Informed Actor–Critic Method for Optimal Control in Economic Dynamics

Kenneth L. Judd[c], Bo Li[a], Serguei Maliar[b,c,*], Peiyuan Zhang[a]

[a] *School of Economics, Peking University, Beijing, China*
[b] *Department of Economics, Santa Clara University, , USA*
[c] *Hoover Institution, Stanford University, , USA*

Abstract

This paper applies deep reinforcement learning to high-dimensional optimal control in dynamic economic models. We introduce an informed actor–critic (iAC) framework that embeds structural equations and constraints directly into the policy optimization loop. Unlike standard model-free reinforcement learning algorithms that treat the economy as a black box—relying entirely on data-driven simulation—the iAC method leverages analytical economic transitions. Computational applications to benchmark consumption–savings, Krusell–Smith, and overlapping-generations economies show that iAC reduces the need for exploration noise and provides a precise, scalable, and sample-efficient framework for complex macroeconomic control problems.

Keywords: Reinforcement learning, Dynamic Programming, Deep Learning, Heterogeneous agents, Overlapping Generations, Curse of Dimensionality, Computational Macroeconomics
JEL classification: C61, C63, C45, C68, C88, D83

1. Introduction

Reinforcement learning (RL) has demonstrated remarkable success across a diverse array of complex applications, including plasma fusion, robotics, locomotion, large language models (LLMs), and self-driving cars. Notably, RL is designed to solve the exact same fundamental class of problems as optimal control in economics.

*This research was supported by NSF grant SES-1949430, National Natural Science Foundation of China (No. 72450002) and Beijing Philosophy and Social Science Foundation (No. 24DTR024).
*Corresponding author: maliars@stanford.edu

Both fields rely on a dynamic feedback loop traditionally structured as

$$\textbf{State} \rightarrow \textbf{Control} \rightarrow \textbf{Reward} \rightarrow \textbf{New state}.$$

In robotics, the state represents the physical body position, the control consists of electrical impulses sent to the rotors, the reward measures the success or failure of a movement, and the new state is the resulting physical position. Conversely, in an economic model, this loop consists of the current-period capital stock, the consumption choice, consumer utility, and the next-period capital stock, respectively.

Can economists leverage these powerful RL methodologies to achieve breakthroughs in their own optimal control problems? Our answer is a definitive yes, but standard RL frameworks must first be adapted to fit the highly specific characteristics of economic models.

The core challenge lies in the trade-off between model-free and model-based frameworks. The most powerful RL algorithms are typically model-free, treating environments as "black boxes" to learn optimal behaviors entirely via trial and error. For example, in robotics, it is often analytically infeasible to derive an exact formula for how a robot's movement is affected by the complex interplay of inertia, momentum, gravity, friction, and changing floor surfaces; thus, the robot must experiment and learn directly from simulated or real-world data. However, this pure black-box approach stumbles in economics because simulated data is inherently noisy, and the convergence rate of model-free methods is notoriously low.

By contrast, optimal control in economics is traditionally model-based: the environment is not a black box but is instead explicitly defined by mathematical equations—such as budget constraints, market-clearing conditions, state transitions, and shock processes. Applying a purely model-free approach to economics is highly inefficient, as it essentially forces us to discard the model's known structural equations only to poorly relearn them from noisy simulation data. By leveraging this structural knowledge, economists can implement solution methods that are highly sample-efficient, guarantee convergence, and produce accurate, reliable solutions; value function iteration via discretization is a canonical example. However, the primary limitation of these traditional model-based methods is that they scale poorly, remaining tractable only for relatively small applications.

In this paper, we merge model-based and model-free RL approaches for optimal control in economic dynamics. Our framework is built around the actor-critic (AC) architecture, which separates the learning of the policy function (the actor) from that of the value function (the critic) to achieve greater efficiency and stability. Standard AC methods in the RL literature are entirely model-free, meaning both the actor and the critic learn strictly from raw simulated or real-world data. To adapt this

architecture to economic optimal control, we modify the standard AC framework to include a model-informed critic. This component provides effective, structural feedback to the actor via a gradient penetration mechanism rooted directly in the model's governing equations. Leveraging the environment's exact structural equations instead of relying on noisy raw data significantly increases both the accuracy and numerical stability of the AC method without increasing its computational cost. We term this novel approach the informed Actor-Critic (iAC) method.

We implement the informed Actor-Critic (iAC) method using deep neural networks to ensure scalability across high-dimensional state spaces. Crucially, we replace the standard Q-learning framework characteristic of the reinforcement learning (RL) literature with V-learning, which we find to be a significantly more effective choice for the studied class of economic dynamics. Furthermore, we establish formal analytical results for the proposed iAC framework, providing a rigorous proof of convergence, precise estimates of the convergence rates, and error bounds for the canonical consumption–savings problem. These formal results highlight the enhanced data efficiency and structural stability of our gradient penetration mechanism and informed model-based critic compared to a conventional, model-free AC critic that learns exclusively from raw simulation data.

Aside from the AC framework, the reinforcement learning literature features a rich variety of alternative computational methods. To evaluate their utility for economic applications and to inform the structural design of our iAC approach, we implemented a comprehensive collection of model-free and model-based RL methods within the Q-learning framework (Watkins and Dayan, 1992) in the context of the standard consumption–savings problem. Monte Carlo rollout, introduced by Tesauro and Galperin (1996) and discussed by Bertsekas and Tsitsiklis (1996), as well as REINFORCE (Williams, 1992) and PPO (Schulman et al., 2017), can produce highly accurate solutions without explicitly relying on the Bellman equation; however, their heavy dependency on repeated local simulations renders them computationally prohibitive for large-scale economic dynamics. Dyna-Q, pioneered by Sutton (1991), integrates two powerful mechanisms that facilitate and stabilize learning—namely, experience replay and offline planning ("dreaming"); nevertheless, it remains a grid-based, tabular method that cannot scale effectively to higher dimensions. Deep Q-Networks (DQN), introduced by Mnih et al. (2015), parameterize the value function using deep neural networks; however, the policy function must still be discretized over a grid format, which inherently restricts the precision of the resulting policy solutions. Deep Deterministic Policy Gradient (DDPG), introduced by Lillicrap et al. (2015), accommodates continuous action and state spaces through an efficient AC architecture; however, it is highly susceptible to getting trapped in local optima if

the critic is insufficiently trained. *Soft Actor-Critic (SAC), introduced by Haarnoja et al. (2018), incorporates entropy regularization into a stochastic actor–critic framework. By balancing expected returns with policy entropy, SAC encourages the agent to explore a broader range of actions and may reduce the risk of becoming trapped in poor local solutions.* Finally, the all-in-One (AiO) Expectation Method proposed by Maliar et al. (2021) combines policy and value function learning into a single objective, replacing standard Q-learning (Watkins and Dayan, 1992) with V-learning; however, simultaneously optimizing both functional approximations under this unified setup frequently induces severe training instability.

In economics, Grüne et al. (2015) introduce nonlinear model predictive control (NMPC) as another reinforcement learning method. NMPC approximates infinite-horizon problems with finite-horizon ones.

In light of the preceding taxonomy, our iAC method can be conceptualized as a DDPG framework augmented with a gradient penetration mechanism designed to guarantee numerical stability and learning efficiency. While our analysis indicates that other RL stabilizers could be deployed alongside or as alternatives to gradient penetration, they introduce distinct computational trade-offs. In this way, experience replay and offline learning (dreaming) break the temporal correlation between specific random simulation draws and gradient updates, thereby preventing the model from falling into local optima traps. Furthermore, entropy objectives incorporating exploration and exploitation phases force a more exhaustive search of the policy space to avoid premature convergence. Although these supplementary stabilizing mechanisms are highly effective in pure model-free environments, integrating them into economic models significantly increases algorithmic complexity, computational overhead, and memory consumption. In our applications, such additional stabilizers proved unnecessary; the gradient penetration mechanism alongside a simple DDPG framework provided highly accurate, stable, and computationally lean solutions across all economic configurations examined.

We apply the iAC method to a suite of prominent heterogeneous-agent models, including the canonical Krusell–Smith economy (Krusell and Smith, 1998), extensions incorporating indivisible labor, and large-scale Overlapping Generations (OLG) environments. Across all setups, the iAC method consistently yields highly reliable and accurate solutions, seamlessly tracking high-dimensional wealth distributions and sharp non-linear policy transitions. Notably, our method matches or exceeds the precision of traditional, computationally intensive projection benchmarks while dramatically reducing execution times and eliminating the need for ad-hoc shape-preserving constraints on policy functions. These numerical experiments demonstrate the robust generalizability of our framework, illustrating how a uniform, structurally

informed training logic can be seamlessly adapted across structurally diverse classes of quantitative macroeconomic models.

The iAC method presented here is related to both earlier computational methods in economics and more recent methods based on machine learning tools. The optimal control problems in economics are typically analyzed using dynamic programming (DP). A well-known DP method is value function iteration via discretization which produces accurate solutions and guarantees convergence but is subject to the severe curse of dimensionality. To achieve scalability, computational economists developed DP approaches based on functional approximation and deterministic integration (Judd, 1998), low-discrepancy sequences (Rust, 1997), Smolyak-style sparse grids (Krueger and Kubler, 2004; Brumm and Scheidegger, 2017), and alternative iterative schemes such as the endogenous grid method (EGM) (Carroll, 2006) and envelope condition method (ECM) (Maliar and Maliar, 2013); see Maliar and Maliar (2014) for a review. For heterogeneous-agent economies, there are methods that use low-dimensional distributional approximations (Schaab, 2020) and methods that approximate high-dimensional distributions with low-dimensional moments (Krusell and Smith, 1998; Den Haan, 1997; Fernández-Villaverde et al., 2023).

In fact, early simulation-based approaches in economics developed by Smith (1993), Den Haan and Marcet (1990), Maliar and Maliar (2005), Stachursky (2009), and Judd et al. (2011) closely resemble reinforcement learning in data science; in particular, Maliar and Maliar (2014) show the tractability of their earlier value-function-iteration-by-simulation method for high-dimensional problems. Furthermore, the application of neural networks to economics dates back to Duffy and McNelis (2001), who construct the PEA method with a neural network. Notably, an early and innovative (but unnoticed and unpublished) working paper by Jirnyi and Lepetyuk (2011) preceded the current machine learning boom, introducing a prescient framework to solve the Bellman equation for the canonical Krusell–Smith economy by synthesizing tabular approximation, approximate dynamic programming, nonparametric statistics, and nearest-neighbor clustering. More recently, Yang et al. (2025) introduce a structural reinforcement learning method based on policy-gradient learning.

At the SCE 2018 meeting, Pablo Winant presented an early version of Maliar et al. (2021), showing how, for the consumption–savings problem, to convert the three fundamental objects of economic dynamics—lifetime reward (REINFORCE), Euler equation, and Bellman equation—into objective functions for deep learning that are minimized in TensorFlow by means of an all-in-one expectation operator.[1]

[1]This presentation is recorded and available at https://www.youtube.com/watch?v=u7_CdytTEe8.

Fernández-Villaverde et al. (2023) illustrate the applicability of neural networks to two-dimensional interpolation, and the papers by Maliar et al. (2021), Azinovic et al. (2022), and Han et al. (2026) illustrate the power of deep learning for solving high-dimensional applications; see Fernández-Villaverde (2025) for a recent review. There are also papers by Maliar and Maliar (2022) and Li et al. (2026) that use deep learning to solve discrete-continuous dynamic choice problems by treating discrete economic decisions as categorical tasks optimized end-to-end via deep neural networks.

Nonetheless, as Cai and Judd (2010, 2013) show, functional approximations such as polynomials tend to produce wiggly, inaccurate solutions for value and policy functions. Attaining higher accuracy requires imposing either additional constraints of monotonicity and concavity, or employing Hermite-type interpolation where the value function is approximated together with its derivatives; see Cai and Judd (2014) for a comprehensive review. However, these shape-preserving and high-order approaches are computationally expensive; thus, tractability remains a persistent barrier for large-scale optimal control applications in economics. This issue led to numerical instability of the Bellman iteration via the all-in-one expectation method in Maliar et al. (2021). The separation of policy and value learning in the iAC method allowed us to correct the numerical instability of the Bellman method in Maliar et al. (2021) and to solve the optimal control problems accurately without imposing shape-preserving restrictions, while the deep-learning basis makes the iAC method tractable for very large applications.

The rest of the paper is as follows: Section 2 formulates the dynamic problem and introduces the iAC method in the benchmark consumption–savings setting. Section 3 establishes the formal results of convergence and accuracy bounds in the context of the canonical consumption–savings problem. Section 4 reviews the RL methods that motivate the iAC design. Section 5 applies the same training logic to heterogeneous-agent applications. Section 6 concludes.

2. Informed Actor–Critic Method

In this section, we formulate the studied class of dynamic economic models, introduce the iAC solution method and illustrate its application in a canonical consumption–savings model.

2.1. A class of dynamic economic problems

We study a broad class of dynamic economic problems that can be written in recursive form. At each date $t \geq 0$, the agent chooses a control variable a_t given the

vector of endogenous and exogenous states x_t and z_t, respectively. The goal is to solve

$$\max_{\{a_t\}_{t\geq 0}} \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t u(x_t, z_t, a_t), \tag{1}$$

subject to feasibility conditions

$$a_t \in \mathcal{A}(x_t, z_t) \text{ and } h(x_t, z_t, a_t) \geq 0, \tag{2}$$

the law of motion for endogenous variables

$$x_{t+1} = G(x_t, z_t, a_t, z_{t+1}), \tag{3}$$

and the process for exogenous states which is represented either as a finite-state Markov chain or as a continuous-state Markov process such as

$$\ln z_{t+1} = \rho \ln z_t + \sigma \varepsilon_{t+1}, \qquad \varepsilon_{t+1} \sim \mathcal{N}(0, 1). \tag{4}$$

where $\beta \in [0, 1)$, $\rho \in (-1, 1)$ and $\sigma > 0$ are the parameters. This class contains canonical consumption–savings problems—including multi-asset portfolios and endogenous labor supply—as well as advanced heterogeneous-agent models where aggregate states and equilibrium variables are embedded in the state vector x_t.

2.2. iAC method

The recursive structure of the problem ((1))–((2)) implies the Bellman equation

$$V(s) = \max_{a \in \mathcal{A}(s)} \{u(s, a) + \beta \, \mathbb{E}\left[V(s') \mid s, a\right]\}, \tag{5}$$

where $s \equiv (x, z)$ is a vector of the economy's endogenous and exogenous states. This abstract state notation is used in the formal results below. In the model-specific sections, the state can be assets, cash-on-hand, wealth, or a larger vector, as long as the notation is kept consistent within each model. The compact-domain conditions behind the formal Bellman arguments are summarized in Assumption 1, and the contraction argument is given in Lemma 1. We parameterize the policy and value functions using deep neural networks π_θ and V_ω, interpreted as the actor and critic, respectively. The actor proposes an action $a = \pi_\theta(s)$, while the critic evaluates the expected long-term value of those actions $V_\omega(s)$. Proposition 1 formalizes why this value-only critic can still induce the one-step action-value object needed for policy improvement. The method operates through two alternating passes. In the notation below, "action" includes the model-specific parameterization used in

the implementation. For example, the actor may output consumption, next-period assets, a consumption rate, or a savings rate. The economic action is then obtained through the model's budget equation and feasibility map. Formally, one can write $b_\theta(s) = \pi_\theta(s)$ and $(c_\theta(s), x'_\theta(s)) = \Psi(s, b_\theta(s))$. To keep the main notation compact, we write this choice simply as $a = \pi_\theta(s)$.

Forward pass (critic update):. The critic aims to minimize an actor-conditioned Bellman residual with respect to its target V_ω on a set of points $s_i \sim \mathbf{S}$. That is, conditional on the frozen actor, the critic fits the left-hand side of the Bellman equation to the one-step right-hand side generated by that actor.

$$\mathcal{L}_V(\omega) = \mathbb{E}_{s_i \sim \mathbf{S}} \left[(V_\omega(s_i) - y_i)^2 \right], \tag{6}$$

taking the frozen copies of actor and critic $\overline{\theta}$ and $\overline{\omega}$ as given

$$y_i \equiv u\big(s_i, \pi_{\overline{\theta}}(s_i)\big) + \beta\, \mathbb{E}_{\varepsilon'} \left[V_{\overline{\omega}}(s'_i) \right], \qquad s'_i = f\big(s_i, \pi_{\overline{\theta}}(s_i), \varepsilon'\big). \tag{7}$$

Here $V_{\overline{\omega}}$ denotes either a soft target critic or a stop-gradient copy of the current critic, depending on the implementation.

Backward pass (actor update):. The actor aims to maximize the expected value of its actions. This is the policy-improvement part of the Bellman maximization: the critic fits the Bellman right-hand side generated by a given actor, while the actor changes the action inside that right-hand side.

$$\mathcal{J}_\pi(\theta) = \mathbb{E}_{s_i \sim \mathbf{S}} \left[u\big(s_i, \pi_\theta(s_i)\big) + \beta\, \mathbb{E}_{\varepsilon'} \left[V_{\overline{\omega}}(s'_i) \right] \right], \qquad s'_i = f\big(s_i, \pi_\theta(s_i), \varepsilon'\big), \tag{8}$$

taking the frozen copy of the critic $\overline{\omega}$ as given. The gradient identity behind this actor update is stated in Proposition 3.

Learning process. The learning process takes place as a sequence of alternating forward and backward passes:

$$\min_\omega \mathcal{L}_V(\omega), \qquad \max_\theta \mathcal{J}_\pi(\theta), \tag{9}$$

subject to transition equations, feasibility conditions, and equilibrium restrictions that define the economic model itself.

2.3. The design of the iAC method

The iAC method uses three ingredients. First, it uses an actor–critic architecture. This separates policy choice from value evaluation. Second, it uses simulation and neural networks. This helps in high-dimensional models. Third, it uses model-based gradient propagation when the relevant model equations are known and cheap to evaluate.

i) Why the actor–critic architecture. The actor–critic architecture separates policy choice from value evaluation. This helps avoid two problems. Pure value-based methods, such as Q-learning, usually need grids over states and actions. They can be accurate, but they scale poorly. Pure policy-gradient methods, such as REINFORCE, work with simulated trajectories. They scale better, but their gradients can have high variance. Actor–critic methods combine the two ideas. The actor learns a policy. The critic gives a value estimate that is less noisy than a pure Monte Carlo return.

ii) Why stochastic simulation and deep learning?. Classical interpolation methods work on grids. They can be accurate, but the grid grows quickly when the state space expands. Simulation avoids this problem by focusing on the part of the state space that is actually visited by the model.[2] Neural networks are useful on this kind of simulated data because the points do not need to lie on a rectangular grid. They also provide global approximations and can fit nonlinear functions. We can implement these approximations with standard software such as TensorFlow and PyTorch.

iii) Why model-based gradient propagation?. Actor–critic is often introduced in the RL literature as a model-free method. In standard DDPG, the actor is updated through a learned state-action critic:

$$J^{MF}(\theta) = \mathbb{E}_s \left[Q_\omega(s, \pi_\theta(s)) \right], \qquad \nabla_\theta J^{MF}(\theta) = \mathbb{E}_s \left[D_\theta \pi_\theta(s)^\top \nabla_a Q_\omega(s,a) \big|_{a=\pi_\theta(s)} \right]. \tag{10}$$

The transition from s to s' enters through sampled data in the replay buffer. The actor gradient passes through the learned object $Q_\omega(s,a)$. It does not directly use the economic transition equation that generated s'.

In economic applications, useful model structure is often known. The iAC implementation uses this structure in a one-step Bellman objective. The next state is written explicitly as

$$s' = f(s, \pi_\theta(s), \varepsilon'). \tag{11}$$

The actor objective becomes

$$J^{iAC}(\theta) = \mathbb{E}_{s,\varepsilon'} \left[r(s, \pi_\theta(s)) + \beta V_{\overline{\omega}}(f(s, \pi_\theta(s), \varepsilon')) \right]. \tag{12}$$

Here r is the one-period reward notation standard in reinforcement learning. In the economic dynamic programs studied below, this reward is the one-period payoff, so

[2] Judd et al. (2011) argue that the ergodic set is an infinitesimal fraction of the rectangular grid in high-dimensional applications.

$r(s,a) = u(s,a)$. Holding the target critic fixed, the actor gradient is

$$\nabla_\theta J^{iAC}(\theta) = \mathbb{E}_{s,\varepsilon'}\left[D_\theta \pi_\theta(s)^\top \left\{ r_a(s,a) + \beta f_a(s,a,\varepsilon')^\top \nabla_{s'} V_{\overline{\omega}}(s') \right\} \Big|_{a=\pi_\theta(s)} \right]. \tag{13}$$

The extra term $f_a(s,a,\varepsilon')^\top \nabla_{s'} V_{\overline{\omega}}(s')$ is the part that model-free DDPG does not use directly. This gives a model-based gradient path

$$\pi_\theta(s) \to s' \to V_{\overline{\omega}}(s')$$

inside the actor update.

In the benchmark consumption–savings problem, let w denote cash-on-hand. If the actor chooses consumption $c_\theta(w,z)$, next-period assets are $k' = w - c_\theta(w,z)$ and next-period cash-on-hand is $w' = Rk' + y(z')$. The corresponding actor derivative is

$$\nabla_\theta J^{iAC}(\theta) = \mathbb{E}\left[\left\{ u_c(c_\theta(w,z)) - \beta R\, \mathbb{E}_{z'|z} V_w(w',z') \right\} \nabla_\theta c_\theta(w,z) \right]. \tag{14}$$

Thus the actor is not trained only from a noisy realized return or from a black-box Q approximation. It receives the derivative of the continuation value with respect to the current action through the known transition map. This is one way the iAC method uses model information inside the actor–critic architecture. Equation (14) is the consumption–savings instance of the general gradient-penetration identity in Proposition 3; its Euler interpretation is given in Corollary 2. The formal error comparison in Proposition 4 clarifies the role of this path: the actor no longer relies on a learned state-action-gradient channel alone, and smaller actor-gradient errors imply smaller stationarity neighborhoods for the actor update.

iv) Why V-learning. The benchmark also shows that V-learning is simpler and cheaper than Q-learning in the studied class of problems. In problems with a discrete set of actions, Q-learning has the advantage that it allows us to recover the policy function with a straightforward maximum operator. In our analysis, the set of actions is continuous, so finding a policy function from the Q-function involves a non-trivial numerical optimization problem.

v) Why one-step Bellman training?. The distinction is also important relative to simulation-heavy methods such as Han et al. (2026). Those methods can work well, but they rely more on multi-period simulation. The iAC method is organized around one-step Bellman targets. The target is current reward plus next-period continuation value. The known transition map gives the next state. This lets us train directly from the local Bellman recursion.

2.4. Implementation in a consumption–savings problem

We describe the implementation and performance of a baseline iAC method in a canonical consumption–savings problem.

2.4.1. The model

We consider a one-agent version of the model (1)–(4) in which the state is cash-on-hand w_t and the agent faces a trade-off between consumption and savings:

$$\max_{\{c_t,k_{t+1}\}_{t\geq 0}} \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t u(c_t) \tag{15}$$

subject to the budget constraint $c_t + k_{t+1} = w_t$, the cash-on-hand transition

$$w_{t+1} = Rk_{t+1} + y(z_{t+1}),$$

the shock process $\ln z_{t+1} = \rho \ln z_t + \sigma\varepsilon_{t+1}$ with $\varepsilon_{t+1} \sim \mathcal{N}(0,1)$, and the feasibility conditions $k_{t+1} \geq \underline{k}$ and $c_t \geq \underline{c} > 0$, where $\beta \in [0,1)$, $\rho \in (-1,1)$ and $\sigma > 0$ are the parameters. The Bellman equation is

$$V(w,z) = \max_{k' \in \mathcal{K}(w)} \left\{ u(w-k') + \beta\, \mathbb{E}\big[V(Rk' + y(z'), z') \mid z\big] \right\}, \tag{16}$$

where $\mathcal{K}(w) = \{k' \geq \underline{k} : w - k' \geq \underline{c}\}$ is the feasible savings set. The Bellman equation is written with next-period assets as the choice variable and cash-on-hand as the state. In the neural implementation, the same feasible choice is parameterized by a bounded consumption ratio, so $c_\theta(w,z) = w\,\psi_\theta(w,z)$ after the feasibility adjustment. Corollary 2 records the corresponding interior Euler interpretation of this actor objective.

2.4.2. Pseudocode

Below, we show pseudocode for solving the consumption–savings problem.

Algorithm 1 Benchmark implementation of the iAC method

Initialize actor parameters θ
Initialize value critic parameters ω
Initialize frozen actor parameters $\overline{\theta} \leftarrow \theta$
Initialize target critic parameters $\overline{\omega} \leftarrow \omega$
repeat
    Sample a batch of replayed states (and stored shock draws, if used) $\{s_i\}_{i=1}^{N}$
    for each state s_i **do**
        Actor outputs an action parameter and the feasibility map converts it into economic choices

$$b_i = \pi_{\overline{\theta}}(s_i), \qquad (c_i, x_i') = \Psi(s_i, b_i)$$

        Use the known transition equation and the chosen integration rule to form the continuation target

$$\widehat{\mathcal{C}}_i = \mathbb{E}_{\varepsilon'}\left[V_{\overline{\omega}}(f(s_i, x_i', \varepsilon'))\right]$$

        Construct Bellman-consistent target

$$y_i = u(c_i) + \beta\, \widehat{\mathcal{C}}_i$$

    end for
    Update critic by minimizing

$$\mathcal{L}_V(\omega) = \frac{1}{N}\sum_{i=1}^{N}\big(V_\omega(s_i) - y_i\big)^2$$

    Update actor by maximizing

$$\mathcal{J}_\pi(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[u(c_{\theta,i}) + \beta\, \mathbb{E}_{\varepsilon'} V_{\overline{\omega}}(f(s_i, x'_{\theta,i}, \varepsilon'))\right], \quad (c_{\theta,i}, x'_{\theta,i}) = \Psi(s_i, \pi_\theta(s_i))$$

    Apply model-based gradient propagation through $\Psi(\cdot)$ and the known transition function $f(\cdot)$ while holding $V_{\overline{\omega}}$ fixed
    Hard-update frozen actor: $\overline{\theta} \leftarrow \theta$
    Soft-update the target critic: $\overline{\omega} \leftarrow \tau_\omega \omega + (1 - \tau_\omega)\overline{\omega}$
until training losses and policy stabilize

In the benchmark implementation, both the policy and value functions are ap-

proximated with feedforward neural networks that take the current state (w, z) as input. The actor uses two hidden layers with 64 tanh units and a sigmoid output layer to produce a bounded consumption ratio $(\frac{c}{w})$, while the critic uses the same hidden-layer architecture with a linear output to approximate the value function. We train the two networks with Adam, using a learning rate of 10^{-4} for the actor and 10^{-3} for the critic, a batch size of 256, and geometric learning-rate decay to a floor of 10^{-5} over 50,000 gradient steps. The critic is trained by minimizing the squared actor-conditioned Bellman residual, and the actor is trained by maximizing the corresponding one-step Bellman objective through backpropagation of gradients through the known transition law. The formal convergence interpretation of the idealized update is summarized in Proposition 2, while the policy-loss interpretation of the residual and greedy-improvement errors is summarized in Proposition 5. We use two training-state designs in the paper. In the simple consumption–savings benchmark, states can be drawn directly from the computational state space. In the more complex heterogeneous-agent models, training states are generated from simulations under the policy being trained. Network weights are left at the default PyTorch initialization.

2.4.3. Numerical solution

We compare the policy and value functions constructed by the iAC method with those from an accurate VFI reference solution based on a wealth grid of size $n_a = 2000$, an income-shock grid of size $n_z = 100$, and an action grid of size $n_{a'} = 2000$; see Appendix B.1.1 for more details about the reference solution.

Figure 1 shows that the iAC benchmark implementation is close to the reference solution.

Table 1 summarizes the numerical accuracy of the preferred multi-sampling specification.

Table 1: iAC accuracy in the benchmark consumption–savings problem

Object	Error metric	Value
Consumption policy	Mean percentage error	0.27%
Consumption policy	Maximum percentage error	0.72%
Value function	RMSE	1.1×10^{-2}
Value function	Maximum absolute error	3.1×10^{-2}
Value function	Mean percentage error, excluding near-zero values	1.9%

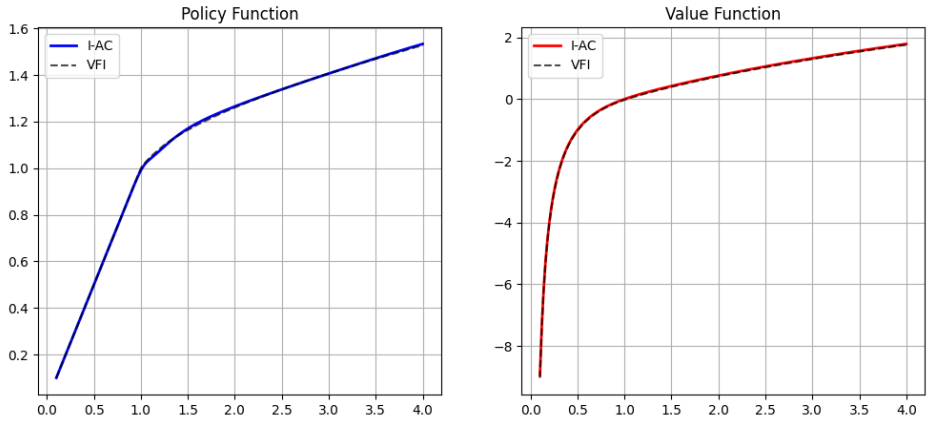


Fig. 1: The iAC method versus accurate VFI in the canonical consumption–savings problem.

3. Formal results

This section states the formal interpretation behind the iAC construction. The detailed proofs are collected in Appendix A. The results are stated for a local one-step improvement operator. This is the object used in the benchmark consumption–savings problem and in the local Bellman validation of the Krusell–Smith benchmarks.

Assumption 1 (Computational dynamic program)**.** The formal results are stated on the compact computational domain used by the numerical algorithms. The state space is compact, the feasible correspondence is nonempty and compact, the one-period payoff is bounded and continuous on the feasible graph, and either the transition map is continuous and maps the computational domain back into itself or the transition kernel is Feller and maps the computational domain back into itself. In models with log or CRRA utility, the computational feasible set imposes a positive consumption floor $c \geq \underline{c} > 0$. The discount factor satisfies $\beta \in (0, 1)$. For gradient statements, the payoff, transition map, critic, and actor are differentiable on the interior of the feasible region, the actor update is computed over a fixed training-state distribution, and differentiation can be interchanged with integration. For policy-distance bounds, the local feasible action set is convex on the region where the bound is applied.

For a policy π, define the actor-conditioned Bellman operator

$$(T^{\pi}V)(s) = u(s, \pi(s)) + \beta E\left[V(f(s, \pi(s), \varepsilon'))\right]. \tag{17}$$

Here $T^{\pi}V$ means that the fixed-policy Bellman operator T^{π} is applied to a candidate continuation value V. Similarly, $T^{\mathrm{loc}}V$ means that the local one-step Bellman improvement operator is applied to V. For local one-step improvement, define

$$(T^{\mathrm{loc}}V)(s) = \max_{a \in A_{\mathrm{loc}}(s)} \{u(s,a) + \beta E\left[V(f(s,a,\varepsilon'))\right]\}. \tag{18}$$

In the simplest benchmark, $A_{\mathrm{loc}}(s)$ is the full feasible action set. In heterogeneous-agent applications, it can represent the local one-household deviation used for validation.

Proposition 1 (Implicit action-value object)**.** *Under Assumption 1, the local Bellman operator is a contraction and has a unique fixed point $V^{\mathrm{loc},*}$. Moreover, for any continuation value V, the known payoff and transition equations induce the one-step action-value object*

$$Q_V(s,a) = u(s,a) + \beta E\left[V(f(s,a,\varepsilon'))\right], \tag{19}$$

so that

$$(T^{\mathrm{loc}}V)(s) = \max_{a \in A_{\mathrm{loc}}(s)} Q_V(s,a). \tag{20}$$

Proposition 2 (Idealized and approximate convergence)**.** *Suppose the idealized iAC update implements the exact local greedy step*

$$V_{n+1} = T^{\mathrm{loc}}V_n. \tag{21}$$

Then

$$\|V_n - V^{\mathrm{loc},*}\|_{\infty} \leq \beta^n \|V_0 - V^{\mathrm{loc},*}\|_{\infty}. \tag{22}$$

More generally, if the implemented update satisfies

$$\|V_{n+1} - T^{\mathrm{loc}}V_n\|_{\infty} \leq \delta_n, \tag{23}$$

then

$$\|V_n - V^{\mathrm{loc},*}\|_{\infty} \leq \beta^n \|V_0 - V^{\mathrm{loc},*}\|_{\infty} + \sum_{j=0}^{n-1} \beta^{n-1-j}\delta_j. \tag{24}$$

In particular, if $\sup_n \delta_n \leq \bar{\delta}$, then

$$\limsup_{n\to\infty} \|V_n - V^{\mathrm{loc},*}\|_{\infty} \leq \frac{\bar{\delta}}{1-\beta}. \tag{25}$$

If $\delta_n \to 0$, then $V_n \to V^{\mathrm{loc},*}$ uniformly.

Proposition 3 (Gradient penetration). *Fix a target critic $V_{\bar{\omega}}$. If the actor objective is*

$$J(\theta;\bar{\omega}) = E_{s,\varepsilon'}\left[u(s,\pi_\theta(s)) + \beta V_{\bar{\omega}}(f(s,\pi_\theta(s),\varepsilon'))\right], \tag{26}$$

then, on the interior of the feasible region,

$$\nabla_\theta J(\theta;\bar{\omega}) = E_{s,\varepsilon'}\left[D_\theta\pi_\theta(s)^\top \left\{u_a(s,a) + \beta f_a(s,a,\varepsilon')^\top \nabla_{s'} V_{\bar{\omega}}(s')\right\}\big|_{a=\pi_\theta(s)}\right]. \tag{27}$$

Here $D_\theta\pi_\theta(s)$ denotes the Jacobian of the actor output with respect to the parameter vector.

Proposition 4 (Gradient-penetration error channel). *Let $g_{\mathrm{iAC}}(\theta;V)$ be the population actor gradient generated by the one-step object Q_V, and let $g_Q(\theta)$ be the actor gradient generated by a learned state-action critic Q_ω. If $\|D_\theta\pi_\theta(s)\|_{\mathrm{op}} \le M_\pi$, then*

$$\|g_Q(\theta) - g_{\mathrm{iAC}}(\theta;V)\| \le M_\pi E_s\left[\|\nabla_a Q_\omega(s,a) - \nabla_a Q_V(s,a)\|_{a=\pi_\theta(s)}\right]. \tag{28}$$

If, in addition, $\|f_a(s,a,\varepsilon')\|_{\mathrm{op}} \le M_f$ and $\|\nabla\hat{V} - \nabla V^{\mathrm{loc},*}\|_\infty \le \eta_V$, then

$$\|g_{\mathrm{iAC}}(\theta;\hat{V}) - g_{\mathrm{iAC}}(\theta;V^{\mathrm{loc},*})\| \le \beta M_\pi M_f \eta_V. \tag{29}$$

Thus gradient penetration replaces the learned action-gradient channel $\nabla_a Q_\omega$ with the known derivatives of the one-step Bellman objective and the value-gradient error of the critic. If an actor step uses an approximate gradient with error bounded by b, and the actor objective is L-smooth and bounded above by $J_{\max}$, then for $0 < \eta \le 1/(4L)$,

$$\min_{0\le t<T} \|\nabla J(\theta_t)\|^2 \le \frac{2(J_{\max} - J(\theta_0))}{\eta T} + \frac{5}{2}b^2. \tag{30}$$

Consequently, a smaller actor-gradient error bound implies a smaller stationarity neighborhood for the actor optimization problem. This result should be read as a gradient-quality and optimization-neighborhood statement, not as a universal finite-sample speed-up theorem for arbitrary nonconvex neural-network training.

Remark 1 in the appendix gives a simple smooth-regression illustration in which the value-gradient channel has a lower-dimensional statistical rate than a learned state-action Q-gradient channel.

Proposition 5 (Local policy-loss bound). *Let $(\hat{V},\hat{\pi})$ be an approximate iAC pair. Suppose*

$$\|\hat{V} - T^{\hat{\pi}}\hat{V}\|_\infty \le \varepsilon_V \tag{31}$$

and

$$0 \le (T^{\mathrm{loc}}\hat{V})(s) - (T^{\hat{\pi}}\hat{V})(s) \le \varepsilon_{\mathrm{loc}} \qquad \textit{for all } s. \tag{32}$$

Then

$$\|V^{\mathrm{loc},*} - V^{\hat{\pi}}\|_\infty \le \frac{2\varepsilon_V + \varepsilon_{\mathrm{loc}}}{1-\beta}. \tag{33}$$

Corollary 1 (Policy-function and validation bounds)**.** *Suppose the one-step objectives $Q_{\hat{V}}(s,\cdot)$ and $Q_{V^{\mathrm{loc},*}}(s,\cdot)$ are μ-strongly concave on the relevant interior feasible set and have unique maximizers. Let*

$$\pi^*(s) = \arg\max_{a \in A_{\mathrm{loc}}(s)} Q_{V^{\mathrm{loc},*}}(s,a). \tag{34}$$

If

$$0 \le Q_{\hat{V}}(s, \pi^g_{\hat{V}}(s)) - Q_{\hat{V}}(s, \hat{\pi}(s)) \le \varepsilon_{\mathrm{loc}}(s) \tag{35}$$

and

$$\sup_{s,a} \|\nabla_a Q_{\hat{V}}(s,a) - \nabla_a Q_{V^{\mathrm{loc},*}}(s,a)\| \le \eta_Q, \tag{36}$$

then, at interior states,

$$\|\hat{\pi}(s) - \pi^*(s)\| \le \sqrt{\frac{2\varepsilon_{\mathrm{loc}}(s)}{\mu}} + \frac{\eta_Q}{\mu}. \tag{37}$$

At binding constraints, the same interpretation should be written in terms of the corresponding Kuhn–Tucker or subgradient gap rather than an interior policy-distance formula. If $\|f_a\|_{\mathrm{op}} \le M_f$ and $\|\nabla\hat{V} - \nabla V^{\mathrm{loc},*}\|_\infty \le \eta_V$, then $\eta_Q \le \beta M_f \eta_V$. Finally, if a bounded validation residual $\rho(s) \in [0, M]$ is evaluated on independent states $s_1, \dots, s_N \sim \nu$, then with probability at least $1-\delta$,

$$E_\nu[\rho(s)] \le \frac{1}{N}\sum_{i=1}^{N} \rho(s_i) + M\sqrt{\frac{\log(1/\delta)}{2N}}. \tag{38}$$

Numerical sampling, integration, or target-network errors can be included by redefining the two residual terms in the same sup norm. The reported average-error diagnostics should therefore be read as numerical analogues of these quantities, not as direct inputs into the sup-norm bound. The high-probability bound in Corollary 1 gives a distributional interpretation of sampled validation errors; a global sup-norm validation statement additionally requires grid coverage and Lipschitz extrapolation.

Summary of the formal results. These statements address three theoretical questions about the method. First, Proposition 2 gives exact convergence and approximate convergence-neighborhood results for the idealized local Bellman iAC update. Second, Proposition 3 and Proposition 4 show how gradient penetration changes the actor-gradient error channel and the resulting stationarity neighborhood. Third, Proposition 5 and Corollary 1 give value, policy-function, and sampled-validation error bounds under strong assumptions. The detailed proofs are in Appendix A.

4. RL methods that informed the iAC design

We use the benchmark consumption–savings problem to test and compare a collection of popular RL methods. For each method, we ask which ingredients are useful in the context of the benchmark problem. On the basis of these experiments, we design the iAC method shown in Section 2.

We will reformulate the consumption–savings problem in terms of the Q-learning framework (Watkins and Dayan, 1992), which is characteristic of the RL literature. In terms of the Q-function, the Bellman equation can be written as follows. The action is the choice of next-period assets, k', and the Q-Bellman equation is

$$Q((w,z),k') = u(w-k') + \beta \mathbb{E}\Big[\max_{k''\in\mathcal{K}(w')} Q((w',z'),k'') \;\Big|\; z\Big], \qquad w' = Rk' + y(z').$$

The corresponding state value is

$$V(w,z) = \max_{k'\in\mathcal{K}(w)} Q((w,z),k').$$

The learning update replaces full-grid Bellman iteration by pointwise updates,

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \beta \max_{a'} Q(s',a') - Q(s,a)\right], \tag{39}$$

where $s = (w,z)$ and a denotes the RL action.

Below, we consider Monte Carlo rollout, introduced by Tesauro and Galperin (1996) and discussed by Bertsekas and Tsitsiklis (1996); REINFORCE (Williams, 1992); PPO (Schulman et al., 2017); Dyna-Q, introduced by Sutton (1991); Deep Q-Networks (DQN), introduced by Mnih et al. (2015); Deep Deterministic Policy Gradient (DDPG), introduced by Lillicrap et al. (2015); Soft Actor-Critic (SAC), introduced by Haarnoja et al. (2018); and the all-in-One (AiO) Expectation Method proposed by Maliar et al. (2021).

4.1. Monte Carlo rollout.

Monte Carlo rollout evaluates candidate actions by simulating trajectories under the known transition law. In simple RL notation, the rollout value of action a at state s can be written as $Q^{\text{roll}}(s,a) = N^{-1}\sum_{i=1}^{N} G_i(s,a)$, where each simulated return $G_i(s,a)$ is generated by the model and a continuation policy after the first action. The action is then chosen as $\pi(s) = \arg\max_a Q^{\text{roll}}(s,a)$. The Monte Carlo rollout produces accurate solutions in our consumption–savings problem. Rollout concen-

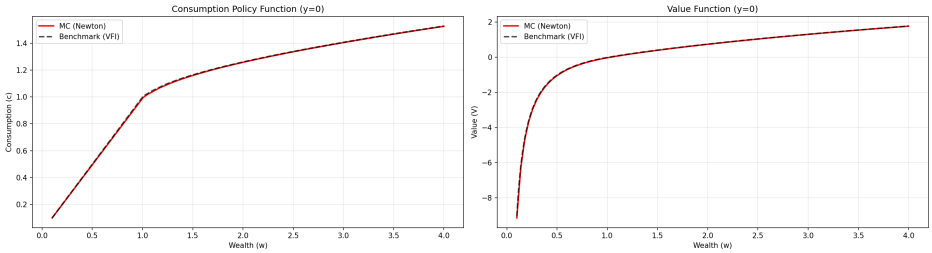


Fig. 2: Monte Carlo rollout results in the consumption–savings benchmark

trates computation on states and actions that are locally relevant for the current decision. A full global approximation is unnecessary for each local action comparison. Repeated simulation can be expensive, especially when accurate comparison across candidate actions requires many sampled trajectories or long planning horizons. The iAC design keeps model-based action evaluation and compresses it into a one-step Bellman target.

4.2. Dyna-Q.

Dyna-Q is implemented using tabular value storage on a discrete state grid and a discrete action grid. In the benchmark consumption–savings problem, the state is cash-on-hand and the action is next-period assets. For each visited grid point, the algorithm records the realized reward and next state generated by the known budget constraint and shock transition. Thus the stored model is a table

$$M(s,a) = (r, s').$$

The direct update is the standard Q-learning update in (39). The planning update samples previously visited pairs $(\tilde{s}, \tilde{a})$ from M and applies the same Bellman update to the synthetic transition $M(\tilde{s}, \tilde{a}) = (\tilde{r}, \tilde{s}')$. When expectations over shocks are evaluated, they are computed by applying the discrete shock transition probabilities on the grid. This makes Dyna-Q a tabular, model-assisted benchmark; see Appendix B.1.4.

Dyna-Q introduces a model-assisted planning mechanism. The direct and planning phases use the same Bellman update. The planning phase reuses stored transitions. In the benchmark, additional planning steps improve accuracy, as shown in Table 2. The limitation is also clear: planning remains tied to a discrete state-action table.

Table 2: Representative Dyna-Q configurations in the benchmark consumption–savings problem

Test	Planning Steps	Time(s)	MSE_C	MSE_V	Interpretation
1	0	4.48	1.12×10^{-2}	2.67×10^{-4}	pure Q-learning baseline
2	5	17.84	2.41×10^{-3}	5.33×10^{-5}	clear gain from planning
3	20	69.30	5.32×10^{-4}	2.16×10^{-5}	higher precision, higher cost

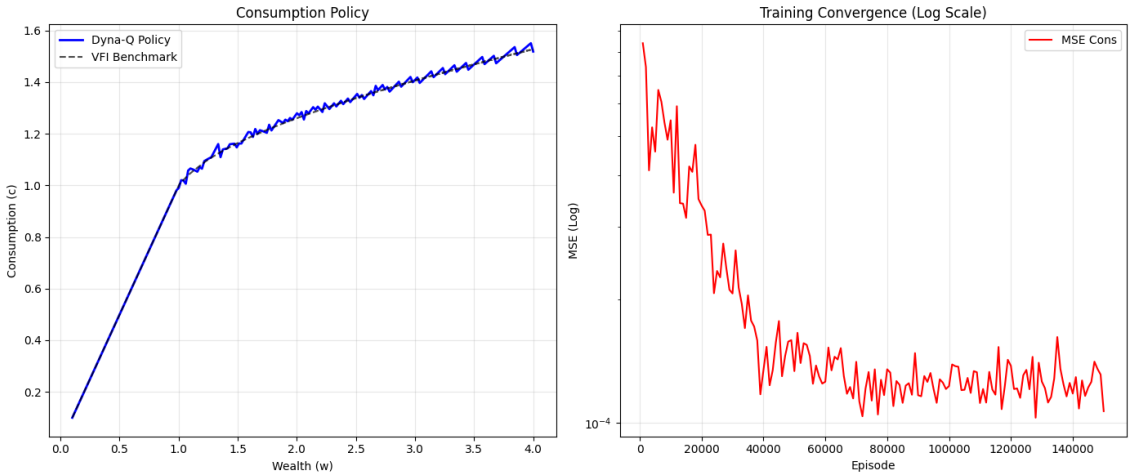


Fig. 3: Dyna-Q in the benchmark consumption–savings problem. The policy panel shows the tabular model-assisted solution against the VFI benchmark; the convergence panel reports the decline in the consumption-policy error.

The Dyna-Q architecture has three useful features. First, learning from stored transitions reduces the dependence on the order of simulated shocks. Second, planning updates propagate value information across previously visited state-action pairs without waiting for new episodes. Third, the method shows that the coverage of state-action pairs matters. Reusing a very narrow set of transitions cannot repair poor coverage of the economically relevant region. The corresponding iAC implication is to use model information inside the Bellman target while choosing training

states that cover the relevant parts of the state space. Repeated updates from a small set of sampled transitions provide a weaker training signal.

4.3. Deep Q-Networks (DQN).

The second source starts from the limits of tabular state-action methods. The next step is to use neural networks and continuous actions. This source identifies useful engineering mechanisms. It also shows where known economic equations can guide training more directly than a learned model-free channel.

The next step is to replace tabular value storage by a neural network. In DQN, the state–action value is approximated by a neural network $Q_\theta(s, a)$ trained from Bellman targets:

$$\mathcal{L}(\theta) = \mathbb{E}\big[(Q_\theta(s_i, a_i) - y_i)^2\big], \tag{40}$$

with $y_i = r_i + \beta \max_{a'} Q_{\overline{\theta}}(s'_i, a')$.

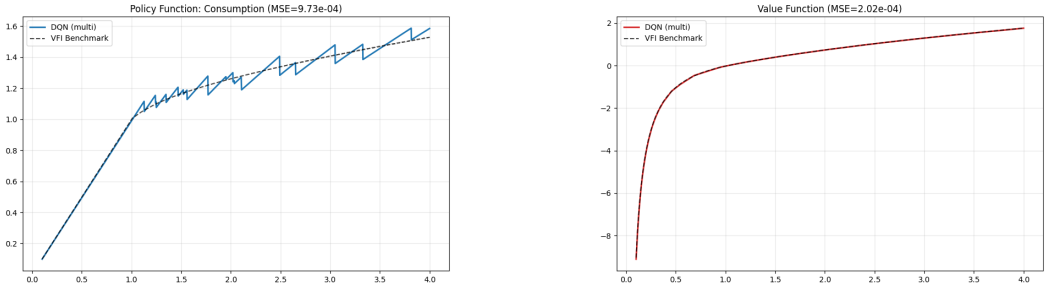


Fig. 4: DQN in the benchmark consumption–savings problem. The value function is already close to the benchmark reference solution. Policy oscillations remain because the action space is still discretized.

In the benchmark consumption–savings problem, DQN improves the value side of the problem. The network smooths over neighboring states and removes the rigid point-by-point structure of the table. The action space remains discrete, so policy recovery remains uneven even when the value fit is already strong. The iAC design keeps neural approximation and uses a continuous actor.

4.4. Deep Deterministic Policy Gradient (DDPG)

DDPG addresses the action-grid problem by using a continuous actor. It separates policy optimization from value fitting and works with continuous actions. It introduces a deterministic actor $\pi_\theta(s)$ and a critic $Q_\omega(s, a)$ with distinct update rules:

$$\mathcal{L}_Q(\omega) = \mathbb{E}\Big[\big(Q_\omega(s, a) - [r + \beta Q_{\overline{\omega}}(s', \pi_{\overline{\theta}}(s'))]\big)^2\Big], \tag{41}$$

and

$$\max_{\theta} \ \mathbb{E}\big[Q_{\omega}(s, \pi_{\theta}(s))\big]. \tag{42}$$

For iAC, the main lesson from DDPG is the actor–critic training structure. The actor gives a continuous policy. The critic gives a value signal used for policy improvement. A deterministic actor is also natural in economic applications because the target object is a state-contingent choice rule. The benchmark consumption–savings problem shows the weak side of a learned model-free Q channel, as shown in Figure 5.

Because the actor is deterministic, exploration must be injected from outside. If the critic is inaccurate early on, the actor can move toward immediate consumption and get trapped there. This is a training-channel lesson. The deterministic actor can be useful when the economic equations give a reliable policy-improvement signal. The iAC response keeps the separated actor–critic structure. It gives the actor a one-step Bellman objective generated by the payoff, feasibility, and transition equations, with the value critic supplying continuation values.

4.5. Soft Actor-Critic (SAC)

SAC keeps the actor–critic split of DDPG. It replaces the deterministic actor by a stochastic one and adds entropy regularization. In compact notation, the actor minimizes the loss

$$\mathcal{L}_{\pi}(\theta) = \mathbb{E}\big[\alpha \log \pi_{\theta}(a \mid s) - \min_{k} Q_{\phi_k}(s, a)\big], \tag{43}$$

while the critic is trained from soft Bellman targets,

$$y = r + \beta \, \mathbb{E}_{a' \sim \pi_{\theta}} \Big[\min_{k} Q_{\bar{\phi}_k}(s', a') - \alpha \log \pi_{\theta}(a' \mid s')\Big]. \tag{44}$$

With adaptive temperature, the entropy weight itself is learned,

$$\mathcal{L}_{\alpha}(\alpha) = \mathbb{E}\big[-\alpha(\log \pi_{\theta}(a \mid s) + \bar{\mathcal{H}})\big]. \tag{45}$$

In the benchmark consumption–savings problem, SAC has the lowest policy error among the pure model-free deep RL candidates considered here.

After DQN, DDPG, and SAC, the benchmark has neural state approximation, continuous actions, separated actor and critic updates, and stochastic exploration. The third panel in Figure 6 serves as a diagnostic. It removes the level component induced by the SAC temperature term. In the soft Bellman target, the continuation value contains the entropy adjustment

$$V^{soft}(s) = \mathbb{E}_{a \sim \pi} \left[Q(s, a) - \alpha \log \pi(a \mid s)\right].$$

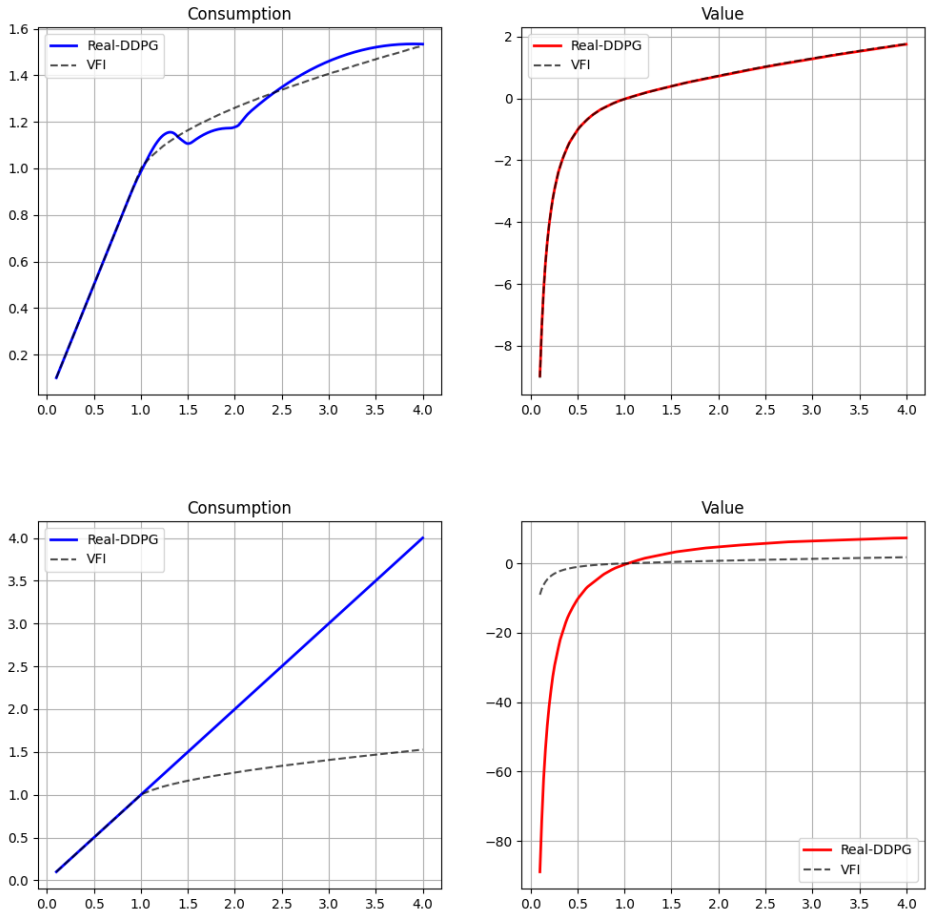


Fig. 5: Two DDPG outcomes in the benchmark consumption–savings problem. The upper panel shows a run that nearly matches the VFI benchmark, although with visible local wiggles in the consumption policy. The lower panel shows a run in which the deterministic actor collapses toward an all-consumption local optimum before the critic has learned the value of savings accurately.

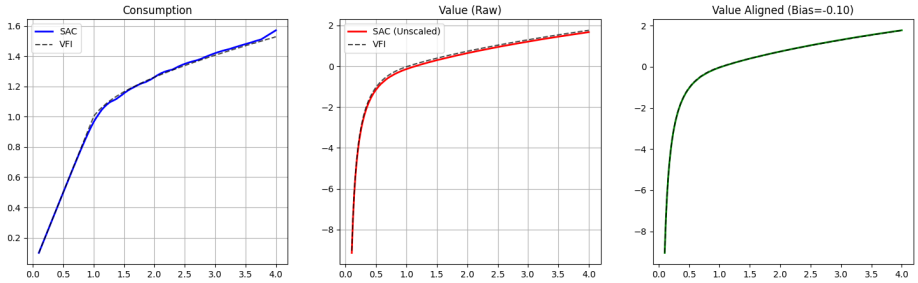


Fig. 6: SAC with adaptive temperature. Relative to DDPG, stochastic exploration stabilizes learning and produces a smoother policy in the benchmark consumption–savings problem. The raw SAC critic estimates the entropy-regularized soft value. The value-aligned panel subtracts the temperature-induced level component before comparing the shape with the VFI value.

The critic estimates an entropy-regularized value object that differs from the VFI benchmark. For plotting, we report

$$V^{aligned}(s) = V^{soft}(s) - \Delta_\alpha, \qquad \Delta_\alpha = \frac{1}{N}\sum_{i=1}^{N}\left(V^{soft}(s_i) - V^{VFI}(s_i)\right),$$

which removes the average temperature-related level difference and makes the value-shape comparison easier to read. The main economic comparison uses the policy panel and the standard value panel. The aligned panel diagnoses whether the remaining difference is mainly a level shift or a shape error.

SAC is useful because it shows that additional exploration and stabilization mechanisms can improve model-free actor–critic learning. Its stochastic policy is valuable for exploration in black-box environments. In the economic problems studied here, the final object is a state-contingent economic choice rule, and the model equations already give strong guidance about payoffs, feasibility, and transitions. The iAC design uses this structure as the main stabilization and policy-improvement channel, with less reliance on entropy-regularized exploration.

4.6. All-in-one expectation method

The Bellman-based deep learning method of Maliar et al. (2021) occupies a different position from the model-free methods above. It is already model-informed: it uses the economic Bellman equation and a neural value function. This makes it an important bridge to iAC. It shows that value-function learning can be used with neural approximation in continuous economic control.

The limitation for the present design lies in the training structure. In the Bellman residual formulation, policy improvement and value fitting are combined inside one objective. The two roles can interfere with each other. Schematically, one may think of this approach as optimizing a single Bellman-based objective in both policy and value parameters at once:

$$\mathcal{L}_B(\omega,\theta) \approx \mathbb{E}\left[\Big(V_\omega(s) - \big\{u\big(s,\pi_\theta(s)\big) + \beta\,\mathbb{E}\big[V_\omega(s') \mid s,\pi_\theta(s)\big]\big\}\Big)^2\right] - \lambda\,\mathbb{E}\left[u\big(s,\pi_\theta(s)\big) + \beta\,\mathbb{E}\big[V_\omega(s') \mid s,\pi_\theta(s)\big]\right]. \quad (46)$$

This joint objective is conceptually appealing. It mixes policy optimization and value fitting in one optimization problem. In the benchmark consumption–savings implementation used here, this mixing is less stable than a separated actor–critic architecture. The design implication is direct. Model information should be used inside a training structure that separates critic fitting from actor improvement.

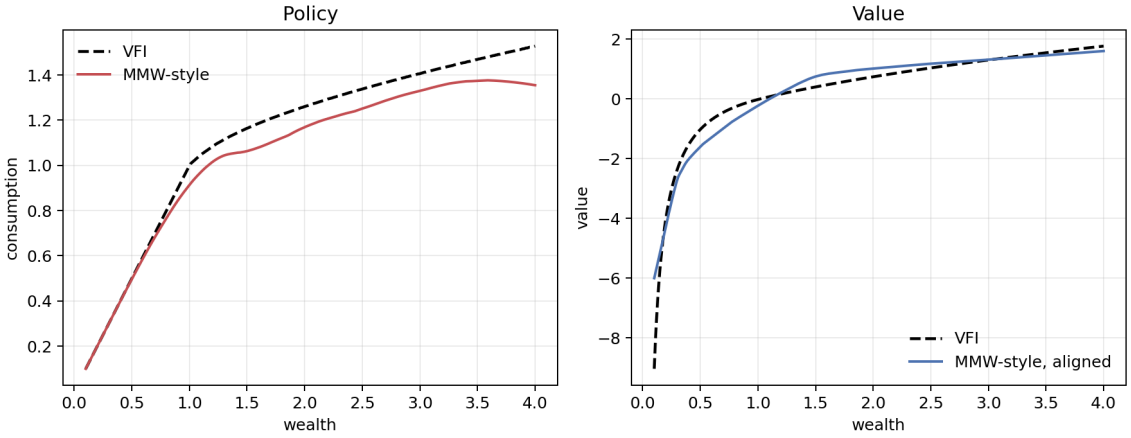


Fig. 7: Bellman-based deep learning in the benchmark consumption–savings problem. The figure reports the policy and value objects from the saved Bellman-based implementation against the VFI benchmark. The value panel is aligned by a constant level shift to emphasize shape.

4.7. The Resulting iAC Design

The benchmark evidence leads to a compact design. iAC keeps neural approximation, a continuous actor, and separated actor–critic updates. It also keeps Bellman discipline because economic dynamic problems are recursive. The main change is the information channel. The actor and critic are trained with the known payoff, feasibility, and transition equations.

Table 3: Benchmark evidence and the resulting iAC design choices

Design requirement	Benchmark motivation	iAC implementation
Continuous economic control	Discrete action grids in Q-learning and DQN create policy recovery problems.	The actor outputs a continuous control or a bounded choice parameter.
Feasibility by construction	Economic choices must satisfy budget and constraint restrictions.	A model-specific map Ψ converts network outputs into feasible actions.
Value critic	A learned $Q(s, a)$ adds an action dimension and can give unstable actor gradients.	The critic approximates $V_\omega(s)$, the continuation value.
One-step Bellman target	Rollout and Dyna-Q show the value of model-based evaluation. Repeated simulation and tables are costly.	The critic target is current payoff plus discounted continuation value after the known transition.
Model-based actor gradient	Deep actor–critic methods need a stable policy-improvement signal.	The actor gradient passes through payoff, feasibility, transition equations, and the fixed value critic.
Local Bellman diagnostics	High-dimensional applications need validation beyond aggregate simulations.	We report actor-conditioned Bellman residuals and local optimality errors.

The key technical point is that iAC can use a value critic without losing the action dependence needed for policy improvement. For any continuation value V, the known payoff and transition equations induce the one-step action-value object

$$Q_V(s,a) = u(s,a) + \beta \mathbb{E}\left[V(f(s,a,\varepsilon'))\right]. \tag{47}$$

The value critic supplies continuation values. The economic equations supply the action dependence. The critic can fit $V(s)$. The actor is improved by differentiating the one-step Bellman right-hand side through the known model equations. This is the sense in which iAC is informed. The training loop uses available economic structure directly.

5. High-Dimensional Applications of iAC

5.1. General logic of iAC heterogeneous-agent applications

The high-dimensional applications below keep the same iAC training logic: continuous actors, value critics, model-specific feasibility maps, one-step Bellman targets, and actor gradients through known economic equations. The surrounding economic objects change. Heterogeneous-agent models add simulated state distributions and equilibrium price or belief objects. OLG models add age, terminal conditions, and macro belief updating.

What changes across models is the implementation. In the benchmark, the transition equation and expectation can be used directly at low cost. In higher-dimensional models, only some structural objects can be imposed directly. The rest is handled by approximation, simulation, or equilibrium iteration.

The applications below show how the iAC method is adapted to different models. We first use the classical two-shock Krusell–Smith benchmark, which is also studied by Han et al. (2026). This model has a reference spline policy, so we can compare policies directly. We then study a Krusell–Smith model with AR(1) aggregate shocks and compare with the Bellman and Euler benchmarks in Maliar et al. (2021). Finally, we consider an extension with endogenous labor supply and an overlapping-generations environment. In each case, the states, controls, equilibrium mappings, and model-based gradient propagation are adapted to the model.

5.2. Two-Shock Krusell–Smith Benchmark

We begin with the classical two-shock Krusell–Smith benchmark. This model has a reference spline policy. We can therefore compare policy accuracy, the implied distribution, aggregate simulation paths, and Bellman errors in the same setting. We follow the Krusell–Smith moment representation and use aggregate capital as the belief state. This makes the comparison with Han et al. (2026) direct.

5.2.1. Model Setup

Households have CRRA preferences and choose consumption and next-period capital,

$$\max_{\{c_t^i, k_{t+1}^i\}} \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \frac{(c_t^i)^{1-\gamma} - 1}{1-\gamma}.$$

The individual budget constraint is

$$k_{t+1}^i = R_t k_t^i + W_t e_t^i - c_t^i, \qquad c_t^i \geq 0, \qquad k_{t+1}^i \geq 0,$$

where $e_t^i \in \{0, 1\}$ is the employment state. Aggregate productivity $z_t \in \{z_g, z_b\}$ and employment follow the standard joint transition matrix

$$\Pi_{(e,z),(e',z')} = \Pr(e_{t+1} = e', z_{t+1} = z' \mid e_t = e, z_t = z).$$

The representative firm has production

$$Y_t = z_t K_t^{\alpha} L_t^{1-\alpha},$$

so factor prices are

$$R_t = 1 - \delta + \alpha z_t K_t^{\alpha-1} L_t^{1-\alpha}, \qquad W_t = (1-\alpha) z_t K_t^{\alpha} L_t^{-\alpha}.$$

Following the Krusell–Smith moment representation, the aggregate state is summarized by aggregate capital K_t. Thus the household policy can be written as a function of individual capital, employment status, aggregate productivity, and aggregate capital,

$$c_i = \pi(k_i, e_i, z_t, K_t).$$

The comparison below evaluates how closely different neural solution methods approximate this reference household policy and the distribution and aggregate paths it implies.

5.2.2. Benchmark Comparison

We compare three objects in the same two-shock benchmark: the reference spline solution, a replication of the released DeepHAM code associated with Han et al. (2026), and our iAC implementation. The comparison is meant to understand where the methods are accurate. DeepHAM is a useful simulation-based method. The results below show that it produces accurate aggregate simulations. The reference spline also lets us check policy-level accuracy.

Figure 8 reports the baseline policy comparison. The iAC run in this figure uses the same low-dimensional moment state representation. Both neural-network

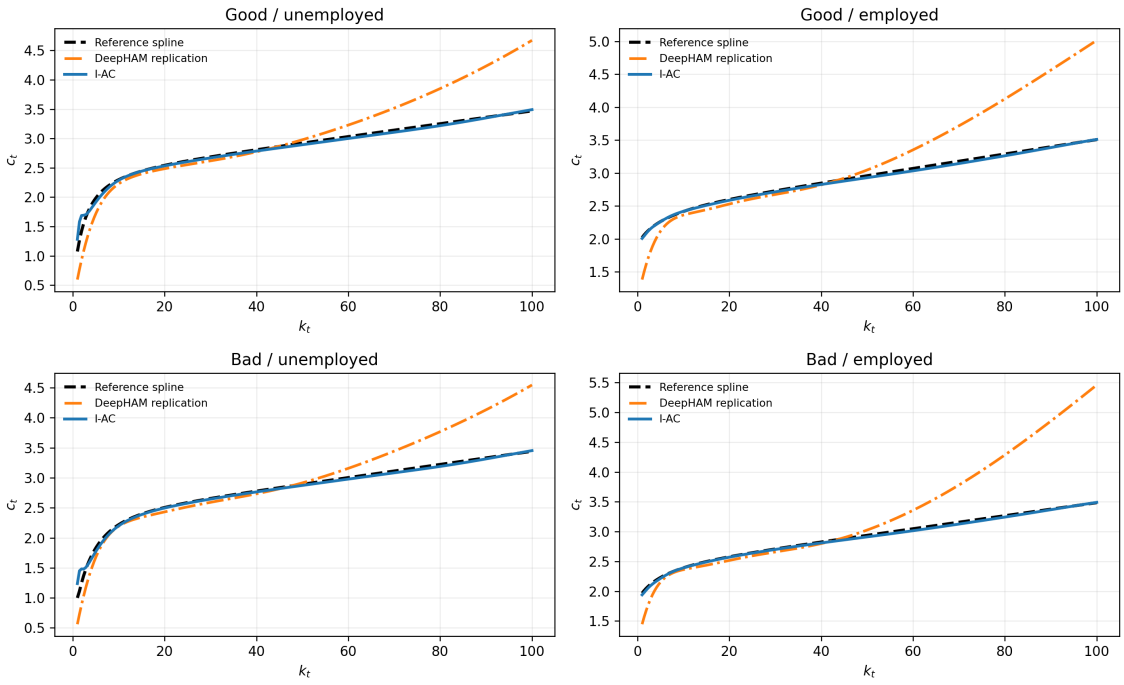


Fig. 8: Policy comparison in the two-shock Krusell–Smith benchmark. The baseline iAC policy uses the same low-dimensional moment state representation as the DeepHAM comparison.

methods therefore face the same coverage issue. Low-capital states are economically important, but long simulations visit them infrequently.

The baseline comparison shows two facts. First, the released-code DeepHAM replication is accurate in the central part of the state space. It has visible deviations from the reference spline in the low-capital kink region and in parts of the high-capital region. Second, the iAC policy is closer to the reference spline over a wider range of capital states. This is consistent with the iAC construction above. The actor is trained against a local one-step Bellman signal at the sampled household states. The low-capital deviations should be read as a sample-coverage issue. They do not show that DeepHAM cannot solve this model. In this two-shock implementation, "local" means that the actor evaluates one-household deviations while holding the aggregate belief and pricing object at the baseline generated by the current policy. The household's candidate next-period capital still enters its continuation value, but the aggregate belief channel is not re-optimized inside this local deviation. This is the Bellman object used in the local validation below and corresponds to the local operator in Proposition 5.

Figure 9 supports this interpretation. With the same iAC architecture, a small amount of low-capital sampling improves the kink region. This refinement is simple. It adds Bellman and actor training points in a region where ordinary simulations generate few observations. A similar sampling refinement could also be used with DeepHAM or other simulation-based methods when low-probability regions matter.

Figure 10 reports the policy-implied capital distributions for the baseline comparison. No upper cap is imposed on the DeepHAM simulation in the main text. This comparison is useful because similar policies near the center of the distribution can still imply different stationary distributions.

Figure 11 compares aggregate simulation paths under common aggregate shocks. The aggregate paths are close for all three solutions. This is true even though Figure 8 shows policy-level differences. Aggregate capital and aggregate consumption are cross-sectional averages, so aggregate paths can hide local policy errors.

All experiments in this comparison were run on a desktop with an Intel Core i7-12700KF CPU and an NVIDIA GeForce RTX 3080 GPU (10 GB VRAM). DeepHAM was run with the default parameter settings from its released public code.

The Bellman validation in Table 4 gives the same message as the policy comparison. The iAC method has lower expectation and local optimality errors in the baseline case. The two-shock benchmark gives two lessons. First, the one-step Bellman iAC construction performs well in these policy-training diagnostics for this heterogeneous-agent model. Second, aggregate paths are not enough when a reference solution is available. It is useful to also check policies, distributions, and local

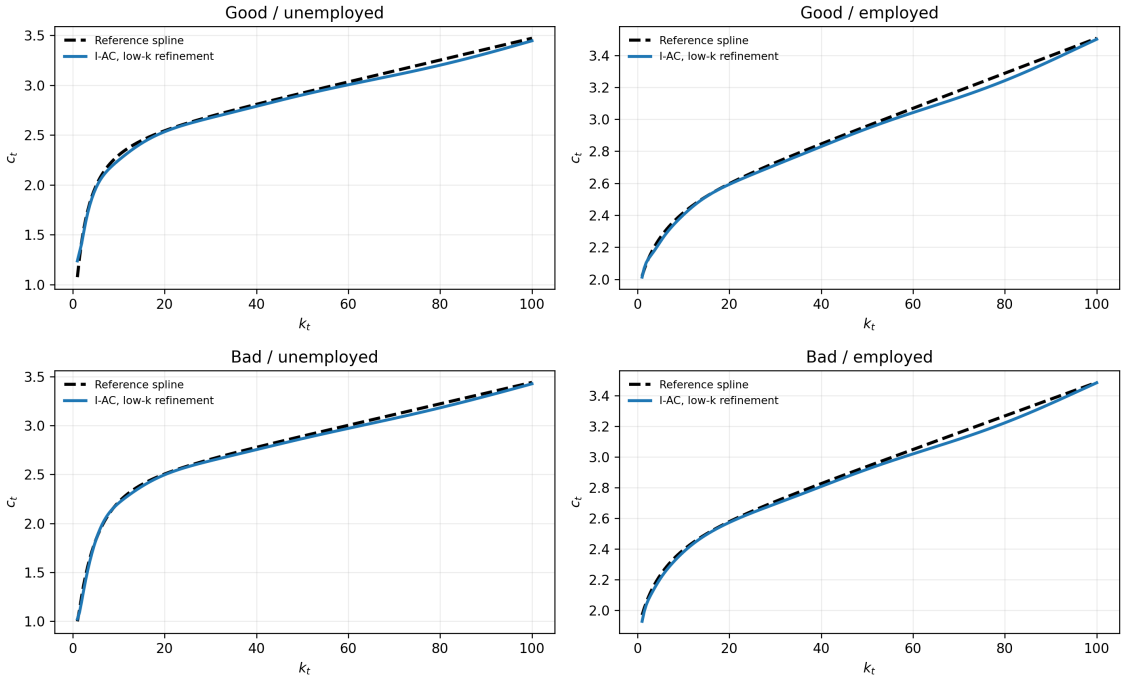


Fig. 9: iAC policy after a small low-capital sample refinement. The figure uses one seed and shows that the kink-region discrepancy is mainly a coverage issue.

Table 4: Bellman validation in the two-shock Krusell–Smith benchmark

Method	Runtime, sec.	Expectation MAE	Optimality MAE
DeepHAM	9660	0.01303	0.01512
iAC method	4740	0.00779	0.00814

Notes: Expectation MAE is the sampled mean absolute actor-conditioned Bellman residual. Optimality MAE is the sampled mean absolute gap from the local one-step Bellman improvement, holding aggregate beliefs and prices fixed at the current policy objects.

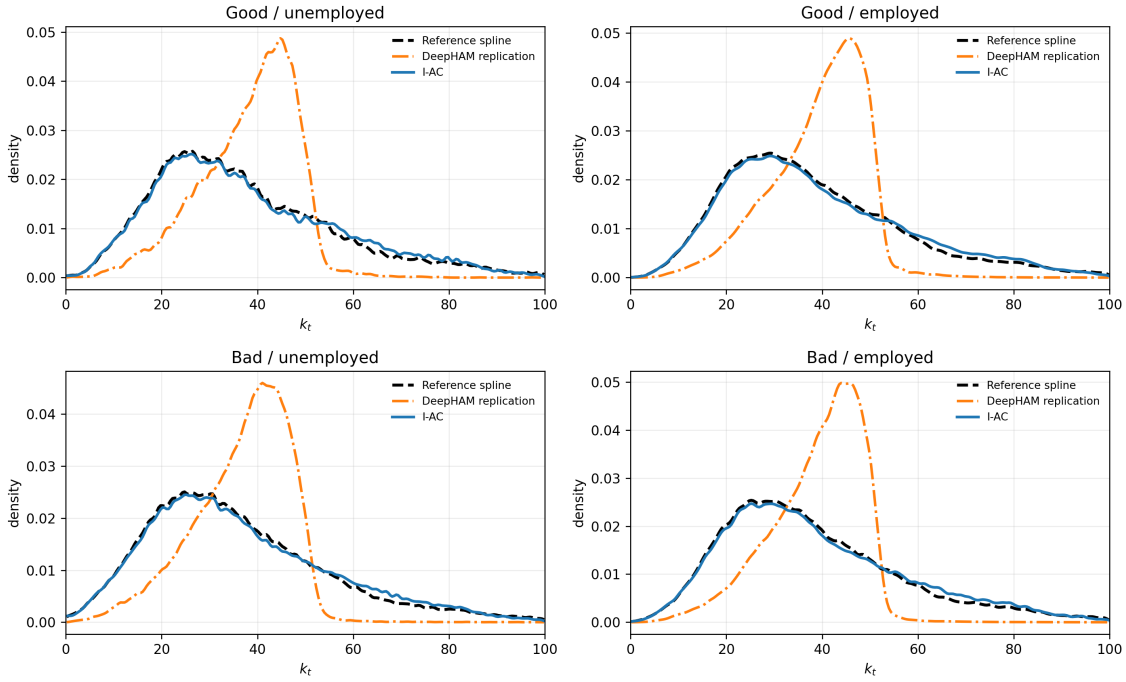


Fig. 10: Policy-implied capital distributions in the baseline comparison, without an upper capital cap in the DeepHAM simulation.

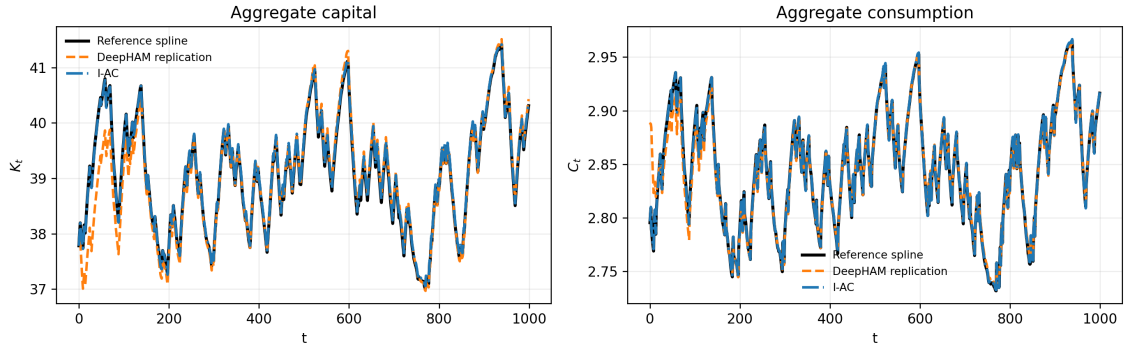


Fig. 11: Aggregate simulation paths under common shocks. The comparison uses the reference spline, the released-code DeepHAM policy, and the iAC policy, with no upper capital cap imposed on the DeepHAM simulation.

Bellman errors. In the terminology of the formal discussion, the expectation error is the sampled analogue of the actor-conditioned Bellman residual, while the local optimality error is the sampled analogue of the local actor greedy error:

$$\text{Bellman expectation error} \approx \varepsilon_V, \qquad \text{Bellman local optimality error} \approx \varepsilon_{\text{loc}}.$$

These are sampled average-error analogues of the residual terms in Proposition 5; the proposition itself is a sup-norm statement. Corollary 1 gives the corresponding high-probability interpretation for sampled mean residuals over the validation distribution.

Appendix C reports additional low-capital multi-seed and capped-simulation figures.

5.3. Krusell–Smith with AR(1) Aggregate Shock: Comparison with Maliar et al. (2021)

The Krusell–Smith model is the first high-dimensional iAC application with continuous aggregate shocks. It differs from the two-shock benchmark above. Aggregate productivity follows an AR(1) process rather than a two-state Markov chain. The comparison is now with the Bellman and Euler implementations in Maliar et al. (2021). The method is the same, but the state transitions are richer. The algorithm must choose which structural objects are useful for equilibrium learning. In this AR(1) implementation, the policy-improvement step is local at sampled household states. Aggregate beliefs and prices are generated by the current simulated economy and are held fixed inside the one-household Bellman improvement. This keeps the training object aligned with the local operator used in the formal results.

5.3.1. Model Setup

The model has ℓ heterogeneous agents. They share the same preferences, but they differ in productivity and asset holdings.

Each agent i chooses consumption c_t^i and next-period assets k_{t+1}^i to maximize expected lifetime CRRA utility:

$$\begin{aligned} &\max_{\{c_t^i, k_{t+1}^i\}_{t=0}^{\infty}} E_0\left[\sum_{t=0}^{\infty} \beta^t \frac{(c_t^i)^{1-\gamma}-1}{1-\gamma}\right] \\ &s.t. \quad k_{t+1}^i = w_t^i - c_t^i, \qquad c_t^i \in (0, w_t^i - \underline{k}] \\ &\qquad w_{t+1}^i = R_{t+1} k_{t+1}^i + W_{t+1} e^{y_{t+1}^i}. \end{aligned}$$

Here, w_t^i is agent i's cash-on-hand at the beginning of period t, y_t^i is their log labor productivity, $\underline{k}$ is the borrowing limit, R_t is the gross return on assets $(1+r_t)$, and W_t

is the wage rate. Each agent provides one unit of labor, and their log productivity y_t^i follows an exogenous AR(1) process:

$$y_{t+1}^i = \rho_y y_t^i + \sigma_y \epsilon_{t+1}^i, \quad \text{with } \epsilon_{t+1}^i \sim N(0,1) \tag{48}$$

A representative firm uses aggregate capital K_t and aggregate effective labor L_t for production via a Cobb-Douglas function:

$$F(K_t, L_t) = z_t K_t^\alpha L_t^{1-\alpha} \tag{49}$$

Aggregate productivity is $z_t = \exp(A_t)$, where log productivity follows an exogenous AR(1) process:

$$A_{t+1} = \rho_z A_t + \sigma_z \epsilon_{t+1}, \quad \text{with } \epsilon_{t+1} \sim N(0,1) \tag{50}$$

Aggregate capital K_t is the sum of beginning-of-period asset holdings, $K_t = \sum_{i=1}^{\ell} k_t^i$, and aggregate effective labor L_t is the sum of individual effective labor units $L_t = \sum_{i=1}^{\ell} \exp(y_t^i)$. Factor prices are determined by marginal products (δ is the depreciation rate):

$$R_t = 1 - \delta + z_t \alpha K_t^{\alpha-1} L_t^{1-\alpha} \tag{51}$$

$$W_t = z_t (1-\alpha) K_t^\alpha L_t^{-\alpha} \tag{52}$$

5.3.2. State Space and Neural Network Input:

In theory, an agent's decision depends on their own state (w_t^i, y_t^i) and the aggregate state. The aggregate state depends on the distribution of agent states $D_t = \{(w_t^i, y_t^i)\}_i^\ell$ and the aggregate productivity shock z_t. In the neural network implementation, this high-dimensional state information is used as input. For agent i, the input vector h_t^i contains:

- Current state information of all agents: $D_t = \{(w_t^i, y_t^i)\}_i^\ell$.
- The current aggregate shock: z_t.
- Agent i's own specific state: (w_t^i, y_t^i).

Thus, for each agent i, the input layer dimension is $2\ell + 1 + 2 = 2\ell + 3$.

5.3.3. Neural Network Approximations

We use neural networks to approximate the key functions:

- Policy Function (Consumption Ratio): $\psi(s_t^i; \theta_1)$ outputs the ratio of consumption to cash-on-hand $\frac{c_t^i}{w_t^i}$, using a sigmoid function to ensure it lies within $[0, 1]$.

$$\frac{c_t^i}{w_t^i} = \sigma(\zeta_0^\psi + \iota^\psi(y_t^i, w_t^i, D_t, z_t; \upsilon^\psi)) = \psi(s_t^i; \theta_1) \tag{53}$$

- Value Function: $V(s_t^i; \theta_2)$ directly outputs the estimated value of state s_t^i.

$$V_t^i = \zeta_0^V + \iota^V(y_t^i, w_t^i, D_t, z_t; \upsilon^V) = V(s_t^i; \theta_2) \tag{54}$$

Here, $\iota(\cdot)$ represents the neural network (with parameters υ), ζ_0 is a learnable adjustment coefficient, and $\theta = \{\zeta_0, \upsilon\}$ denotes all parameters for that network. These networks share the same input structure h_t^i.

5.3.4. Pseudocode

Algorithm 2 writes the AR(1) Krusell–Smith implementation in the same iAC notation used above. The policy network outputs a consumption-rate parameter, the feasibility map converts it into consumption and savings, and the memory pool is refreshed by simulations under the current policy. The actor update uses local household Bellman improvements at sampled states.

5.3.5. Results

We apply the proposed solver to the Krusell–Smith model and compare the results with the Bellman and Euler benchmarks in Maliar et al. (2021). This is the AR(1) aggregate-shock comparison; the two-shock comparison with Han et al. (2026) was reported separately above.

Overall Solution Quality and Steady-State Simulation. In this implementation, the learned policy functions have the expected monotonicity patterns for the Krusell–Smith model. The learned policy function ψ in Figure 12b shows the expected comparative statics: the consumption rate declines with assets and increases with productivity. When the policy is used in long-run simulations, the main aggregate variables, including aggregate capital, remain bounded and fluctuate around a steady-state region, as shown in Figure 12c. This behavior is consistent with the economic structure of the model and with existing benchmark results.

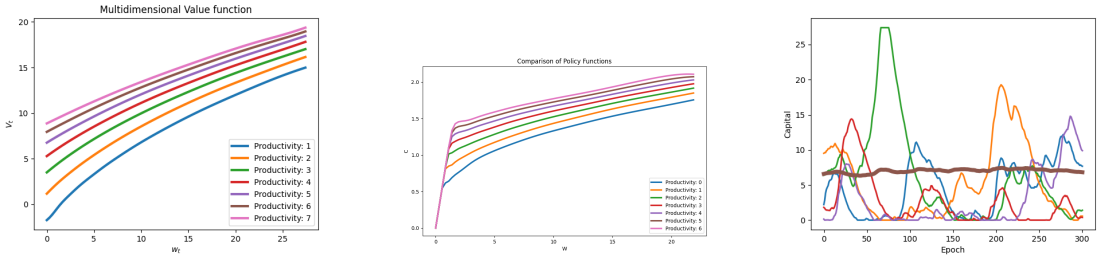
Algorithm 2 iAC training for the AR(1) Krusell–Smith model

Neural Network Settings
$\Phi^P(S;\Theta^P)$: Policy network
$\Phi^V(S;\Theta^V)$: Value network
$S^i = (\{w^j, y^j\}_{j\in L}, z, w^i, y^i)$ ▷ Agent state representation
Initialization
Initialize Θ_0^P, Θ_0^V with normal initialization
Initialize memory pool $MP \leftarrow \emptyset$
Draw initial economy states $Data_0^{sim} \in \mathbb{R}^{Batch_{sim}\times L\times 2}$
Fill MP with initial simulated states from $Data_0^{sim}$
for $k = 1$ to $Epochs_{pretrain}$ **do**
    Pretrain Value Network
    for $iter = 1$ to $train\_value\_times\_each\_step$ **do**
        Sample batch $\{w^i, y^i\}_{i=1}^L \sim MP$
        Compute $b^i = \Phi^P(S^i;\Theta^P_{fixed})$ and $(c^i, k^{i\prime}) = \Psi(S^i, b^i)$
        Compute target $\hat{v}^i = u(c^i) + \beta E_M[\Phi^V(S^{i\prime};\Theta^V_{fixed})]$
        Update Θ^V: $\min \frac{1}{L}\sum_{i=1}^L \frac{1}{M}\sum_{m=1}^M [\Phi^V(S^i;\Theta^V) - \hat{v}^i_m]^2$
    end for
end for
for $k = 1$ to $Epochs_{total}$ **do**
    for $phase = 1$ to $alternating\_steps$ **do**
        Policy Network Update
        for $iter = 1$ to $train\_policy\_times\_each\_step$ **do**
            Sample batch $\{w^i, y^i\}_{i=1}^L \sim MP$
            Compute $b^i = \Phi^P(S^i;\Theta^P)$ and $(c^i, k^{i\prime}) = \Psi(S^i, b^i)$
            Compute local actor gradient: $\nabla_{\Theta^P}\mathbb{E}_M[u(c^i) + \beta\Phi^V(S(k^{i\prime});\Theta^V_{fixed})]$
            Update Θ^P using Adam
        end for
        Value Network Update
        for $iter = 1$ to $train\_value\_times\_each\_step$ **do**
            Sample new batch $\{w^i, y^i\}_{i=1}^L \sim MP$
            Compute $b^i = \Phi^P(S^i;\Theta^P_{fixed})$ and $(c^i, k^{i\prime}) = \Psi(S^i, b^i)$
            Compute $\hat{v}^i = u(c^i) + \beta E_M[\Phi^V(S^{i\prime};\Theta^V_{fixed})]$
            Update Θ^V: $\min \frac{1}{L}\sum_{i=1}^L \frac{1}{M}\sum_{m=1}^M [\Phi^V(S^i;\Theta^V) - \hat{v}^i_m]^2$
        end for
    end for
    Memory Pool Update
    for $sim = 1$ to $simulate\_times$ **do**
        Simulate new policy states and maintain queue: $MP \leftarrow f(Data_{current}, \Phi^P, \varepsilon')$
    end for
end for

Table 5: AR(1) Krusell–Smith comparison with Maliar et al. (2021) benchmarks

Metric	iAC method	Maliar et al. Bellman	Maliar et al. Euler
Model moments			
Number of agents	50	50	50
Gini of wealth	0.4565	–	0.447
Corr.(c, y)	0.701	0.7275	0.681
Solver diagnostics			
Time, sec.	2939	14204	1721
R^2	0.996741	0.992532	0.9898
Bellman expectation error	0.010502	0.026395	–
Bellman optimality error	0.026260	0.026284	–

Notes: Bellman expectation error is the sampled absolute Bellman residual for the current policy. Bellman optimality error is the sampled absolute gap from the local one-step Bellman improvement.



(a) Value function (b) Policy function (c) Simulation

Fig. 12: Krusell–Smith with AR(1) aggregate shock: value function, policy function, and simulation path.

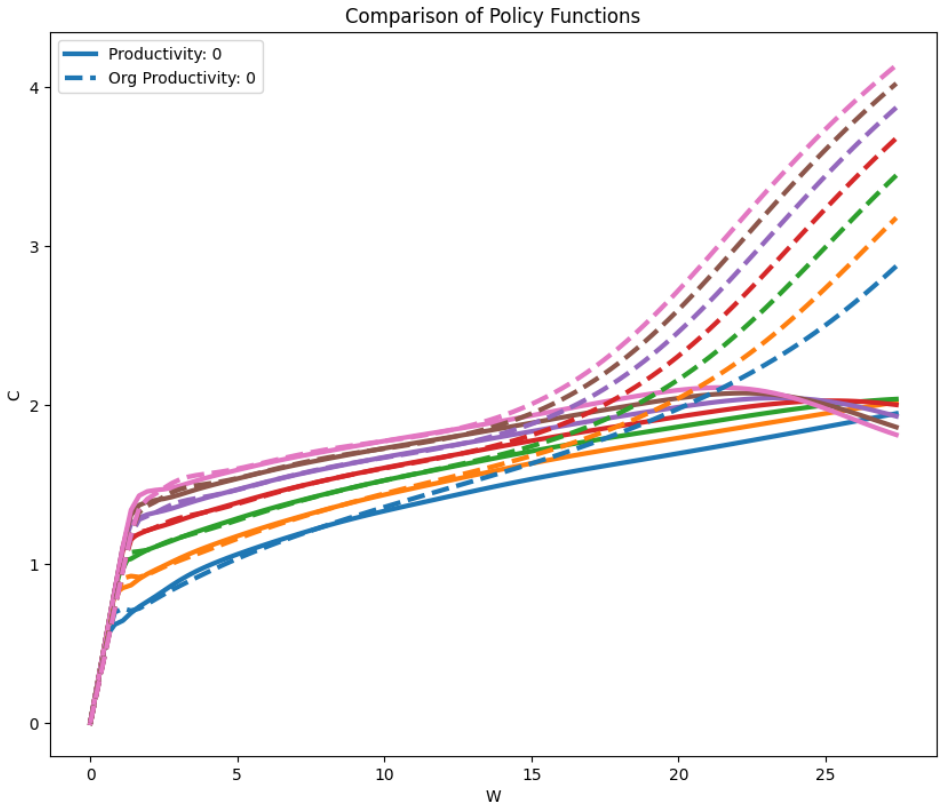


Fig. 13: Compared policy functions

Policy Function Accuracy and Smoothness. Figure 13 compares our policy function with the benchmark results in Maliar et al. (2021). The main differences are in the tail region and in the lower-asset region. In the tail region, our policy function behaves more regularly and displays fewer large local deviations. In the lower-asset region, it is smoother and contains fewer visible kinks in this comparison. These differences are relevant for long-run simulation because local non-smoothness in the policy function can generate distortions in simulated aggregate dynamics.

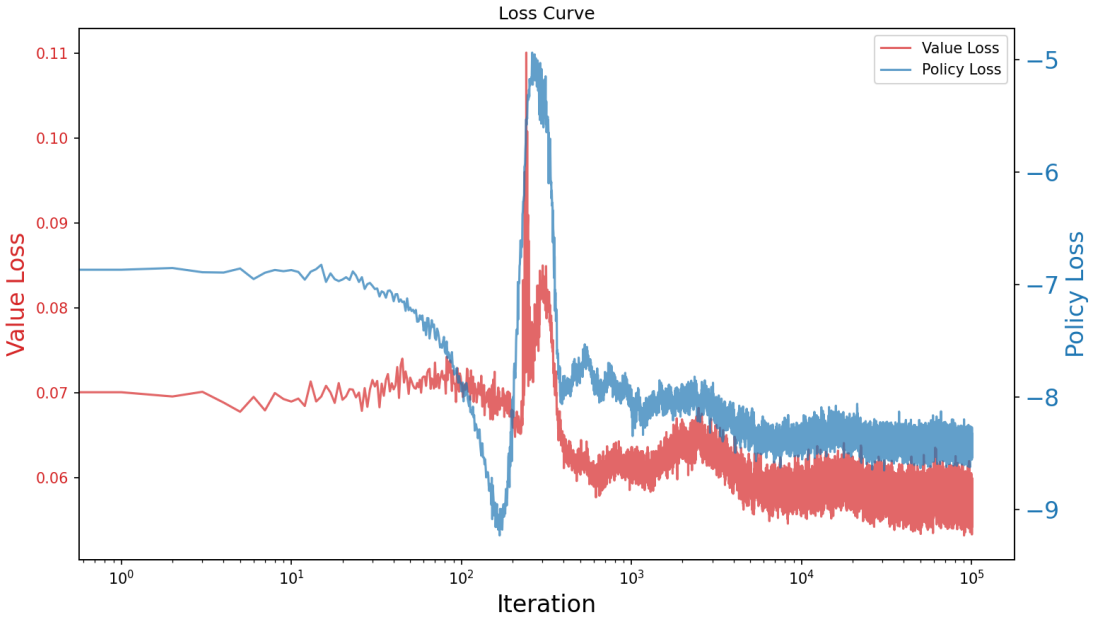
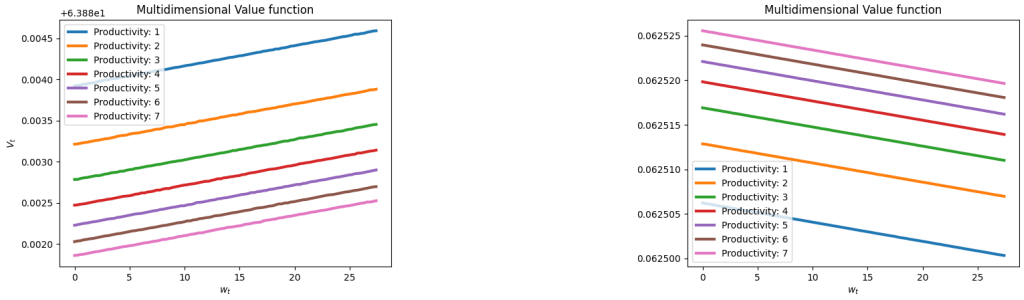


Fig. 14: Training losses

Training Losses. Because the method uses an actor and a critic, the losses differ from standard supervised-learning losses. The policy and value networks are trained together. Convergence therefore means that the two networks become stable, not that each loss disappears. As shown in Figure 14, both losses may fluctuate early in training. They later become more stable. This pattern is consistent with the policy and value networks moving toward a stable Bellman pair.

Sensitivity to Initialization. To examine sensitivity to initialization, we run the training procedure under different random initializations. As shown in Figure 15, the resulting policy functions and steady-state outcomes are similar across runs in this experiment. This reduces the concern that the reported solution comes from one favorable initialization.



(a) Random initialization 1

(b) Random initialization 2

Fig. 15: Different initializations

iAC extension takeaway. The main point is not only that the solver can be applied to the Krusell–Smith model. The same iAC training logic can also be used when the state space is large and equilibrium feedback is endogenous. Relative to the benchmark problem, the method class does not change. Only the implementation becomes more selective.

5.4. iAC with Endogenous Labor Supply

We next allow households to choose labor supply. This adds one continuous control variable. The same iAC method still applies. The actor now learns both savings and labor decisions under model-implied feasibility conditions.

5.4.1. Model Setup

We modify the baseline Krusell–Smith model to allow agents to choose their labor supply l_t.

Continuous Labor Supply. Agents can now continuously choose their labor supply l_t within the interval $[0, 1]$. The utility function is accordingly modified to include the disutility of labor:

$$u(c, l) = \frac{(c^{\kappa}(1-l)^{1-\kappa})^{1-\gamma} - 1}{1-\gamma} \tag{55}$$

where $\kappa > 0$ controls the trade-off between leisure $(1 - l)$ and consumption c. The labor income term in the budget constraint now depends on wages, idiosyncratic productivity, and labor supply:

$$w^i_{t+1} = R_{t+1}(w^i_t - c^i_t) + W_{t+1} l^i_t e^{y^i_{t+1}} \tag{56}$$

For the neural network implementation, the output dimension of the policy network increases because the network must output both the consumption decision and the labor supply decision l_t. Since l_t is constrained to $[0, 1]$, a sigmoid activation function can be used for the output corresponding to l_t.

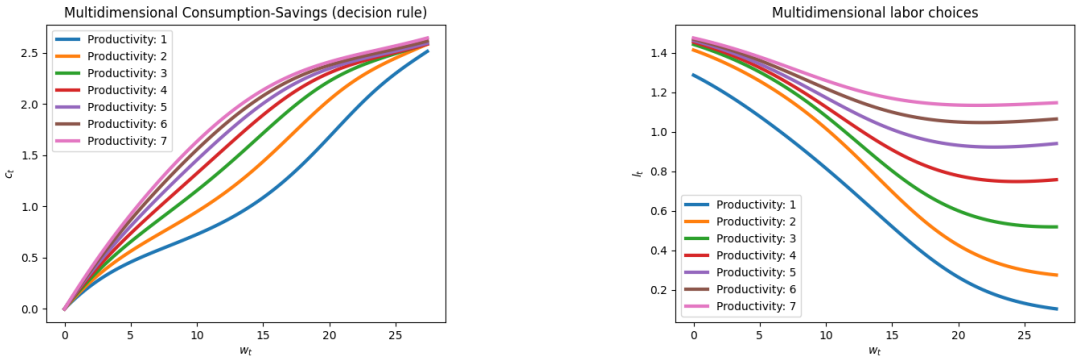
Discrete Labor Supply. An alternative scenario involves discrete labor participation choices, for example

$$l_t \in \{0, 1\} \quad \text{or} \quad l_t \in \{0, 0.5, 1\}.$$

These values represent unemployment, part-time work, and full-time work, respectively. We employ the Gumbel-Softmax trick at the end of the policy network. In this case, the policy network outputs the probabilities of choosing each discrete labor option, rather than the level itself.

5.4.2. Results

We apply the same iAC training method to both labor-supply extensions. Figures 16, 17, and 18 show the consumption, savings, and labor-supply policies in the continuous, binary, and ternary labor settings. The plotted policies have the expected qualitative patterns. We use this section as a portability check for the iAC training logic rather than as a full quantitative validation exercise.

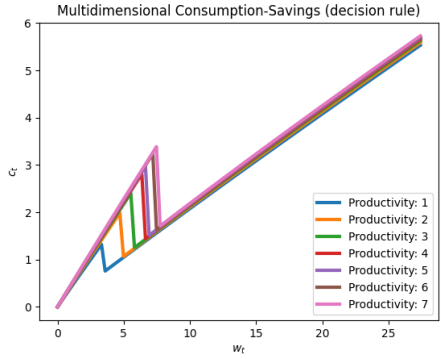


(a) Consumption choice

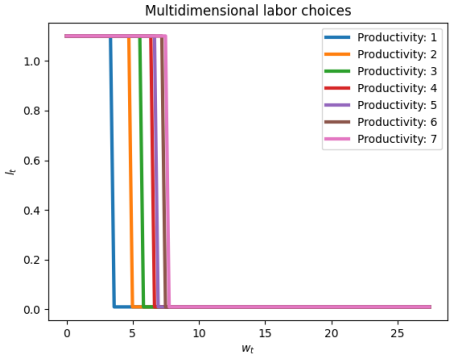
(b) Labor choice

Fig. 16: Continuous labor choice model

iAC extension takeaway. This extension shows how the iAC method can be written when the household problem adds labor choices. The action space is richer, and the feasibility conditions change. The training logic itself does not change.

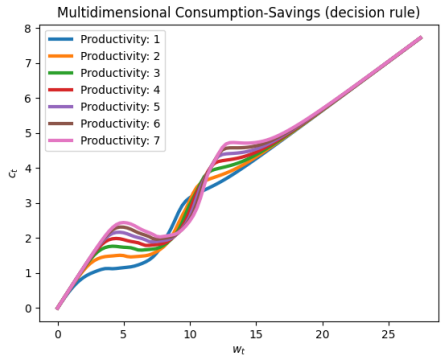


(a) Consumption choice



(b) Labor choice

Fig. 17: Binary labor-supply model



(a) Consumption choice

(b) Labor choice

Fig. 18: Ternary labor-supply model

5.5. iAC in OLG: non-stationarity, terminal structure, and macro belief

The final application is an overlapping-generations (OLG) economy. This is still an iAC application. The model adds age dependence, non-stationarity, terminal conditions, and macro equilibrium feedback. OLG models are harder than the Krusell–Smith model because the problem changes with age. Individual decisions depend on assets, aggregate shocks, and the household's place in the life cycle. The model therefore has age-specific policy rules, terminal conditions, and changes in the composition of generations.

5.5.1. Model Setup

We consider a simple OLG economy closely related to the benchmark environment studied in Azinovic et al. (2022). Households live for $A = 6$ periods and maximize expected lifetime log utility:

$$\max_{\{c^h_{t+h}, a^h_{t+h}\}_{h=0}^{A-1}} \mathbb{E}_t \sum_{h=0}^{A-1} \beta^h \log(c^h_{t+h}). \tag{57}$$

Let h denote age. Individuals only supply labor when young, so that $l^0_t = 1$ and $l^{h>0}_t = 0$. The household budget constraint is

$$c^h_t + a^h_t = R_t k^h_t + W_t l^h_t, \tag{58}$$

with capital accumulation

$$k^{h+1}_{t+1} = a^h_t, \tag{59}$$

and the terminal saving rule is

$$a^{A-1}_t = 0. \tag{60}$$

Here c^h_t is consumption, a^h_t is savings, k^h_t is beginning-of-period capital, R_t is the gross return on capital, and W_t is the wage.

The production side is given by a Cobb–Douglas technology with two aggregate shocks, total factor productivity η_t and depreciation δ_t:

$$F(K_t, L_t) = \eta_t K^\alpha_t L^{1-\alpha}_t + K_t(1 - \delta_t). \tag{61}$$

In the simple version studied here, the macro shock takes four discrete states corresponding to the combinations of high/low productivity and high/low depreciation. Aggregate factor prices satisfy

$$R_t = \alpha \eta_t K^{\alpha-1}_t L^{1-\alpha}_t + (1 - \delta_t), \qquad W_t = (1 - \alpha)\eta_t K^\alpha_t L^{-\alpha}_t. \tag{62}$$

Thus the model combines finite-horizon life-cycle behavior with aggregate general-equilibrium feedback.

5.5.2. Why iAC Requires Additional Adjustments in OLG

The main computational difference is age. In an infinite-horizon model, one can learn one stationary policy rule. In an OLG model, a young household and an old household choose differently even at the same wealth level. The final period also creates a hard terminal boundary. A small approximation error at the end of life can propagate backward and distort earlier-age policies.

To address these issues, we modify the iAC method in three ways.

First, age is included explicitly in the state and encoded by a one-hot vector. This allows one network to represent different policy rules across the life cycle. The state input is constructed as

$$s_t = (k_t^h, z_t, K_{\text{belief},t}, \text{Age}_t), \tag{63}$$

where k_t^h is individual capital, z_t is the aggregate shock, $K_{\text{belief},t}$ is the perceived aggregate capital level used to form expectations, and Age_t is the one-hot age representation.

Second, we introduce a scalar macro belief K_{belief} and update it through a Deep Aggregate Law of Motion (Deep ALM). This variable is a compressed equilibrium object. Instead of feeding the full wealth distribution into the network, agents condition their decisions on a belief about aggregate capital. The inner loop trains life-cycle policies for a given belief. The outer loop simulates the economy and updates the belief toward realized aggregate capital. This approximates rational-expectations consistency without putting the full wealth distribution into the network.

Third, we impose terminal anchoring. The last period has no continuation value. The final-age policy is therefore known: the household consumes all remaining resources. We impose this condition directly in training. This pins down the final-age value target and reduces errors that would otherwise move backward through the life cycle.

5.5.3. iAC training logic in OLG

The OLG solver uses the same iAC training logic. Training now has two layers.

At the micro level, conditional on a given macro belief K_{belief}, the actor outputs a savings rate (or, equivalently, a consumption rate) as a function of the individual state, age, and current macro conditions. The critic approximates the corresponding continuation value. Using the known transition equations, current prices, and the next-age law of motion, we construct Bellman-consistent targets exactly as in the benchmark models, except that the state now includes age and a changing macro environment.

At the macro level, we simulate a large population of agents over many periods using the learned micro policies, compute the realized aggregate capital path, and update the macro belief toward the simulated long-run average. Thus, the OLG implementation combines an inner model-based actor–critic problem with an outer belief-updating loop. The procedure uses reinforcement learning for optimization and uses available economic structure to stabilize the solution.

5.5.4. Generalized market clearing via macro belief

A useful way to interpret this iAC extension in the OLG setting is as a generalized market-clearing procedure. In a standard heterogeneous-agent general-equilibrium model, an individual would in principle need to forecast the evolution of the full cross-sectional distribution in order to form expectations about future prices. In the simple OLG environment studied here, we instead compress this aggregate information into a scalar macro belief, denoted by K_{belief}.

The reason this compression is useful in this simple model is that each individual is atomistic: when solving the household problem, a single agent does not internalize the effect of their own savings choice on aggregate capital. Thus, conditional on the current macro shock and a perceived aggregate capital level, the individual optimization problem can be treated as a price-taking problem. Since labor supply is pinned down by the life-cycle structure and aggregate shocks are exogenous, a scalar belief about aggregate capital is sufficient to construct current prices and continuation expectations in the inner loop.

More formally, given $K_{\text{belief},t}$ and the current aggregate shock z_t, prices are computed as

$$R_t = R(K_{\text{belief},t}, z_t), \qquad W_t = W(K_{\text{belief},t}, z_t). \tag{64}$$

The individual policy problem is then solved conditional on these prices. In the outer loop, the economy is simulated using the learned household policy, producing a realized aggregate capital path K_t^{sim}. Generalized market clearing is achieved by updating the belief toward the realized capital:

$$K_{\text{belief}} \leftarrow (1-\lambda)K_{\text{belief}} + \lambda K_{\text{sim}}, \tag{65}$$

where $\lambda \in (0, 1]$ is a damping parameter. In equilibrium, the perceived aggregate capital used by households to form decisions becomes consistent with the realized aggregate capital generated by those same decisions.

5.5.5. Algorithmic implementation

Algorithm 3 summarizes the iAC solver for the OLG model. The key ingredients are: (i) age one-hot encoding to represent non-stationary life-cycle policies; (ii) a

macro belief K_{belief} that is updated in an outer Deep ALM loop; and (iii) terminal anchoring to stabilize backward propagation from the last period of life.

Algorithm 3 iAC Method with Deep ALM for the OLG Model

Initialize actor parameters θ, critic parameters ω
Initialize macro belief $K_{\text{belief}} \leftarrow K_{\text{belief},0}$
Choose belief update frequency and damping parameter λ
for epoch $= 1, 2, \ldots$ **do**
    // **Inner layer: micro policy/value training conditional on** K_{belief}
    for each mini-batch **do**
        Sample individual capital k, macro shock z, and age h
        Encode age h as a one-hot vector
        Construct current state $s = (k, z, K_{\text{belief}}, \text{Age})$
        Optionally perturb K_{belief} slightly for generalization
        Compute current prices R, W from (K_{belief}, z)
        Actor outputs savings rate (or consumption rate) $\mu_\theta(s)$
        Compute current reward $u(c)$ from the budget constraint
        if $h = A - 1$ **then**
            // **Terminal anchoring**
            Set terminal target using the analytically known last-period rule
        else
            Compute next-period individual capital k'
            Draw / integrate over next macro shock z'
            Construct next state $s' = (k', z', K_{\text{belief}}, \text{Age} + 1)$
            Form Bellman-consistent target

$$y = u(c) + \beta\, \mathbb{E}\big[V_\omega(s')\big]$$

        end if
        Update critic by minimizing

$$\mathcal{L}_V = \mathbb{E}\big[(V_\omega(s) - y)^2\big]$$

        Update actor by maximizing the same Bellman-consistent objective
    end for
    if epoch mod belief_update_freq $= 0$ **then**
        // **Outer layer: macro belief update (Deep ALM)**
        Simulate a large population of agents for many periods using current policy
        Compute realized aggregate capital path K_t^{sim}
        Let K_{sim} be the average capital over the tail of the simulation
        Update belief:

$$K_{\text{belief}} \leftarrow (1 - \lambda) K_{\text{belief}} + \lambda K_{\text{sim}}$$

    end if
end for

The algorithm makes the economic logic transparent. In the inner loop, households solve a conditional life-cycle problem while taking the macro environment summarized by K_{belief} as given. In the outer loop, this perceived macro environment is gradually reconciled with the realized aggregate path generated by household decisions. Thus, the entire procedure can be interpreted as a neural generalized market-clearing algorithm: policy learning and equilibrium consistency are solved jointly through alternating micro optimization and macro belief updating.

5.5.6. Results

The adapted method is applied to the simple OLG model. Figure 19 compares the neural network consumption-rate policy with the benchmark solution across age groups. The two policies are close. The remaining small fluctuations reflect undertraining. Newborn agents begin with zero assets, so the Age 0 comparison is concentrated at the boundary. For later ages, the neural network closely follows the benchmark life-cycle shape.

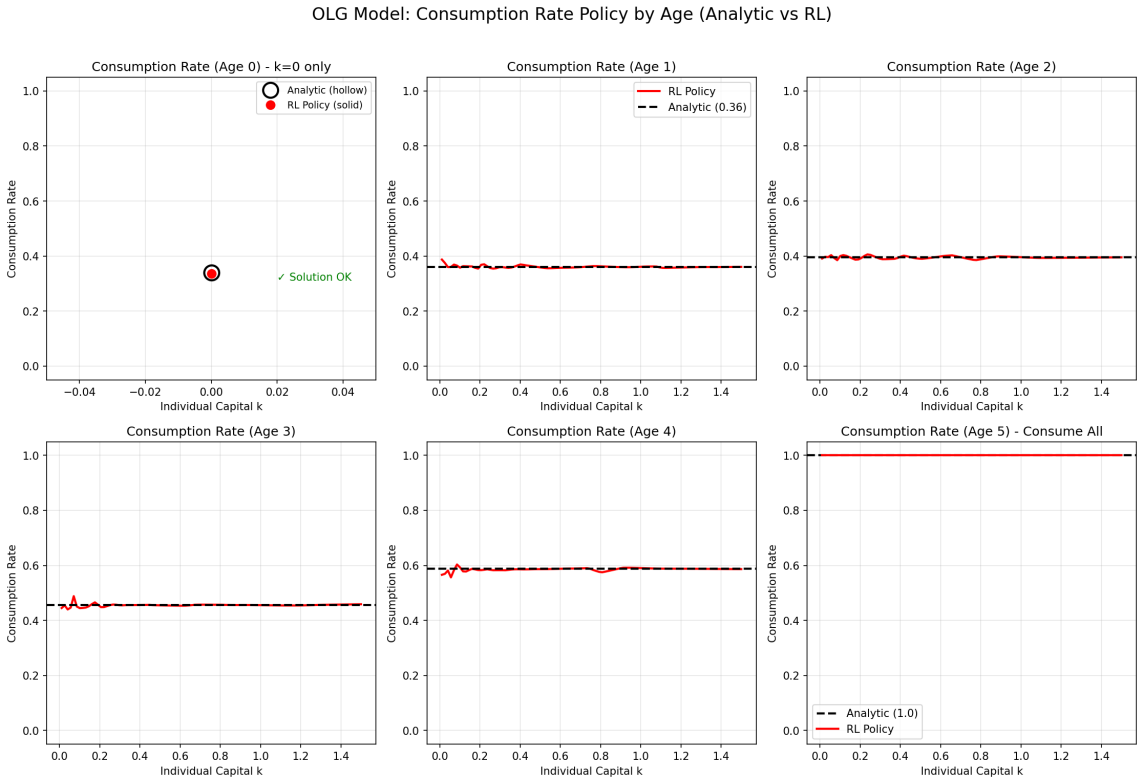


Fig. 19: OLG model: consumption-rate policy by age (analytic benchmark versus RL solution).

The main lesson is that the iAC method can be used beyond stationary infinite-horizon models. Age information, a macro belief, and terminal anchoring provide a parameterization for this simple finite-life model. The example shows how the training logic can be adapted when the economic structure changes.

Cross-model takeaway. Taken together, the Krusell–Smith model, the labor-supply extension, and the OLG environment are three iAC applications. The state spaces are different. In the benchmark problem, the transition equation is simple and model-based gradient propagation is direct. In stationary heterogeneous-agent models, the implementation must account for equilibrium. In OLG models, it must also account for age, terminal conditions, and macro belief updating. The training logic remains the same. The actor and critic are implemented differently across models.

6. Conclusion

This paper studies an informed actor–critic method for dynamic economic models. The idea is simple. Reinforcement learning is useful for continuous control, but economic models are not black boxes. Bellman equations, transition laws, feasibility conditions, and equilibrium mappings are often known. A useful solver should use this information when it is available.

The benchmark consumption–savings problem explains the design of the method. Grid-based dynamic programming is accurate but hard to scale. Pure model-free methods are more flexible, but they are sensitive to targets, expectations, and exploration. The benchmark shows why we use a continuous actor, a value critic, multi-sampling, and known model structure in training.

The iAC method combines these ingredients in one solver. In the reported benchmark problem, it improves stability and accuracy relative to the model-free actor–critic baselines considered here. The same training logic can also be adapted to larger models. We illustrate this in the Krusell–Smith model, in labor-supply extensions, and in an OLG environment. The states, controls, and equilibrium conditions change across models, but the basic training logic stays the same.

The main lesson is that reinforcement-learning tools are more useful in these benchmarks when they are combined with model structure. Neural networks help with scale. Economic equations help with targets, gradients, feasibility, and interpretation. The iAC method is one way to combine these two parts.

Future work can test the method in models with richer aggregate states, multiple assets, and harder equilibrium structures. It can also compare deterministic and stochastic actors in models with both discrete and continuous decisions. Another

useful direction is to combine actor–critic learning with equilibrium belief updating in larger general-equilibrium models.

Appendix A. Formal Results and Proofs

This appendix provides the proofs for the formal statements in Section 3. The proofs are written for the local one-step improvement operator. This keeps the proof aligned with the benchmark consumption–savings problem and with the local Bellman validation used in the two-shock Krusell–Smith comparison.

Lemma 1 (Contraction of the local Bellman operator)**.** *Under Assumption 1, T^{loc} is a contraction under the sup norm:*

$$\|T^{\mathrm{loc}}V_1 - T^{\mathrm{loc}}V_2\|_\infty \leq \beta \|V_1 - V_2\|_\infty. \tag{A.1}$$

The same contraction property holds for any fixed-policy operator T^π.

Proof. For any state s and any two bounded value functions V_1, V_2,

$$\begin{aligned} |(T^{\mathrm{loc}}V_1)(s) - (T^{\mathrm{loc}}V_2)(s)| &\leq \beta \sup_{a \in A_{\mathrm{loc}}(s)} E\left[|V_1(f(s,a,\varepsilon')) - V_2(f(s,a,\varepsilon'))|\right] \\ &\leq \beta \|V_1 - V_2\|_\infty. \end{aligned} \tag{A.2}$$

Taking the supremum over s gives the result. The proof for T^π is the same, without the maximization over a. □

Proof of Proposition 1. Lemma 1 and Banach's fixed point theorem imply that T^{loc} has a unique fixed point $V^{\mathrm{loc},*}$. The representation result follows directly from the definition

$$Q_V(s,a) = u(s,a) + \beta E[V(f(s,a,\varepsilon'))]. \tag{A.3}$$

Substituting this expression into the local Bellman operator gives

$$(T^{\mathrm{loc}}V)(s) = \max_{a \in A_{\mathrm{loc}}(s)} Q_V(s,a). \tag{A.4}$$

Thus a value critic and the known economic transition equation induce the one-step action-value object used by the actor. □

Proof of Proposition 2. Let $T = T^{\mathrm{loc}}$ and $V^* = V^{\mathrm{loc},*}$. In the idealized case, the exact greedy actor makes the implemented update equal to $V_{n+1} = TV_n$. Since $V^* = TV^*$ and T is a contraction by Lemma 1,

$$\|V_{n+1} - V^*\|_\infty = \|TV_n - TV^*\|_\infty \leq \beta \|V_n - V^*\|_\infty. \tag{A.5}$$

Iterating this inequality gives equation (22).

For the approximate update, define $d_n = \|V_n - V^*\|_\infty$. The triangle inequality and the contraction property give

$$\begin{aligned} d_{n+1} &\leq \|V_{n+1} - TV_n\|_\infty + \|TV_n - TV^*\|_\infty \\ &\leq \delta_n + \beta d_n. \end{aligned} \tag{A.6}$$

Recursive substitution yields

$$d_n \leq \beta^n d_0 + \sum_{j=0}^{n-1} \beta^{n-1-j} \delta_j, \tag{A.7}$$

which is equation (24). If $\delta_j \leq \bar{\delta}$, the weighted sum is bounded by $\bar{\delta} \sum_{k=0}^{n-1} \beta^k \leq \bar{\delta}/(1-\beta)$, giving the limsup bound. If $\delta_n \to 0$, the finite initial part of the weighted sum vanishes geometrically and the tail is arbitrarily small; hence the weighted sum converges to zero and $V_n \to V^*$ uniformly. □

Proof of Proposition 3. Fix $V_{\bar{\omega}}$ and write

$$a = \pi_\theta(s), \qquad s' = f(s, a, \varepsilon'). \tag{A.8}$$

On the interior of the feasible region, the chain rule gives

$$\nabla_\theta u(s, \pi_\theta(s)) = D_\theta \pi_\theta(s)^\top u_a(s, a), \tag{A.9}$$

$$\nabla_\theta V_{\bar{\omega}}(f(s, \pi_\theta(s), \varepsilon')) = D_\theta \pi_\theta(s)^\top f_a(s, a, \varepsilon')^\top \nabla_{s'} V_{\bar{\omega}}(s'). \tag{A.10}$$

Combining the two derivatives and interchanging differentiation and integration gives equation (27). If the network output is a consumption or savings rate rather than the final economic action, the same calculation applies to the composite map $(c_\theta(s), x'_\theta(s)) = \Psi(s, \pi_\theta(s))$. At binding borrowing or consumption constraints, the corresponding statement is interpreted as a Kuhn–Tucker or subgradient condition. □

Proof of Proposition 4. For a continuation value V, define

$$g_{\mathrm{iAC}}(\theta; V) = E_s \left[D_\theta \pi_\theta(s)^\top \nabla_a Q_V(s, a)\big|_{a=\pi_\theta(s)} \right], \tag{A.11}$$

where

$$\nabla_a Q_V(s, a) = u_a(s, a) + \beta E \left[f_a(s, a, \varepsilon')^\top \nabla V(f(s, a, \varepsilon')) \right]. \tag{A.12}$$

The learned-Q actor gradient is

$$g_Q(\theta) = E_s\left[D_\theta \pi_\theta(s)^\top \nabla_a Q_\omega(s,a)\big|_{a=\pi_\theta(s)}\right]. \tag{A.13}$$

Therefore

$$\begin{aligned}\|g_Q(\theta) - g_{\mathrm{iAC}}(\theta;V)\| &\le E_s\left[\|D_\theta \pi_\theta(s)\|_{\mathrm{op}}\,\|\nabla_a Q_\omega(s,a) - \nabla_a Q_V(s,a)\|_{a=\pi_\theta(s)}\right] \\ &\le M_\pi E_s\left[\|\nabla_a Q_\omega(s,a) - \nabla_a Q_V(s,a)\|_{a=\pi_\theta(s)}\right], \end{aligned}\tag{A.14}$$

which proves equation (28).

For the value-gradient channel, subtract the iAC gradients generated by $\hat{V}$ and $V^{\mathrm{loc},*}$. The current-payoff derivative cancels, so

$$\begin{aligned}&g_{\mathrm{iAC}}(\theta;\hat{V}) - g_{\mathrm{iAC}}(\theta;V^{\mathrm{loc},*}) \\ &= \beta E_{s,\varepsilon'}\left[D_\theta \pi_\theta(s)^\top f_a(s,a,\varepsilon')^\top\left(\nabla \hat{V}(s') - \nabla V^{\mathrm{loc},*}(s')\right)\right]. \end{aligned}\tag{A.15}$$

Using the assumed operator-norm bounds gives

$$\|g_{\mathrm{iAC}}(\theta;\hat{V}) - g_{\mathrm{iAC}}(\theta;V^{\mathrm{loc},*})\| \le \beta M_\pi M_f \eta_V, \tag{A.16}$$

which proves equation (29).

It remains to prove the stationarity bound. Let $g_t = \nabla J(\theta_t)$ and $\widetilde{g}_t = g_t + e_t$, with $\|e_t\| \le b$. By L-smoothness,

$$J(\theta_{t+1}) \ge J(\theta_t) + \eta g_t^\top \widetilde{g}_t - \frac{L\eta^2}{2}\|\widetilde{g}_t\|^2. \tag{A.17}$$

Young's inequality gives $g_t^\top e_t \ge -\|g_t\|^2/4 - b^2$, so $g_t^\top \widetilde{g}_t \ge 3\|g_t\|^2/4 - b^2$. Also, $\|\widetilde{g}_t\|^2 \le 2\|g_t\|^2 + 2b^2$. Hence

$$J(\theta_{t+1}) - J(\theta_t) \ge \eta\left(\frac{3}{4} - L\eta\right)\|g_t\|^2 - \eta(1+L\eta)b^2. \tag{A.18}$$

For $\eta \le 1/(4L)$, this implies

$$J(\theta_{t+1}) - J(\theta_t) \ge \frac{\eta}{2}\|g_t\|^2 - \frac{5\eta}{4}b^2. \tag{A.19}$$

Summing over $t = 0, \dots, T-1$ and using $J(\theta_T) \le J_{\max}$ gives

$$\frac{1}{T}\sum_{t=0}^{T-1}\|g_t\|^2 \le \frac{2(J_{\max} - J(\theta_0))}{\eta T} + \frac{5}{2}b^2. \tag{A.20}$$

The minimum over $t < T$ is no larger than the average, proving equation (30). □

Remark 1 (Lower-dimensional gradient-learning channel)**.** The previous proposition gives a deterministic error-channel comparison. A simple statistical illustration is obtained by assuming that first-derivative mean-square error for a smooth critic on a d-dimensional domain satisfies

$$E\|\widehat{\nabla F} - \nabla F\|^2 \le C_d N^{-r(d)}, \qquad r(d) = \frac{2(\alpha-1)}{2\alpha+d}, \quad \alpha > 1. \tag{A.21}$$

The learned-Q channel has dimension $d_s + d_a$, while the value-gradient channel has dimension d_s. Under comparable constants,

$$b^2_{\mathrm{MF}}(N) \lesssim M_\pi^2 C_Q N^{-r(d_s+d_a)}, \qquad b^2_{\mathrm{iAC}}(N) \lesssim \beta^2 M_\pi^2 M_f^2 C_V N^{-r(d_s)}. \tag{A.22}$$

Since $r(d_s) > r(d_s+d_a)$ whenever $d_a > 0$, the value-gradient channel can reach a given gradient-accuracy tolerance with fewer samples in this simple smooth-regression model. This is an illustration of the mechanism, not a claim that arbitrary neural-network training attains this nonparametric rate.

Corollary 2 (Euler interpretation in the consumption–savings problem)**.** *In the benchmark consumption–savings model, suppose the actor parameterizes consumption as $c_\theta(w,z)$ and the transition is*

$$w' = R(w - c_\theta(w,z)) + y(z'). \tag{A.23}$$

Then the gradient-penetration calculation gives the Euler residual

$$\mathcal{E}_\theta(w,z) = u_c(c_\theta(w,z)) - \beta R E[V_w(w',z') \mid z]. \tag{A.24}$$

In a statewise or sufficiently rich policy class, setting this residual to zero gives the standard Euler equation. With a finite neural-network parameterization, the first-order condition is the orthogonality of this Euler residual to the policy-network gradients.

Proof. The actor objective is

$$J(\theta) = E\left[u(c_\theta(w,z)) + \beta E[V(w',z') \mid z]\right]. \tag{A.25}$$

Since $\partial w'/\partial c = -R$, differentiation gives

$$\nabla_\theta J(\theta) = E\left[\{u_c(c_\theta(w,z)) - \beta R E[V_w(w',z') \mid z]\}\, \nabla_\theta c_\theta(w,z)\right]. \tag{A.26}$$

At an interior optimum in a statewise policy class, the bracketed term is zero. With a finite neural network, the displayed expression gives the orthogonality condition. The usual envelope condition then gives $u_c(c_t) = \beta R E_t[u_c(c_{t+1})]$ in the statewise case. At a constraint, the equality is replaced by the appropriate Kuhn–Tucker inequality. □

Proof of Proposition 5. Let $T = T^{\mathrm{loc}}$ and let $V^* = V^{\mathrm{loc},*}$. The critic residual assumption gives

$$\|\hat{V} - T^{\hat{\pi}}\hat{V}\|_\infty \le \varepsilon_V. \tag{A.27}$$

The local greedy-error assumption gives

$$0 \le T\hat{V} - T^{\hat{\pi}}\hat{V} \le \varepsilon_{\mathrm{loc}}. \tag{A.28}$$

Therefore

$$\|\hat{V} - T\hat{V}\|_\infty \le \|\hat{V} - T^{\hat{\pi}}\hat{V}\|_\infty + \|T^{\hat{\pi}}\hat{V} - T\hat{V}\|_\infty \le \varepsilon_V + \varepsilon_{\mathrm{loc}}. \tag{A.29}$$

By Lemma 1,

$$\|\hat{V} - V^*\|_\infty \le \frac{\varepsilon_V + \varepsilon_{\mathrm{loc}}}{1-\beta}. \tag{A.30}$$

Since $T^{\hat{\pi}}$ is also a contraction and $V^{\hat{\pi}}$ is its fixed point,

$$\|\hat{V} - V^{\hat{\pi}}\|_\infty \le \frac{\varepsilon_V}{1-\beta}. \tag{A.31}$$

The triangle inequality gives

$$\|V^* - V^{\hat{\pi}}\|_\infty \le \|\hat{V} - V^*\|_\infty + \|\hat{V} - V^{\hat{\pi}}\|_\infty \le \frac{2\varepsilon_V + \varepsilon_{\mathrm{loc}}}{1-\beta}. \tag{A.32}$$

Sampling error, numerical integration error, and target-network lag can be absorbed into the residual terms when they are bounded in the same sup norm. □

Proof of Corollary 1. First fix a state s and write

$$a_g = \pi^g_{\hat{V}}(s), \qquad a^* = \pi^*(s). \tag{A.33}$$

Strong concavity of $Q_{\hat{V}}(s,\cdot)$ implies the quadratic-growth inequality

$$Q_{\hat{V}}(s,a_g) - Q_{\hat{V}}(s,a) \ge \frac{\mu}{2}\|a - a_g\|^2. \tag{A.34}$$

Taking $a = \hat{\pi}(s)$ and using the assumed local greedy gap gives

$$\|\hat{\pi}(s) - a_g\| \le \sqrt{\frac{2\varepsilon_{\mathrm{loc}}(s)}{\mu}}. \tag{A.35}$$

To compare a_g with the true local optimizer a^*, use the interior first-order conditions

$$\nabla_a Q_{\hat{V}}(s,a_g) = 0, \qquad \nabla_a Q_{V^{\mathrm{loc},*}}(s,a^*) = 0. \tag{A.36}$$

Strong concavity of $Q_{V^{\mathrm{loc},*}}(s,\cdot)$ implies strong monotonicity of $-\nabla_a Q_{V^{\mathrm{loc},*}}(s,\cdot)$, hence

$$\mu\|a_g - a^*\| \le \|\nabla_a Q_{V^{\mathrm{loc},*}}(s,a_g) - \nabla_a Q_{V^{\mathrm{loc},*}}(s,a^*)\| = \|\nabla_a Q_{V^{\mathrm{loc},*}}(s,a_g)\|. \tag{A.37}$$

Since $\nabla_a Q_{\hat{V}}(s,a_g) = 0$,

$$\|\nabla_a Q_{V^{\mathrm{loc},*}}(s,a_g)\| = \|\nabla_a Q_{V^{\mathrm{loc},*}}(s,a_g) - \nabla_a Q_{\hat{V}}(s,a_g)\| \le \eta_Q. \tag{A.38}$$

Therefore $\|a_g - a^*\| \le \eta_Q/\mu$. Combining this inequality with the previous bound by the triangle inequality proves equation (37). If $\|f_a\|_{\mathrm{op}} \le M_f$ and $\|\nabla\hat{V} - \nabla V^{\mathrm{loc},*}\|_\infty \le \eta_V$, then

$$\begin{aligned}
&\|\nabla_a Q_{\hat{V}}(s,a) - \nabla_a Q_{V^{\mathrm{loc},*}}(s,a)\| \\
&\le \beta E\left[\|f_a(s,a,\varepsilon')\|_{\mathrm{op}}\|\nabla\hat{V}(f(s,a,\varepsilon')) - \nabla V^{\mathrm{loc},*}(f(s,a,\varepsilon'))\|\right] \\
&\le \beta M_f \eta_V,
\end{aligned} \tag{A.39}$$

so $\eta_Q \le \beta M_f \eta_V$.

For the validation statement, the random variables $\rho(s_i)$ are independent and bounded in $[0,M]$. Hoeffding's inequality gives

$$P\left(E_\nu[\rho(s)] - \frac{1}{N}\sum_{i=1}^{N}\rho(s_i) \ge t\right) \le \exp\left(-\frac{2Nt^2}{M^2}\right). \tag{A.40}$$

Choosing $t = M\sqrt{\log(1/\delta)/(2N)}$ proves equation (38). □

Remark 2 (Scope of the local validation)**.** The proof above uses the local improvement operator because this is the object used in the consumption–savings benchmark and in the Krusell–Smith Bellman validations. It controls a one-household improvement problem with the aggregate belief and price objects held fixed. It is not a proof of global equilibrium uniqueness or of convergence of the outer simulation loop.

Scope of the proof strategy. The convergence result is a perturbed-contraction result for the local Bellman operator. The usefulness of gradient penetration is stated as a bound on the actor-gradient error channel and the resulting stationarity neighborhood. The error-bound results add policy-function and high-probability validation interpretations. All conclusions are local and use strong assumptions. The appendix does not claim a global proof for nonconvex neural-network training or for the outer general-equilibrium loop.

Appendix B. Detailed Algorithms for the Benchmark Methods

This appendix provides the detailed benchmark algorithms used in the main text. The goal is not to document every implementation detail line by line, but to make clear how each solver updates its value object, policy object, and continuation target. For the high-dimensional Krusell–Smith and OLG applications, the corresponding algorithmic logic is already described in the main text and is therefore not repeated here.

Appendix B.1. Model-Based Benchmark Methods

Appendix B.1.1. Value Function Iteration (VFI)

Algorithm 4 Value Function Iteration for the benchmark consumption–savings problem

Discretize the state grid $\mathcal{S}$ and feasible action grid $\mathcal{A}(s)$
Initialize the value function guess $V^{(0)}(s)$ for all $s \in \mathcal{S}$
Set iteration counter $n \leftarrow 0$
repeat
    $\Delta \leftarrow 0$
    for each state $s \in \mathcal{S}$ **do**
        $V_{\text{old}} \leftarrow V^{(n)}(s)$
        for each feasible action $a \in \mathcal{A}(s)$ **do**
            Compute one-period utility $u(s, a)$
            Compute next-period endogenous state
            Compute continuation value

$$\mathcal{C}(s, a) = \mathbb{E}\big[V^{(n)}(s') \mid s, a\big]$$

            Compute Bellman candidate

$$\mathcal{B}(s, a) = u(s, a) + \beta \mathcal{C}(s, a)$$

        end for
        Set

$$V^{(n+1)}(s) = \max_{a \in \mathcal{A}(s)} \mathcal{B}(s, a)$$

        Store policy

$$\pi^{(n+1)}(s) = \arg\max_{a \in \mathcal{A}(s)} \mathcal{B}(s, a)$$

        $\Delta \leftarrow \max\{\Delta, |V^{(n+1)}(s) - V_{\text{old}}|\}$
    end for
    $n \leftarrow n + 1$
until $\Delta <$ tol or $n \geq n_{\max}$
return $V^{(n)}, \pi^{(n)}$

Appendix B.1.2. Bellman-Based Policy Function Iteration (PFI)

Algorithm 5 Bellman-based Policy Function Iteration

Discretize $\mathcal{S}$ and $\mathcal{A}(s)$
Initialize policy guess $\pi^{(0)}(s)$
Set outer iteration counter $m \leftarrow 0$
repeat
    Policy evaluation step:
    Solve for $V^{\pi^{(m)}}$ from

$$V^{\pi^{(m)}}(s) = u\big(s, \pi^{(m)}(s)\big) + \beta \mathbb{E}\big[V^{\pi^{(m)}}(s') \mid s, \pi^{(m)}(s)\big]$$

    Policy improvement step:
    stable $\leftarrow$ true
    for each state $s \in \mathcal{S}$ **do**
        $a_{\text{old}} \leftarrow \pi^{(m)}(s)$
        for each feasible action $a \in \mathcal{A}(s)$ **do**
            Compute

$$\mathcal{B}^{\pi}(s, a) = u(s, a) + \beta \mathbb{E}\big[V^{\pi^{(m)}}(s') \mid s, a\big]$$

        end for
        Set

$$\pi^{(m+1)}(s) = \arg\max_{a \in \mathcal{A}(s)} \mathcal{B}^{\pi}(s, a)$$

        if $\pi^{(m+1)}(s) \neq a_{\text{old}}$ **then**
            stable $\leftarrow$ false
        end if
    end for
    $m \leftarrow m + 1$
until stable = true or $m \geq m_{\max}$
return $\pi^{(m)}, V^{\pi^{(m)}}$

Appendix B.1.3. Monte Carlo Rollout

Algorithm 6 Monte Carlo Rollout with a base continuation policy

Fix a base continuation policy π^{base}
Fix rollout horizon H and number of simulations N
function RolloutValue(s, a)
    sum_return $\leftarrow 0$
    for $i = 1, \ldots, N$ **do**
        $s_0 \leftarrow s$, $a_0 \leftarrow a$, $G_i \leftarrow 0$
        for $t = 0, \ldots, H-1$ **do**
            Apply action a_t in state s_t
            Observe reward r_t and next state s_{t+1}
            $G_i \leftarrow G_i + \beta^t r_t$
            Set continuation action $a_{t+1} \leftarrow \pi^{\text{base}}(s_{t+1})$
        end for
        sum_return $\leftarrow$ sum_return $+ G_i$
    end for
    return sum_return$/N$
end function
for each decision state s **do**
    for each feasible action $a \in \mathcal{A}(s)$ **do**
        Compute $Q^{\text{roll}}(s, a) =$ RolloutValue(s, a)
    end for
    Set rollout policy

$$\pi(s) = \arg\max_{a \in \mathcal{A}(s)} Q^{\text{roll}}(s, a)$$

end for
return π

Appendix B.1.4. Dyna-Q

Algorithm 7 Dyna-Q

Initialize action-value table $Q(s, a)$
Initialize model memory M as empty
for each episode **do**
    Initialize starting state s
    while s is not terminal **do**
        Choose action a using ϵ-greedy(Q)
        Execute a and observe (r, s')
        Direct RL update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \beta \max_{a'} Q(s',a') - Q(s,a) \right]$$

        Store model transition $M(s, a) \leftarrow (r, s')$
        Planning updates:
        for $k = 1, \ldots, K$ **do**
            Sample previously visited $(\tilde{s}, \tilde{a})$ from M
            Retrieve $(\tilde{r}, \tilde{s}') = M(\tilde{s}, \tilde{a})$
            Update

$$Q(\tilde{s},\tilde{a}) \leftarrow Q(\tilde{s},\tilde{a}) + \alpha \left[\tilde{r} + \beta \max_{a'} Q(\tilde{s}',a') - Q(\tilde{s},\tilde{a}) \right]$$

        end for
        $s \leftarrow s'$
    end while
end for
return Q and induced greedy policy

Algorithm 8 Bellman-based deep learning solver

Initialize value network $V_\omega(s)$
if a separate policy network is used **then**
    Initialize policy network $\pi_\theta(s)$
end if
repeat
    Sample a batch of states $\{s_i\}_{i=1}^N$
    for each sampled state s_i **do**
        for each feasible action $a \in \mathcal{A}(s_i)$ **do**
            Compute one-period reward $r(s_i, a)$
            Compute next-state continuation

$$\mathcal{C}(s_i, a) = \mathbb{E}\big[V_\omega(s_i') \mid s_i, a\big]$$

            Form Bellman candidate

$$\mathcal{B}(s_i, a) = r(s_i, a) + \beta \mathcal{C}(s_i, a)$$

        end for
        Set Bellman target

$$y_i = \max_{a \in \mathcal{A}(s_i)} \mathcal{B}(s_i, a)$$

    end for
    Update ω by minimizing

$$\mathcal{L}_B(\omega) = \frac{1}{N} \sum_{i=1}^{N} \big(V_\omega(s_i) - y_i\big)^2$$

    if a policy network is used **then**
        Update θ so that $\pi_\theta(s_i)$ tracks the Bellman-improving action
    end if
until loss and policy/value objects stabilize

Appendix B.2. Model-Free Benchmark Methods

Appendix B.2.1. Q-learning

Algorithm 9 Tabular Q-learning

Initialize $Q(s, a)$ arbitrarily
for each episode **do**
    Initialize starting state s
    while s is not terminal **do**
        Choose action a using ϵ-greedy(Q)
        Execute a and observe reward r and next state s'
        Update

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \beta \max_{a'} Q(s',a') - Q(s,a) \right]$$

        $s \leftarrow s'$
    end while
end for
return Q and greedy policy

Appendix B.2.2. Q-learning Ablation Design

The full Q-learning study varies the following dimensions:

1. **State visitation rule:** Full Batch, Sweep, Shuffled Sweep, and Random sampling.
2. **Expectation construction:** exact expectation, single-sampling, and multi-sampling with different Monte Carlo sample sizes.
3. **Off-grid continuation evaluation:** nearest-neighbor versus interpolation.
4. **Grid density:** coarse, intermediate, and fine state–action discretizations.
5. **Learning dynamics:** training length, batch size, minimum learning rate, and seed sensitivity.

These ablations are used in the main text to distinguish theoretical upper bounds from practical bottlenecks.

Appendix B.2.3. DQN

Algorithm 10 Deep Q-Network

Initialize online network $Q_\theta(s, a)$
Initialize target network $Q_{\bar{\theta}}(s, a)$ with $\bar{\theta} \leftarrow \theta$
Initialize replay buffer D
for each episode **do**
    Initialize state s
    while s is not terminal **do**
        Choose action a by ϵ-greedy exploration using $Q_\theta(s, a)$
        Execute a and observe (r, s', done)
        Store $(s, a, r, s', \text{done})$ in replay buffer D
        Sample minibatch B from D
        for each transition $(s_i, a_i, r_i, s'_i, \text{done}_i)$ in B **do**
            if done_i **then**
                $y_i \leftarrow r_i$
            else
                Form continuation target

$$y_i \leftarrow r_i + \beta \max_{a'} Q_{\bar{\theta}}(s'_i, a')$$

            end if
        end for
        Update online network by minimizing

$$\mathcal{L}_{\text{DQN}}(\theta) = \frac{1}{|B|} \sum_i (Q_\theta(s_i, a_i) - y_i)^2$$

        Periodically copy target parameters $\bar{\theta} \leftarrow \theta$
        $s \leftarrow s'$
    end while
end for

Algorithm 11 Deep Deterministic Policy Gradient

Initialize deterministic actor $\pi_\theta(s)$
Initialize critic $Q_\phi(s, a)$
Initialize target networks $\pi_{\overline{\theta}}, Q_{\overline{\phi}}$
Initialize replay buffer D
for each episode **do**
    Initialize state s
    while s is not terminal **do**
        Choose exploratory action

$$a \leftarrow \pi_\theta(s) + \text{noise}$$

        Execute a and observe (r, s', done)
        Store $(s, a, r, s', \text{done})$ in D
        Sample minibatch B from D
        for each transition $(s_i, a_i, r_i, s'_i, \text{done}_i)$ in B **do**
            if done_i **then**
                $y_i \leftarrow r_i$
            else
                $y_i \leftarrow r_i + \beta Q_{\overline{\phi}}(s'_i, \pi_{\overline{\theta}}(s'_i))$
            end if
        end for
        Update critic by minimizing

$$\mathcal{L}_Q(\phi) = \frac{1}{|B|} \sum_i (Q_\phi(s_i, a_i) - y_i)^2$$

        Update actor by maximizing

$$\mathcal{J}_\pi(\theta) = \mathbb{E}_{s \sim B}[Q_\phi(s, \pi_\theta(s))]$$

        Soft-update target networks

$$\overline{\theta} \leftarrow \tau\theta + (1 - \tau)\overline{\theta}, \qquad \overline{\phi} \leftarrow \tau\phi + (1 - \tau)\overline{\phi}$$

        $s \leftarrow s'$
    end while
end for

Appendix B.2.5. SAC

Algorithm 12 Soft Actor-Critic

Initialize stochastic policy $\pi_\theta(a \mid s)$
Initialize critics Q_{ϕ_1}, Q_{ϕ_2}
Initialize target critics or target value object
Initialize replay buffer D
for each episode **do**
    Initialize state s
    while s is not terminal **do**
        Sample action $a \sim \pi_\theta(\cdot \mid s)$
        Execute a and observe (r, s', done)
        Store $(s, a, r, s', \text{done})$ in D
        Sample minibatch B from D
        for each transition $(s_i, a_i, r_i, s'_i, \text{done}_i)$ in B **do**
            Form soft critic target

$$y_i = r_i + \beta\, \mathbb{E}_{a' \sim \pi_\theta(\cdot \mid s'_i)} \Big[\min_k Q_{\bar{\phi}_k}(s'_i, a') - \alpha \log \pi_\theta(a' \mid s'_i) \Big]$$

        end for
        Update each critic by minimizing

$$\mathcal{L}_{Q_k}(\phi_k) = \frac{1}{|B|} \sum_i \big(Q_{\phi_k}(s_i, a_i) - y_i \big)^2$$

        Update policy by minimizing

$$\mathcal{L}_\pi(\theta) = \mathbb{E}_{s \sim B,\, a \sim \pi_\theta} \big[\alpha \log \pi_\theta(a \mid s) - \min_k Q_{\phi_k}(s, a) \big]$$

        If adaptive temperature is used, update α toward target entropy
        $s \leftarrow s'$
    end while
end for

Appendix B.3. iAC Method for the Benchmark Problem
Appendix B.3.1. DDPG with Value Critic and Model-Based Gradient Propagation

Algorithm 13 iAC method for the benchmark consumption–savings problem

Initialize deterministic actor $\pi_\theta(s)$
Initialize value critic $V_\omega(s)$
Initialize frozen actor parameters $\overline{\theta} \leftarrow \theta$
Initialize target critic $V_{\overline{\omega}}(s) \leftarrow V_\omega(s)$
repeat
    Sample a batch of replayed states (and stored shock draws, if used) $\{s_i\}_{i=1}^N$
    for each state s_i **do**
        Actor outputs an action parameter and the feasibility map converts it into economic choices

$$b_i = \pi_{\overline{\theta}}(s_i), \qquad (c_i, x_i') = \Psi(s_i, b_i)$$

        Use the known transition equation and the chosen integration rule to form the continuation target

$$\widehat{\mathcal{C}}_i = \mathbb{E}_{\varepsilon'}\left[V_{\overline{\omega}}(f(s_i, x_i', \varepsilon'))\right]$$

        Construct Bellman-consistent target

$$y_i = u(c_i) + \beta\, \widehat{\mathcal{C}}_i$$

    end for
    Update critic by minimizing

$$\mathcal{L}_V(\omega) = \frac{1}{N}\sum_{i=1}^{N}\big(V_\omega(s_i) - y_i\big)^2$$

    Update actor by maximizing

$$\mathcal{J}_\pi(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[u(c_{\theta,i}) + \beta\, \mathbb{E}_{\varepsilon'} V_{\overline{\omega}}(f(s_i, x'_{\theta,i}, \varepsilon'))\right], \quad (c_{\theta,i}, x'_{\theta,i}) = \Psi(s_i, \pi_\theta(s_i))$$

    Apply model-based gradient propagation through $\Psi(\cdot)$ and the known transition function $f(\cdot)$ while holding $V_{\overline{\omega}}$ fixed
    Hard-update frozen actor: $\overline{\theta} \leftarrow \theta$
    Soft-update the target critic: $\overline{\omega} \leftarrow \tau_\omega \omega + (1 - \tau_\omega)\overline{\omega}$
until training losses and policy stabilize

Appendix C. Supplementary Figures for the Two-Shock Krusell–Smith Comparison

This appendix reports supplementary diagnostics for the two-shock Krusell–Smith comparison in Section 5.2. The main text uses the no-capital-cap simulations and separates the baseline comparison from the single-seed low-capital refinement. The figures below document the same logic under multiple seeds and under the alternative capped DeepHAM simulation.

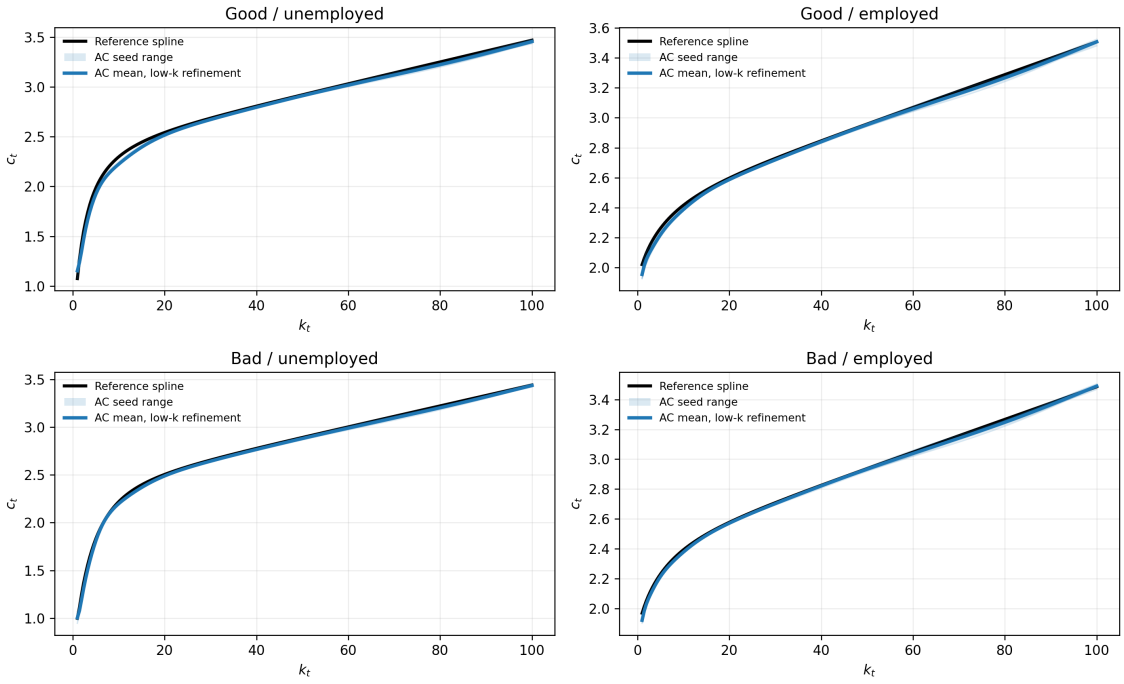


Fig. C.20: iAC policy with low-capital sample refinement across multiple seeds. The thick line is the seed average and the shaded region shows the cross-seed range.

Appendix D. All experiments on Q-learning

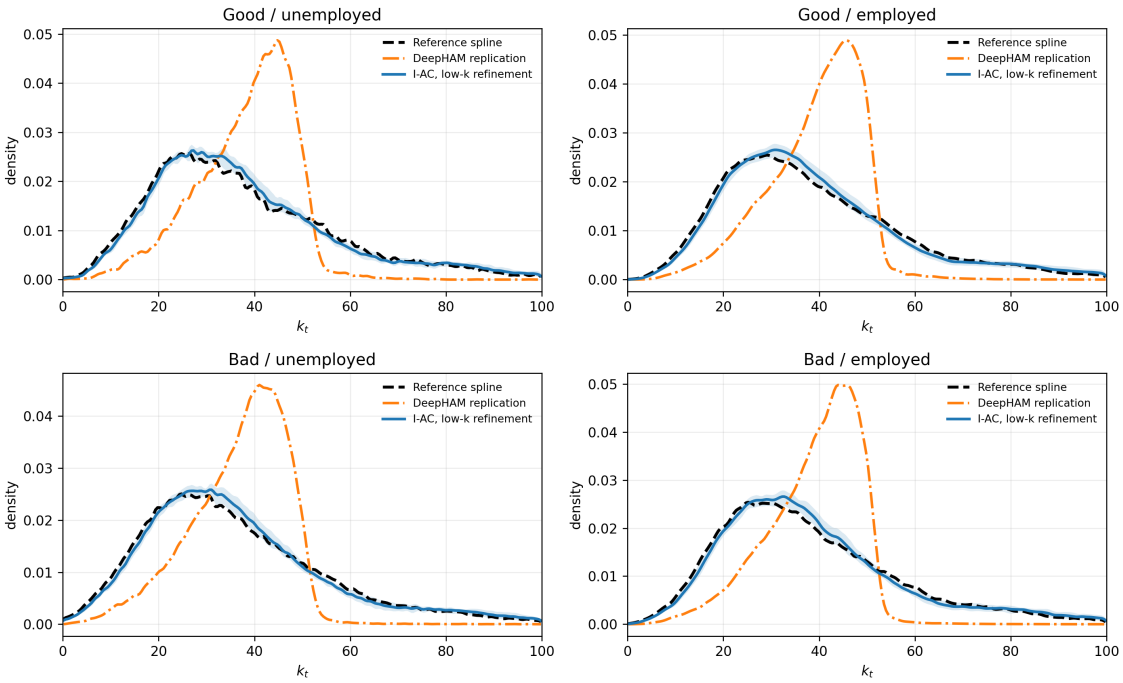


Fig. C.21: Policy-implied capital distributions with the low-capital iAC refinement across multiple seeds. No upper capital cap is imposed on the DeepHAM simulation.

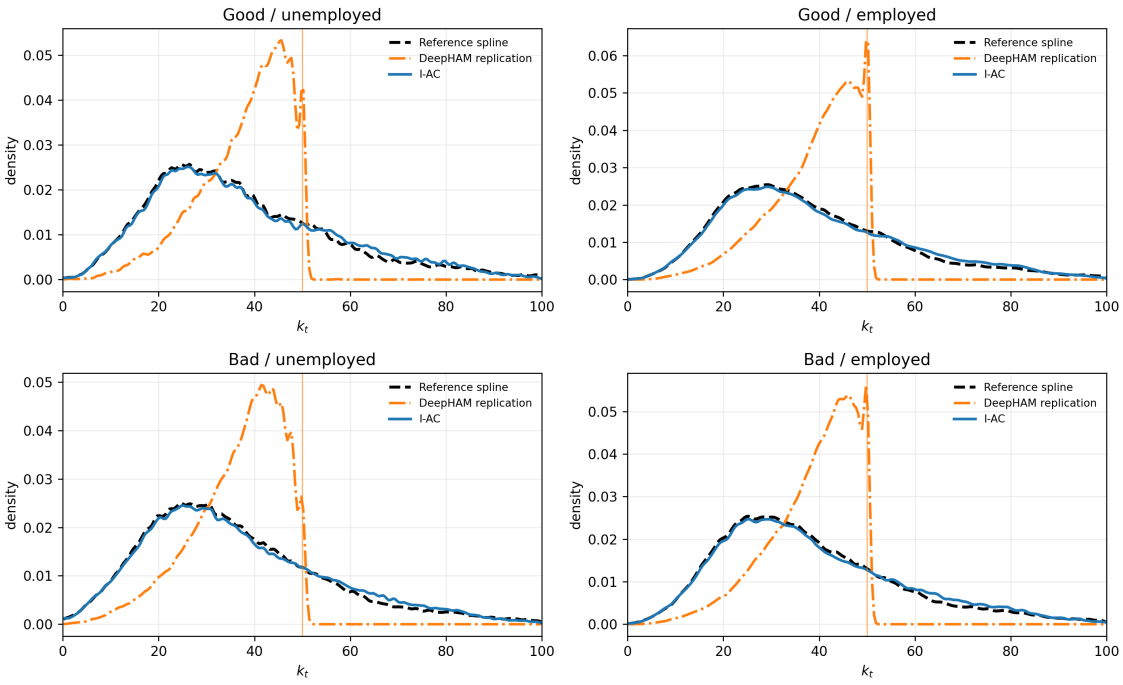


Fig. C.22: Policy-implied capital distributions for the baseline comparison when the DeepHAM simulation is capped at $k \leq 50$.

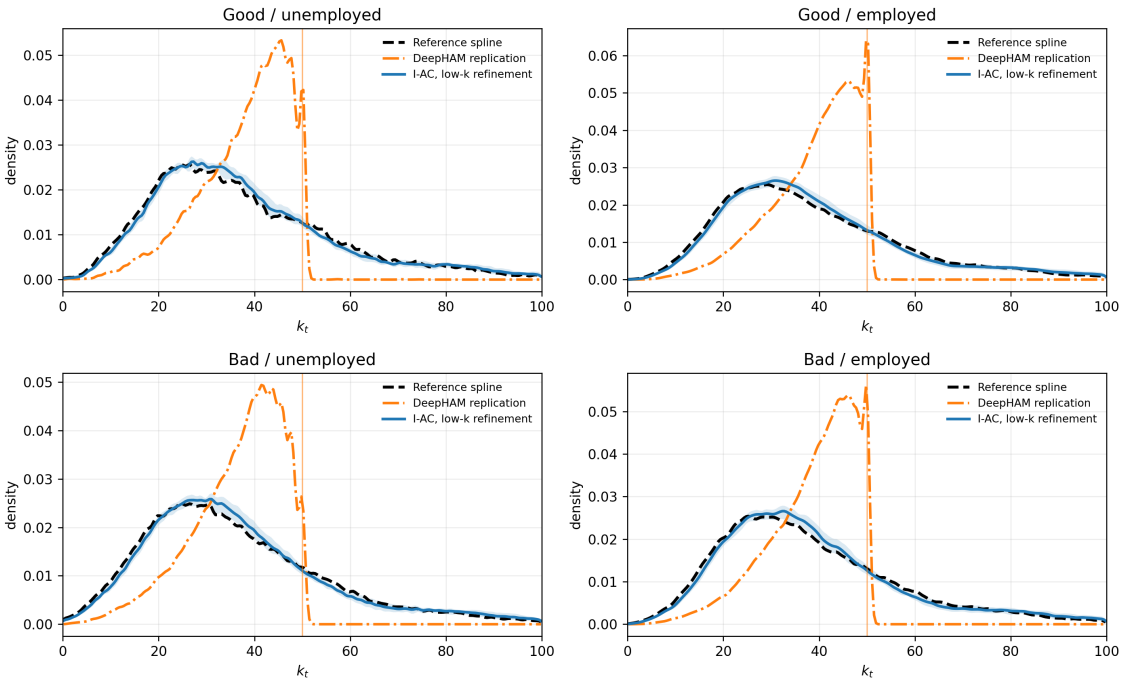


Fig. C.23: Policy-implied capital distributions with the low-capital iAC refinement across multiple seeds when the DeepHAM simulation is capped at $k \leq 50$.

Table D.6: Summary of All Experimental Results (Test 1 - Test 78)

Test	Samp	Exp	Grid	Upd/Int	Batch	Ep	Time(s)	MSE_C
1	Ran	MC50	200x200	S/Y	200	1M	395.63	3.65e-04
2	Ran	Ex	200x200	S/Y	200	1M	165.44	1.50e-04
3	Swp	MC50	200x200	S/Y	200	1M	363.79	3.93e-04
4	Swp	Ex	200x200	S/Y	200	1M	129.99	1.68e-04
5	Ran	MC100	200x200	S/Y	200	500k	293.18	3.61e-04
6	Ran	MC100	200x200	S/Y	200	1M	584.47	2.80e-04
7	Ful	MC50	200x200	S/Y	1400	500k	2530.94	3.63e-04
8	Ran	MC300	200x200	S/Y	200	1M	1536.45	2.69e-04
9	Ran	MC50	200x200	S/Y	200	5M	1860.12	3.80e-04
10	Ran	Ex	200x200	S/Y	200	5M	777.16	1.36e-04
11	Ful	MC20	200x100	A/Y	1400	50k	13096.53	3.77e-04
12	Ful	Ex	200x200	A/Y	1400	10k	454.56	1.21e-05
13	Shf	Ex	200x200	A/Y	1400	50k	2228.26	1.21e-05
14	Swp	MC1	200x200	S/N	200	200k	21.84	2.75e-03
15	Ful	MC1	200x200	S/N	1400	200k	183.85	2.52e-03
16	Swp	MC1	200x200	A/N	200	200k	176.35	3.26e-03
17	Ful	MC1	200x200	A/N	1400	200k	1210.63	4.97e-03
18	Shf	MC1	200x200	S/N	200	200k	26.23	4.22e-03
19	Ran	MC1	200x200	S/N	200	200k	26.25	4.77e-03
20	Swp	MC10	200x200	S/N	200	200k	30.40	8.98e-04
21	Swp	Ex	200x200	S/N	200	200k	26.97	5.41e-04
22	Swp	MC1	200x200	S/Y	200	200k	22.77	3.44e-03
23	Swp	MC50	200x200	S/Y	200	200k	75.02	4.34e-04
24	Ful	Ex	200x200	S/Y	1400	200k	228.66	1.21e-05
25	Ran	MC50	200x200	S/Y	200	100k	78.71	5.90e-04
26	Ran	MC50	200x200	S/Y	200	500k	187.22	4.14e-04
27	Ran	MC100	200x200	S/Y	200	200k	115.80	5.60e-04
28	Ran	Ex	200x200	S/Y	200	200k	30.14	2.17e-04
29	Shf	MC50	200x200	S/Y	200	500k	202.90	5.20e-04
30	Ran	MC200	200x200	S/Y	200	5M	7309.94	2.17e-04
31	Ran	MC200	200x200	S/Y	500	1M	2236.57	1.57e-04
32	Ran	MC200	200x200	S/Y	1000	1M	4684.81	1.57e-04
33	Shf	MC50	200x200	S/Y	200	1M	559.88	6.38e-04
34	Ran	MC200	200x200	S/Y	200	1M	940.74	2.23e-04
35	Ran	MC200	200x200	S/Y	200	1M	933.43	1.99e-04

Table D.6 – Continued

Test	Samp	Exp	Grid	Upd/Int	Batch	Ep	Time(s)	MSE_C
36	Ran	MC200	200x200	S/Y	200	1M	939.83	3.05e-04
37	Ran	MC200	200x200	S/Y	500	1M	2194.61	2.53e-04
38	Ran	MC200	200x200	S/Y	500	1M	2205.91	1.25e-04
39	Ran	MC200	200x200	S/Y	200	2M	2064.59	1.87e-04
40	Ran	MC200	200x200	S/Y	200	2M	2221.33	2.80e-04
41	Ran	MC200	200x200	S/Y	200	2M	2806.14	2.91e-04
42	Ran	MC100	200x200	S/Y	200	1M	599.03	2.66e-04
43	Ran	MC25	200x200	S/Y	200	1M	247.58	3.01e-04
44	Ran	MC1	200x200	S/Y	200	1M	125.25	2.13e-03
45	Ran	MC500	200x200	S/Y	200	1M	2392.71	2.18e-04
46	Ran	Ex	200x200	S/Y	200	1M	156.16	2.16e-04
47	Ran	MC100	200x200	S/Y	200	2M	1193.79	2.44e-04
48	Ran	MC200	200x200	S/Y	2000	500k	5442.37	1.63e-04
49	Ran	MC200	200x200	S/Y	200	5M	5249.64	2.93e-04
50	Ran	MC500	200x200	S/Y	200	5M	11718.95	1.96e-04
51	Swp	MC200	200x200	S/Y	200	5M	5030.94	2.07e-04
52	Ran	MC200	200x200	S/Y	200	5M	5145.03	2.20e-04
53	Ran	MC200	200x200	S/Y	200	5M	5150.92	1.41e-04
54	Ran	MC200	200x200	S/Y	2000	1M	11451.08	2.05e-04
55	Swp	Ex	200x200	S/Y	200	200k	26.62	1.84e-04
56-1	Ran	MC100	200x200	S/Y	200	200k	122.61	4.49e-04
56-2	Ran	MC100	200x200	S/Y	200	200k	120.67	4.24e-04
56-3	Ran	MC100	200x200	S/Y	200	200k	119.65	6.31e-04
57	Ran	MC100	100x200	S/Y	200	200k	126.99	2.56e-04
58	Ran	MC100	300x200	S/Y	200	200k	125.22	4.01e-04
59	Ran	MC100	500x200	S/Y	200	200k	134.94	1.88e-03
60	Ran	MC100	200x100	S/Y	200	200k	74.39	2.96e-04
61	Ran	MC100	200x300	S/Y	200	200k	180.67	7.69e-04
62	Ran	MC100	500x300	S/Y	200	200k	176.56	5.45e-03
63	Ran	MC100	500x500	S/Y	200	200k	278.48	1.85e-02
64	Ran	MC200	500x500	S/Y	200	200k	510.78	1.75e-02
65-1	Ran	MC200	100x100	S/Y	200	1M	797.74	1.63e-04
66	Ran	MC200	500x500	S/Y	200	2M	5254.79	2.08e-04
67	Ran	MC400	500x500	S/Y	200	1M	4959.58	1.49e-04
68	Ran	MC400	1000x500	S/Y	200	1M	4930.93	1.70e-04

Table D.6 – Continued

Test	Samp	Exp	Grid	Upd/Int	Batch	Ep	Time(s)	MSE_C
69	Ran	MC400	100x100	S/Y	200	1M	1088.55	1.45e-04
70	Ran	Ex	100x100	S/Y	200	1M	114.73	1.25e-04
71-1	Ran	Ex	200x200	S/Y	200	200k	37.98	2.64e-04
71-2	Ran	Ex	200x200	S/Y	200	200k	32.23	2.15e-04
71-3	Ran	Ex	200x200	S/Y	200	200k	32.98	1.76e-04
72-1	Ran	Ex	200x200	S/Y	200	1M	245.79	1.77e-04
72-2	Ran	Ex	200x200	S/Y	200	1M	289.38	1.40e-04
72-3	Ran	Ex	200x200	S/Y	200	1M	205.52	2.24e-04
73-1	Ran	MC50	200x200	S/Y	200	1M	588.03	3.56e-04
73-2	Ran	MC50	200x200	S/Y	200	1M	329.96	4.42e-04
73-3	Ran	MC50	200x200	S/Y	200	1M	326.83	3.45e-04
74-1	Ran	MC50	200x200	S/Y	200	200k	64.85	5.05e-04
74-2	Ran	MC50	200x200	S/Y	200	200k	64.48	4.60e-04
74-3	Ran	MC50	200x200	S/Y	200	200k	64.91	4.39e-04
75-1	Ran	Ex	100x100	S/Y	200	1M	104.42	7.83e-05
75-2	Ran	Ex	100x100	S/Y	200	1M	104.73	1.10e-04
75-3	Ran	Ex	100x100	S/Y	200	1M	103.37	7.14e-05
76	Ran	Ex	50x50	S/Y	200	1M	87.64	2.07e-04
77	Ran	MC200	50x50	S/Y	200	1M	335.20	2.65e-04
78	Ran	MC100	50x50	S/Y	200	1M	214.68	3.72e-04

Appendix E. Supplementary Design Notes

Appendix E.1. Return to Structure in the Benchmark Problem

This appendix collects design notes that are useful for interpreting the benchmark comparison but are secondary to the main results. The benchmark implementation of iAC is introduced in Section 2, while Section 4 compares the RL and Bellman-based ingredients that motivate the final design.

The benchmark comparison highlights several useful ingredients: global approximation, continuous control, actor–critic separation, and exploration or replay when they are needed. It also highlights a cost of pure model-free learning in this setting. A model-free method treats the transition map mainly as something to be sampled from, even when the main economic equations are already known.

In the benchmark consumption–savings problem, the transition law, reward function, and feasibility conditions are all known and cheap to evaluate. Using these ob-

jects directly inside the informed actor–critic architecture reduces the amount that must be learned from simulated data.

In this sense, the benchmark instance can be read as a relatively structure-rich iAC implementation, not as a separate method. In high-dimensional models, the actor–critic shell and the Bellman discipline remain the same, but the use of known structure becomes more selective because not every structural component can be imposed directly at low cost.

Table E.7: Representative numerical results of the model-free methods shown in the main text

Method	Configuration	MSE_C	MSE_V	$\mathrm{MSE}_V^{\mathrm{aligned}}$
DQN	Multi	9.73×10^{-4}	2.02×10^{-4}	–
DDPG	Single	1.64×10^{-3}	1.77×10^{-4}	–
SAC	Multi $\alpha = 0.05$	3.37×10^{-2}	8.95×10^{0}	3.73×10^{-2}
SAC	Multi adaptive α	1.72×10^{-4}	9.91×10^{-3}	1.28×10^{-4}

Appendix E.2. Supplementary Synthesis of the iAC Design Lines

The benchmark leaves a design logic. VFI and Bellman-PFI provide the numerical anchor and show the value of recursive structure, but they also show the scaling cost of grids. Monte Carlo rollout and Dyna-Q show that known transitions and model-assisted updates can improve action evaluation, while also revealing the cost of repeated simulation and tabular state-action storage. Q-learning connects this Bellman logic to RL notation: with accurate information it can approach the VFI benchmark, but its double-discrete state-action representation produces policy-recovery problems in continuous-control settings.

The scalable-control line gives the other half of the design. DQN shows the usefulness of neural approximation but keeps a discrete action set. DDPG removes the action grid and separates policy improvement from value fitting, while SAC shows that extra exploration mechanisms can stabilize purely model-free learning. The Bellman residual method of Maliar et al. (2021) is already model-informed, but its mixed policy–value objective motivates a separated training structure. The iAC method combines these messages: it keeps neural approximation, continuous economic controls, and separated actor–critic updates, while using known payoff, feasibility, and transition equations inside the one-step Bellman objective.

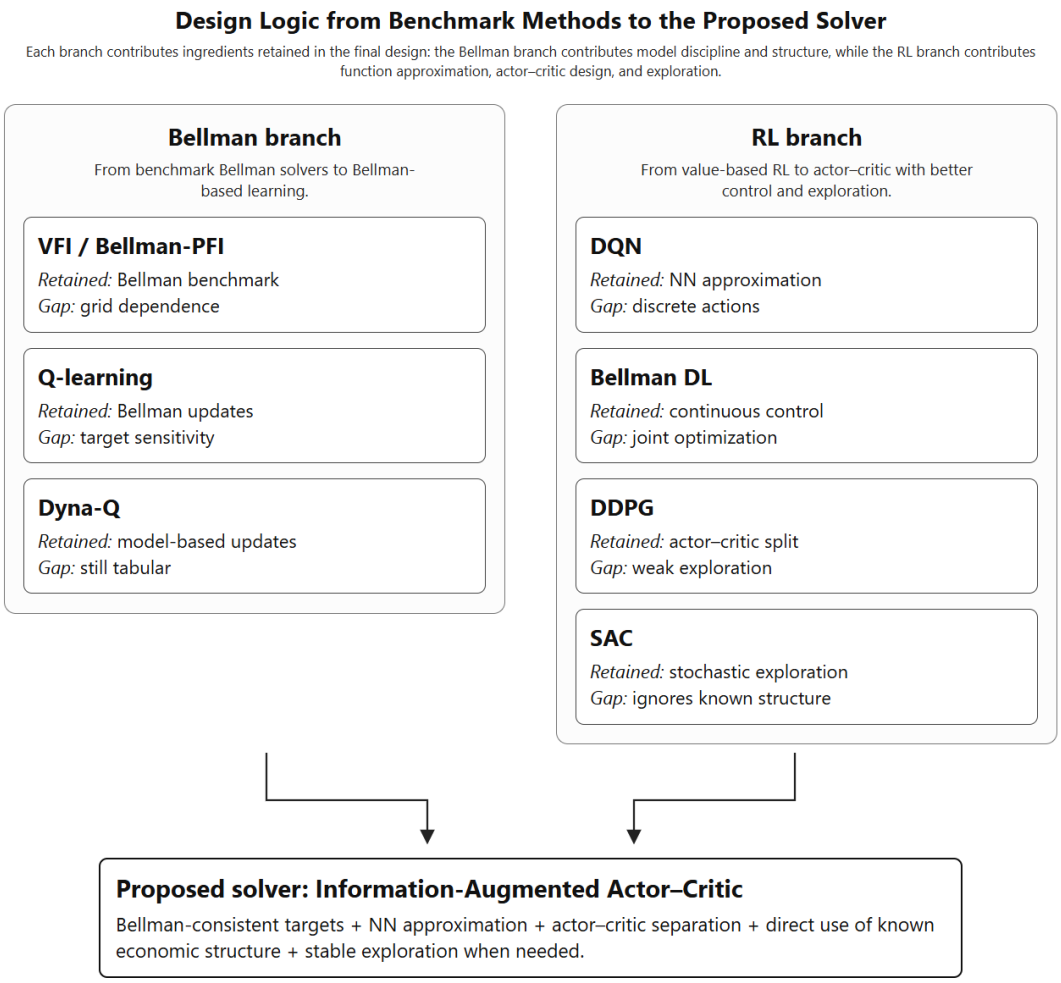


Fig. E.24: Design logic from benchmark methods to the proposed iAC method. The two columns should be read as two mechanism sources rather than two competing method blocks: model-information methods contribute recursive discipline and direct use of known structure, while scalable-control methods contribute neural approximation, continuous actions, and separated actor–critic training.

References

Azinovic, M., Gaegauf, L., Scheidegger, S., 2022. Deep equilibrium nets. International Economic Review 63, 1471–1525.

Bertsekas, D.P., Tsitsiklis, J.N., 1996. Neuro-Dynamic Programming. Athena Scientific, Belmont, MA.

Brumm, J., Scheidegger, S., 2017. Using adaptive sparse grids to solve high-dimensional dynamic models. Econometrica 85, 1575–1612. doi:10.3982/ECTA12216.

Cai, Y., Judd, K.L., 2010. Stable and efficient computational methods for dynamic programming. Journal of the European Economic Association 8, 626–634.

Cai, Y., Judd, K.L., 2013. Shape-preserving dynamic programming. Mathematical Methods of Operations Research 77, 407–421. doi:10.1007/s00186-012-0406-5.

Cai, Y., Judd, K.L., 2014. Advances in numerical dynamic programming and new applications, in: Schmedders, K., Judd, K.L. (Eds.), Handbook of Computational Economics. Elsevier. volume 3. chapter 8, pp. 479–516. doi:10.1016/B978-0-444-52980-0.00008-6.

Carroll, C.D., 2006. The method of endogenous gridpoints for solving dynamic stochastic optimization problems. Economics letters 91, 312–320.

Den Haan, W.J., 1997. Solving dynamic models with aggregate shocks and heterogeneous agents. Macroeconomic dynamics 1, 355–386.

Den Haan, W.J., Marcet, A., 1990. Solving the stochastic growth model by parameterizing expectations. Journal of Business & Economic Statistics 8, 31–34. doi:10.1080/07350015.1990.10509770.

Duffy, J., McNelis, P.D., 2001. Approximating and simulating the stochastic growth model: Parameterized expectations, neural networks, and the genetic algorithm. Journal of Economic Dynamics and Control 25, 1273–1303.

Fernández-Villaverde, J., 2025. Deep Learning for Solving Economic Models. NBER Working Paper 34250. National Bureau of Economic Research. doi:10.3386/w34250.

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

Grüne, L., Semmler, W., Stieler, M., 2015. Using nonlinear model predictive control for dynamic decision problems in economics. Journal of Economic Dynamics and Control 60, 112–133. doi:10.1016/j.jedc.2015.08.010.

Haarnoja, T., Zhou, A., Abbeel, P., Levine, S., 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: Proceedings of the 35th International Conference on Machine Learning, pp. 1861–1870.

Han, J., Yang, Y., E, W., 2026. DeepHAM: A global solution method for heterogeneous agent models with aggregate shocks. Quantitative Economics 17, 297–341. doi:10.3982/QE2190.

Jirnyi, A., Lepetyuk, V., 2011. A Reinforcement Learning Approach to Solving Incomplete Market Models with Aggregate Uncertainty. Working Paper Serie AD 2011-21. Instituto Valenciano de Investigaciones Económicas. doi:10.2139/ssrn.1832745.

Judd, K.L., 1998. Numerical Methods in Economics. MIT Press, Cambridge, MA.

Judd, K.L., Maliar, L., Maliar, S., 2011. Numerically stable and accurate stochastic simulation approaches for solving dynamic economic models. Quantitative Economics 2, 173–210. doi:10.3982/QE14.

Krueger, D., Kubler, F., 2004. Computing equilibrium in OLG models with stochastic production. Journal of Economic Dynamics and Control 28, 1411–1436. doi:10.1016/S0165-1889(03)00111-8.

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

Li, B., Maliar, S., Zhang, P., 2026. Solving discrete-continuous dynamic choice models: A soft-to-hard AI solver with an application to sovereign default. Working paper.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 .

Maliar, L., Maliar, S., 2005. Solving nonlinear dynamic stochastic models: An algorithm computing value function by simulations. Economics Letters 87, 135–140.

Maliar, L., Maliar, S., 2013. Envelope condition method versus endogenous grid method for solving dynamic programming problems. Economics Letters 120, 262–266.

Maliar, L., Maliar, S., 2014. Numerical methods for large-scale dynamic economic models, in: Schmedders, K., Judd, K.L. (Eds.), Handbook of Computational Economics. Elsevier. volume 3. chapter 7, pp. 325–477. doi:10.1016/B978-0-444-52980-0.00007-4.

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

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533. doi:10.1038/nature14236.

Rust, J., 1997. Using randomization to break the curse of dimensionality. Econometrica 65, 487–516.

Schaab, A., 2020. Micro and macro uncertainty. Available at SSRN 4099000.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 .

Smith, A.A., 1993. Estimating nonlinear time-series models using simulated vector autoregressions. Journal of Applied Econometrics 8, S63–S84.

Stachursky, J., 2009. Economic Dynamics: Theory and Computation. MIT Press, Cambridge, MA.

Sutton, R.S., 1991. Dyna, an integrated architecture for learning, planning, and reacting. SIGART Bulletin 2, 160–163. doi:10.1145/122344.122377.

Tesauro, G., Galperin, G.R., 1996. On-line policy improvement using monte-carlo search, in: Mozer, M.C., Jordan, M.I., Petsche, T. (Eds.), Advances in Neural Information Processing Systems 9, MIT Press. pp. 1068–1074.

Watkins, C.J.C.H., Dayan, P., 1992. Q-learning. Machine Learning 8, 279–292. doi:10.1007/BF00992698.

Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8, 229–256. doi:10.1007/BF00992696.

Yang, Y., Wang, C., Schaab, A., Moll, B., 2025. Structural Reinforcement Learning for Heterogeneous Agent Macroeconomics. Research Paper 25-109. Swiss Finance Institute. doi:10.2139/ssrn.5988734.