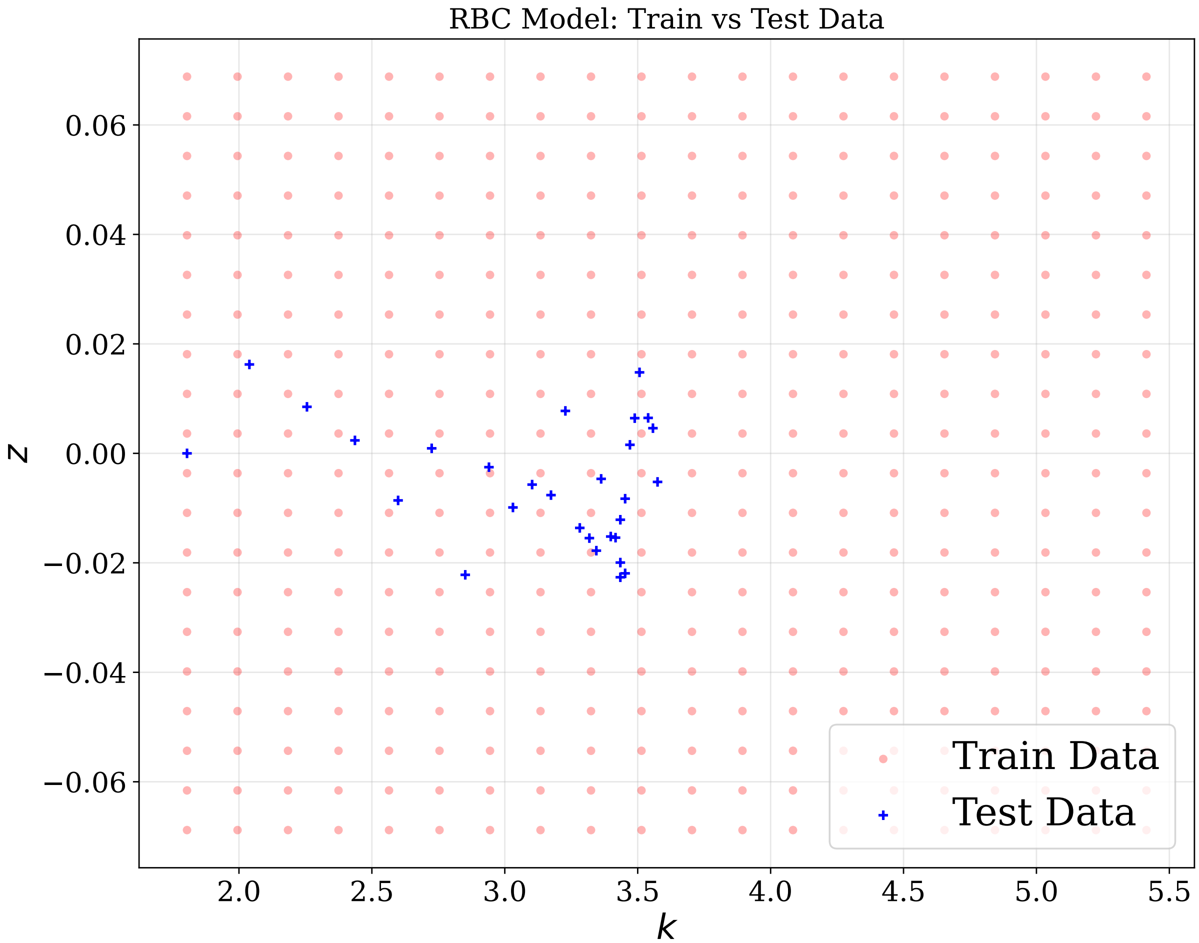

Figure 7: RBC model: Training data (red dots: 20×20 Cartesian grid) and test data (blue crosses: $T = 29$ capital path simulated under the VFI policy with shocks, starting at $k_0 = 0.5\,k^*$). The test path visits the interior of the state space and is never on a training node.

Since VFI solves the model to high precision, any discrepancy reflects only neural network approximation error. Given the trained network, we simulate a $T = 29$ period capital path under the neural network policy, using the same shocks as the VFI test path. The absolute relative error at period t is

$$\varepsilon_{k,t} \equiv \frac{|k_t^{\mathrm{NN}} - k_t^{\mathrm{VFI}}|}{|k_t^{\mathrm{VFI}}|}.$$

For each seed s we summarize the path-level errors by two statistics: the maximum across time,

$$\bar{\varepsilon}^{(s)} \equiv \max_t \{\varepsilon_t^{(s)}\},$$

which is a demanding metric since a single poorly-approximated period is enough to raise it, and the median across time,

$$\tilde{\varepsilon}^{(s)} \equiv \text{median}_t \{\varepsilon_t^{(s)}\},$$

which is less sensitive to individual outlier periods. For each statistics we report:

- **Median**: $\text{median}_s\{\bar{\varepsilon}^{(s)}\}$ and $\text{median}_s\{\tilde{\varepsilon}^{(s)}\}$, the error for a typical seed.
- **10th percentile**: $\text{quantile}_{0.1,s}\{\bar{\varepsilon}^{(s)}\}$ and $\text{quantile}_{0.1,s}\{\tilde{\varepsilon}^{(s)}\}$, a lower bound on what the method can achieve.
- **90th percentile**: $\text{quantile}_{0.9,s}\{\bar{\varepsilon}^{(s)}\}$ and $\text{quantile}_{0.9,s}\{\tilde{\varepsilon}^{(s)}\}$, a conservative upper bound on the error.

The gap between the 10th and 90th percentiles measures stability. Our main finding is that this band collapses as width grows, documenting the stability blessing.

Results. Figure 8 plots train and test Euler-residual loss as functions of the number of network parameters. Both losses decline as the network grows wider, with no sign of overfitting. The shaded band (10th–90th percentiles across 50 seeds) narrows sharply as width increases, showing that overparameterization simultaneously improves accuracy and reduces initialization sensitivity.

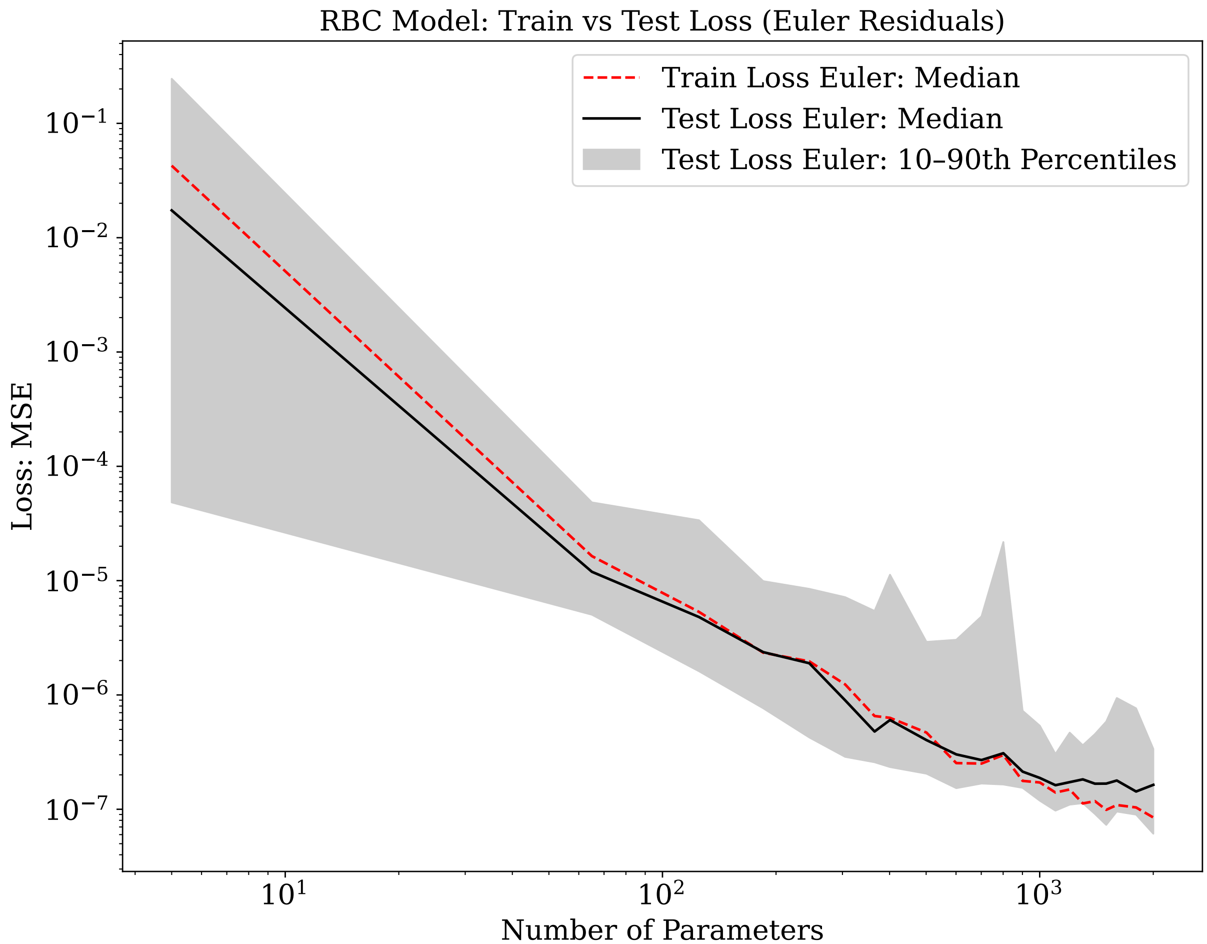

Figure 8: RBC model: Train (dashed red) and test (solid black) Euler-residual MSE as a function of the number of network parameters, aggregated over 50 random seeds. The solid line is the median across seeds; the shaded band spans the 10th to 90th percentiles. Both losses fall and solutions become more stable, with the across-seed variance collapsing as the network grows wider.

Figure 9 translates the Euler-residual losses into economically interpretable units. Both the median and maximum absolute relative error of the capital policy relative to VFI fall and their across-seed bands collapse as the network grows wider, confirming that larger networks produce policies that are both more accurate and more reliable.

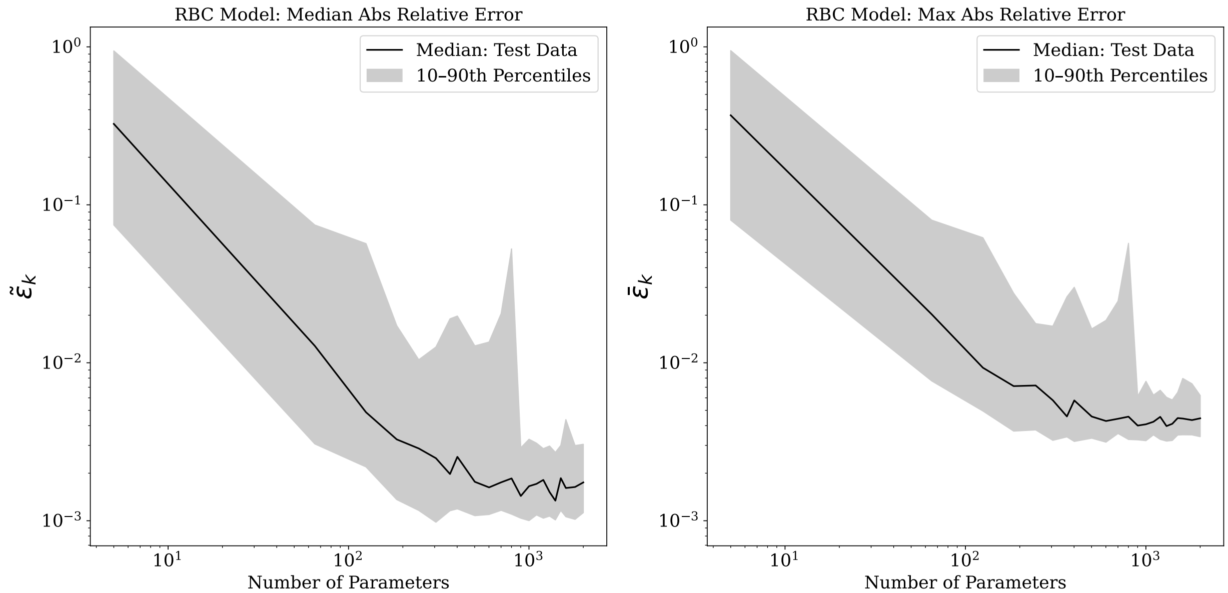

Figure 9: RBC model: Median (left) and maximum (right) absolute relative error of the NN capital policy versus the VFI benchmark as a function of the number of parameters. The solid line is the median across 50 seeds; the shaded band spans the 10th to 90th percentiles. Both statistics decline and solutions become more stable, concentrating around the median seed as network width increases.

Figure 10 contrasts the simulated capital paths for an underparameterized network ($H = 32$) and an overparameterized network ($H = 512$). With $H = 32$ the median path is close to the VFI benchmark, yet the 10th–90th percentile band across 50 seeds is wide, revealing high sensitivity to initialization and poor reliability: the median network is accurate but not robust or stable. With $H = 512$ the median tracks the VFI path almost exactly over the full $T = 29$ horizon *and* the across-seed band collapses to near zero, demonstrating that overparameterization delivers both accuracy and stability.

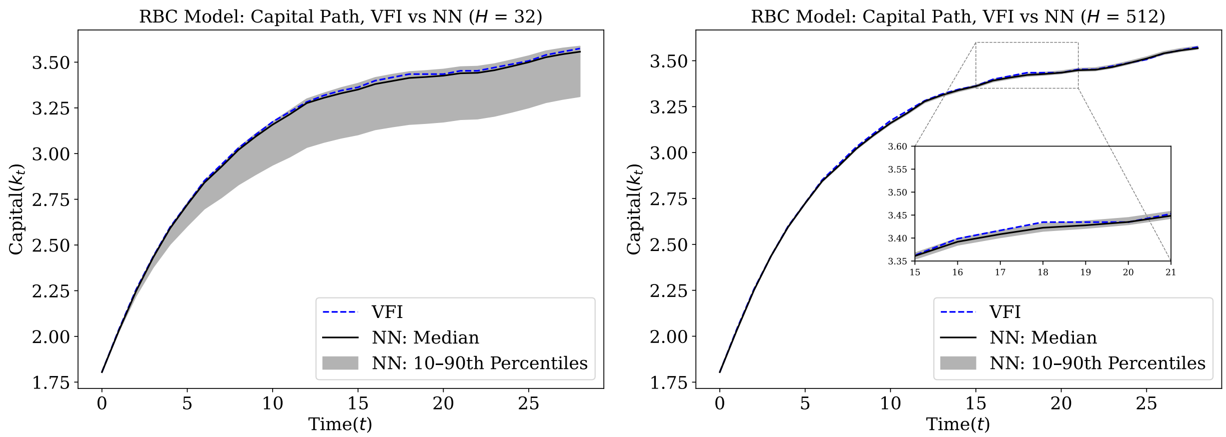

Figure 10: RBC model: Capital paths under VFI (dashed blue) versus the NN policy for an underparameterized ($H = 32$, left) and an overparameterized ($H = 512$, right) network (solid black: median across 50 seeds; shaded band: 10th–90th percentiles). Under-parameterization yields a median that approximates the VFI path but with a large across-seed dispersion, indicating the solution is unreliable and sensitive to initialization. Over-parameterization achieves both a close median and a negligible band, illustrating the robustness and stability gains from operating beyond the interpolation threshold.

Robustness of these results to alternative activation functions and a two-hidden-layer architecture is documented in Appendix C.3.

4 Conclusion

This paper documents two blessings of overparameterization in the neural network solution of dynamic economic models. First, there is no overfitting: as network width increases, out-of-sample Euler and Bellman residuals decline, and approximate policy and value functions converge toward their benchmark counterparts. Second, and more fundamentally, overparameterization delivers algorithmic stability. Small, underparameterized networks produce solutions that are highly sensitive to random initialization, leaving no reliable way to assess solution quality. Wide, overparameterized networks concentrate across seeds, making the algorithm reliable and initialization-independent. We document both findings consistently across three canonical models: an LQ regulator with a closed-form solution, a McCall job search model with a kink in the value function, and a stochastic RBC model without a closed-form solution.

These findings are robust across the dimensions most relevant to applied practice. The decline in approximation error and the collapse in across-seed dispersion survive changes in activation function (from ReLU to Leaky ReLU and Sigmoid) and in network depth, with results for two-hidden-layer architectures matching those for single-hidden-layer networks throughout.

The implication is direct. Economists solving dynamic models with neural networks need not be concerned about scale. The evidence points in the opposite direction: when in doubt, use a wider network, or a deeper one. Overparameterization is not a source of risk to be managed but a feature to be exploited (Telgarsky, 2016; He et al., 2016).

References

Adenbaum, J., F. Babalievsky, and W. Jungerman (2024): "Learning on the Job," Working Paper.

Andreasen, M. M., J. Fernández-Villaverde, and J. F. Rubio-Ramírez (2018): "The pruned state-space system for non-linear DSGE models: Theory and empirical applications," *The Review of Economic Studies*, 85, 1–49.

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

Azinovic, M. and J. Žemlička (2024): "Intergenerational consequences of rare disasters," University of Waterloo Economics.

Azinovic-Yang, M. and J. Žemlička (2025): "Deep learning in the sequence space," *arXiv preprint*.

Barnett, M., W. Brock, L. P. Hansen, R. Hu, and J. Huang (2023): "A deep learning analysis of climate change, innovation, and uncertainty," .

Bartlett, P. L., A. Montanari, and A. Rakhlin (2020): "Benign overfitting in linear regression," *Proceedings of the National Academy of Sciences*, 117, 30063–30070.

Belkin, M., D. Hsu, S. Ma, and S. Mandal (2019): "Reconciling modern machine-learning practice and the classical bias–variance trade-off," *Proceedings of the National Academy of Sciences*, 116, 15849–15854.

Druedahl, J. and J. Røpke (2026): "Deep learning algorithms for solving convex finite-horizon models," Working Paper.

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

Gopalakrishna, G., J. Payne, and Z. Gu (2024): "Asset pricing, participation constraints, and inequality," Proceedings of the EUROFIDAI-ESSEC Paris December Finance Meeting.

Gu, Z., M. Laurière, S. Merkel, and J. Payne (2024): "Global solutions to master equations for continuous time heterogeneous agent macroeconomic models," *arXiv preprint arXiv:2406.13432*.

Han, J., Y. Yang, and W. E (2022): "Deepham: A global solution method for heterogeneous agent models with aggregate shocks," .

He, K., X. Zhang, S. Ren, and J. Sun (2016): "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Jungerman, W. (2023): "Dynamic monopsony and human capital," Working Paper.

Kahou, M. E., J. Fernández-Villaverde, S. Gomez-Cardona, J. Perla, and J. Rosa (2024): "Spooky Boundaries at a Distance: Inductive Bias, Dynamic Models, and Behavioral Macro," Working Paper 32850, National Bureau of Economic Research.

Kahou, M. E., J. Fernández-Villaverde, J. Perla, and A. Sood (2021): "Exploiting symmetry in high-dimensional dynamic programming," Tech. rep., National Bureau of Economic Research.

Kase, H., L. Melosi, and M. Rottner (2022): "Estimating nonlinear heterogeneous agents models with neural networks," Tech. rep., Centre for Economic Policy Research.

Kim, J., S. Kim, E. Schaumburg, and C. A. Sims (2008): "Calculating and using second-order accurate solutions of discrete time dynamic equilibrium models," *Journal of Economic Dynamics and Control*, 32, 3397–3414.

Kydland, F. E. and E. C. Prescott (1982): "Time to build and aggregate fluctuations," *Econometrica*, 50, 1345–1370.

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

McCall, J. J. (1970): "Economics of information and job search," *The Quarterly Journal of Economics*, 84, 113–126.

Payne, J., A. Revei, and Y. Yang (2024): "Deep learning for search and matching models," SSRN 4768566.

Phan, N. (2025): "The welfare consequences of countercyclical fiscal transfers," Working paper, Queen's University.

Telgarsky, M. (2016): "Benefits of depth in neural networks," in *Conference on Learning Theory*, 1517–1539, arXiv:1602.04485.

Appendix

A Model Parameters

Table 1 lists the parameter values used in all experiments.

Table 1: Parameter values for the three models.

Description	Symbol	Value	
Panel A: Linear-Quadratic Regulator			
Discount factor	β	0.9	
Demand intercept	α_0	1.0	inverse demand level
Demand slope	α_1	1.3	price sensitivity to aggregate
Adjustment cost	γ	100	convex investment cost
Aggregate drift	h_0	0.05	constant in $Y' = h_0 + h_1 Y$
Aggregate persistence	h_1	0.9	AR coefficient for Y
Initial firm output	y_0	0.1	starting value on test path
Initial aggregate	Y_0	0.1	starting value on test path
Panel B: McCall Job Search			
Discount factor	β	0.9	
Wage support	B	1.0	upper bound of $\mathcal{U}[0, B]$ wage distribution
Unemployment benefit	c	0.1	flow payoff while searching
Panel C: Real Business Cycle			
Discount factor	β	0.96	
Capital share	α	1/3	Cobb-Douglas exponent
Depreciation rate	δ	0.1	per-period capital depreciation
TFP persistence	ρ	0.9	AR(1) coefficient for z_t
TFP volatility	σ	0.01	standard deviation of ε_t
Utility function	$u(c)$	$\ln c$	log utility (CRRA, $\eta = 1$)

B LQ Regulator: Value Function Results

Algorithm. The pre-trained policy $u(\cdot;\theta_u^*)$ is fixed throughout and was obtained using the algorithm of Section 3.1 with $H = 8$ hidden units.

Given this fixed policy, we train a value function network $v(y, Y; \theta_v)$ by minimizing the mean squared Bellman residual over a Cartesian training grid $\mathcal{D} = \{y_1, \ldots, y_{n_y}\} \times \{Y_1, \ldots, Y_{n_Y}\}$, with $n_y = 12$ points on $[0.1, 1.2]$ and $n_Y = 12$ points on $[0.1, 0.5]$ ($N =$

$n_y \times n_Y = 144$ points in total):

$$\min_{\theta_v} \frac{1}{N} \sum_{(y_i, Y_i) \in \mathcal{D}} \Big[v(y_i, Y_i; \theta_v) - \Big(-\tfrac{\gamma}{2} u(Y_i; \theta_u^*)^2 + (\alpha_0 - \alpha_1 Y_i) y_i + \beta\, v(y_i', Y_i'; \theta_v) \Big) \Big]^2,$$

where the next-period states are implied by the pre-trained policy:

$$\begin{aligned} y_i' &= y_i + u(Y_i; \theta_u^*), \\ Y_i' &= h_0 + h_1 Y_i. \end{aligned}$$

The value function network is a two-input (y and Y), one-hidden-layer ReLU network with a linear output layer. The optimizer is Adam with a StepLR scheduler (decay factor $\gamma = 0.99$ every 100 epochs) and early stopping when the Bellman loss falls below 10^{-8}. We vary the number of hidden units over the grid $H \in \{1, 21, 41, 61, 81, 100, 150, 200, \ldots, 1400\}$ and repeat each width with 50 random seeds.

Evaluation. We simulate a $T = 20$ period path from (y_0, Y_0) using the **true LQ policy** $u^*(Y_t) = -F\, x_t$, where $x_t = (1, y_t, Y_t)^\top$ and $F_y = 0$ (Section 3.1). The state path evolves as:

$$\begin{aligned} Y_{t+1} &= h_0 + h_1\, Y_t, \\ y_{t+1} &= y_t + u^*(Y_t). \end{aligned}$$

Because the simulation uses the true policy, the sequence $\{(y_t, Y_t)\}_{t=0}^{T-1}$ carries no approximation error, so any discrepancy in the evaluation below reflects only value function approximation error.

The NN value function is evaluated at each point along the path:

$$\hat{v}_t = v(y_t, Y_t;\, \theta_v^*).$$

The closed-form benchmark is $v_t = -x_t^\top P\, x_t$, where P is the solution to the discrete Riccati equation. The accuracy metric is the absolute relative error:

$$\varepsilon_{v,t} = \frac{|\hat{v}_t - v_t|}{|v_t|}.$$

We summarize each seed by two statistics: the maximum across the test path, $\bar{\varepsilon}_v^{(s)} \equiv \max_t\{\varepsilon_{v,t}^{(s)}\}$, and the median, $\tilde{\varepsilon}_v^{(s)} \equiv \text{median}_t\{\varepsilon_{v,t}^{(s)}\}$. For each statistics we report:

- **Median**: $\text{median}_s\{\bar{\varepsilon}_v^{(s)}\}$ and $\text{median}_s\{\tilde{\varepsilon}_v^{(s)}\}$, the error for a typical seed.

- **10th percentile**: $\text{quantile}_{0.1,s}\{\bar{\varepsilon}_v^{(s)}\}$ and $\text{quantile}_{0.1,s}\{\tilde{\varepsilon}_v^{(s)}\}$, a lower bound on what the method can achieve.
- **90th percentile**: $\text{quantile}_{0.9,s}\{\bar{\varepsilon}_v^{(s)}\}$ and $\text{quantile}_{0.9,s}\{\tilde{\varepsilon}_v^{(s)}\}$, a conservative upper bound on the error.

The gap between the 10th and 90th percentiles measures stability.

Figure 11 shows the 12×12 Cartesian training grid (red dots) and the $T = 20$ test path (blue crosses) in the (y, Y) state space. With the exception of the initial condition (y_0, Y_0), test points lie in the interior of the grid and never coincide with a training node.

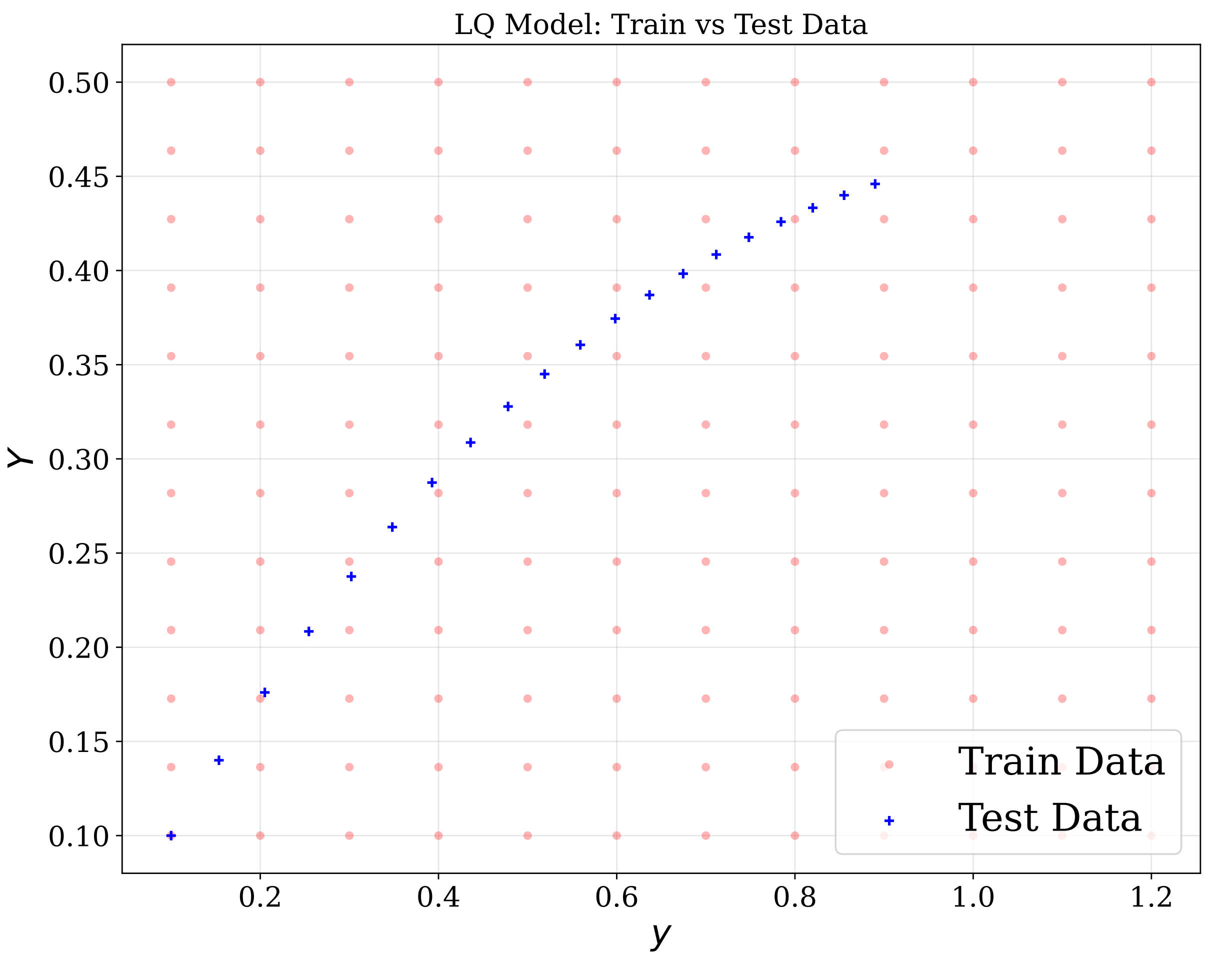

Figure 11: Training grid (red dots) and test path (blue crosses) for the LQ value function experiment. The training grid is a 12×12 Cartesian product of y and Y values; the test path is a $T = 20$ period trajectory simulated from (y_0, Y_0) using the true LQ policy. Test points lie in the interior of the state space and never fall on a training node.

Results. Figure 12 plots train and test Bellman-residual loss as functions of the number of parameters. We vary hidden units from $H = 1$ to $H = 1400$; both losses decline steadily across the full width range and the generalization gap remains small throughout. The across-seed band collapses as width grows, confirming that the value function approximation becomes both more accurate and more stable.

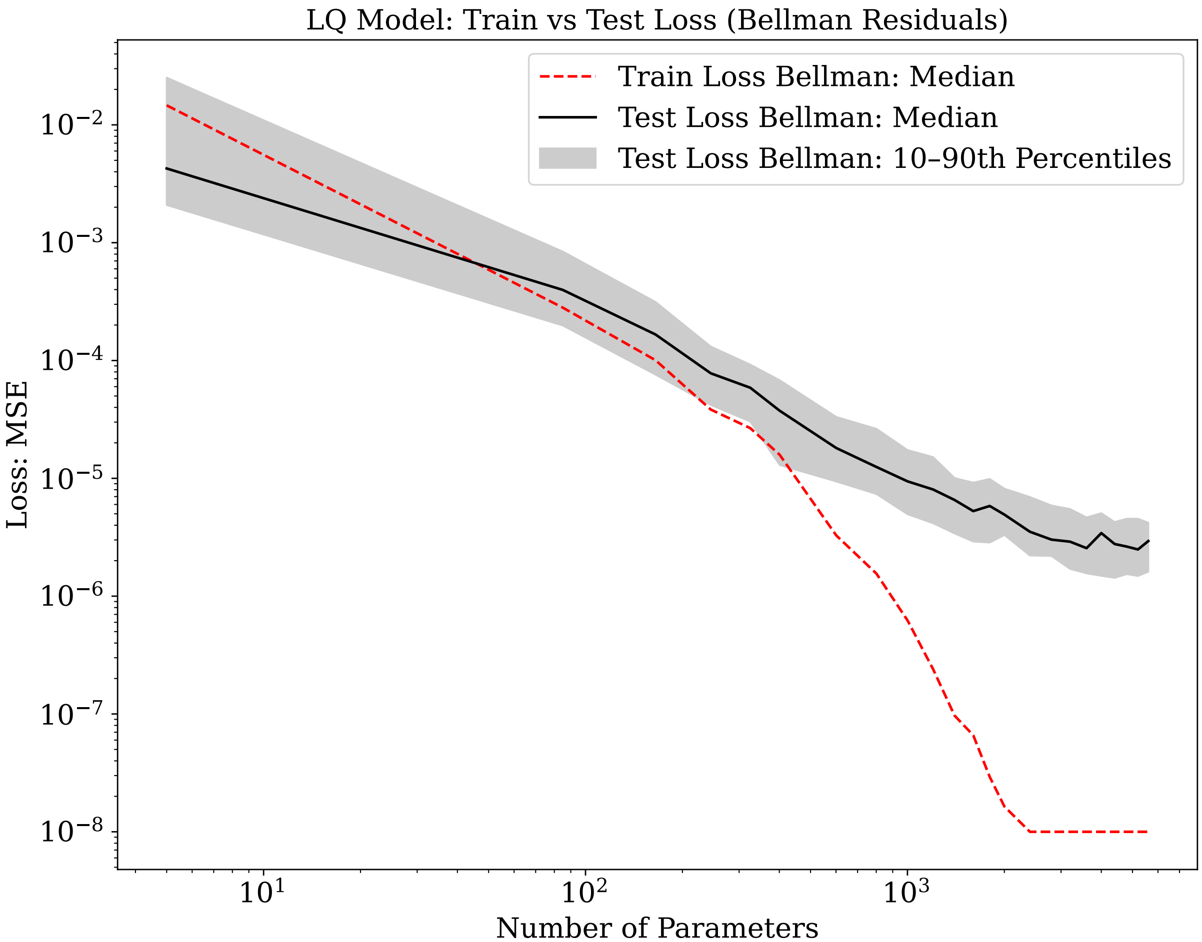

Figure 12: Train and test Bellman MSE for the LQ value function network as a function of the number of parameters. The solid line is the median across 50 seeds; the band spans the 10th to 90th percentiles. Both losses decline with width and solutions become more stable, with the across-seed dispersion collapsing as the network is overparameterized.

Figure 13 reports the median and maximum absolute relative error of the NN value function against the closed-form solution along the test path. Narrow networks produce large and highly variable errors across seeds. As the network widens, both statistics fall and the across-seed band collapses, mirroring the pattern seen for the policy function.

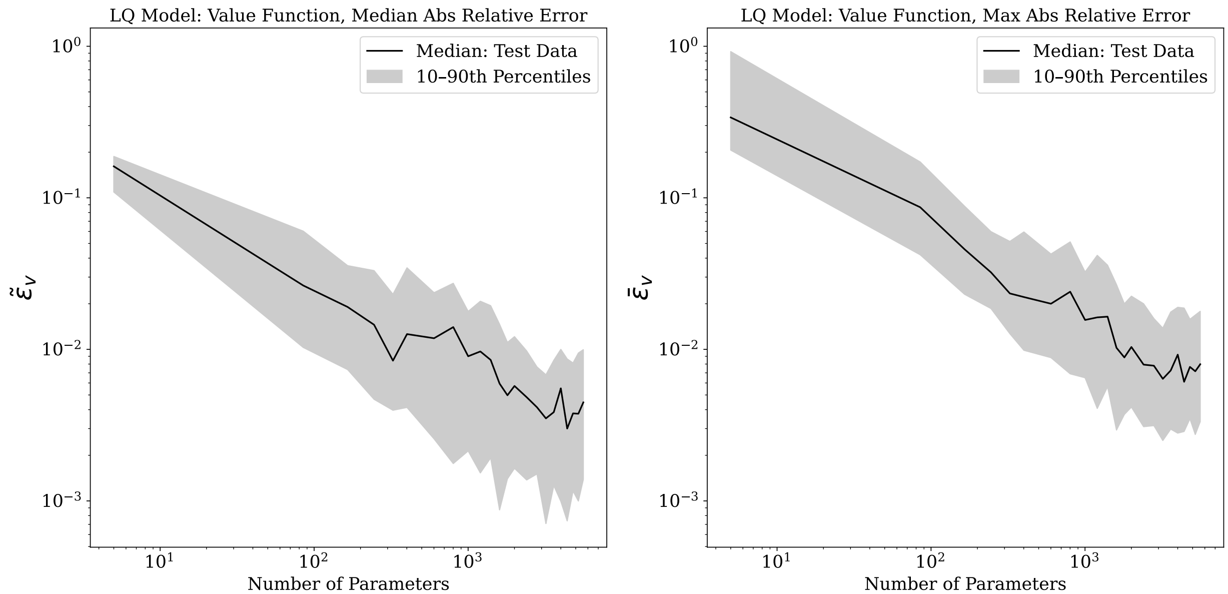

Figure 13: LQ model: Median (left) and maximum (right) absolute relative error of the NN value function versus the closed-form solution as a function of the number of parameters. Solid line: median across 50 seeds; band: 10th–90th percentiles. Both statistics fall and solutions become more stable, with cross-seed dispersion declining sharply as the network widens.

C Robustness

C.1 LQ Regulator: Alternative Activation Functions

The baseline LQ policy results in Section 3.1 use ReLU activations. Here we verify that the main findings are robust to two alternative activation functions: sigmoid and Leaky ReLU.

Activation functions. The *sigmoid* activation is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

which maps $\mathbb{R}$ to $(0, 1)$ and is smooth everywhere. The *Leaky ReLU* activation is defined as

$$\text{LeakyReLU}(z) = \begin{cases} z & \text{if } z \geq 0, \\ \alpha z & \text{if } z < 0, \end{cases} \tag{5}$$

where $\alpha > 0$ is a small slope (we use the PyTorch default $\alpha = 0.01$). Unlike standard ReLU, Leaky ReLU allows a small gradient for negative inputs, which avoids the dying-neuron problem.

Results. Figure 14 reports train and test Euler-residual loss and the median absolute relative error $\tilde{\varepsilon}_u$ for both activation functions, following the same protocol as the baseline. For both activations we vary hidden units from $H = 1$ to $H = 7$ (4 to 22 parameters), the same range as the ReLU baseline, since the generalization gap closes within this range. The patterns closely mirror the ReLU baseline: accuracy improves with width and the across-seed dispersion collapses as the network is overparameterized.

These results confirm that the accuracy and stability blessing documented in the main text is not an artifact of the ReLU activation function. The paper's central message, that overparameterization improves both the accuracy of the NN solution and the reliability across random initializations, is robust to the choice of activation function.

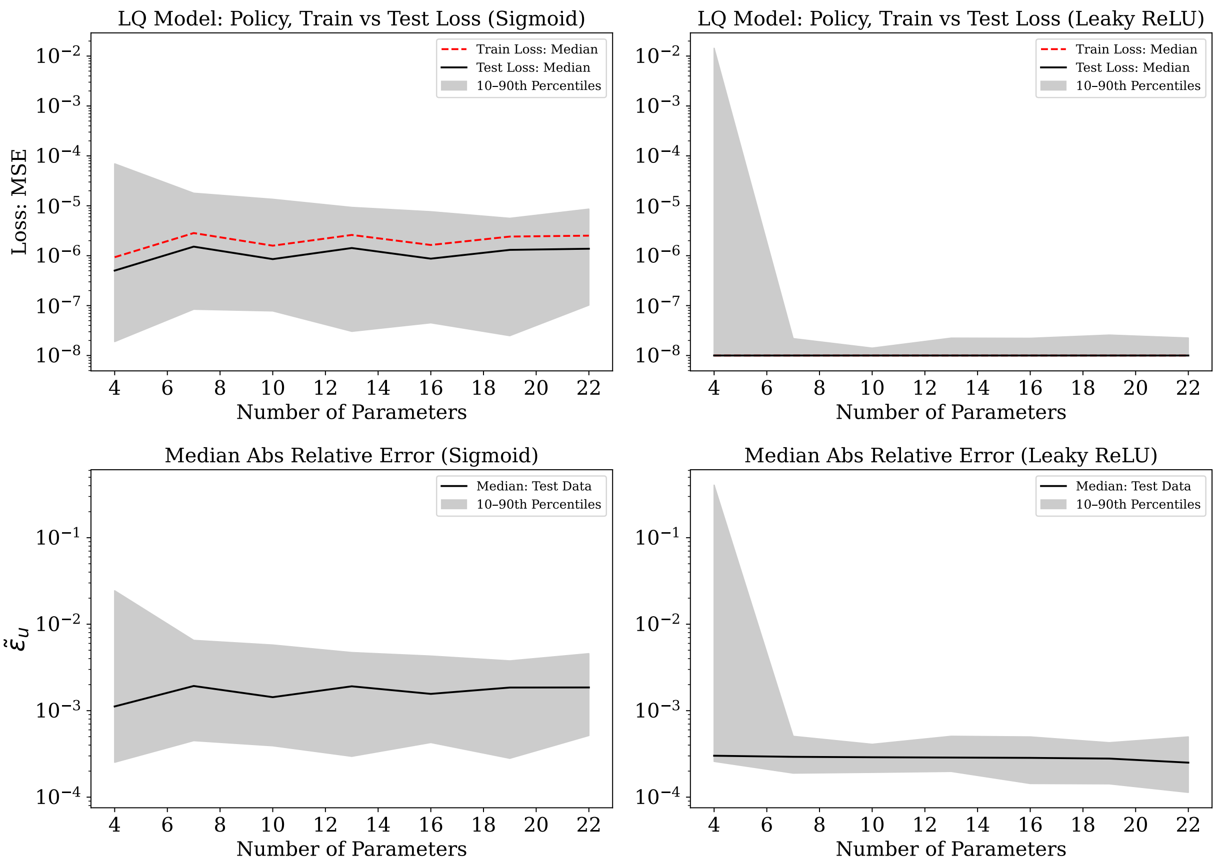

Figure 14: LQ model: Train and test Euler-residual MSE (top row) and median absolute relative error $\tilde{\varepsilon}_u$ (bottom row) for Sigmoid (left column) and Leaky ReLU (right column) policy networks, as functions of the number of parameters. Solid lines are medians across 50 seeds; shaded bands span the 10th to 90th percentiles. Both activation functions replicate the main finding: as the network widens, solutions become more accurate and more stable, with across-seed dispersion collapsing in the overparameterized regime.

C.2 McCall Job Search Model: Deep Learning Robustness

The baseline McCall results in Section 3.2 use a one-hidden-layer network with ReLU activations. Here we verify that the main findings extend to a deeper, two-hidden-layer architecture with ReLU and Leaky ReLU activations (defined in equation 5 of Appendix C.1).[1]

[1]We do not report results for the sigmoid activation for this model. The McCall value function has a kink at the reservation wage $\bar{w}$, arising from the threshold structure of the optimal policy. ReLU and Leaky ReLU, being piecewise-linear activations, are a natural fit for approximating such functions. Sigmoid can in principle approximate a kink, but it is not a natural choice for this problem.

Architecture. We use the two-hidden-layer network defined in equation (2). We increase the number of hidden units H across the grid $H \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 16, 24, 32, 64, 128\}$, yielding between 6 and 16,897 parameters. All other training details follow the baseline.

Results. Figure 15 reports train and test Bellman-residual MSE (top row) and median absolute relative error $\tilde{\varepsilon}_v$ (bottom row) for ReLU and Leaky ReLU. Both activation functions replicate the main findings: as the network widens, the solution becomes more accurate and more stable, with across-seed disagreement collapsing in the overparameterized regime. The results are quantitatively similar across the two activations.

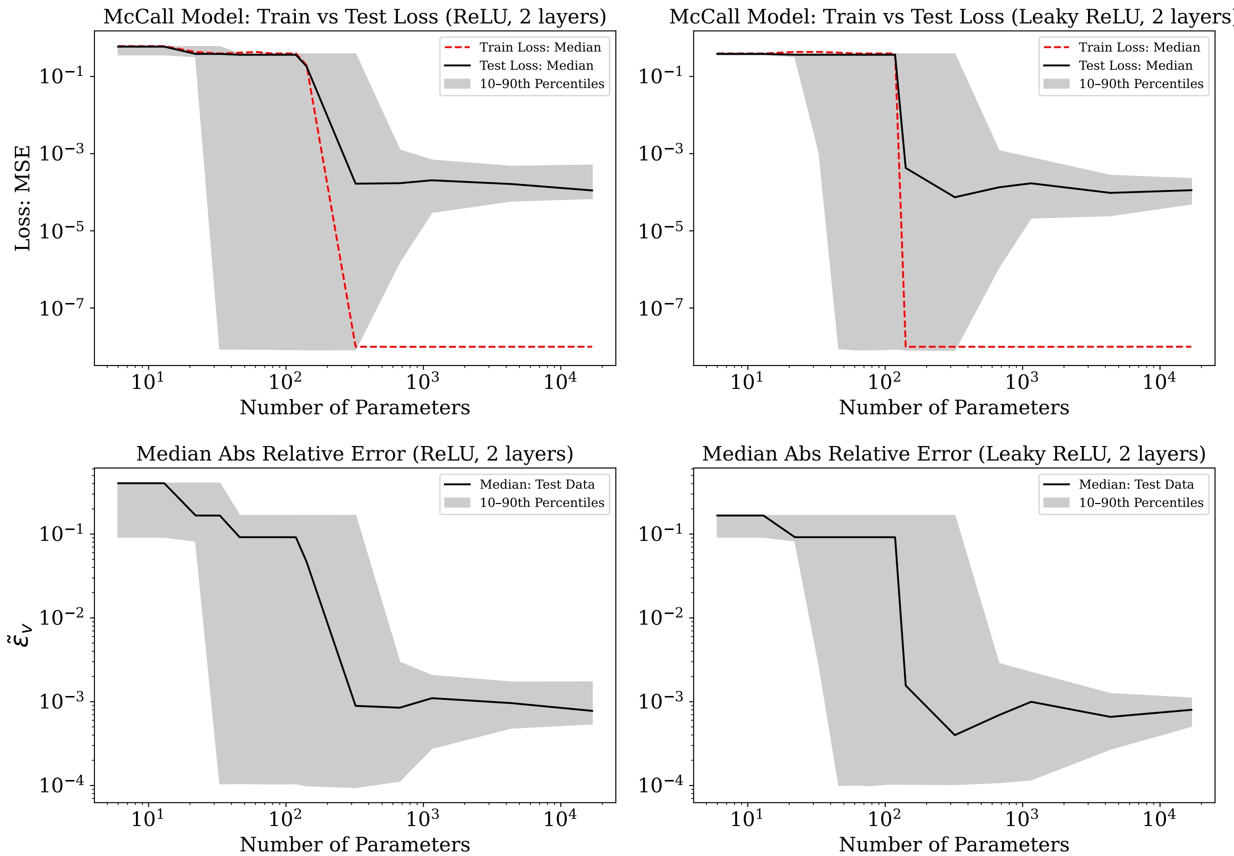

Figure 15: McCall model: Train and test Bellman-residual MSE (top row) and median absolute relative error $\tilde{\varepsilon}_v$ (bottom row) for two-hidden-layer networks with ReLU (left column) and Leaky ReLU (right column) activations, as functions of the number of parameters. Solid lines are medians across 50 seeds; shaded bands span the 10th to 90th percentiles. Both activation functions replicate the main finding: as the network widens, the solution becomes more accurate and more stable, with across-seed dispersion collapsing in the overparameterized regime.

C.3 Real Business Cycle Model: Deep Learning Robustness

The baseline RBC results in Section 3.3 use a one-hidden-layer network with ReLU activations. Here we verify that the main findings extend to a deeper, two-hidden-layer architecture with three alternative activation functions: ReLU, Leaky ReLU, and Sigmoid.

Setup. The architecture follows the two-hidden-layer specification defined in equation 2, applied to the two-dimensional RBC state (k_t, z_t). We vary hidden units over the grid $H \in$

$\{1, 2, \ldots, 10, 16, 24, 32, 64, 128\}$, yielding between 7 and 17,025 parameters, with 50 seeds per configuration. All other training details follow the baseline.

Results. Figure 16 reports the train and test Euler-residual loss and the median absolute relative error $\tilde{\varepsilon}_k$ for all three activation functions. The pattern is consistent across activation functions: both loss and relative errors decline as the network widens, and the across-seed band narrows substantially as the network becomes overparameterized. ReLU and Leaky ReLU perform near-identically; Sigmoid delivers comparable accuracy with a similarly tight band, confirming the blessings are not specific to piecewise-linear activations.

Taken together, these results confirm that the accuracy and stability blessing documented in the main text for the RBC model holds across architectures and activation functions, even for a stochastic model with no closed-form solution.

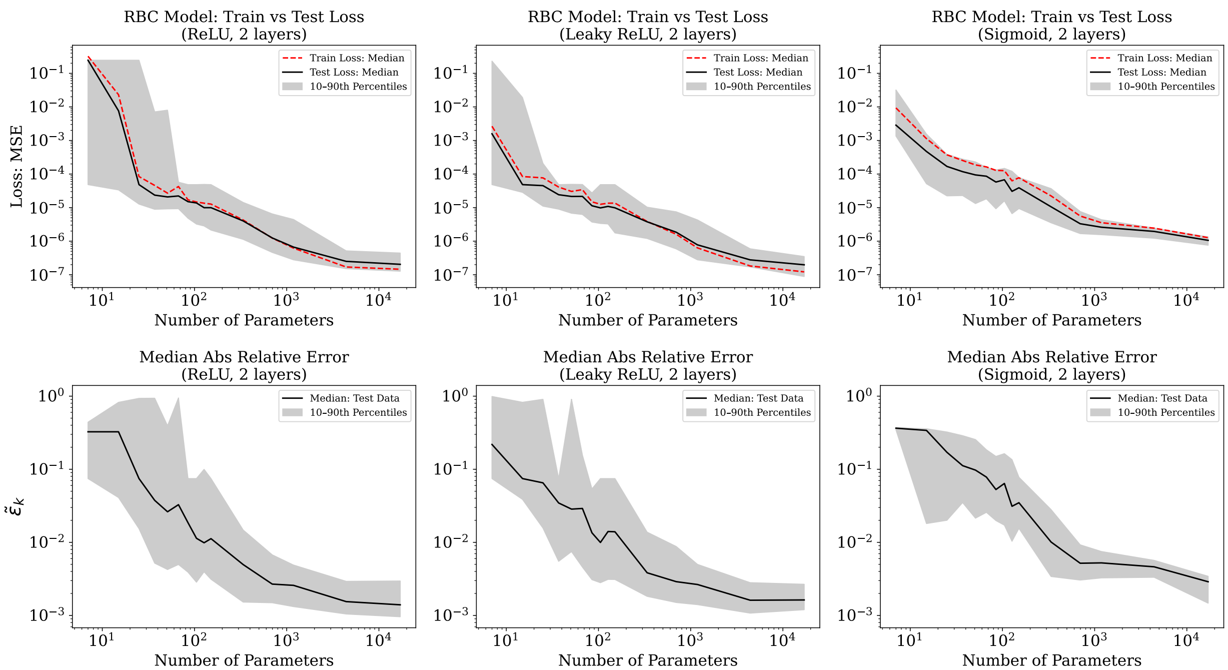

Figure 16: RBC model: Train and test Euler-residual MSE (top row) and median absolute relative error $\tilde{\varepsilon}_k$ (bottom row) for two-hidden-layer networks with ReLU (left column), Leaky ReLU (center column), and Sigmoid (right column) activations, as functions of the number of parameters. Solid lines are medians across 50 seeds; shaded bands span the 10th to 90th percentiles. All three activation functions replicate the main finding: as the network widens, the solution becomes more accurate and more stable, with across-seed dispersion collapsing in the overparameterized regime.