SNAP-FM: Sparse Nonlinear Accelerated Projection for Physics-Constrained Generative Modeling

Alaina Kolli [*1] **Theodoros Xenakis** [*1 2] **Utkarsh Utkarsh** [*1] **Pengfei Cai** [1] **Rafael Gómez-Bombarelli** [1] **Alan Edelman** [1] **Christopher V. Rackauckas** [1]

Abstract

Generative models have emerged as scalable surrogates for physical simulation, yet they offer no guarantee that their outputs respect the conservation laws, boundary conditions, and nonlinear invariants that govern the underlying physics. Constrained sampling closes this gap, enforcing such constraints exactly at inference time without retraining, but at a computational cost: projection, correction and trajectory-optimization steps are repeated during sampling, with these steps becoming expensive for nonlinear constraints. Standard ML frameworks exacerbate this: their dense tensor algebra and limited sparse solver composability obscure the structure that physical constraints naturally induce, making efficient batched nonlinear optimization difficult to realize in practice. We address this bottleneck by exploiting the structure that sample-wise batching and local PDE couplings induce in the projection subproblems – namely, block-sparse Jacobian and KKT systems – exposing this structure using `ExaModels.jl` and solving the resulting sparse nonlinear programs with `MadNLP.jl` and GPU sparse factorization. Applied to Physics-Constrained Flow Matching (PCFM), on PDE benchmarks with linear, nonlinear, one-dimensional, and two-dimensional constraints, this approach accelerates nonlinear constraint projection while maintaining constraint satisfaction. These results show that sparse GPU nonlinear optimization is a practical foundation for constrained generative sampling in scientific machine learning.

[1]Massachusetts Institute of Technology [2]Norwegian University of Science and Technology. Correspondence to: Christopher V. Rackauckas <crackauc@mit.edu>.

Proceedings of the AI4Physics Workshop at the 43rd International Conference on Machine Learning (AI4Physics@ICML 2026), Seoul, South Korea. 2026. Copyright 2026 by the author(s).

1. Introduction

Generative models have emerged as flexible surrogates for physical simulation, learning solution distributions for partial differential equations (PDEs) and amortizing inference across varying physical conditions (Price et al., 2023; Yuan et al., 2023; Huang et al., 2024; Utkarsh et al., 2025a). Yet their deployment in scientific settings is limited by a fundamental gap: unconstrained generative models do not, by themselves, guarantee physical fidelity. Conservation of mass, momentum, and energy, nonlinear boundary conditions, and invariants tied to the governing equations are central to classical numerical simulation, but are routinely violated by learned surrogates unless explicitly enforced (Raissi et al., 2019; Li et al., 2021). Closing this gap without sacrificing the amortized inference advantage of generative models is the central challenge of physics-constrained generative modeling.

Constraint enforcement in generative sampling can be broadly divided into soft and hard approaches. Soft methods, including training-time penalty losses (Baldan et al., 2025; Huang et al., 2024), PINN-style residual regularization, physics-informed neural operators, and architecture-level inductive biases (Greydanus et al., 2019; Richter-Powell et al., 2022), encourage constraint satisfaction approximately. These approaches are computationally attractive and scale naturally within modern deep learning pipelines, but they generally provide no exact feasibility guarantees and can exhibit increased constraint violation under distribution shift.

Hard-constrained methods instead aim to enforce feasibility exactly, either through inference-time correction and optimization (Utkarsh et al., 2025a; Christopher et al., 2024; Cheng et al., 2025; Römer et al., 2024; Yuan et al., 2023) or through end-to-end constrained formulations (Utkarsh et al., 2025b). In the simplest case, one can apply a final post-hoc projection that maps a generated sample onto the feasible manifold, but such a late correction can introduce substantial distributional distortion. To reduce this mismatch, many methods interleave corrections with the sampling dynamics, repeatedly guiding or projecting intermediate states toward feasibility (Utkarsh et al., 2025a; Cheng et al., 2025;

Christopher et al., 2024; Ben-Hamu et al., 2024). Relaxed variants further delay or soften early corrections, reflecting the fact that early iterates often remain noise-like and may be harmed by overly strict constraint enforcement.

Despite these algorithmic differences, hard-constrained test-time methods share a common computational burden: feasibility is enforced through repeated optimization during sampling (Utkarsh et al., 2025a; Cheng et al., 2025; Christopher et al., 2024). In projection-based methods, this burden is especially explicit: each correction maps an intermediate state onto, or closer to, the constraint manifold by solving a constrained optimization problem. For linear constraints, this projection often reduces to a single matrix solve. For the nonlinear conservation laws governing many physical systems, including flux constraints, integral invariants, and nonlinear boundary conditions, the same step requires iterative nonlinear optimization. Since this optimization must be performed repeatedly across sampling steps and over batches of generated samples, it can dominate the total cost of hard-constrained generation. The central question is therefore not merely whether to enforce hard constraints, but how to make the resulting projection and correction steps fast enough to be practical at scale.

The algorithmic response is well known from large-scale optimization: exploit the sparsity induced by the constraints, and solve the resulting KKT systems using sparse linear algebra (Nocedal & Wright, 2006; Pacaud & Shin, 2024; Rennich et al., 2014; Lu & Yang, 2025). However, this strategy is difficult to deploy inside modern generative sampling pipelines. Standard ML frameworks are optimized primarily for dense batched tensor algebra, and although sparse tensor support exists, it does not yet provide a mature end-to-end stack for batched nonlinear programming, interior-point globalization, GPU-resident sparse KKT factorization, and integration with the sampling loop (Paszke et al., 2019; Bradbury et al., 2018). This gap between constrained optimization practice and deep generative modeling infrastructure motivates our approach.

The projection NLPs arising in batched hard-constrained sampling are highly structured. Across the batch, the constraint Jacobian is block diagonal because each sample's constraints are independent of the others. Within each block, the Jacobian is sparse because physical conservation laws and PDE discretizations couple only local spatial or temporal degrees of freedom. This two-level structure yields sparse KKT systems that are amenable to GPU-resident sparse factorization, avoiding the dense tensorization that would otherwise make nonlinear projection prohibitively expensive.

We realize this idea through SNAP-FM, a sparse nonlinear projection framework based on Physics-Constrained Flow Matching (PCFM) (Utkarsh et al., 2025a). SNAP-FM uses `ExaModels.jl` to symbolically compile the structured projection NLP, `MadNLP.jl` as the interior-point solver, and GPU sparse factorization for the resulting KKT systems. Built on PCFM, which enforces arbitrary nonlinear constraints zero-shot in pretrained flow models, SNAP-FM targets the main computational bottleneck of constraint enforcement: repeated nonlinear projection during sampling. Across heat, reaction-diffusion, Burgers, and two-dimensional Navier–Stokes benchmarks, SNAP-FM accelerates the projection step relative to generic optimization baselines while maintaining constraint satisfaction.

2. Physics Constrained Flow Matching

This work builds directly on the Physics-Constrained Flow Matching framework introduced by Utkarsh et al. (2025a), with the goal of extending it to a domain the current implementation doesn't support: the efficient enforcement of nonlinear constraints at scale. Among recent constraint-aware generative models, PCFM is the natural foundation for our work, also showing promising extension to different scientific domains such as atomistic generative models (Cai et al., 2026). It ensures constraint satisfaction exactly at inference time, requiring no retraining or architectural modifications of the underlying flow model, operating entirely post-hoc. It projects intermediate flow states onto constraint manifolds at inference time, without requiring gradient information during training, enabling zero-shot constraint enforcement up to machine precision for pretrained flow matching models. Unlike prior zero-shot approaches restricted to linear or non-overlapping constraints, it admits arbitrary nonlinear and coupled constraints within a single framework. These characteristics make PCFM a solid foundation on which to build a more general-purpose constrained sampling mechanism.

2.1. Sampling

The sampling algorithm utilized in PCFM is a constraint-guided algorithm that interleaves lightweight constraint corrections with marginally consistent flow updates. The procedure consists of four main steps: forward shooting, Gauss-Newton projection, reverse updating, and relaxed constraint corrections.

Let $v_\theta(u, \tau)$ denote the pretrained flow model, which defines an ODE transporting samples $u_0 \sim \pi_0$ from a tractable prior to solution-like outputs $u_1 \sim \pi_1$ over flow time $\tau \in [0, 1]$. Given a constraint function $h(u)$, PCFM enforces $h(u_1) = 0$ by discretizing $[0, 1]$ into N uniform substeps. At each timestep, $\tau \to \tau' = \tau + \Delta\tau$, three operations are performed: a one-step extrapolation to the terminal time using the learned velocity, a projection of the resulting candidate onto the constraint manifold $\mathcal{M} = \{u : h(u) = 0\}$, and a linear interpolation back to τ' along the optimal-

transport displacement (Utkarsh et al., 2025a):

$$\hat{u}_1 = u_\tau + (1-\tau)\, v_\theta(u_\tau, \tau), \tag{1}$$

$$u_1 = \arg\min_u \ \tfrac{1}{2}\|u - \hat{u}_1\|^2 \quad \text{s.t.} \quad h(u) = 0, \tag{2}$$

$$u_{\tau'} = u_0 + \tau'(u_1 - u_0). \tag{3}$$

The full PCFM algorithm of (Utkarsh et al., 2025a) additionally permits a penalized correction at each step, $\arg\min_u \|u - \hat{u}_{\tau'}\|^2 + \lambda \|h(u + (1-\tau')v_\theta(u, \tau'))\|^2$, to compensate for nonlinearity under coarse discretization. However, we omit it, as in our experiments, the streamlined three-step variant is sufficient.

The key structural feature exploited in this work is that the loop above is constraint-agnostic. Every operation except the projection in step 2 is a fixed arithmetic kernel, learning the entire specifications of the problem to be funneled into the projection subproblem. The cost of a single sampling pass therefore can be divided into two main terms: the constraint projections and the velocity evaluations. The forward shoot and OT pullback are independent of the constraints being enforced, whereas the projection step scales with the number of imposed constraints and their complexities, causing the projection step to be the bottleneck of the sampling, especially in these more complex cases. The remainder of this work focuses on optimizing the projection step via exploitation of the sparsity in the Jacobians, since constraints are independent across samples, and temporal couplings are typically local.

3. Theory

In this section, we discuss common constraints imposed on PDE systems and derive structural properties of the associated constraint Jacobians.

We adopt notation similar to that of Utkarsh et al. (2025a). Specifically, we consider partial differential equations on a bounded spatiotemporal domain $\Omega \times [0, T]$, where $\Omega \subset \mathbb{R}^d$. The true PDE solution is a field $u : \Omega \times [0, T] \to \mathbb{R}$, which we approximate using a discretized tensor $\mathbf{u} \in \mathbb{R}^{N_x \times N_t}$, where N_x and N_t are the number of spatial and temporal grid points, respectively.

In addition to satisfying the governing PDE, the solution must also satisfy a collection of physical constraints. These constraints are represented as

$$(\mathcal{H}_k(u))_k = \mathbf{0}$$

where $\mathcal{H}_k$ is the k-th constraint acting on the solution u. For convenience, we let $\mathcal{H}(u)$ denote the concatenation of all imposed constraints. Further, it is implied that the constraint operator can also work on the discretized solution $\mathbf{u}$.

Although the discussion in this section focuses on one-dimensional PDEs, the main ideas extend naturally to higher-dimensional settings.

Table 1. Overview of common constraint types for PDEs.

CONSTRAINT TYPE	FORM
DIRICHLET IC/BC	$Au - b$
GLOBAL MASS CONSERVATION	$\int_\Omega u\, dx - C$
NONLINEAR CONSERVATION LAW	$\frac{d}{dt}\int_\Omega \rho(u)\, dx - C$
NEUMANN OR FLUX BC	$\partial_n u(x,t) - g(x,t)$

3.1. Constraints

Constraints imposed for PDEs typically arise from physical principles such as conservation laws, initial conditions, and boundary conditions (LeVeque, 2004; 1992). These constraints may be linear or nonlinear in u, and may act locally or globally in space and time. A selection of commonly encountered constraints is summarized in table 1.

Most constraints encountered in practice are local in time. Meaning, they depend only on a single time step or a small number of neighboring time steps. For instance, an initial value constraint applies only at $t = 0$, while a mass conservation constraint typically applies to each time step independently. Even nonlinear conservation laws involving temporal derivatives can often be discretized using finite-difference schemes that couple only a small number of neighboring time steps, thereby preserving temporal locality (Quarteroni et al., 2007). The general Dirichlet IC/BC constraint formulated as $Au - b$ in 1 could, in principle, define a global coupling, depending on the structure of A. In practice, however, A is usually sparse and only couples nearby spatial or temporal nodes.

3.2. Jacobian Structure

The Gauss-Newton projection step presented in 2.1 can be reformulated as solving the constrained optimization problem

$$u_{\text{proj}} = \min_u ||u - u_1||_2^2 \quad \text{subject to } \mathcal{H}(u) = 0 \tag{4}$$

where the variables u can be interpreted as flattened vectors. Solving such a constrained optimization problem generally involves repeatedly solving large linear systems derived from the Karush-Kuhn-Tucker (KKT) conditions (Nocedal & Wright, 2006). These systems typically take the form

$$\begin{bmatrix} W & J_{\mathcal{H}}^T \\ J_{\mathcal{H}} & 0 \end{bmatrix} \tag{5}$$

where $J_{\mathcal{H}} = \frac{\partial \mathcal{H}}{\partial u}$ is the Jacobian of the constraints $\mathcal{H}$, and W is the Hessian of the Lagrangian. We defer a detailed

discussion of the KKT derivation to Appendix B. Regardless of the specific optimization method employed, computing the Jacobian - or an approximation to it - is generally a necessary part of the solution procedure (Nocedal & Wright, 2006).

A key observation is that physically motivated PDE constraints typically induce highly sparse Jacobians after discretization. Exploiting this sparsity is crucial for efficiently solving the associated linear systems. Moreover, for a fixed set of constraints, the sparsity pattern remains constant throughout the optimization process, allowing repeated linear solves to reuse structural information.

To understand why $J_{\mathcal{H}}$ is sparse, consider the discretized solution as a flattened vector $\mathbf{u} \in \mathbb{R}^{N_x N_t}$. The Jacobian then has dimensions $J_{\mathcal{H}} \in \mathbb{R}^{N_c \times N_x N_t}$, where N_c denotes the number of constraints.

If a constraint depends only on a single time step, at most N_x entries of the corresponding Jacobian row will be nonzero. Similarly, if a constraint involves only a few spatial steps, but is global in time, then each row contains only a small number of nonzero entries per time step, yielding at most $\mathcal{O}(N_t)$ non-zero entries overall. Furthermore, constraints that are global in one domain but local in the other often induce repeated block structures in the Jacobian, since the same constraint operator is applied repeatedly across spatial or temporal points. For an involved example involving both mass and an initial value constraint, we refer to Appendix C.

3.2.1. Solution Batches

It is also useful to consider the case where a batch of N_s samples is generated simultaneously. In that setting, the Jacobian has dimension $J_{\mathcal{H}} \in \mathbb{R}^{N_c N_s \times N_x N_t N_s}$. Since each sample is independent, the resulting Jacobian exhibits a block-diagonal structure, with each block corresponding to the constraints of an individual sample.

3.3. Sparse Solvers and GPU acceleration

Several optimization frameworks have recently been developed to exploit the structure and sparsity of the objective and constraint functions in large-scale nonlinear programs (NLP). Two notable libraries are `ExaModels.jl` (Shin et al., 2024a) and `MadNLP.jl` (Shin et al., 2024b).

ExaModels is an algebraic modeling and automatic differentiation framework designed for high-performance nonlinear optimization (Shin et al., 2024a). When using a general-purpose modeling framework such as `JuMP.jl` (Lubin et al., 2023a), it has been found that automatic differentiation may constitute a significant fraction of the total solver time (Shin et al., 2024b). ExaModels is designed to mitigate this issue by generating sparsity-aware computational graphs and compiling them into SIMD-parallel kernels through `ExaCore`. This allows objectives, constraints, Jacobians and Hessians to be evaluated efficiently in a single sparsity-preserving pass, while exploiting the parallelism available on modern GPU architectures. (Shin et al., 2024b).

MadNLP implements a primal-dual interior point solver, working together with ExaModels to form and solve sparse KKT systems efficiently. Since the sparsity pattern of the Jacobian typically remains fixed throughout the optimization process, MadNLP can leverage this to optimize the linear system solves, significantly reducing overhead associated with repeated factorizations of the Jacobian (Shin et al., 2024b).

Furthermore, MadNLP works across both CPU and GPU. On GPUs, MadNLP interfaces with NVIDIA's GPU sparse linear solver, cuDSS (2025). cuDSS solves sparse linear systems using variants of LU, LDL^T or Cholesky factorization, and is specifically designed for exploiting the parallelism of NVIDIA GPU architectures. Large, sparse KKT systems can thus be solved substantially faster than with traditional CPU-based solvers (Pacaud & Shin, 2024).

4. Methodology and Setup

We evaluate SNAP-FM via benchmarking of the PCFM sampling loop across a set of PDE problems chosen to span the constraint regimes of practical interest, and for each problem, compare the runtime and constraint-satisfaction behavior of numerous optimization backends inside the projection step. For implementation details and scripts used to reproduce the experiments, we provide the code at https://github.com/xenakistheo/PCFM.jl. Additional details on the experimental setup, hardware, and benchmarking procedure are given in Appendix E.

4.1. Optimization Methods

Our main proposed solver, **ExaModels + MadNLP on GPU** is benchmarked against five baselines that utilize different algebraic modeling layers, NLP solvers, and execution hardware. We chose this set of methods such that comparisons against the adjacent baselines allow for isolation of contributions of each of the three axes.

ExaModels + MadNLP (GPU) As mentioned earlier, `ExaModels.jl` (Shin et al., 2024a) compiles the projection NLP into SIMD-parallel GPU kernels via `ExaCore`, allowing for evaluation of objectives, constraints and ADD in a single sparsity-preserving pass. The resulting structured KKT system is then solved entirely on the GPU by `MadNLP.jl` (Shin et al., 2024b), an interior-point (IP) solver, with the sparse linear systems broken down via

cuDSS. This combination is a promising direction for large-scale batches where the full Jacobian structure can be exploited for GPU-accelerated NLP.

ExaModels + MadNLP (CPU) This optimization method is of identical modeling and solver to our main method, but instead executed on the CPU. This baseline isolates the contribution of GPU acceleration. Namely, any performance gap that is found between this method and our above approach is purely attributed to hardware utilization.

JuMP + MadNLP (CPU) We utilize the same NLP solver, but with the model instead expressed in `JuMP.jl` (Lubin et al., 2023b), a general-purpose algebraic modeling package. JuMP differs from ExaModels in that it does not preserve SIMD structure or compile to GPU kernels. Therefore, this allows for isolation of the contribution of the modeling layer, specifically the value of structure-preserving compilation.

JuMP + Ipopt Ipopt (Wächter & Biegler, 2006) is one of the most widely deployed NLP solvers in scientific computing. We include it as a standard benchmark any practitioner would reach for when including nonlinear constraints in `PCFM.jl`, and serves as empirical evidence that projection remains the sampling bottleneck in the nonlinear regime.

Optimization.jl + IPNewton We implemented Optimization.jl with IPNewton to benchmark with a second interior-point baseline through the `Optimization.jl` (Dixit & Rackauckas, 2023) unified interface. This ensures that any observed performance improvements are attributed correctly to MadNLP implementations of interior-point methods, not just the IP method itself.

Optimization.jl + L-BFGS We include L-BFGS as an unconstrained optimization baseline, ensuring the structured-NLP framing is truly advantageous and necessary. Unlike the other methods, L-BFGS does not impose hard equality constraints. Instead, the constrained projection problem is converted into a different mathematical problem in which the constraint residuals enter through a penalty term. This distinction is important when interpreting runtime and feasibility results, since a faster L-BFGS should not be read as a faster solution of the hard-constrained projection problem.

4.2. Test Problems

We consider generative modeling for a collection of six different PDE benchmark problems with varying physical constraints. These problems are chosen to evaluate how different constraint sets influence both inference quality and computational cost. More importantly, we are also interested in how these results vary across the different solvers. The benchmark problems are as follows

- One dimensional heat equation with initial value and mass-conservation constraints.
- One dimensional heat equation with initial value, mass-conservation, and energy evolution constraints.
- One dimensional reaction diffusion equation with Neumann boundary conditions and a nonlinear conservation constraint.
- One dimensional Burgers' equation with boundary and mass-conservation constraints.
- One dimensional Burgers' equation with initial value, mass-conservation, and local flux constraints.
- Two-dimensional Navier-Stokes equation in vorticity form with initial-value, total-vorticity conservation, and enstrophy constraints.

The test problems with their constraints are described in detail in Appendix A.

For both the heat equation and Burgers' equation, the paired test problems use identical pretrained FFM models. In each case, the training dataset and network architecture are unchanged, and only the inference-time constraints differ.

For the Burgers' with initial value constraint, we also experiment with removing the mass-conservation, and the flux constraints in a scaling study.

The discretization procedures used to enforce the constraints are described in Appendix D.

5. Experiments and Results

We evaluate the proposed methods on the benchmark problems outlined in Section 4.2. These problems span constraint regimes of increasing difficulty, including linear and nonlinear one-dimensional constraints, as well as a nonlinear two-dimensional problem. For each benchmark, we compare all optimization backends from Section 4.1 in terms of runtime, and constraint violation. Beyond the main benchmark results, we include two scaling studies to better characterize the computational behavior of the projection step. First, we consider the heat equation with fixed initial condition and mass-conservation constraints, and vary the number of generated samples to study scaling with batch size. Second, we consider the Burgers equation under increasingly rich constraint sets, allowing us to assess how solver performance changes as additional nonlinear constraints are imposed.

Unless otherwise stated, all reported runtimes are averaged over four repetitions with 32 generated samples — an exception being the two-dimensional Navier-Stokes problem

Table 2. Heat Equation - 1

OPTIMIZATION SETUP	TIME (S)	INFEASIBILITY
EXA, MADNLP, GPU	$\mathbf{11.73 \pm 0.34}$	3.19E-6
EXA, MADNLP, CPU	14.29 ± 0.15	3.23E-7
JUMP, MADNLP, CPU	137.94 ± 4.42	7.28E-7
JUMP, IPOPT, CPU	182.72 ± 4.73	7.44E-7
IPNEWTON	6225.35 ± 38.60	5.31E-2
L-BFGS	1623.98 ± 107.94	3.94E-2

Table 3. Heat Equation - 2

OPTIMIZATION SETUP	TIME (S)	INFEASIBILITY
EXA, MADNLP, GPU	$\mathbf{110.26 \pm 1.92}$	0.222
EXA, MADNLP, CPU	3501.11 ± 89.36	0.222
JUMP, MADNLP, CPU	DNF	-
JUMP, IPOPT, CPU	DNF	-
IPNEWTON	DNF	-
L-BFGS	977.87 ± 39.22	4.49E-5

Table 4. Burgers' (BC)

OPTIMIZATION SETUP	TIME (S)	INFEASIBILITY
EXA, MADNLP, GPU	153.53 ± 1.50	0.564
EXA, MADNLP, CPU	$\mathbf{109.74 \pm 7.67}$	0.550
JUMP, MADNLP, CPU	438.26 ± 38.94	0.550
JUMP, IPOPT, CPU	1177.00 ± 75.58	1.747
IPNEWTON	DNF	-
L-BFGS	170.04 ± 21.89	2.038

Table 5. Burgers' (IC, Mass, Flux)

OPTIMIZATION SETUP	TIME (S)	INFEASIBILITY
EXA, MADNLP, GPU	$\mathbf{38.83 \pm 0.55}$	0.920
EXA, MADNLP, CPU	2048.83 ± 187.58	1.594
JUMP, MADNLP, CPU	DNF	-
JUMP, IPOPT, CPU	DNF	-
IPNEWTON	DNF	-
L-BFGS	DNF	-

for which we generated 2 samples due to the substantially larger state dimension. Wall-clock timings are reported using `BenchmarkTools.jl`'s `@btime` $\pm$ a standard deviation. *DNF* denotes runs that did not converge within the allotted runtime budget of six hours.

5.1. Runtime

Tables 2, 3, 4, 5, 6 & 7 present the central results of this paper: end-to-end sampling time for a full pass of PCFM under each method and constraint regime, broken down into the model-evaluation and projection components for specification. The fastest solver in each constraint regime is highlighted in bold.

A few observations worth highlighting. Notably, the ExaModels + MadNLP combination beat the other methods in all six test problems. Moreover, the gap between the GPU and CPU implementation isolates the contribution of the hardware. This difference between the two was especially pronounced in the case of the heat equation with an energy-evolution constraint, as well as the Burgers' PDE with Godunov's flux constraints. We hypothesize that the nonlinearity of these two sets of constraints made it significantly harder to solve as an optimization problem. Hence, the difference in performance became more pronounced. Secondly, the difference in the results of JuMP + MadNLP and ExaModels + MadNLP both run on CPU isolates the modeling layer, which is our main focus here. What we observe is that there is a consistent speedup from the exploitation of the sparsity structure of the Jacobian matrices at compile time. Even for the simple constraints in the heat equation, 2, the difference is significant.

We also note that missing entries correspond to runs that did not complete within the runtime budget. These failures are themselves informative: all methods except for the ExaModels + MadNLP failed to converge on the Burgers' Equation constraint that included IC and flux constraints within a reasonable amount of time. IPNewton also failed on Burgers' with boundary constraints, as well as on the Navier-Stokes' equation.

5.2. Constraint Violation

Beyond analyzing the runtimes, we want to examine constraint violations along the sampling trajectory.

Constraint satisfaction is quantified using a metric that measures the average magnitude of the constraint violations across all samples in a batch. Violations are normalized within each constraint type before being aggregated, ensuring that constraints with different scales or numbers of instances contribute comparably. The resulting scalar score provides a fair basis for comparing optimization backends on a given set of constraints. We refer to this metric as *Infeasibility*.

Figure 1 follows the mass-conservation residual along the sampling trajectory for our Burgers' equation constraint under each method. The structure-exploiting methods, namely `ExaModels + MadNLP` on GPU and CPU as well as `JuMP + MadNLP` produce residuals that oscillate around zero with a magnitude $\sim 10^{-3}$. To contrast, the unstructured methods produce residuals that appear biased. IPNewton undershoots the constraint in the early steps and overshoots in the latter, while L-BFGS decays negatively throughout. While the overall magnitudes are relatively similar, the qualitative behavior of our methods differs.

The infeasibility values in our results also highlight an im-

Table 6. Reaction-Diffusion Equation

OPTIMIZATION SETUP	TIME (S)	INFEASIBILITY
EXA, MADNLP, GPU	35.68 ± 0.74	4.60E-4
EXA, MADNLP, CPU	$\mathbf{24.58 \pm 1.69}$	4.49E-4
JUMP, MADNLP, CPU	173.15 ± 3.56	4.49E-4
JUMP, IPOPT, CPU	228.80 ± 4.35	4.49E-4
IPNEWTON	2314.20 ± 62.47	3.67E-4
L-BFGS	484.19 ± 45.62	1.69E-3

Table 7. Navier-Stokes' Equation

OPTIMIZATION SETUP	TIME (S)	INFEASIBILITY
EXA, MADNLP, GPU	$\mathbf{45.69 \pm 0.96}$	0.678
EXA, MADNLP, CPU	118.49 ± 0.84	0.678
JUMP, MADNLP, CPU	79.20 ± 2.02	1.131
JUMP, IPOPT, CPU	59.38 ± 0.74	1.131
IPNEWTON	DNF	-
L-BFGS	298.30 ± 14.26	1.766

portant nuance. The constraints are solved up to the numerical tolerances of the nonlinear optimizer. Thus, the reported infeasibility should be interpreted as the residual constraint violation remaining after convergence, rather than as evidence that the constraints are ignored. In all experiments, we used the solver's default feasibility tolerance of 10^{-4}. Tighter or problem-specific tolerances may further reduce constraint violation, but could also increase runtime or affect solver robustness. Investigating how to tune these tolerances, and how this trade-off impacts both feasibility and sampling cost, is an important direction for future work.

5.2.1. SCALING STUDY - NUMBER OF SAMPLES

Figure 2 shows the runtime of each optimization backend as the number of generated samples is increased. The runtime grows approximately linearly with the number of samples for all methods. This is consistent with the block-diagonal structure of the batched projection problem: increasing the batch size adds independent constraint blocks, but does not introduce additional cross-sample coupling. The relative ordering of the methods is also largely preserved across batch sizes.

5.2.2. SCALING STUDY - NUMBER OF CONSTRAINTS

Table 8 shows the runtime and infeasibility values as additional constraints are imposed on the Burgers equation. Even for the smaller nonlinear constraint sets, MadNLP-based methods are substantially faster than the alternative baselines. As local flux constraints are added, the projection problem becomes significantly more difficult. In this regime, GPU acceleration becomes increasingly important. The ExaModels + MadNLP GPU backend is the only method that remains tractable for the largest tested constraint set.

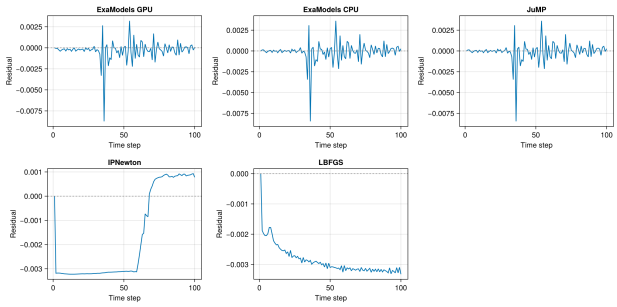

Figure 1. Mass-conservation residual along the PCFM sampling trajectory for Burgers' equation. Structure-exploiting methods (top row) produce residuals that oscillate around zero, consistent with constraint satisfaction modulo numerical noise. Unstructured methods (bottom row) produce systematically biased residuals: `IPNewton` undershoots then overshoots, `L-BFGS` drifts monotonically.

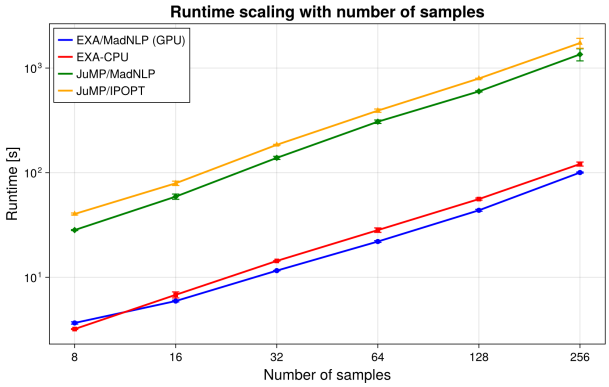

Figure 2. Runtime scaling with the number of generated samples for the first heat-equation constraint regime. Time scales linearly with number of samples, and ordering is preserved.

The constraint sets increase in complexity. *IC-only* fixes the initial condition, while *IC, Mass* additionally enforces conservation of the spatial integral over time. The Flux(k) constraints add k local Godunov update residuals – see Appendix D.

6. Conclusion

We present an acceleration of the projection subproblem in Physics-Constrained Flow Matching by exploiting the sparsity structure inherent in the constraints. This work is motivated by the observation that the computational complexity of the projection step scales poorly during the PCFM sampling process as the number and complexity of constraints increase. By analyzing the Jacobian sparsity pattern, we reformulate the problem as a collection of independent per-sample subproblems while additionally leveraging the fixed sparsity structure arising from the discretized conservation constraints. To exploit this structure efficiently on the

Table 8. Runtime scaling with the number of imposed constraints for the Burgers equation. DNF denotes runs that did not finish within the runtime budget.

OPTIMIZATION SETUP	IC-ONLY	IC, MASS	IC, MASS, FLUX(1)	IC, MASS, FLUX(5)	IC, MASS, FLUX(10)
EXA, MADNLP, GPU	6.90±0.49	19.32±0.49	24.23±0.09	39.03±0.36	62.95±2.07
EXA, MADNLP, CPU	25.56±0.17	24.40±0.51	1632.12±73.94	2038.96±215.10	DNF
JUMP, MADNLP, CPU	155.33±1.61	346.27±3.75	DNF	DNF	DNF
JUMP, IPOPT, CPU	156.28±1.87	DNF	DNF	DNF	DNF
IPNEWTON	50.38±3.56	DNF	DNF	DNF	DNF
L-BFGS	3.92±0.01	DNF	DNF	DNF	DNF

GPU, we employed `ExaModels.jl` as the modeling layer together with `MadNLP.jl` as the nonlinear programming solver. Across our benchmarks, this approach yields a significant speedup for computationally demanding nonlinear constraints, while not compromising on correctness.

Acknowledgments

We thank the reviewers for their thoughtful comments and constructive suggestions, which helped improve the clarity and quality of this paper.

This material is based upon work supported by the U.S. National Science Foundation under award Nos CNS-2346520, RISE-2425761, and DMS-2325184, by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00112490488, by the Department of Energy, National Nuclear Security Administration under Award Number DE-NA0004266 and by the United States Air Force Research Laboratory under Cooperative Agreement Number FA8750-19-2-1000. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Impact Statement

This paper presents work whose goal is to make physically consistent generative simulation more practical for scientific applications. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Baldan, G., Liu, Q., Guardone, A., and Thuerey, N. Flow matching meets pdes: A unified framework for physics-constrained generation. *arXiv preprint arXiv:2506.08604*, 2025.

Ben-Hamu, H., Puny, O., Gat, I., Karrer, B., Singer, U., and Lipman, Y. D-flow: Differentiating through flows for controlled generation, 2024. URL https://arxiv.org/abs/2402.14017.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., et al. Jax: composable transformations of python+ numpy programs. 2018.

Cai, P., Utkarsh, U., Segal, N., Subramanian, A., Jager, K., Pan, E., Edelman, A., Rackauckas, C. V., and Gomez-Bombarelli, R. Enforcing constraints in molecular and crystalline generative models via physics-constrained flow matching. In *AI for Accelerated Materials Design - ICLR 2026*, 2026. URL https://openreview.net/forum?id=QVQjDmIFg0.

Cheng, C., Han, B., Maddix, D. C., Ansari, A. F., Stuart, A., Mahoney, M. W., and Wang, Y. Gradient-free generation for hard-constrained systems, 2025. URL https://arxiv.org/abs/2412.01786.

Christopher, J. K., Baek, S., and Fioretto, F. Constrained synthesis with projected diffusion models. *Advances in Neural Information Processing Systems*, 37:89307–89333, 2024.

Dixit, V. and Rackauckas, C. Optimization.jl: A unified interface for mathematical optimization in julia. In *JuliaCon Proceedings*, 2023.

Greydanus, S., Dzamba, M., and Yosinski, J. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

Huang, J., Yang, G., Wang, Z., and Park, J. J. Diffusion-pde: Generative pde-solving under partial observation. *Advances in Neural Information Processing Systems*, 37: 130291–130323, 2024.

LeVeque, R. J. *Numerical Methods for Conservation Laws*. Lectures in Mathematics ETH Zürich. Birkhäuser, Basel, 1992.

LeVeque, R. J. *Finite-Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2004.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations, 2021. URL https://arxiv.org/abs/2010.08895.

Lu, H. and Yang, J. cupdlp. jl: A gpu implementation of restarted primal-dual hybrid gradient for linear programming in julia. *Operations Research*, 73(6):3440–3452, 2025.

Lubin, M., Dowson, O., Dias Garcia, J., Huchette, J., Legat, B., and Vielma, J. P. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 15:581–589, 2023a. doi: 10.1007/s12532-023-00239-3.

Lubin, M., Dowson, O., Dias Garcia, J., Huchette, J., Legat, B., and Vielma, J. P. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 15:581–589, 2023b. doi: 10.1007/s12532-023-00239-3.

Nocedal, J. and Wright, S. *Numerical Optimization*. Springer, 2006.

NVIDIA Corporation. NVIDIA cuDSS: A high-performance CUDA library for direct sparse solvers, 2025. URL https://docs.nvidia.com/cuda/cudss/. Accessed: 2026-05-07.

Pacaud, F. and Shin, S. GPU-accelerated dynamic nonlinear optimization with ExaModels and MadNLP. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 5963–5968, 12 2024. doi: 10.1109/CDC56724.2024.10886720.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Price, I., Sanchez-Gonzalez, A., Alet, F., Andersson, T. R., El-Kadi, A., Masters, D., Ewalds, T., Stott, J., Mohamed, S., Battaglia, P., et al. Gencast: Diffusion-based ensemble forecasting for medium-range weather. *arXiv preprint arXiv:2312.15796*, 2023.

Quarteroni, A., Sacco, R., and Saleri, F. *Numerical Mathematics*. Springer, 2007.

Raissi, M., Perdikaris, P., and Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Rennich, S. C., Stosic, D., and Davis, T. A. Accelerating sparse cholesky factorization on gpus. In *2014 4th Workshop on Irregular Applications: Architectures and Algorithms (IAˆ3)*, pp. 9–16. IEEE, 2014.

Richter-Powell, J., Lipman, Y., and Chen, R. T. Neural conservation laws: A divergence-free perspective. *Advances in Neural Information Processing Systems*, 35: 38075–38088, 2022.

Römer, R., von Rohr, A., and Schoellig, A. P. Diffusion predictive control with constraints. *arXiv preprint arXiv:2412.09342*, 2024.

Shin, S., Anitescu, M., and Pacaud, F. Accelerating optimal power flow with GPUs, 2024a. ExaModels.jl + MadNLP.jl v0.8 release, Julia Discourse, March 2024.

Shin, S., Pacaud, F., and Anitescu, M. Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods. *Electric Power Systems Research*, 236, 2024b.

Utkarsh, U., Cai, P., Edelman, A., Gomez-Bombarelli, R., and Rackauckas, C. Physics-constrained flow matching: Sampling generative models with hard constraints. *NeurIPS*, 2025a.

Utkarsh, U., Maddix, D. C., Ma, R., Mahoney, M. W., and Wang, Y. End-to-end probabilistic framework for learning with hard constraints. *arXiv preprint arXiv:2506.07003*, 2025b.

Wächter, A. and Biegler, L. T. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, March 2006. doi: 10.1007/s10107-004-0559-y.

Yuan, Y., Song, J., Iqbal, U., Vahdat, A., and Kautz, J. Physdiff: Physics-guided human motion diffusion model. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 16010–16021, 2023.

A. Model Problems

Many of the constraints considered in this work are formulated in terms of conserved integral quantities. We therefore define the mass and energy of a solution u at time t by

$$m(t) := \int_\Omega u(x,t)\,dx \tag{6}$$

$$E(t) := \frac{1}{2}\int_\Omega u^2(x,t)\,dx \tag{7}$$

A.1. Heat Equation Problems

We first consider the one-dimensional heat equation

$$u_t = \alpha u_{xx}, \quad \Omega = [0, 2\pi], \quad t \in [0,1] \tag{8}$$

with periodic boundary conditions $u(0,t) = u(2\pi, t)$, and initial values given by

$$u(x,0) = u^{\mathrm{H}}_{IC}(x) := \sin(x+\phi)$$

The spatiotemporal gridsizes are $N_x = N_t = 100$.

For this specific problem, there is an analytical solution given by

$$u_{\text{analytic}}(x,t) = \exp(-\alpha t)\sin(x+\phi) \tag{9}$$

The FFM model was trained using a dataset constructed by computing the analytical solution for different diffusion coefficients and phases. Namely, the parameters were sampled from the following distributions.

$$\alpha \sim \mathcal{U}(1,5), \quad \phi \sim \mathcal{U}(0,\pi)$$

The training dataset contained $10,000$ such solutions.

During inference, we evaluate on two different constraint regimes.

In the first regime, we fix $\phi = \pi/4$ to keep the initial condition constant, and we enforce the initial value and mass conservation constraints.

$$\mathcal{H}^{\mathrm{H},1}(u) = \begin{bmatrix} u(x,0) - u^{\mathrm{H}}_{IC}(x) \\ m(t) - m(0) \end{bmatrix} \tag{10}$$

In the second set of constraints, we additionally impose an energy-evolution constraint together with a local PDE constraint. The resulting constraint operator is given by

$$\mathcal{H}^{\mathrm{H},2}(u) = \begin{bmatrix} u(x,0) - u^{\mathrm{H}}_{IC}(x) \\ m(t) - m(0) \\ u_t - \alpha u_{xx} \\ \frac{d}{dt}\frac{1}{2}\int_\Omega (u(x,t))^2\,dx + \alpha\int_\Omega (u_x)^2\,dx \end{bmatrix} \tag{11}$$

A derivation and motivation for the energy-evolution constraint are provided in Appendix D.

A.2. Burgers'

Secondly, consider the one-dimensional inviscid Burgers' equation

$$u_t + \frac{1}{2}(u^2)_x = 0, \quad \Omega = [0,1], \quad t \in [0,1] \tag{12}$$

The training dataset was constructed by generating solutions with the Godunov finite volume method (LeVeque, 2004), on a $N_x = N_t = 101$ grid. For the training set, the following three conditions were imposed

1. Initial value:
$$u(x,0) = u^{\text{B}}_{\text{IC}}(x; p_{\text{loc}}) := \frac{1}{1+\exp(\frac{x-p_{\text{loc}}}{\epsilon})}$$
with $\epsilon = 0.02$ fixed, and the location sampled randomly from $p_{\text{loc}} \sim \mathcal{U}(0.2, 0.8)$.

2. Dirichlet condition on the left boundary
$$u(0,t) = u_{\text{bc}} \quad \text{with } u_{\text{bc}} \sim \mathcal{U}(0,1)$$

3. Von Neumann condition on the right
$$\frac{d}{dx}u(1,t) = 0$$

80 different values of p_{loc} and u_{bc} were sampled each, generating a total of 6400 solutions for the training dataset.

The first regime of constraints considered for the Burgers' equation imposes a Dirichlet condition on the left boundary, and a Neumann condition on the right boundary. The Dirichlet value is not fixed, but varies across samples - as in the training data. There is also a mass conservation constraint. Together the constraints are given by

$$\mathcal{H}^{\text{B,BC}}(u) = \begin{bmatrix} u(0,t) - u_L \\ m(t) - m_0 \\ \partial_n u(1,t) \end{bmatrix} \tag{13}$$

In the second regime, we impose an initial value constraint, with p_{loc} sampled as in the training set. In addition to the typical mass constraint we also impose a sequence of local conservation updates based on Godunov's flux method. The constraint function then becomes

$$\mathcal{H}^{\text{B,IC}}(u) = \begin{bmatrix} u(x,0) - u^{\text{B}}_{\text{IC}}(x; p_{\text{loc}})) \\ m(t) - m(0) \\ R^{(k)}_{\text{Flux}}(u) \ (k = 1, .., 5) \end{bmatrix} \tag{14}$$

A derivation and description of the flux constraint based on the Godunov method are provided in Appendix D.

A.3. Reaction-Diffusion Equation

Next, we consider a nonlinear reaction-diffusion equation given by

$$u_t = \rho u(1-u) - \nu u_{xx} \tag{15}$$
$$\Omega = [0,1], \quad t \in [0,1] \tag{16}$$

with parameters $(\rho, \nu) = (0.01, 0.005)$. The equation is discretized on a grid with spatial and temporal resolutions $N_x = 128$ and $N_t = 100$, respectively.

The initial conditions are sampled from randomized combinations of sinusoidal and localized bump functions. The training dataset is generated by pairing 80 distinct initial conditions with 80 boundary conditions, resulting in a total of 6400 PDE solutions. The solutions used for training are computed using a semi-implicit finite difference scheme.

$$\mathcal{H}^{\text{RD}}(u) = \begin{bmatrix} u(x,0) - u_{\text{IC}}(x) \\ m(t) - \Big(m_0 + \int_0^t \rho u(1-u)\, d\tau \\ + \int_0^t (g_L(\tau) - g(\tau))\, d\tau\Big) \end{bmatrix} \tag{17}$$

A.4. Navier-Stokes Equation

Lastly we consider the Navier-Stokes equation in two dimensions, given in its vorticity form with periodic boundary conditions as

$$w_t + u \cdot \nabla w = \nu \nabla^2 w + f(x) \tag{18}$$

$$\Omega = [0,1]^2, \quad t \in [0, 49] \tag{19}$$

where $w := \nabla \times u$ is the vorticity, and we set the viscosity $\nu = 10^{-3}$. We let the forcing function $f(x) = 0.1\sqrt{2}\sin(2\pi(x+y)+\phi)$. The spatiotemporal resolutions are as follows $N_x = N_y = 16$, $N_t = 50$. The training dataset was constructed by solving the governing problem with a Crank–Nicolson spectral solver. Specifically, $10,000$ simulations were generated from 100 randomly sampled initial vorticities w_0, drawn from a Gaussian random field, and 100 forcing phases $\phi \sim \mathcal{U}(0, \pi/2)$. With periodic boundary conditions the vorticity-mass is conserved. We further impose an initial value constraint, and an energy conservation constraint on the vorticity. In total, these are the constraints:

$$\mathcal{H}^{\text{NS}}(w) = \begin{bmatrix} w(x,y,0) - w_{\text{IC}}(x,y) \\ \displaystyle\int_\Omega w(x,y,t)\,dx\,dy - \int_\Omega w_{\text{IC}}(x,y)\,dx\,dy \\ \displaystyle\int_\Omega w(x,y,t)^2\,dx\,dy - \int_\Omega w_{\text{IC}}(x,y)^2\,dx\,dy \end{bmatrix} \tag{20}$$

B. Newton-KKT system

Consider the equality constrained optimisation problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to } g(x) = 0 \tag{21}$$

for some objective function $f : \mathbb{R}^n \to \mathbb{R}$, and constraint function $g : \mathbb{R}^n \to \mathbb{R}^m$

Our goal is to show that approximating this solution can be reduced to repeatedly solving linear systems of the form $Au = b$, where A has a characteristic sparse structure that is highly dependent on the Jacobian of g.

The key idea to solving (21) is that at the optimum, we cannot improve the objective while remaining in the feasible set. Mathematically, this means that if we suppose that x^* solves

Suppose x^* solves (21). Consider a small perturbation Δx. To remain within the feasible set we must require

$$g(x^* + \Delta x) = 0$$

Linearization gives

$$g(x^*) + J(x^*)\Delta x = 0$$

where we have introduced the Jacobian of g, $J(u) := \nabla_x g(u) \in \mathbb{R}^{m \times n}$. As x^* by assumption is in the feasible set, we have that $J(x^*)\Delta x = 0$. In other words, all feasible perturbations must lie in the nullspace of the Jacobian.

Moreover, as x^* is an optimum, one cannot decrease the objective function in any feasible direction. This implies that

$$\nabla f(x^*)^T \Delta x = 0, \quad \forall \Delta x \text{ satisfying } J(x^*)\Delta x = 0$$

A basic linear algebra result is that a vector v which is orthogonal to the nullspace of a matrix Q, is necessarily in the rowspace of Q. This implies that we can write

$$\nabla f(x^*) = -J(x^*)^T \lambda$$

for some $\lambda \in \mathbb{R}^m$.

These two conditions together form the Karush-Kuhn-Tucker (KKT) conditions that we require the solution to satisfy

$$\begin{aligned} \nabla f(x) + (J(x))^T \lambda &= 0 \\ g(x) &= 0 \end{aligned}$$

By introducing the Lagrangian $\mathcal{L}(x, \lambda) = f(x) + (J(x))^T \lambda$ we can also reformulate the problem of finding x^* so that $F(x, \lambda) = 0$, where we define

$$F : \mathbb{R}^{n+m} \to \mathbb{R}^{n+m}, \quad F(x, \lambda) := \begin{bmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ g(x) \end{bmatrix} \tag{22}$$

One can recall that a single iteration of the multivariate Newton-Raphson method for solving the problem $G(u) : \mathbb{R}^p \to \mathbb{R}^q$ takes the form

$$u^{(k+1)} = u^{(k)} - (\nabla G(u^{(k)}))^{-1} G(u^{(k)})$$

which can be simplified to solving the linear equation

$$\nabla G(u^{(k)}) \Delta_k = -G(u^{(k)}), \quad \Delta_k := u^{(k+1)} - u^{(k)} \tag{23}$$

Hence, to solve $F(x, \lambda) = 0$ using the Newton-Raphson method, we need to compute $\nabla_{(x,\lambda)} F(x, \lambda)$.

A straightforward computation yields the linear system

$$\begin{bmatrix} W & J^T \\ J & 0 \end{bmatrix} \begin{bmatrix} \Delta x_n \\ \Delta \lambda_n \end{bmatrix} = - \begin{bmatrix} \nabla f(x_n) + (J(x_n))^T \lambda_n \\ g(x_n) \end{bmatrix}, \quad W := \nabla^2_{xx} \mathcal{L}(x_n, \lambda_n) \tag{24}$$

The matrix on the left-hand side of (24) is called the *Newton-KKT system*. Its sparse block structure plays a central role in large-scale constrained optimisation algorithms.

C. Jacobian Structure Example

This appendix presents an example illustrating the sparsity structure in the Jacobian arising from a common set of PDE constraints.

Consider a PDE subject to the following constraints.

$$\mathcal{H}(u) = \begin{bmatrix} u(x, 0) - f(x) \quad \forall x \in \Omega \\ \int_\Omega u(x, t)\, dx - \int_\Omega u(x, 0)\, dx \quad \forall t \in [0, T] \end{bmatrix} = \mathbf{0}$$

Assume the solution is discretized on a spatiotemporal grid with $(N_x, N_t) = (3, 3)$ grid points: x_1, x_2, x_3, and t_1, t_2, t_3. Let $\mathbf{u}_i^{(k)}$ denote the approximation of $u(x_i, t_k)$.

The initial value constraint then becomes

$$\mathbf{u}_i^{(1)} - f_i = 0, \qquad f_i := f(x_i), \quad i = 1, 2, 3$$

Using a simple left Riemann sum approximation to the integral, the mass conservation constraint takes the form

$$\sum_{i=1}^{2} \mathbf{u}_i^{(k)} - \sum_{i=1}^{2} \mathbf{u}_i^{(1)} = 0, \quad k = 2, 3$$

where the constraint at the first timestep is omitted.

In total, this yields 5 constraints. For convenience, we represent the approximation $\mathbf{u}$ as the flattened vector

$$\mathbf{u} = (\mathbf{u}_1^{(1)}, \mathbf{u}_2^{(1)}, \mathbf{u}_3^{(1)}, ..., \mathbf{u}_1^{(3)}, \mathbf{u}_2^{(3)}, \mathbf{u}_3^{(3)})^T \in \mathbb{R}^9$$

.

The Jacobian of the constraint operator, $J_{\mathcal{H}} \in \mathbb{R}^{5\times 9}$ has entries $(J_{\mathcal{H}})_{i,j} = \frac{\partial \mathcal{H}_i}{\partial \mathbf{u}_j}$, where the constraints are ordered such that the initial value constraints appear first.

A straightforward computation gives

$$J_{\mathcal{H}} = \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{-1} & \mathbf{-1} & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ \mathbf{-1} & \mathbf{-1} & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \end{bmatrix} \tag{25}$$

The 3×3 identity block in the upper-left corner arises from the initial value constraints. More generally, initial value constraints typically contribute block-diagonal structure to the Jacobian.

Similarly, regardless of the number of grid points, a mass conservation constraint discretized using a left Riemann sum produces rows containing a sequence of $N_x - 1$ nonzero entries followed by a zero, reflecting the structure of the underlying quadrature rule.

D. Constraints

D.1. Energy Evolution Constraint

In generating solutions for the heat equation we considered an energy evolution constraint. Here, ”energy” is meant as

$$E(t) := \frac{1}{2}\int_\Omega (u(x,t))^2 \, dx$$

not to be confused with the physical ”energy” of the system. The energy referred to here is more of a measure of variance of u. We might want to enforce that the temperature profile, u is smoothed out over time. In other words, that the temporal derivative $\frac{d}{dt}E(t) \leq 0$. If one assumes zero flux on the boundary, then this is a purely mathematical artifact of the heat equation.

$$\begin{aligned} \frac{d}{dt}\frac{1}{2}\int_\Omega (u(x,t))^2 \, dx &= \int_\Omega u u_t \, dx \\ &= \int_\Omega \alpha u u_{xx} \, dx = -\alpha \int_\Omega (u_x)^2 \, dx \end{aligned}$$

where the second-to-last equality comes from substituting in the heat equation, and the last equality comes from integration by parts - assuming zero flux on boundary. Importantly, the last expression must be negative, hence, we can enforce

$$\frac{d}{dt}\frac{1}{2}\int_\Omega (u(x,t))^2 \, dx = -\alpha \int_\Omega (u_x)^2 \, dx$$

This constraint enforces that the energy of the solution, $E(t)$, which is to be interpreted as the variability of the solution in space, decreases over time - much as one would expect from a diffusive process such as the heat equation.

D.2. Godunov’s Flux Conservation

For the Burgers’ equation with fixed initial condition, we additionally imposed a sequence of local conservation constraints based on Godunov’s finite-volume method. The purpose of this constraint was to encourage the neural solution to satisfy not only global mass conservation, but also the local conservative transport structure of the PDE.

Specifically, the constraint operator included the residual terms

$$R_{\text{Flux}}^{(k)}(u), \qquad k = 1, \ldots, 5,$$

where each residual corresponds to one conservative update step of a finite-volume discretization.

Let the spatial domain be partitioned into cells indexed by i, with cell averages

$$u_i^k \approx \frac{1}{\Delta x} \int_{x_i}^{x_{i+1}} u(x, t_k)\, dx.$$

For Burgers' equation

$$u_t + \frac{1}{2}(u^2)_x = 0,$$

Godunov's method updates the solution according to

$$u_i^{k+1} = \nu_i^k - \frac{\Delta t}{\Delta x}\left(F_{i+\frac{1}{2}}^k - F_{i-\frac{1}{2}}^k\right),$$

where $F_{i+\frac{1}{2}}^k$ denotes the numerical flux through the interface between neighboring cells.

For Burgers' equation, the Godunov flux is given by

$$F(a, b) = \begin{cases} \min_{u \in [a,b]} \frac{u^2}{2}, & a \leq b, \\ \max_{u \in [b,a]} \frac{u^2}{2}, & a > b, \end{cases}$$

with $a = u_i^k$ and $b = u_{i+1}^k$.

Using the neural network prediction evaluated on the spatial grid, we defined the flux residual as the mismatch between consecutive predicted states and a single Godunov update:

$$R_{\text{Flux}}^{(k)}(u) = u^{k+1} - \left[u^k - \frac{\Delta t}{\Delta x}\left(F_{i+\frac{1}{2}}^k - F_{i-\frac{1}{2}}^k\right)\right].$$

Thus, the constraint enforces that the predicted solution approximately evolves according to a locally conservative finite-volume scheme over several successive time steps.

D.3. Discretization

To evaluate the conservation constraints in practice, the continuous space-time domain was discretized on a uniform grid

$$x_i = i\Delta x, \qquad t_k = k\Delta t,$$

where Δx and Δt denote the spatial and temporal step sizes, respectively. Using the representation that $\mathbf{u}_i^k$ is the approximation to $u(x_i, t_k)$, the continuous conservation operators could be approximated through discrete sums and local update relations.

Global conservation quantities were approximated through Riemann sums over the spatial grid. For example, the total mass at time t_k was approximated as

$$m(t_k) \approx \sum_i u_i^k \Delta x.$$

D.3.1. Neumann Boundary Conditions

For problems with homogeneous Neumann boundary conditions, the continuous condition

$$\frac{\partial u}{\partial n} = 0$$

was enforced discretely by requiring the normal derivative at the boundary to vanish. In one spatial dimension, this corresponds to

$$\frac{u_2^k - u_1^k}{\Delta x} = 0, \qquad \frac{u_{N_x}^k - u_{N_x-1}^k}{\Delta x} = 0,$$

which implies that the boundary values mirror their neighboring interior values,

$$u_1^k = u_2^k, \qquad u_{N_x}^k = u_{N_x-1}^k.$$

This prevents artificial fluxes through the boundary and ensures consistency with the conservation constraints.

D.3.2. Discretization of the Conservation Constraint

Consider the continuous conservation constraint

$$\frac{d}{dt}\int_\Omega \rho(u)\,dx - C = 0,$$

where $\rho(u)$ denotes some quantity whose integral in space we'd like to conserve, and C is some constant.

Using the spatial discretization, the integral term was approximated through a Riemann sum,

$$\int_\Omega \rho(u)\,dx \approx \sum_i \rho(u_i^k)\Delta x.$$

The temporal derivative was then approximated using a finite difference,

$$\frac{d}{dt}\int_\Omega \rho(u)\,dx \approx \frac{\sum_i \rho(u_i^{k+1})\Delta x - \sum_i \rho(u_i^k)\Delta x}{\Delta t}.$$

Hence, the fully discretized conservation constraint becomes

$$\frac{\sum_i \rho(u_i^{k+1})\Delta x - \sum_i \rho(u_i^k)\Delta x}{\Delta t} - C.$$

E. Implementation Details

All sampling runs were initialized from white noise – iid Gaussian entries – matched to the grid of the target PDE as a starting point for our models.

For all the methods, the forward pass of the pretrained neural operator is executed on the GPU. This choice is intentional to isolate the cost of the projection step, rather than benchmark CPU versus GPU neural-network inference. A fully CPU-resident implementation would likely incur additional cost from evaluating the neural operator on the CPU, and should therefore be expected to have larger end-to-end runtimes than those reported here.

E.1. Hardware

All experiments were conducted on a high-performance computing cluster with 1 × NVIDIA L40S GPU (46 GB VRAM), 4 CPU cores, and 64 GB RAM.

E.2. Test for Correctness

We verify that all methods produce solutions to the projection subproblem of comparable quality.

Figure 3 shows generated samples under each backend, as compared with an analytic solve for reference. The `ExaModels + MadNLP` variants (both CPU and GPU) reproduce the analytic sample to visual accuracy, demonstrating that the GPU execution does not compromise the correctness of the solution. The remaining backends, `JuMP`, `IPNewton`, and `L-BFGS`, produce samples that satisfy the constraints nominally but are visibly distinct from the reference.

As PCFM is a generative method, even with exact constraint satisfaction, different optimization trajectories can occur. Therefore, the figure highlights differences in reproducibility rather than correctness.

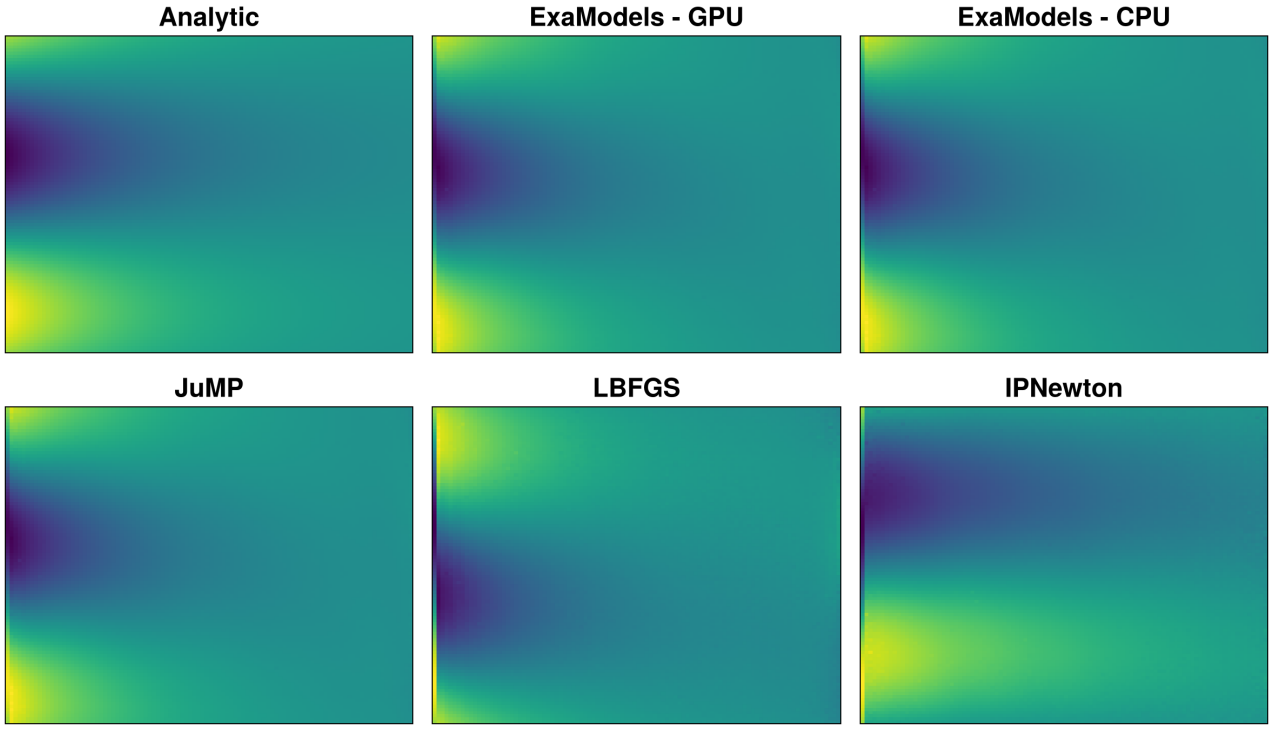

Figure 3. Generated heat-equation samples under each optimization backend, alongside an analytic reference. `ExaModels+MadNLP` (CPU and GPU) reproduces the analytic sample; `JuMP`, `IPNewton`, and `L-BFGS` produce visibly different samples despite using the same input noise, indicating that solver choice affects the generated sample beyond runtime.