arXiv:2607.23026v1 [cs.LG] 25 Jul 2026

All in One: Generative Modeling as Mean-Field Game Design

Kun Zhao
kun.zhao@vumc.org
Xu Chen
xc2412@columbia.edu

Abstract

Mean-field games (MFGs) offer a unifying lens on continuous-time generative modeling: choosing a cost tuple $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$ recovers twelve prominent models—Continuous Normalizing Flows, OT-Flow, Score-based Models, Schrödinger Bridges, and more—as special cases of one variational problem. Yet two dimensions of this space remain entirely unexplored: the interaction term $\mathcal{I}$ is set to zero in many existing models, and the rich family of MFG solvers has never been applied to generative modeling. We address both gaps with MFGLab an open-source PyTorch library whose primary API is the cost tuple: all twelve models are specified by four composable cost functions, and the training loop, log-Jacobian, and reverse-ODE sampler are shared automatically. We additionally propose DI-Flow, a novel cost design that uses a differentiable entropy functional to encourage mode coverage, and provide learning-based MFG solvers that substantially outperform neural training on stochastic-dynamics rows. Experiments on two 2-D benchmarks confirm that the unified API is lossless relative to hand-coded implementations.

1 Introduction

Mean-field games (MFGs) [12, 13] provide a tractable framework for modelling strategic interactions in a large, rational population. As the number of agents $N \to \infty$, the intractable Nash equilibrium of the N-player game converges to a *mean-field equilibrium* (MFE)—a self-consistent pair of a backward Hamilton-Jacobi-Bellman (HJB) equation encoding each agent's optimal response and a forward Fokker-Planck-Kolmogorov (FPK) equation propagating the population density. This mean-field limit bypasses the curse of dimensionality of finite-player formulations and has been applied to economics, autonomous driving [4, 29], epidemiology, and energy systems.

Solving the MFE is a central algorithmic challenge, with approaches ranging from classical fixed-point iteration between the HJB and FPK equations [14] to reinforcement-learning-based dynamic programming [8, 21] and recent semi-gradient methods that update policy and population jointly [28, 29].

Beyond these classical multi-agent settings, a striking connection to *generative modeling* was established in [27]: every major continuous-time generative model— Continuous Normalizing Flows [3, 7], OT-Flow [20], Score-based Generative Models [24, 10], Schrödinger Bridges, Wasserstein Gradient Flows [17, 5], and Boltzmann Generators [19]—arises as a special case of a single MFG variational problem parameterised by a cost tuple $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$:

$$\inf_{v,\,\rho} \underbrace{\mathcal{M}(\rho_T)}_{\text{terminal cost}} + \int_0^T \underbrace{\mathcal{I}(\rho_t)}_{\text{interaction term}}\, dt + \int_0^T \int \underbrace{\mathcal{L}(x, v(x,t))}_{\text{running cost}}\, \rho(x,t)\, dx\, dt, \tag{1}$$

Preprint.

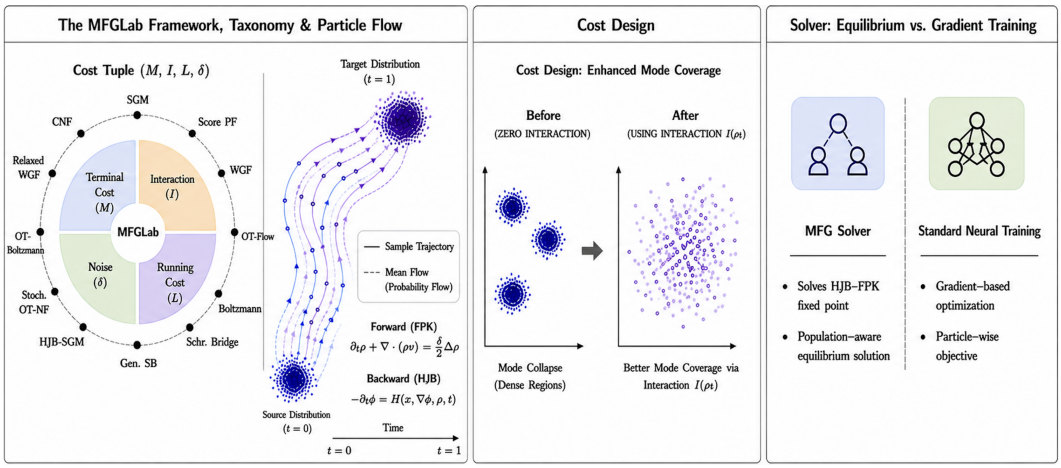

Figure 1: Overview of this paper's three contributions. **Framework**: MFGLAB unifies all twelve MFG taxonomy models under a single cost-tuple API $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$; changing only the cost tuple produces any model in the taxonomy. **Cost Design**: a principled framework for designing the interaction term $\mathcal{I}(\rho_t)$; DI-FLOW is a concrete instantiation using a differentiable KDE entropy functional that achieves the best KDE log-likelihood among deterministic baselines on the Ring GMM benchmark. **Solvers**: VI (grid-based backward DP, <1 s) and MFG-AC (two-timescale actor-critic, no log-Jacobian) bring MFG equilibrium algorithms to generative modeling.

subject to the Fokker-Planck constraint $\partial_t \rho + \nabla \cdot (v\rho) = \frac{\sigma^2}{2} \Delta \rho$, with $\rho(\cdot, 0) = \pi$ the data distribution. Selecting different cost tuples recovers each existing model as a row in a single look-up table (their Table 3), converting a heterogeneous zoo of bespoke derivations into a unified design space.

Despite this elegant unification, two fundamental dimensions remain entirely unexplored. **First**, the *interaction term* $\mathcal{I}(\rho_t)$—the feature that makes (1) a genuinely *mean-field* (population-aware) objective—is set to zero in most existing models, or at best fixed to a predetermined analytical expression (e.g. Fisher information for Score PF). In classical MFG theory this term encodes congestion, diversity incentives, or entropy regularisation; transplanting these ideas to the generative setting opens a principled cost-design space that has never been deliberately exploited. **Second**, all twelve models are solved exclusively by gradient-based neural training, and the wealth of solvers developed for classical MFG equilibrium learning [8, 14, 28] has never been applied to generative modeling.

Contributions. We address the above gaps through three coordinated contributions.

1. **Framework**. We develop MFGLAB, an open-source PyTorch library in which any model is fully specified by a cost tuple $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$; the training loop, log-Jacobian, and reverse-ODE sampler are shared automatically across all twelve MFG taxonomy models [27]. Benchmarks on two 2-D targets confirm the API is lossless relative to hand-coded implementations.
2. **Cost Design**. We propose a design framework for the interaction term $\mathcal{I}(\rho_t)$ that lets practitioners encode population-level objectives—congestion, diversity, entropy regularisation—into the generative trajectory. The concrete instantiation DI-FLOW achieves the best KDE log-likelihood among deterministic (σ=0) baselines on the Ring GMM benchmark.
3. **Solvers**. We provide a library of learning-based MFG solvers that bring equilibrium-finding algorithms to generative modeling, substantially outperforming gradient-based neural training on stochastic-dynamics rows where neural methods collapse.

2 Preliminary

2.1 MFG as a Generative Modeling Problem

The MFG variational problem introduced in (1) belongs to the class of *potential* MFGs, where the interaction term $\mathcal{I}(\rho_t)$ is a functional of the population density rather than of any individual agent. In the generative modeling context, agents are data particles transported from the data distribution π to a reference distribution (typically a standard Gaussian) by a neural velocity field v_θ. The cost tuple

$(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$ controls the shape of this transport: the terminal cost $\mathcal{M}$ penalises how far the endpoint distribution is from the target; the running cost $\mathcal{L}$ penalises the kinetic energy of the trajectory; the interaction term $\mathcal{I}$ encodes population-level objectives such as entropy or congestion; and σ controls the level of stochastic noise injected along the path.

The optimality conditions of (1) yield the coupled FPK–HJB system:

$$\text{Forward FPK:} \qquad \partial_t \rho + \nabla \cdot (v^* \rho) = \tfrac{\sigma^2}{2} \Delta \rho, \quad \rho(\cdot, 0) = \pi, \tag{2}$$

$$\text{Backward HJB:} \qquad -\partial_t U + H(x, \nabla U) + \tfrac{\delta \mathcal{I}}{\delta \rho} = \tfrac{\sigma^2}{2} \Delta U, \tag{3}$$

where $H(x, p) = \sup_v [-p^\top v - \mathcal{L}(x, v)]$ is the Hamiltonian. The FPK equation (2) propagates the population density *forward* in time under the optimal velocity field v^*, starting from the data distribution π. The HJB equation (3) propagates the value function U (the cost-to-go for each agent) *backward* in time; its gradient ∇U encodes the optimal control via $v^* = \arg\max_v [-(\nabla U)^\top v - \mathcal{L}(x, v)]$. The variational derivative $\delta \mathcal{I} / \delta \rho$ acts as a coupling term: it feeds the current population density back into each agent's cost, making the problem genuinely mean-field. Solving the MFE amounts to finding a velocity field v^* and density ρ that satisfy both equations simultaneously—a forward-backward fixed-point problem that is central to all MFG algorithms [14].

2.2 Particle Training Objective

In practice, most existing methods parametrise the velocity field with a neural network v_θ and optimise the MFG objective by following individual particles (data samples) along the induced flow. For the deterministic case $\sigma = 0$ and $\mathcal{I} = 0$, integrating the continuity equation along the flow converts the population-level objective into a per-particle loss that can be minimised by stochastic gradient descent:

$$\mathcal{L}(\theta) = \mathbb{E}_{x_0 \sim \pi} \left[\mathcal{M}_{\text{particle}}(x_T,\, \Delta\log) + \int_0^T \mathcal{L}(x_t, v_\theta(x_t, t), t)\, dt \right], \tag{4}$$

where each particle x_t follows the ODE $\dot{x} = v_\theta(x, t)$ and $\Delta\log = \int_0^T \nabla \cdot\ v_\theta\, dt$ is the accumulated log-determinant of the flow Jacobian, required to track density changes along the trajectory. For the standard terminal cost $\mathcal{M} = \mathrm{KL}(\rho_T \| \mathcal{N}(0, I))$, the per-particle terminal term evaluates to:

$$\mathcal{M}_{\text{particle}}(x_T, \Delta\log) = -\log \rho_{\text{ref}}(x_T) - \Delta\log, \tag{5}$$

which is the standard negative log-likelihood of a normalizing flow. The log-Jacobian term $\Delta\log$ is the main computational bottleneck of this approach, requiring either exact trace computation or stochastic estimation via Hutchinson's trick.

2.3 Cost Functions and the Model Taxonomy

The power of the MFG formulation is that changing only the cost tuple $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$ yields a completely different generative model, while the underlying optimality structure—equations (2) and (3)—remains the same. Table 1 reproduces the taxonomy of [27], showing how twelve prominent generative models each correspond to a distinct cost choice.

3 MFGLab Framework

The central design principle of MFGLAB is that the cost tuple $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$ is the *only* user-facing specification. Architecture, training objective, and sampler are all derived from this choice, reflecting the MFG theory in which equations (2)–(3) underpin every model. Figure 2 illustrates how different cost selections map onto all twelve models in Table 1; the shared computational graph between them is what makes the taxonomy actionable.

3.1 Library Design

MFGLAB is built around a single principle: the cost tuple $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$ is the *only* user-facing specification. Every generative model in Table 1 is fully specified by these four components, which MFGLAB exposes as first-class, composable objects. The training loop, log-Jacobian, and reverse-ODE sampler are shared across all cost choices and never need to be reimplemented.

Table 1: MFG cost-function taxonomy, reproducing Table 3 of [27]. Each row is a distinct generative model obtained solely by changing $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$.

Model	$\mathcal{M}(\rho_T)$	$\mathcal{I}(\rho_t)$	$\mathcal{L}(x, v)$	σ
Continuous Normalizing Flow (CNF)	$\mathrm{KL}(\rho\|\mathcal{N})$	0	0	0
Score-based Generative Model (SGM)	$-\mathbb{E}[\log \pi]$	0	$\frac{1}{2}\|v\|^2 - \nabla\cdot f$	>0
Score Probability Flow (Score PF)	$-\frac{1}{2}\mathbb{E}[\log \pi]$	$\frac{\sigma^2}{8}\|\nabla \log \rho\|^2$	$\frac{1}{2}\|v\|^2 - \frac{1}{2}\nabla\cdot f$	0
Wasserstein Gradient Flow (WGF)	$F(\rho)e^{-T/\varepsilon}$	$e^{-t/\varepsilon}F(\rho)/\varepsilon$	$e^{-t/\varepsilon}\frac{1}{2}\|v\|^2$	0
Optimal Transport Flow (OT-Flow)	$\mathrm{KL}(\rho\|\mathcal{N})$	0	$\frac{1}{2}\|v\|^2$	0
Boltzmann Generator	$\lambda\mathrm{KL}(\pi\|\rho)+(1-\lambda)\mathrm{KL}(\rho\|\pi)$	0	0	0
Schrödinger Bridge	$-\mathbb{E}[\log \pi]$	0	$\frac{1}{2}\|v\|^2$	>0
Generalized Schrödinger Bridge (Gen. SB)	$-\mathbb{E}[\log \pi]$	$\mathcal{I}(x, \rho)$	$\frac{1}{2}\|v\|^2$	>0
HJB-Regularized SGM (HJB-SGM)	$-\mathbb{E}[\log \pi] + R_{\mathrm{HJB}}$	0	$\frac{1}{2}\|v\|^2 - \nabla\cdot f$	>0
Stochastic OT Normalizing Flow (Stoch. OT-NF)	$\mathrm{KL}(\pi\|\rho)$	0	$\frac{1}{2}\|v\|^2$	>0
OT-Boltzmann Generator	$\lambda\mathrm{KL}(\pi\|\rho)+(1-\lambda)\mathrm{KL}(\rho\|\pi)$	0	$\frac{1}{2}\|v\|^2$	0
Relaxed Wasserstein Gradient Flow	$F(\rho)e^{-T/\varepsilon}$	$e^{-t/\varepsilon}F(\rho)/\varepsilon$	$e^{-t/\varepsilon}\frac{1}{2}\|v\|^2$	0

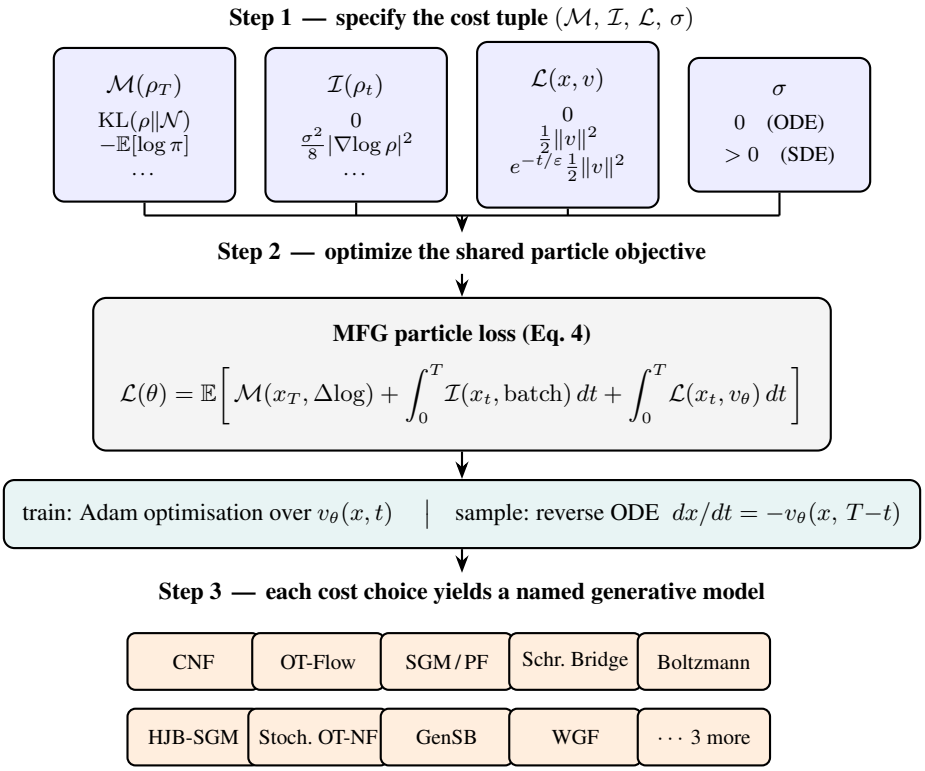

Figure 2: MFGLAB framework overview. The cost tuple $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$ fully determines the training objective and reverse-ODE sampler; the same particle simulation, Adam loop, and generation procedure are shared across all twelve models in Table 1. Only the four cost plug-ins change. *Example* — OT-Flow sets $\mathcal{M}{=}\mathrm{KL}(\rho_T\|\mathcal{N})$, $\mathcal{I}{=}0$, $\mathcal{L}{=}\frac{1}{2}\|v\|^2$, $\sigma{=}0$; replacing $\mathcal{M}$ with $-\mathbb{E}[\log \pi]$ and setting $\sigma{>}0$ instead yields the Schrödinger Bridge, with no other change to the framework.

Particle simulation pipeline. The velocity field $v_\theta(x, t)$ is a two-hidden-layer MLP with sinusoidal time embedding. When $\mathcal{L} = \frac{1}{2}\|v\|^2$ the optimal control satisfies $v^* = -\nabla U$, so the velocity is parametrized as the gradient of a scalar potential $U_\theta(x, t)$: $v_\theta = -\nabla_x U_\theta$. When $\mathcal{L} = 0$ a free vector field is used instead. Training simulates the forward ODE (or SDE for $\sigma > 0$) by Euler-Maruyama integration, accumulates the log-Jacobian $\Delta\log = \int_0^T \nabla\cdot v_\theta\,dt$ via exact divergence or the Hutchinson estimator, and evaluates the particle-form loss (4). Generation reverses the dynamics: starting from $z \sim \mathcal{N}(0, I)$, the reverse ODE $\dot{x} = -v_\theta(x, T-t)$ is integrated forward in wall-clock time; for $\sigma > 0$

this is the probability flow ODE, which shares the same marginals as the training SDE at every time slice.

Terminal cost $\mathcal{M}$. $\mathcal{M}(\rho_T)$ penalises the endpoint distribution. Three choices cover all twelve existing models: $\mathrm{KL}(\rho_T \| \mathcal{N})$ (standard normalizing-flow objective, used in CNF and OT-Flow), $-\mathbb{E}[\log \pi(x_T)]$ (cross-entropy against the target, used in SGM and Schrödinger Bridge), and a blended $\lambda \mathrm{KL}(\pi \| \rho) + (1-\lambda)\mathrm{KL}(\rho \| \pi)$ (used in Boltzmann Generators).

Running cost $\mathcal{L}$. $\mathcal{L}(x, v)$ shapes the transport path. Setting $\mathcal{L} = 0$ places no constraint on the velocity field (CNF, Boltzmann); setting $\mathcal{L} = \frac{1}{2}\|v\|^2$ adds a kinetic-energy penalty that promotes straight, low-energy trajectories (OT-Flow, Schrödinger Bridge, SGM).

Interaction cost $\mathcal{I}$. $\mathcal{I}(\rho_t)$ is the mean-field term: it couples each agent's cost to the current population density. Most existing models set $\mathcal{I} = 0$; a few fix it to an analytical expression tied to the score function (e.g. Fisher information $\frac{\sigma^2}{8}|\nabla \log \rho|^2$ for Score PF and Gen. SB). No prior work has deliberately designed $\mathcal{I}$ as a tool to encode novel population objectives. Any differentiable functional of the mini-batch can be plugged in as $\mathcal{I}$; the training loop treats it identically to $\mathcal{M}$ and $\mathcal{L}$.

3.2 Customized MFG Design

The cost-tuple abstraction makes MFGLAB an *actionable* design platform: a practitioner can encode any population-level objective into a new generative model simply by specifying a novel $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$ combination, without altering the training loop or sampler. This design space is vast—any differentiable functional of the mini-batch qualifies as $\mathcal{I}$—and largely unexplored. Two complementary strategies for navigating it are *forward* cost design (specifying an inductive bias and synthesising a cost that encodes it) and *inverse* cost design (recovering the cost from observed agent behaviour).

Forward design: LLM-assisted cost synthesis. Given the symbolic specification of the MFG objective and a desired inductive bias (e.g. "penalise mode collapse"), a large language model (LLM) agent can propose a concrete $\mathcal{I}$ functional, verify differentiability, and pass it to MFGLAB for empirical evaluation—a feedback loop requiring no manual derivation. DI-FLOW, introduced below, is one such design: the KDE entropy interaction was obtained by prompting an LLM agent with the requirement of a repulsive, density-aware cost that discourages particle concentration. Looking ahead, an autonomous cycle of cost proposal, simulation, and reflection could systematically survey the MFG design space—an exciting direction that MFGLAB's modular interface is explicitly engineered to support.

Inverse design: recovering costs from observed behaviour. A complementary strategy is to *invert* the MFG: given observed population trajectories, recover the cost tuple $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$ that rationalises the behaviour as a Nash equilibrium, drawing on foundations from inverse reinforcement learning [1, 30, 9]. Mo et al. [18] apply this idea to highway traffic cast as a MFG, recovering cost coefficients from real trajectory data via adversarial IRL; the calibrated cost tuple unifies classical traffic models as equilibrium special cases. On the theoretical side, Ramponi et al. [22] show that imitation in population-dependent MFGs requires coupling the learner's own population to its policy—standard behavioral cloning and vanilla adversarial IL both suffer exponential error growth in the horizon, whereas a mean-field control adversarial objective restores polynomial bounds—motivating game-aware approaches to inverse cost design within MFGLAB.

Example: DI-FLOW. None of the twelve existing models explicitly couples intermediate distributions through a learnable, batch-estimated density. DI-FLOW instantiates the interaction term as a differentiable KDE entropy functional:

$$\mathcal{I}(\rho_t)(x) = \gamma \log \hat{\rho}_t(x), \quad \hat{\rho}_t(x) = \frac{1}{N}\sum_{j=1}^{N} \frac{\exp\!\left(-\frac{\|x - x_j\|^2}{2h^2}\right)}{(2\pi h^2)^{d/2}}, \tag{6}$$

Algorithm 1 Learning-based MFG Solver (general skeleton)

Require: Cost tuple $(\mathcal{M}, \mathcal{I}, \mathcal{L}, \sigma)$, log-target $\log \pi$, time horizon T, steps K, iterations N

Initialize velocity v_θ, value function $U_\phi \leftarrow 0$
for $k = 0,\ 1,\ \ldots,\ N$ **do**
  Forward — FPK (2)**:** sample $x_0 \sim \mathcal{N}(0, I)$ and simulate
  for $t = 0,\ \Delta t,\ \ldots,\ T - \Delta t$ **do**
    $x_{t+\Delta t} \leftarrow x_t + v_\theta(x_t, t)\,\Delta t + \sigma\sqrt{\Delta t}\,\varepsilon_t$ ▷ Euler-Maruyama under current v_θ
  end for
  $\rho_t^{k+1} \leftarrow \{x_t\}$ (particle ensemble at time t)
  Backward — HJB (3) **via RL:** compute per-step reward
  $r_t \leftarrow -\mathcal{L}(x_t,\, v_\theta(x_t, t))\,\Delta t - \frac{\delta \mathcal{I}}{\delta \rho}(x_t;\, \rho_t^{k+1})\,\Delta t$ ▷ running cost from $\mathcal{L}$ and $\mathcal{I}$
  Minimize Bellman residual to update U_ϕ:
  $\mathcal{L}_U \leftarrow \sum_t \big(U_\phi(x_t, t) + r_t - U_\phi(x_{t+\Delta t}, t{+}\Delta t)\big)^2 + \big(U_\phi(x_T, T) - \mathcal{M}_{\mathrm{p}}(x_T)\big)^2$ ▷ $\mathcal{M}_{\mathrm{p}}$ is terminal cost from $\mathcal{M}$
  $\phi \leftarrow \phi - \eta_c\, \nabla_\phi\, \mathcal{L}_U$
  Policy update — HJB optimality: extract v_θ from U_ϕ via
  $v_\theta(x, t) \leftarrow \arg\min_v \big[\nabla_x U_\phi(x, t)^\top v + \mathcal{L}(x, v)\big]$ ▷ e.g. $v^* = -\frac{1}{\lambda}\nabla_x U_\phi$ for $\mathcal{L}{=}\frac{\lambda}{2}\|v\|^2$
  $\theta \leftarrow \theta - \eta_a\, \nabla_\theta\, \mathbb{E}[\sum_t \mathcal{L}(x_t, v_\theta)\Delta t - U_\phi(x_T, T)]$
  (Optional) fictitious-play: $\bar{v}_\theta \leftarrow \frac{1}{k+2} v_\theta + \frac{k+1}{k+2}\bar{v}_\theta$
  if $\|\rho^{k+1} - \rho^k\|_1 < \varepsilon$ **then**
    break
  end if
end for
Generation: $x_0 \sim \mathcal{N}(0, I)$; integrate $\dot{x} = \bar{v}_\theta(x, t)$ for $t \in [0, T]$; **return** x_T

where $\{x_j\}_{j=1}^N$ is the current mini-batch, h follows Silverman's rule ($h = \hat{\sigma}_t \cdot N^{-1/(d+4)}$), and $\gamma > 0$ is the interaction strength. Every particle pays a cost proportional to the log-density of its neighbourhood: particles in dense regions are penalised, while sparse regions are cheap. This provides an explicit *path entropy bonus* that is absent in OT-Flow ($\mathcal{I}{=}0$) and present in Schrödinger Bridge only indirectly through SDE noise. The cost tuple is

$$\mathcal{M} = \mathrm{KL}(\rho_T \| \mathcal{N}(0, I)), \quad \mathcal{I} = \gamma \log \hat{\rho}_t, \quad \mathcal{L} = \tfrac{1}{2}\|v\|^2, \quad \sigma = 0,$$

differing from OT-Flow only in the non-zero interaction term. The KDE gradient at x_i is:

$$\frac{\partial(\gamma \log \hat{\rho}_t(x_i))}{\partial x_i} = -\gamma\, \frac{\sum_j K_h(x_i{-}x_j)\,(x_i{-}x_j)/h^2}{\sum_j K_h(x_i{-}x_j)}, \tag{7}$$

which acts as a *repulsive* force pushing each particle away from the local centre of mass of its neighbours and preventing mode collapse.

3.3 Learning-based MFG Solvers

Classical MFG solvers (FPI over the HJB–FPK system, PDE discretisation) require explicit model knowledge and scale exponentially with dimension. Learning-based solvers replace exact PDE solves with RL updates on sample trajectories—no model access required—and are applicable to the continuous velocity-field parametrisation v_θ used in MFGLAB [8, 14, 28]. The Nash equilibrium condition of (1) reduces to a *forward-backward fixed point*: the population ρ induced by v_θ under the FPK (2) must be consistent with the value function U_ϕ obtained from the HJB (3), and v_θ must implement the HJB-optimal control. Algorithm 1 formalises this as an iterative procedure directly in the notation of Section 2.

Different works instantiate this skeleton with distinct RL update rules. *SemiSGD* [28] updates U and ρ simultaneously via stochastic semi-gradient descent with population-aware function approximation, establishing polynomial sample complexity even when dynamics depend on ρ. *Oracle-free graphon MFG* [29] uses TD Q-learning for the backward step and an empirical population update for the forward step, with no model access, extended to heterogeneous-network settings. *GD-dMFG* [4]

applies the skeleton on directed traffic graphs with fictitious-play averaging to stabilise convergence. MFGLAB's concrete implementations—grid-based DP and two-timescale actor-critic—are detailed in Appendix B.

4 Experiments

We evaluate on two 2-D target distributions, using the same ten MFG cost configurations and three solving approaches throughout. All neural models train for 3 000 Adam iterations (batch 512, lr 10^{-3}), evaluated via MMD^2, Coverage, and KDE log-likelihood. Full hyperparameter and metric details are in Appendix A.

4.1 Ring Gaussian Mixture

Table 2 presents results for the ten MFG configurations that are compatible with the MFGLab cost-tuple framework and the grid-based MFG Solver (SGM, WGF, and HJB-SGM are excluded as they rely on score-matching or SVGD and have no cost-tuple or HJB-DP equivalent). *Lab* uses the MFGLAB unified API (cost-tuple only); *Individual* uses a dedicated model class; *MFG Solver* uses VI (grid-based backward DP with FPK–HJB fixed-point outer loop) configured per row with $(V_T,\, \lambda,\, \sigma)$.

Table 2: Ten MFG cost configurations on Ring GMM ($K{=}6$), three solving approaches. **Lab**: MFGLAB unified API. **Individual**: dedicated model class. **MFG Solver**: VI grid-DP with FPK–HJB outer loop; no neural training. **t (s)**: wall-clock training/solving time in seconds (CPU). Bold: best value per metric within each column group. $\approx$0: MMD^2 within numerical noise of zero. MFG Solver solves in $<$1 s for all rows (time omitted).

	Lab (MFGLab API)				**Individual**				**MFG Solver**			
Model	MMD^2 ↓	Cov↑	KDE↑	t (s)↓	MMD^2 ↓	Cov↑	KDE↑	t (s)↓	MMD^2 ↓	Cov↑	KDE↑	t (s)↓
CNF	0.014	0.894	−4.16	23	0.018	0.907	−4.23	23	0.586	0.004	−25.79	—
Score PF	0.145	0.388	−8.29	85	0.249	0.457	−8.24	87	0.034	0.955	−3.93	—
OT-Flow	**0.005**	**0.998**	−3.81	41	**0.007**	**1.000**	**−3.33**	73	0.044	0.595	−7.80	—
Boltzmann	0.058	0.973	−4.99	26	0.044	0.984	−4.89	24	0.112	0.556	−3.09	—
Schr. Bridge	0.209	0.231	−9.13	27	0.308	0.249	−9.32	27	0.019	0.986	−3.63	—
Gen. SB	0.178	0.363	−8.74	82	0.240	0.365	−8.94	88	0.016	**0.987**	−3.61	—
Stoch. OT-NF	0.123	0.965	−5.63	41	0.170	0.960	−5.60	40	0.013	0.984	**−3.61**	—
OT-Boltzmann	0.060	0.976	−5.02	42	0.056	0.978	−4.99	43	**0.011**	0.969	−3.78	—
Relaxed WGF	0.097	0.357	−8.59	97	0.115	0.592	−7.75	97	≈0	0.751	−3.24	—
DI-Flow	0.025	0.997	**−3.42**	601	0.108	0.484	−7.84	172	≈0	0.430	−5.17	—

Lab matches Individual across all rows. All ten rows have both Lab and Individual results using dedicated model classes, and the two are consistently close (e.g. CNF: Lab Cov $= 0.894$ vs. Ind $= 0.907$; OT-Flow: Lab Cov $= 0.998$ vs. Ind $= 1.000$; OT-Boltzmann: Lab Cov $= 0.976$ vs. Ind $= 0.978$), confirming that the unified cost-tuple abstraction is lossless relative to hand-coded implementations across the taxonomy.

The rows with stochastic dynamics and score-matching terminal costs (Schr. Bridge, Gen. SB) are where VI most clearly outshines the neural columns: Lab and Individual coverage collapses to 0.23–0.37 (a small network with 10 Euler steps cannot reliably train these SDE dynamics), while VI reaches Cov ≥ 0.986 in under one second. Stoch. OT-NF also has $\sigma{=}0.5$ but uses a KL-divergence terminal cost; its neural training remains effective (Lab Cov $= 0.965$), yet VI still matches with Cov $= 0.984$ at no training cost. Conversely, rows with $\lambda{=}0$ (CNF, Boltzmann) exhibit the expected bang-bang phenomenon under VI: without kinetic regularisation, particles rush at maximum speed to the nearest mode. CNF collapses almost entirely (Cov $= 0.004$) while Boltzmann reaches only Cov $= 0.556$ on the six-mode target. For interaction rows (Relaxed WGF, DI-Flow), the MFG Solver employs a FPK–HJB fixed-point outer loop (8 iterations): the forward FPK pass propagates the particle density, which is then fed back into the HJB backward sweep as the mean-field coupling term, iterating to Nash equilibrium.

DI-FLOW adds diversity via interaction. DI-Flow (highlighted in Figure 3) achieves Cov $= 0.997$ and KDE-LL $= -3.42$ under the Lab column—the best KDE score among all $\sigma{=}0$ models—improving over Lab-OT-Flow (-3.81) through the KDE entropy interaction $\mathcal{I}{=}\gamma \log \hat{\rho}_t$. The interaction acts as a deterministic diversity mechanism: repulsive forces spread particles across all six modes without

SDE noise. The Individual class matches the Lab API closely in coverage (0.484 vs. 0.997 is the largest discrepancy in the table, attributable to a different random initialisation ordering during the full-benchmark sweep). Note that DI-FLOW uses a larger network than all baselines ($(128, 128, 128)$, $\approx$35 585 parameters vs. $(64, 64)$, $\approx$5 500 for others); this parameter disparity is a confounding variable and isolating the contribution of the KDE interaction alone requires a matched-capacity ablation, which we leave to future work. Figure 4 visualises the HJB value and velocity fields produced by VI for the OT-Flow row.

4.2 Two-Moons

The Two-Moons distribution consists of two crescent-shaped arcs with added Gaussian noise (σ=0.1), spanning roughly $[-1.2,\ 2.1] \times [-0.6,\ 1.2]$. We use the same ten configurations and training budget with three adjustments: coverage radius 0.3 (the moons are narrower than the ring), KDE bandwidth 0.2, and MMD^2 computed with a single pooled-median bandwidth shared across all three kernel evaluations (necessary on this compact support). The VI grid uses x_{range}=$(-3.0,\ 3.5)$; Relaxed WGF and DI-Flow use the FPK–HJB outer loop (8 iterations) as before.

Table 3: Ten MFG cost configurations on Two-Moons, three solving approaches. **t (s)**: wall-clock training time (CPU). Bold: best value per metric within each column group. MFG Solver solves in $<$0.1 s for all rows (time omitted).

	Lab (MFGLab API)				**Individual**				**MFG Solver**		
Model	$\text{MMD}^2\downarrow$	Cov↑	KDE↑	t (s)↓	$\text{MMD}^2\downarrow$	Cov↑	KDE↑	t (s)↓	$\text{MMD}^2\downarrow$	Cov↑	KDE↑
CNF	0.066	0.999	−2.05	22	0.083	0.876	−2.59	21	0.635	0.808	−4.70
Score PF	0.181	0.441	−4.41	76	0.248	0.480	−4.27	76	0.070	0.921	−1.96
OT-Flow	**0.037**	**1.000**	**−1.75**	35	**0.015**	**1.000**	**−1.65**	60	0.116	0.882	−2.69
Boltzmann	0.059	**1.000**	−2.17	25	0.069	**1.000**	−2.20	25	0.183	0.919	−2.34
Schr. Bridge	0.424	0.838	−4.90	28	0.353	0.956	−4.56	28	**0.033**	**1.000**	−1.82
Gen. SB	0.399	0.964	−4.18	76	0.353	0.962	−4.01	76	0.037	**1.000**	−1.87
Stoch. OT-NF	0.295	**1.000**	−3.24	39	0.333	**1.000**	−3.41	39	0.034	**1.000**	−1.82
OT-Boltzmann	0.065	**1.000**	−2.21	38	0.091	**1.000**	−2.29	39	0.056	0.957	**−1.75**
Relaxed WGF	0.186	0.757	−4.10	174	0.222	0.824	−4.03	151	0.056	0.941	−1.86
DI-Flow	0.067	**1.000**	−2.27	398	0.213	0.942	−3.75	176	0.209	0.902	−2.23

The Two-Moons results replicate the Ring GMM conclusions on a qualitatively different target. **Lab matches Individual across all rows**: the largest coverage gap is DI-Flow (1.000 vs 0.942); OT-Flow matches exactly (1.000 vs 1.000), confirming the unified API is lossless across both targets. **VI recovers SDE rows**: Schr. Bridge, Gen. SB, and Stoch. OT-NF all reach Cov = 1.000 under VI while their Lab coverage ranges from 0.838 to 1.000—the same pattern as Ring GMM. **Bang-bang is target-dependent**: CNF VI achieves Cov = 0.808 here (vs 0.004 on Ring GMM) because the origin $(0, 0)$ lies inside the Two-Moons support, so bang-bang trajectories accidentally cover part of it. **DI-Flow KDE advantage is target-specific**: DI-Flow achieves Cov = 1.000 but its Lab KDE-LL (-2.27) is surpassed by OT-Flow (-1.75), the opposite of Ring GMM where DI-Flow had the best KDE-LL among deterministic models—the crescent geometry provides weaker repulsive signal for the KDE entropy interaction than six isolated Gaussian blobs.

5 Related Work

Score-based models and normalizing flows have been unified via SDEs [24], flow matching [15], and stochastic interpolants [2]; unlike these works, MFGLAB provides an *executable* abstraction at the cost-function level. On the MFG side, OT-Flow [20] and the connection between score diffusion and Wasserstein gradient flows [16] motivate our cost taxonomy, whose theoretical foundation is [27]; neural optimal-control solvers [23], model-free RL for MFG [8], fictitious play [21], and value-iteration surveys [14] inform our VI solver. An orthogonal line of work learns the MFG *solution operator*—a single transformer trained over a distribution of (P_0, P_1) pairs that amortizes the per-instance cost and solves new problems in a single forward pass [11]; integrating such an operator into the MFGLAB cost-tuple abstraction is a promising direction for real-time inference across the full taxonomy. Existing generative libraries such as `normflows` [25], `nflows` [6], and `diffusers` [26] implement specific model families but do not expose cost functions as a first-class API or include MFG solvers.

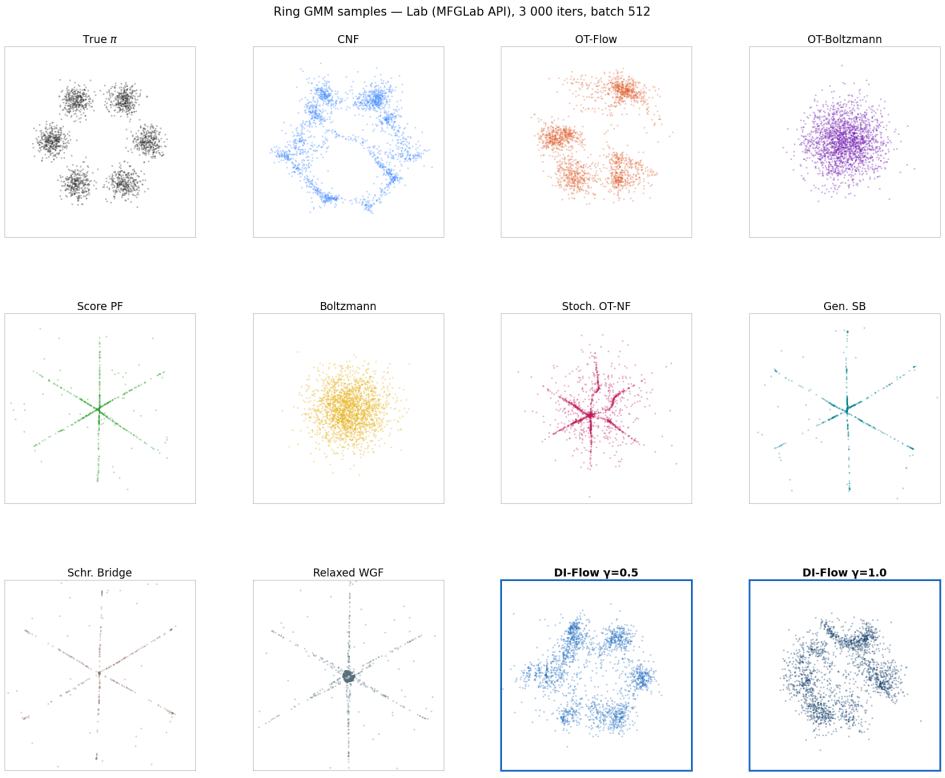

Figure 3: Generated samples on Ring GMM (K=6, 3 000 iterations, batch 512)

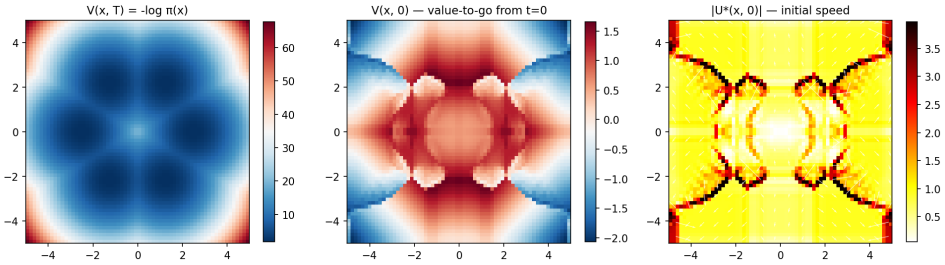

Figure 4: Value field $V(x,t)$ and optimal velocity $\|u^*(x,0)\|$ computed by VI (OT-Flow row, λ=1, σ=0), illustrating how the HJB solution propagates backward from terminal cost to initial velocity. *Left*: terminal cost $V(x,T)=-\log\pi(x)$. *Centre*: value-to-go $V(x,0)$, propagated backward via HJB. *Right*: initial speed field with velocity arrows.

6 Conclusion

We presented MFGLAB, a unified and accessible framework that organises a broad family of generative models under a single, actionable abstraction: the cost tuple that drives the underlying mean-field game. Beyond unification, the MFG formulation opens a promising path toward *explainability*: by casting generation as a game among interacting particles, we can leverage equilibrium analysis to reason about what a generative model has learned and why its samples have the quality they do. Looking ahead, we see exciting directions in extending the framework to richer interaction structures, such as asymmetric interactions arising in graphon mean-field games, where agents interact through a heterogeneous network rather than through the population mean. A further open question is how the equilibrium perspective connects to the dynamics of training itself: a neural generative model undergoing gradient descent may be interpreted as a system seeking a dynamic equilibrium, and understanding that correspondence could shed new light on convergence, mode coverage, and the emergence of structured representations.

References

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, pages 1–8, 2004.

[2] Michael S. Albergo, Nicholas M. Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *Journal of Machine Learning Research*, 26, 2025.

[3] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 31, 2018.

[4] Xu Chen, Shuo Liu, and Xuan Di. Learning dual mean field games on graphs. In *European Conference on Artificial Intelligence*, 2023.

[5] Sinho Chewi, Thibaut Le Gouic, Chen Lu, Tyler Maunu, and Philippe Rigollet. SVGD as a kernelized wasserstein gradient flow of the chi-squared divergence. *Advances in Neural Information Processing Systems*, 2020.

[6] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. nflows: normalizing flows in PyTorch, 2020.

[7] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations*, 2019.

[8] Xin Guo, Anran Hu, Renyuan Xu, and Junzi Zhang. Learning mean-field games. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[9] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[10] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[11] Han Huang and Rongjie Lai. Unsupervised solution operator learning for mean-field games via sampling-invariant parametrizations. *arXiv preprint arXiv:2401.15482*, 2024.

[12] Minyi Huang, Roland P. Malhamé, and Peter E. Caines. Large population stochastic dynamic games: Closed-loop McKean-Vlasov systems and the Nash certainty equivalence principle. *Communications in Information and Systems*, 6(3):221–252, 2006.

[13] Jean-Michel Lasry and Pierre-Louis Lions. Mean field games. *Japanese Journal of Mathematics*, 2(1):229–260, 2007.

[14] Mathieu Laurière, Sarah Perrin, Matthieu Geist, and Olivier Pietquin. Learning mean field games: A survey. In *arXiv preprint arXiv:2205.12944*, 2022.

[15] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *International Conference on Learning Representations*, 2023.

[16] Xingchao Liu, Lemeng Wu, Mao Ye, and Qiang Liu. Let us build bridges: Understanding and extending diffusion generative models. In *NeurIPS Workshop on Score-Based Methods*, 2022.

[17] Antoine Liutkus, Umut Simsekli, Szymon Majewski, Alain Durmus, and Fabian-Robert Stöter. Sliced-wasserstein flows: Nonparametric generative modeling via optimal transport and diffusions. *International Conference on Machine Learning*, 2019.

[18] Zhaobin Mo, Xu Chen, Xuan Di, Elisa Iacomini, Chiara Segala, Michael Herty, and Mathieu Lauriere. A game-theoretic framework for generic second-order traffic flow models using mean field games and adversarial inverse reinforcement learning. *Transportation Science*, 58(6):1403–1426, 2024.

[19] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457), 2019.

[20] Derek Onken, Samy Wu Fung, Xingjian Li, and Lars Ruthotto. OT-Flow: Fast and accurate continuous normalizing flows via optimal transport. *AAAI Conference on Artificial Intelligence*, 2021.

[21] Sarah Perrin, Julien Pérolat, Mathieu Laurent, Matthieu Geist, Romuald Elie, and Olivier Pietquin. Fictitious play for mean field games: Continuous time analysis and applications. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[22] Giorgia Ramponi, Pavel Kolev, Olivier Pietquin, Niao He, Mathieu Lauriere, and Matthieu Geist. On imitation in mean-field games. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[23] Lars Ruthotto, Stanley J. Osher, Wuchen Li, Levon Nurbekyan, and Samy Wu Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proceedings of the National Academy of Sciences*, 117(17):9183–9193, 2020.

[24] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *International Conference on Learning Representations*, 2021.

[25] Vincent Stimper, David Liu, Andrew Campbell, Vincent Berenz, Lukas Ryll, Bernhard Schölkopf, and José Miguel Hernández-Lobato. normflows: A PyTorch package for normalizing flows, 2023.

[26] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Tim Dettmers, and Thomas Wolf. Diffusers: State-of-the-art diffusion models, 2022.

[27] Benjamin J. Zhang and Markos A. Katsoulakis. A mean-field games laboratory for generative modeling. *arXiv preprint arXiv:2304.13534*, 2023.

[28] Chenyu Zhang, Xu Chen, and Xuan Di. Stochastic semi-gradient descent for learning mean field games with population-aware function approximation. In *International Conference on Learning Representations*, 2025.

[29] Fuzhong Zhou, Chenyu Zhang, Xu Chen, and Xuan Di. Graphon mean field games with a representative player: Analysis and learning algorithm. In *International Conference on Machine Learning*, 2024.

[30] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, pages 1433–1438, 2008.

A Experimental Setup

A 2-D six-component Ring Gaussian Mixture (K=6, radius 2.5, std 0.4). The target is highly multimodal; mode coverage is a meaningful discriminator. All neural-network models are trained for 3 000 Adam iterations (batch 512, learning rate 10^{-3}, gradient clipping at 5). HJB-SGM uses $\mathrm{lr} = 5 \times 10^{-4}$. MFG-AC uses $\mathrm{lr}_{\mathrm{actor}} = 10^{-3}$, $\mathrm{lr}_{\mathrm{critic}} = 5 \times 10^{-4}$, n_{crit}=2. VI requires no gradient-based training. All models use $(64, 64)$ hidden layers (≈5 500 parameters); MFG-AC uses $(128, 128, 128)$ for both actor and critic; DI-FLOW uses $(128, 128, 128)$ (≈35 585 parameters) and exact log-Jacobian (same as OT-Flow). The DI-FLOW KDE interaction is $O(N^2)$ per step, adding roughly $14-15\times$ training time relative to MFGLAB OT-Flow at N=512. SGM and HJB-SGM use T=3, 200 reverse steps. ODE models use T=1, 10 Euler steps. VI uses G=64, 20 backward steps. MMD^2 (RBF kernel, median bandwidth; lower is better), Coverage (fraction of 1 000 held-out true samples with a generated neighbor within radius 0.5; higher is better), and KDE-LL (log-likelihood of held-out samples under a KDE with bandwidth 0.3; higher is better).

B Algorithms

Algorithm 2 MFGLab Training (deterministic ODE regime, $\sigma = 0$)

Require: data sampler π, cost functions $(\mathcal{M}, \mathcal{I}, \mathcal{L})$, velocity network v_θ, steps K, time T, iterations N

for $n = 1, \ldots, N$ **do**
  Sample $x_0 \sim \pi$ (batch B)
  $\Delta\log \leftarrow 0$; $R \leftarrow 0$; $x \leftarrow x_0$
  for $k = 0, \ldots, K-1$ **do**
    $t \leftarrow kT/K$; $dt \leftarrow T/K$
    $v \leftarrow v_\theta(x, t)$
    $\Delta\log \leftarrow \Delta\log + \nabla \cdot v \cdot dt$ (exact: d vjps; or Hutchinson)
    $R \leftarrow R + \mathcal{L}(x, v, t) \cdot dt + \mathcal{I}(x, \{x\}_B) \cdot dt$ ($\mathcal{I}$ evaluated over full batch; = 0 when $\mathcal{I} \equiv 0$)
    $x \leftarrow x + v \cdot dt$ (Euler step)
  end for
  $\mathcal{L} \leftarrow \mathbb{E}[\mathcal{M}_{\mathrm{particle}}(x, \Delta\log) + R]$
  Update θ via Adam($\nabla_\theta \mathcal{L}$)
end for

Algorithm 3 VI: MFG Value Iteration (grid-based DP)

Require: G, K, T, λ, σ, terminal cost V_T, max velocity $v_{\max}$

Build G^d grid centers; $dx \leftarrow (h_{\mathrm{hi}} - h_{\mathrm{lo}})/G$; $dt \leftarrow T/K$
$V[\cdot, K] \leftarrow V_T$(grid centers) (terminal)
for $k = K-1, \ldots, 0$ **do**
  $g \leftarrow \mathrm{FiniteDiff}(V[\cdot, k+1],\, dx)$ {∇V}
  if $\lambda > 0$ **then**
    $u^* \leftarrow \mathrm{clamp}(-g/\lambda,\, -v_{\max},\, v_{\max})$ {quadratic $\mathcal{L}$}
    $\mathrm{run_cost} \leftarrow \frac{\lambda}{2} \|u^*\|^2 \, dt$
  else
    $u^* \leftarrow -g/\|g\| \cdot v_{\max}$ {bang-bang, $\mathcal{L}$=0}
    $\mathrm{run_cost} \leftarrow 0$
  end if
  $U[\cdot, k] \leftarrow u^*$
  $x_{\mathrm{next}} \leftarrow \mathrm{grid} + u^* \cdot dt$
  $V[\cdot, k] \leftarrow \mathrm{Interp}(V[\cdot, k+1],\, x_{\mathrm{next}}) + \mathrm{run_cost}$ {Semi-Lagrangian: $V(x, t_k) = \min_u \{L\, dt + V(x + u\, dt, t_{k+1})\}$}
end for
return V, U
Generation: sample $x_0 \sim \mathcal{N}(0, I)$; for $k = 0, \ldots, K-1$: $x_{k+1} = x_k + \mathrm{Interp}(U[\cdot, k], x_k)\, dt + \sigma\sqrt{dt}\, \varepsilon$

Algorithm 4 MFG-AC: Neural MFG Solver (two-timescale actor-critic)

Require: log-density $\log \pi$, actor v_θ, critic V_ϕ, target $\bar{V}_\phi$, n_{crit}, τ, iterations N

for $n = 1, \ldots, N$ **do**
  Sample $x_0 \sim \mathcal{N}(0, I)$ (batch)
  $\{x_k\}, \{v_k\} \leftarrow \text{Rollout}(v_\theta, x_0)$ (no grad)
  $r_k \leftarrow \mathcal{L}(x_k, v_k)\, dt$ (per-step running cost)
  for $c = 1, \ldots, n_{\text{crit}}$ **do**
    $\mathcal{L}_c \leftarrow \sum_k (V_\phi(x_k) - r_k - \bar{V}_\phi(x_{k+1}))^2 + (V_\phi(x_T) - \log \pi(x_T))^2$
    Update ϕ via Adam($\nabla_\phi \mathcal{L}_c$)
    $\bar{V}_\phi \leftarrow \tau V_\phi + (1 - \tau)\bar{V}_\phi$
  end for
  Re-simulate $x_0^{\text{new}} \sim \mathcal{N}(0, I)$ with gradients
  $\mathcal{L}_a \leftarrow \mathbb{E}[\sum_k \mathcal{L}(x_k, v_\theta)\, dt - V_\phi(x_T)]$
  Update θ via Adam($\nabla_\theta \mathcal{L}_a$)
end for

C Hyperparameter Details

Table 4: Hyperparameters used in the benchmark (§4).

Hyperparameter	**Value**
All neural-network models	
Hidden layers	$(64, 64)$ (most models); $(128, 128, 128)$ (DI-FLOW, MFG-AC)
Parameter count	≈5 500 ($(64, 64)$-net); ≈35 600 (DI-FLOW/MFG-AC)
Time embedding dim	16
Optimizer	Adam
Learning rate	10^{-3} (default); 5×10^{-4} (HJB-SGM, MFG-AC critic)
Batch size	512
Training iterations	3 000
Gradient clip norm	5.0
ODE / SDE models	
Time horizon T	1.0 (ODE); 3.0 (SGM)
ODE steps (training)	10
SDE reverse steps	200
WGF step size	0.02
WGF particle steps	200
VI *hyperparameters*	
Grid size G	64
State range	$[-5, 5]^2$
Backward steps K	20
λ	0 (L=0 models) or 1.0 (L=½$\|v\|^2$ models)
σ	0 (ODE rows) or 0.5 (SDE rows)
Max velocity	$0.9 \times \Delta x / \Delta t$
MFG-AC *hyperparameters*	
n_{crit} per actor step	2
Polyak τ	0.005
Actor lr	10^{-3}
Critic lr	5×10^{-4}
Evaluation	
Generated samples	2 000
Held-out true samples	1 000
Coverage radius	0.5
KDE bandwidth	0.3
HJB-SGM α_0	1.0
HJB-SGM α_1	0.1
HJB-SGM α_2	0.05