Data-Centric Engineering (2026), 7: e48
doi:10.1017/dce.2026.10079

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

GPU-accelerated ECSW hyper-reduction: implementation and performance evaluation

Nicolas Lepage[1,2], Samir Beneddine[2], Michele Giuliano Carlino[3], Camilla Fiorini[4], Iraj Mortazavi[1], Denis Sipp[2] and Nicolas Thome[5]

[1]M2N, CNAM, France
[2]ONERA, France
[3]Equipe-projet MONHADE, INRIA, France
[4]CNAM, France
[5]Sorbonne Universite, France
Corresponding author: Nicolas Lepage; Email: nicolas.lepage3@lecnam.net

Received: 01 December 2025 UTC; **Revised:** 24 June 2026 UTC; **Accepted:** 05 August 2026 UTC

Keywords: ECSW; finite elements; fluid mechanics; GPU computing; hyper-reduction

Abstract

This paper investigates the computational relevance of the Energy-Conserving Sampling and Weighting (ECSW) method for hyper-reduction in reduced-order models (ROMs), with particular attention to modern Graphical Processing Unit (GPU)-accelerated and deep-learning-based frameworks. While hyper-reduction techniques such as ECSW are traditionally introduced to enable tractable evaluation of nonlinear terms in Galerkin ROMs, their actual benefit for computable nonlinearities with traditional Proper Orthogonal Decomposition (POD)–Galerkin ROMs combined with deep learning and on deep-learning GPU hardware remains a question. We provide a full ECSW implementation for finite element Navier–Stokes solvers, compatible with automatic differentiation and batched tensor operations, and derive exact operation counts for both standard tensorial and hyper-reduced formulations. Theoretical analysis shows that the computational complexity scales as $\mathcal{O}(r^3)$ for tensorial ROMs and $\mathcal{O}(rm)$ for ECSW, where r is the reduced dimension and m the number of retained quadrature points. Extensive CPU and GPU benchmarks are conducted on the fluidic pinball configuration to assess asymptotic scaling, memory effects, and hardware bottlenecks. Results confirm the linear complexity and efficiency of ECSW on CPUs but reveal that, on GPUs, the regular tensorial formulation can outperform hyper-reduction for common reduction dimensions due to highly optimized batched matrix kernels and limited bandwidth constraints. Overall, the study clarifies when hyper-reduction remains advantageous in practice and provides concrete guidelines for integrating ECSW within deep-learning-augmented ROM frameworks.

Impact statement

Fast and accurate reduced-order models (ROMs) are essential in many areas of science and engineering. To fully benefit from the recent GPU computing revolution and from deep-learning-based error correction, we need ROM techniques that remain efficient on modern hardware. This work provides the first detailed GPU implementation and systematic benchmarking of the ECSW hyper-reduction method, compared against standard tensor-based

This research article was awarded Open Data and Open Materials badges for transparent practices. See the Data Availability Statement for details.

© The Author(s), 2026. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (http://creativecommons.org/licenses/by/4.0), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

ROMs. Focusing on quadratic convective nonlinearities, we quantify how performance depends on the number of reduced modes and sampled points, and identify when hyper-reduction is faster than the quadratic ROM. Our results give clear, practical guidelines on when the extra approximation of ECSW is worthwhile, helping researchers design ROM and hybrid ROM–deep learning pipelines that are accurate and computationally efficient.

1. Introduction

Model order reduction has become an important class of methods to reduce the computational cost of high-fidelity numerical simulations. Among classical reduction techniques, the Proper Orthogonal Decomposition (POD) combined with Galerkin projection remains the cornerstone of projection-based reduced-order modeling (Lumley, 1979; Sirovich, 1987; Holmes et al., 1996; Noack et al., 2003). POD extracts an optimal set of orthonormal modes that capture the dominant energetic structures of the flow, while Galerkin projection yields a low-dimensional dynamical system governing the temporal evolution of their amplitudes. This formulation provides a mathematically grounded and physically interpretable surrogate directly derived from the full-order equations. POD–Galerkin ROMs have been successfully applied to canonical fluid flows such as the cylinder wake or cavity oscillations, offering drastic cost reductions while preserving key physical mechanisms.

By projecting the governing equations onto a low-dimensional subspace spanned by data-driven modes, reduced-order models (ROMs) can accelerate transient fluid flow computations by several orders of magnitude. However, this gain in efficiency is often achieved at the expense of accuracy and robustness, especially for nonlinear, advection-dominated systems. Moreover, even in their simplest formulations, classical ROMs can become computationally expensive when nonlinear terms must be evaluated in the full space of the original degrees of freedom.

For quadratic nonlinearities, which appear in many canonical fluid dynamics equations such as Burgers, Kuramoto–Sivashinsky, Korteweg–de Vries, or the incompressible Navier–Stokes equations, it is possible to derive a closed reduced model entirely in the latent space through tensorial projection of the quadratic term. While this removes the need to return to the full-order space, two limitations remain. First, evaluating the quadratic term in reduced coordinates scales cubically with the number of retained modes, resulting in rapidly increasing online cost and memory requirements. Second, standard Galerkin ROMs remain poorly suited to strongly convective dynamics, often requiring intrusive stabilization or closure modeling to achieve predictive accuracy.

To address these issues, two main research directions have emerged almost independently. A first class of approaches seeks to alleviate the cubic cost of nonlinear evaluation by approximating the nonlinear operator without resorting to full-space computations. These so-called *hyper-reduction* methods approximate the nonlinear residual using a restricted set of sampling points or degrees of freedom, and reconstruct the reduced projection through interpolation or weighted quadrature. The most widely used among these are the Discrete Empirical Interpolation Method (DEIM) (Barrault et al., 2004; Chaturantabut and Sorensen, 2010) and Gappy POD (Everson and Sirovich, 1995), which follows a similar philosophy and is often used for sensor placement and incomplete-data problems. More recent methods, such as the Energy-Conserving Sampling and Weighting (ECSW) technique (Chapman et al., 2017), approximate the nonlinear term through a weighted sampling of quadrature points, with optimized weights ensuring that the reduced operator reproduces the full-order projection. All share the same objective: replacing the global nonlinear evaluation with a reduced, efficiently computable surrogate that operates entirely in a reduced subspace, thereby circumventing the need for costly evaluations in the full-order space. However, these methods do not aim to improve the intrinsic accuracy of ROMs for a given level of reduction; they merely introduce additional approximations to make the nonlinear evaluation tractable without addressing the underlying limitations of the reduced representation itself.

Among these, ECSW has proven particularly well suited for finite element formulations. It preserves the symmetry and positive definiteness of reduced operators and guarantees numerical stability during time integration, features crucial for conservative systems. Furthermore, ECSW is structure-preserving,

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

compatible with conservation laws and Hamiltonian formulations, and has demonstrated greater robustness and accuracy than DEIM for highly nonlinear or stiff dynamics. Recent implementations (Carlino, 2021; Riffaud et al., 2021) have also shown that ECSW naturally extends to nonlinear manifolds and mappings, improving its generality for complex geometries and multiphysics problems.

A second body of work has attempted to improve the accuracy of ROMs using data-based corrections and closure. In particular, with the recent rise of deep learning in physics applications, neural networks have been used as data-driven error correctors, enhancing ROM fidelity while retaining some level of physical interpretability. A first closure modeling approach, inspired by the Mori–Zwanzig formalism and Takens' embedding theorem (Zwanzig, 1961; Mori, 1965; Takens, 1981), introduces time-dependency and memory effects to account for the influence of truncated modes. Representative examples include LSTM-ROMs (Wang et al., 2020) and CDROM (Menier et al., 2023), which employ recurrent or convolutional architectures to reconstruct the temporal correlations neglected by standard Galerkin projections. Other studies, such as our recent hybrid autoencoder–Galerkin framework (Lepage et al., 2025), exploit deep learning both to enhance the latent-space representation and to correct integration errors of the reduced model. Finally, alternative formulations, such as the PROM-ANN approach of Barnett et al. (2023), use neural networks to reconstruct the dynamics of subdominant POD coefficients from the leading ones, and even couple this strategy with ECSW hyper-reduction for efficient physics-based time stepping.

Overall, these efforts demonstrate a growing convergence between deep learning and model reduction, with neural networks being used not only for closure modeling but also for representation learning and computational acceleration. This emerging community provides the broader context for our present work, which focuses on assessing the effective cost of hyper-reduction within such deep-learning-augmented ROM frameworks.

Despite the growing popularity of both paradigms, their interaction remains largely unexplored. In particular, it is unclear whether hyper-reduction techniques such as ECSW still offer a net computational benefit when executed on GPU hardware. Deep learning correction increases accuracy for a given latent dimension but also shifts the computational burden toward neural network inference, often already optimized for massively parallel architectures. Consequently, the relative cost of the ROM evaluation may change drastically compared to traditional CPU-based implementations, altering the efficiency of the additional hyper-reduction approximation in this new context.

In this work, we aim to answer a fundamental question:

When is ECSW hyper-reduction still computationally beneficial in a deep-learning-based hybrid ROM, and what are the related hardware-specific bottlenecks?

To this end, we implement ECSW within a differentiable GPU-accelerated framework and compare its theoretical and practical cost against the standard tensorial ROM formulation. The implementation follows the formulation and practical details proposed by Carlino (2021) and Riffaud et al. (2021), and is fully integrated into a deep learning framework supporting GPU acceleration. Our contributions are threefold:

1. We provide a detailed implementation of ECSW adapted to GPU-based deep learning frameworks, ensuring compatibility with automatic differentiation and batched tensor operations (details in Section 2).
2. We derive analytical expressions for the asymptotic and exact operation counts of both standard and hyper-reduced ROMs, highlighting their respective memory and computational scaling (details in Section 4.1).
3. We perform an extensive benchmarking campaign on both CPU and GPU hardware, analyzing wall-clock performance, scaling behavior, and numerical bottlenecks, and provide guidelines to help practitioners assess whether hyper-reduction is advantageous for their specific hybrid architectures (details in Section 4.4).

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

2. Hyper-reduction and Galerkin models

Having established the motivation for replacing the standard ROM evaluation with ECSW, we now describe its implementation in our workflow. To address the scarcity of documentation regarding ECSW in open-source finite element settings, particularly concerning the Navier–Stokes equations and GPU-based inference, we detail the extraction of empirical operators and the associated modeling choices.

Although we operationalize this workflow using FreeFem++ and PyTorch, the underlying principles remain agnostic to the specific FEM solver or GPU framework. However, because the performance of hyper-reduction depends on low-level implementation decisions, a transparent description of this architecture is required to contextualize the cost analysis. In this section, we therefore highlight the components of the ECSW workflow that are most relevant to the present study, while a detailed description of the method construction and inner workings is provided in Appendix A.

2.1. Hyper-reduction and Galerkin projection at a glance

Let us consider the simple PDE problem:

$$\partial_t u(x,t) = f(u(x,t)), \tag{2.1}$$

where $u(x,t)$ is a scalar dependent on space and time, while $f : u \to f(u)$ is a nonlinear function, possibly involving space derivatives of solution u. The general procedure is the following:

1. **Finite element discretization**. Introducing a basis of local functions $\Theta(x) = (\theta_1(x), \cdots, \theta_N(x)) \in \mathbb{R}^{1\times N}$ where N is the total number of degrees of freedom. Introducing the unknown $u(x,t) = \Theta(x)\mathbf{u}(t)$, where $\mathbf{u}(t) \in \mathbb{R}^{N\times 1}$, the finite element methodology allows one to write the weak-form:

$$\mathbf{M}\partial_t \mathbf{u} = \mathbf{F}(\mathbf{u}), \tag{2.2}$$

where

$$\mathbf{F}(\mathbf{u}) = \sum_{\ell} \Theta(x_\ell)^T f(\Theta(x_\ell)\mathbf{u})\delta_\ell, \quad \mathbf{M} = \sum_{\ell} \Theta(x_\ell)^T \Theta(x_\ell)\delta_\ell \tag{2.3}$$

with (x_ℓ, δ_ℓ) being $\tilde{m} \sim N$ quadrature points in the mesh.

2. **Generation of snapshots**. The finite element code allows to generate P snapshots stacked in $\mathbf{U} = (\mathbf{u}_1, \cdots, \mathbf{u}_P) \in \mathbb{R}^{N\times P}$. These snapshots are rewritten as $\mathbf{u}_p = \overline{\mathbf{u}} + \mathbf{u}'_p$, where $\overline{\mathbf{u}}$ is an arbitrary flow-field (lifting function) containing the non-homogeneous boundary conditions (for example, the time-average of the snapshots $\mathbf{u}_p$ or a fixed point of $f(\cdot) = 0$). The homogeneous snapshots are then stacked in $\mathbf{U}' = \left(\mathbf{u}'_1 \cdots \mathbf{u}'_P\right) \in \mathbb{R}^{N\times P}$.
3. **Generation of POD basis**. In a classical way, we generate POD modes by solving:

$$\left(\mathbf{U}'^T \mathbf{M} \mathbf{U}'\right)\mathbf{T} = \mathbf{T}\Lambda, \tag{2.4}$$

where $\mathbf{T}$ are the chronos modes and Λ is a diagonal matrix containing positive eigenvalues ranked in decreasing order. We can then select the leading $r \le P$ eigenvectors/eigenvalues and compute r POD modes $\Phi \in \mathbb{R}^{N\times r}$, with $r \ll N$:

$$\Phi = \mathbf{U}'\mathbf{T}\Lambda^{-1/2}, \ \Phi^T \mathbf{M} \Phi = \mathbf{I}. \tag{2.5}$$

4. **Projection of the finite element solver on the POD basis**:

$$\partial_t \mathbf{a} = \sum_{\ell} \Phi(x_\ell)^T f(\overline{u}(x_\ell) + \Phi(x_\ell)\mathbf{a})\delta_\ell, \tag{2.6}$$

where $\mathbf{u}(t) = \Phi\mathbf{a}(t)$ in which $\mathbf{a}(t) \in \mathbb{R}^{r\times 1}$ is the dynamics within the POD basis, $\Phi(x) = \Theta(x)\Phi = (\phi_1(x), \cdots, \phi_r) \in \mathbb{R}^{1\times r}$ and $\overline{u}(x) = \Theta(x)\overline{\mathbf{u}}$. This model still scales on the $\tilde{m} \approx N$ and is therefore not yet a reduced-order model.

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

Then, we have two cases, depending on the reduction technique: either hyper-reduction (Section 2.1.1), which is valid in the general case, or Galerkin projection (Section 2.1.2), which is valid only for polynomial nonlinearities.

2.1.1. Hyper-reduction

The algorithm for hyper-reduction follows in two steps:

5. **NNLS (offline)**. The Non-Negative Least Squares (NNLS) algorithm allows us to approximate the sum within (Equation 2.6) with a subset of $m \ll \tilde{m}$ points listed in $\mathcal{S}$ weighted by new quadrature coefficients δ'_ℓ:

$$\sum_\ell \Phi(x_\ell)^T f(\overline{u}_\ell + u_\ell)\delta_\ell \approx \sum_{\ell \in \mathcal{S}} \Phi(x_\ell)^T f(\overline{u}_\ell + u_\ell)\delta'_\ell. \tag{2.7}$$

   This approximation should be valid for all snapshots $\mathbf{u}$ within the learning basis $\mathbf{u} \in \mathbf{U}'$ and the notations $\overline{u}_\ell$, u_ℓ refer to the ℓ^{th} element within $\overline{\mathbf{u}}$ and $\mathbf{u}$.
6. **Hyper-Reduced-Order model (online)**. The model then reads

$$\partial_t \mathbf{a} = \sum_{\ell \in \mathcal{S}} \Phi(x_\ell)^T f(\overline{u}(x_\ell) + \Phi(x_\ell)\mathbf{a})\delta'_\ell, \tag{2.8}$$

   where we have the scaling $r \times m$ compared to the initial $N \times \tilde{m}$.

2.1.2. Galerkin projection

In the case where function f is polynomial, for example, considering here quadratic symmetric non-linearities, $f(u) = f(u,u)$ with $f(u,v) = f(v,u)$, steps 5 and 6 are replaced by:

5. **Tensorization (offline)**. The k^{th} component of the r.h.s. of Equation (2.6) reads

$$\begin{aligned}
&\sum_\ell \phi_k(x_\ell) f\left(\overline{u}(x_\ell) + \sum_i \phi_i(x_\ell)a_i, \overline{u}(x_\ell) + \sum_j \phi_j(x_\ell)a_j\right)\delta_\ell \\
&= \underbrace{\sum_\ell \phi_k(x_\ell) f(\overline{u}(x_\ell), \overline{u}(x_\ell))\delta_\ell}_{b_k} + \sum_i a_i \underbrace{\sum_\ell 2\phi_k(x_\ell) f(\overline{u}(x_\ell), \phi_i(x_\ell))\delta_\ell}_{L_{ki}} \\
&\quad + \sum_{i,j} a_i a_j \underbrace{\sum_\ell \phi_k(x_\ell) f\left(\phi_i(x_\ell), \phi_j(x_\ell)\right)\delta_\ell}_{N_{kij}}.
\end{aligned} \tag{2.9}$$

   All coefficients b_k, L_{ki} and N_{kij} can be computed offline.
6. **Galerkin model (online)**. The model then reads:

$$\partial_t a_k = b_k + \sum_i L_{ki} a_i + \sum_{i,j} N_{kij} a_i a_j, \tag{2.10}$$

   where the scaling is in $(r)^3$.

2.2. NNLS algorithm and considerations

The ECSW hyper-reduction method relies on solving large-scale NNLS problems to identify sparse quadrature weights. As established in Section 2.1, each hyper-reduction problem has the form:

$$\min_{\boldsymbol{\omega} \geq 0} \|\mathcal{A}\boldsymbol{\omega} - \mathbf{b}\|_2^2,$$

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

where $\mathcal{A} \in \mathbb{R}^{rN_{\text{snap}} \times \tilde{m}}$ contains nonlinear evaluations at Gauss points, and $\boldsymbol{\omega} \in \mathbb{R}^{\tilde{m}}$ is the empirical quadrature weight vector. In our application:

- The reduced dimension is typically $r \sim \mathcal{O}(10)$;
- The number of snapshots is $N_{\text{snap}} \sim \mathcal{O}(10^3)$;
- The number of Gauss points is $\tilde{m} \sim \mathcal{O}(10^5)$.

This leads to NNLS problems of size $\sim 10^4 \times 10^5$, which are challenging to even store in memory—let alone solve using dense methods. Indeed, the classical Lawson and Hanson (1995) active set method for NNLS has worst-case complexity:

$$\mathcal{O}\left(rN_{\text{snap}}\tilde{m}^2 + \tilde{m}^3\right),$$

which, when inserting our scaling, gives:

$$\mathcal{O}\left(\left(10 \cdot 10^3\right) \cdot \left(10^5\right)^2 + \left(10^5\right)^3\right) = \mathcal{O}\left(10^{13} + 10^{15}\right).$$

Such complexity renders direct solvers impractical for our setting.

2.2.1. Greedy algorithm for efficient sparsification

To alleviate this computational burden, we use a greedy subset selection strategy (Yaghoobi et al., 2015) classically used in orthogonal matching pursuit on top of the classical Lawson and Hanson method. The algorithm iteratively constructs a sparse active set $S \subset \{1, \ldots, \tilde{m}\}$ of retained quadrature points by maximizing correlation with the residual. At each iteration, a small NNLS problem of size $\#S \ll \tilde{m}$ is solved over the active set using the Lawson and Hanson method from SciPy (Virtanen et al., 2020).

Algorithm 1: Greedy Non-Negative Selection for ECSW.

Input: Matrix $\mathbf{A} \in \mathbb{R}^{rN_{\text{snap}} \times \tilde{m}}$, vector $\mathbf{b} \in \mathbb{R}^{rN_{\text{snap}}}$, sparsity budget k, precision ϵ.
Output: Weights $\mathbf{x}$
$\mathbf{r} \leftarrow \mathbf{b}$; $S \leftarrow \varnothing$; $\mathbf{x} \leftarrow \mathbf{0}$;
for $t \leftarrow 1$ **to** k $\|\mathbf{r}\| \leq \epsilon$ **do.**
  Compute correlations $c_j \leftarrow \mathbf{A}_{:j}^{\top}\mathbf{r}$ for all $j \notin S$;
  $j^{\star} \leftarrow \arg\max_{j \notin S} c_j$;
  $S \leftarrow S \cup \{j^{\star}\}$;
  Solve NNLS: $\mathbf{x}_S \leftarrow \arg\min_{\mathbf{z} \geq 0} \|\mathbf{A}_S\mathbf{z} - \mathbf{b}\|_2$;
  $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}_S\mathbf{x}_S$;
Set $\mathbf{x}_j \leftarrow (\mathbf{x}_S)_j$ for $j \in S$ (zero elsewhere);
return $\mathbf{x}$;

This algorithm reduces the cost of weight selection by restricting the search to a small subset of highly informative quadrature points, often representing less than 1% of the original set. However, as a greedy method, it does not guarantee the optimal selection of Gauss-point locations for a given budget. Results presented in Section 3 show that our more economical Gauss-point selection performs effectively, although a comparison with the full-space Lawson and Hanson approach is not included in this study.

2.3. Online evaluation after hyper-reduction

Let us now summarize the structure and complexity of the reduced-order model during the online phase after hyper-reduction.

Recall the full-order Navier–Stokes system in semi-discrete perturbed form $\mathbf{w}_t = \overline{\mathbf{w}} + \mathbf{w}$ with

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

$$\mathbf{M}\partial_t \mathbf{w} = \mathcal{L}\mathbf{w} + \mathcal{B}(\mathbf{w}, \mathbf{w}), \tag{2.11}$$

where $\mathbf{M}$ is the mass matrix, $\mathcal{L}$ the discrete linear operator, and $\mathcal{B}(\cdot,\cdot)$ the quadratic convection operator.

The reduced-order model obtained via projection onto the reduced basis Φ reads:

$$\partial_t \mathbf{a} = \Phi^\top \mathcal{L}\Phi\mathbf{a} + \Phi^\top \mathcal{B}(\Phi\mathbf{a}, \Phi\mathbf{a}), \tag{2.12}$$

with $\mathbf{a} \in \mathbb{R}^r$ the reduced coordinates. When the nonlinear term is treated explicitly, it becomes a time-dependent source term:

$$\mathbf{f}_{\text{red}}(t) := \Phi^\top \mathcal{B}\left(\Phi\mathbf{a}^{t-\Delta t}, \Phi\mathbf{a}^{t-\Delta t}\right).$$

Using the matrix formulation derived earlier and applying ECSW, this nonlinear term is computed via:

$$\begin{aligned}\mathbf{f}_{\text{red}} = {} & \Phi^\top \Theta_u^\top W\left(\Theta_u\Phi\mathbf{a} \odot \partial_x\Theta_u\Phi\mathbf{a} + \Theta_v\Phi\mathbf{a} \odot \partial_y\Theta_u\Phi\mathbf{a}\right) \\ & + \Phi^\top \Theta_v^\top W\left(\Theta_u\Phi\mathbf{a} \odot \partial_x\Theta_v\Phi\mathbf{a} + \Theta_v\Phi\mathbf{a} \odot \partial_y\Theta_v\Phi\mathbf{a}\right).\end{aligned} \tag{2.13}$$

2.3.1. Implementation and cost

After hyper-reduction, the integration involves only a sparse subset of quadrature points of size $m \ll \tilde{m}$, where $\tilde{m}$ denotes the total number of Gauss points in the original problem. The online evaluation of the nonlinear convection term is then reduced to a sequence of matrix–vector operations with matrices of size $m \times r$ or $r \times m$, together with a few element-wise operations over m-dimensional vectors. The complete numerical cost of the method is therefore $\mathcal{C}_{\text{nonlin}} = \mathcal{O}(rm + m)$, which scales linearly with both r and m and remains drastically cheaper than the full-order computation or the tensorial ROM formulation. A detailed analysis of implementation cost, including operation counts and hardware runtimes on both CPU and GPU, is provided in Section 4.

2.3.2. Time integration

Assuming a backward Euler scheme and denoting by $\mathbf{A} = \Phi^\top \mathcal{L}\Phi$ the reduced linear operator, the time integration step becomes:

$$\mathbf{a}^{t+\Delta t} = (I - \Delta t\mathbf{A})^{-1}(\mathbf{a}^t + \Delta t\mathbf{f}_{\text{red}}(t)).$$

The cost of solving this reduced system scales as $\mathcal{O}(r^3)$ for a direct solve, but can be reduced to $\mathcal{O}(r^2)$ using an lower-upper (LU) decomposition pre-computed once and reused at each time step.

2.3.3. Conclusion

After hyper-reduction, each time step of the reduced model requires only:

- A single evaluation of the nonlinear term with cost $\mathcal{O}(rm)$,
- One linear solve of size $r \times r$.

This ensures that the online complexity is independent of the full-order degrees of freedom (DOF) size n and scales only with the reduced basis size r and the retained quadrature size m, enabling orders-of-magnitude speed-up over the original model.

3. Numerical experiments for hyper-reduction alone

3.1. Test case selection and setup

To validate the proposed hyper-reduction methodology, we consider the *fluidic pinball* configuration. This geometry consists of three identical cylinders placed at the vertices of an equilateral triangle, generating a rich variety of wake dynamics as the Reynolds number increases. It provides a compact yet versatile benchmark that captures essential features of vortex shedding, symmetry breaking, and

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

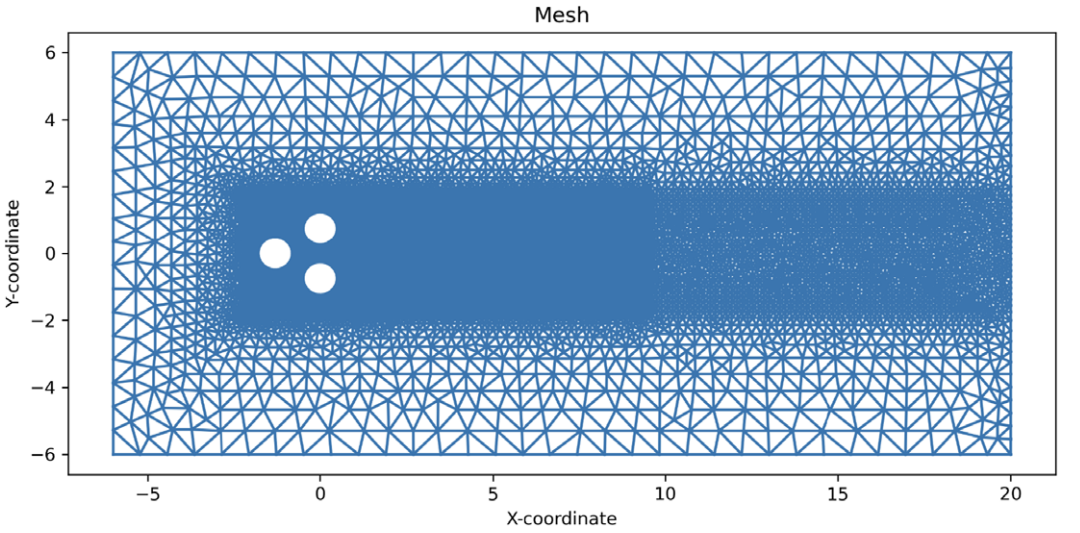


Figure 1. *Mesh for the fluidic pinball test case, featuring three cylinders in an equilateral triangle configuration.*

nonlinear coupling between wake modes, while remaining computationally manageable for systematic analysis.

The validation test case corresponds to the periodic regime of the fluidic pinball, obtained at $Re = 30$. Dynamically, this configuration behaves similarly to the classical cylinder flow in the Von Kármán vortex shedding regime. A single trajectory is simulated from the steady base flow toward the periodic limit cycle, and 1000 snapshots are extracted along this trajectory. These snapshots are used both to construct the reduced POD basis and to assemble the ECSW NNLS hyper-reduction problem. The resulting hyper-reduced model is then evaluated on the same trajectory to verify correct implementation and basic dynamical consistency.

This periodic configuration serves as a controlled validation case. It is simple enough that any properly implemented reduced-order model should reproduce the correct limit cycle dynamics, yet dynamic enough to test stability and temporal accuracy. The regime's transient growth rate, shedding frequency, and limit cycle amplitude are well documented in the literature and have been used extensively in earlier work to benchmark ROM performance, enabling direct consistency checks with earlier results.

Only the periodic regime is considered here, as it provides a transparent and interpretable sanity check confirming that the ECSW implementation reproduces expected reduced-order behavior. Since the goal of this work is to analyze computational cost rather than model accuracy, extending the study to more complex or chaotic regimes would not provide additional insight into numerical performance. The following sections, therefore, focus exclusively on quantifying the computational efficiency and scaling properties of ECSW hyper-reduction (Figure 1).

3.2. Validating the implementation on a periodic test case

We evaluate the hyper-reduction method on the periodic regime of the fluidic pinball. This configuration provides a controlled yet representative test case: the dynamics evolve from a steady base flow toward a stable limit cycle, allowing us to assess both transient behavior and long-term accuracy.

The reduced-order model is constructed using a quadratic Galerkin formulation with a basis size chosen as the minimal number of modes required to capture both the transient evolution and the limit cycle dynamics accurately. In this regime, the reference ROM provides a qualitatively correct representation of the full-order dynamics, both in terms of growth toward the attractor and its final periodic behavior.

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

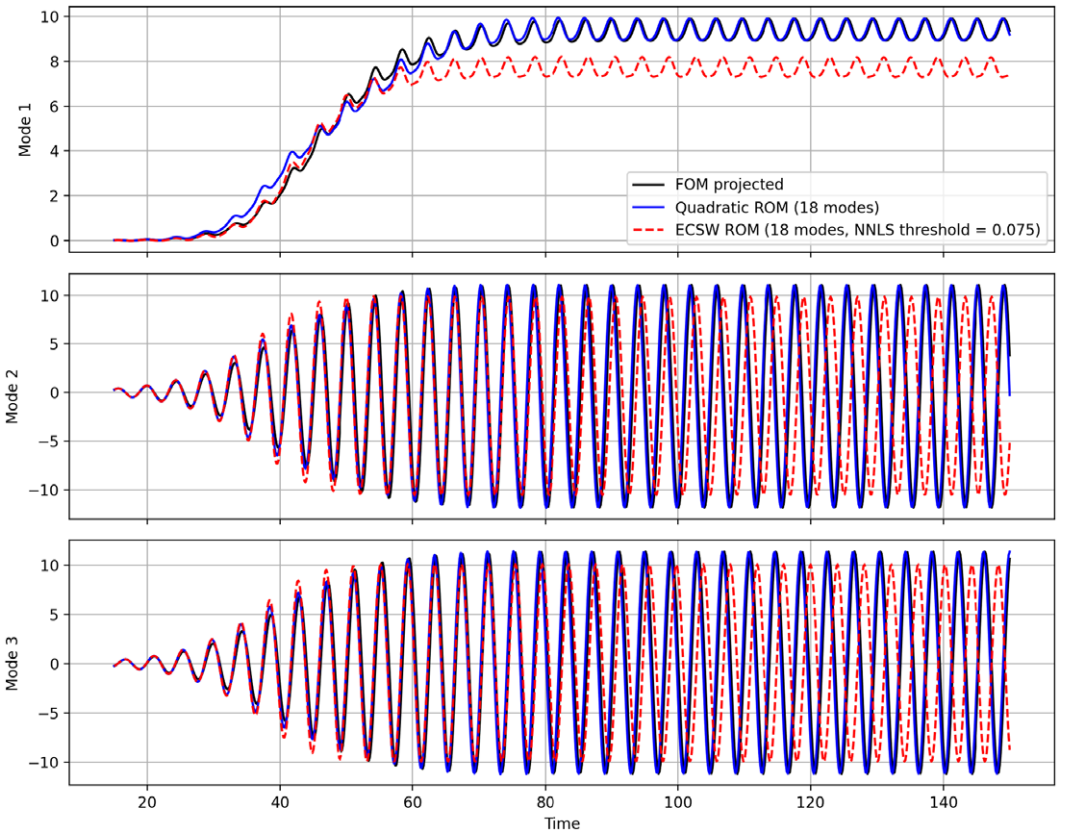


Figure 2. *First three POD coordinates over time for the periodic test case. Comparison between full-order solution (black), quadratic Galerkin ROM (blue), and hyper-reduced model with NNLS tolerance* $\varepsilon = 7.5 \cdot 10^{-2}$ *(red).*

We then investigate whether the ECSW hyper-reduction is able to recover a similar level of accuracy under different sparsity regimes. Two configurations are considered, corresponding to different tolerances in the NNLS greedy selection of quadrature points.

In the first case, a relative residual threshold of $\varepsilon = 7.5 \cdot 10^{-2}$ is used. This results in retaining 50 Gauss points for the u-component and 54 for the v-component in the ECSW method. As shown in Figure 2, the hyper-reduced model at this size does not match the performance of the quadratic ROM or the FOM. It appears to overdamp the system, resulting in a relatively accurate transient but an incorrect limit cycle.

In the second configuration, the relative tolerance is tightened to $\varepsilon = 5 \cdot 10^{-2}$, leading to 65 Gauss points for both the u and v components. Notably, the two selected sets share only a single common index out of 130 unique points, confirming that the nonlinear terms for each velocity component are supported in largely distinct spatial regions and justifying the use of separate ECSW decompositions. In this case (Figure 3), the hyper-reduced model recovers dynamics that are qualitatively indistinguishable from the quadratic ROM, both during the transient growth and on the limit cycle. This indicates that a moderate increase in quadrature size is sufficient to fully restore the reduced model's accuracy.

These results validate ECSW on this classical benchmark and identify the minimal quadrature size required for an accurate representation of this test case with the chosen POD quadratic ROM.

From a computational perspective, we compare the wall-clock time required for full trajectory integration on CPU. The quadratic ROM requires a total simulation time of 1.60 seconds. The hyper-reduced models significantly reduce this cost, with total simulation times of 0.76 seconds and 0.81

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

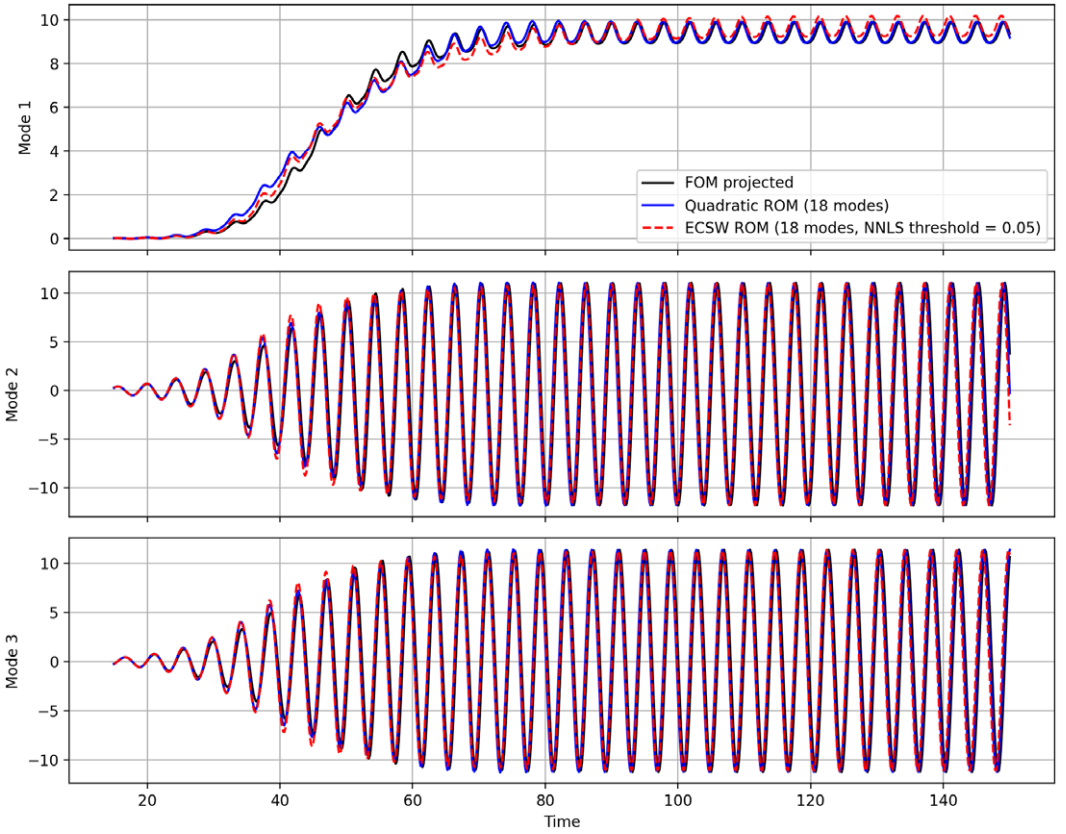


Figure 3. *Same setup as Figure 2, but with NNLS tolerance* $\varepsilon = 5 \cdot 10^{-2}$. *The hyper-reduced model and quadratic ROM are nearly indistinguishable.*

seconds for the two configurations, respectively. More precisely, when isolating the right-hand side evaluation, the computational cost drops from $5.2 \cdot 10^{-4}$ s to $1.45 \cdot 10^{-4}$ s and $1.95 \cdot 10^{-4}$ s. This corresponds to an overall speed-up of approximately $2\times$, and up to $2.6\times$ when considering only the nonlinear term, while maintaining comparable accuracy in the refined regime.

3.3. Assessment of numerical cost for the considered architecture

One of the main advantages of hyper-reduction is its lower numerical complexity compared to the tensorial quadratic ROM method. In earlier standalone tests on CPU, we showed clear gains, with the hyper-reduced model running about twice as fast in wall-clock time for the chosen number of retained modes and Gauss points. In the present experiments, however, the context is different: the models now run directly on the GPU and are optimized for this hardware. Under these conditions, the results are surprising. Despite using the same reduction sizes as before, the hyper-reduced model no longer shows the expected advantage. To understand this, we measured the runtime of the quadratic ROM alone and the hyper-reduced ROM alone, isolating their computation while keeping the same optimization strategy for the algorithm and hardware. For the full trajectory (18,001 time steps), the quadratic ROM integrates in about 6.1 s, with each nonlinear evaluation costing 153 μs on average. By contrast, the hyper-reduced model requires 6.9 s, with nonlinear evaluations taking 208 μs each on average. These results run counter to both previous empirical evidence and the theoretical scaling of the method, which should favor hyper-reduction. Section 4 investigates this discrepancy by analyzing cost scaling in more detail and examining how GPU hardware influences the benefits of hyper-reduction.

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

Although the isolated nonlinear evaluation is about 36% more time-consuming for the hyper-reduced model (208 vs 153 μs), the full trajectory cost increases by only about 13% (6.9 vs 6.1 s). This is because the nonlinear right-hand side is only one component of each time step: the remainder is spent on the implicit $r \times r$ linear solve and on time-stepping bookkeeping, both of which are identical for the two models. Indeed, over the 18 001 steps the right-hand-side evaluation accounts for 2.75 s for the quadratic ROM and 3.74 s for the hyper-reduced model, leaving 3.35 s and 3.16 s respectively for the shared remaining workload—consistent up to timing noise, as expected for model-independent operations. The right-hand-side penalty is therefore amortized over these shared costs.

4. Numerical cost assessment considering deep learning hardware

4.1. Theoretical operation count scaling for nonlinearity computation

Before examining wall-clock timings for the tensorized quadratic ROM and the hyper-reduced ROM, we first estimate the exact number of operations required for both approaches. Earlier, we described their asymptotic complexity: $\mathcal{O}(r^3)$ for the tensorial ROM and $\mathcal{O}(mr)$ for the hyper-reduction, where r is the size of the retained POD subspace and m the number of retained Gauss points in ECSW. Here, we go beyond asymptotics and compute the exact non-asymptotic operation counts, which provide a hardware-agnostic baseline for comparing methods and identifying where the trade-off between them shifts.

4.1.1. Numerical cost of the quadratic ROM nonlinearity evaluation

The quadratic ROM computes the quadratic nonlinearity entirely in the reduced space. As expressed in Section 2.1.2, the explicit nonlinear term for one unbatched evaluation is

$$f_k = \sum_{i=1}^{r} \sum_{j=1}^{r} N_{ijk} a_i a_j, \quad N \in \mathbb{R}^{r \times r \times r}, \quad a \in \mathbb{R}^r.$$

In practice, this is not implemented with triple loops but with the `torch.einsum` operator, which supports batching and calls optimized basic linear algebra subprograms (BLAS) kernels. The batched version can be written as

$$f_{lk} = \sum_{i=1}^{r} \sum_{j=1}^{r} N_{ijk} a_{li} a_{lj}, \quad a \in \mathbb{R}^{b \times r}, \quad \mathbf{f} \in \mathbb{R}^{b \times r}.$$

with b the batch size.

The contraction is typically performed in two steps. First, compute the intermediate tensor

$$T_{ljk} = \sum_{i=1}^{r} N_{ijk} a_{li}.$$

This requires br^3 multiplications and $br^3 - br^2$ additions.

Then compute

$$f_{lk} = \sum_{j=1}^{r} T_{ljk} a_{lj},$$

which requires br^2 multiplications and $br^2 - br$ additions.

In total, the quadratic ROM costs

$$\left(br^3 + br^2\right) \text{multiplications}, \quad \left(br^3 - br\right) \text{additions},$$

for an overall count of $2br^3 + br^2 - br$ operations (not accounting for MAC optimization).

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

4.1.2. Numerical cost of hyper-reduction nonlinearity evaluation

For hyper-reduction, the nonlinear evaluation is written as in Section 2.3:

$$\begin{aligned}\mathbf{f} = M_1[(M_3\mathbf{a})\odot(M_4\mathbf{a}) + (M_5\mathbf{a})\odot(M_6\mathbf{a})] \\ + M_2[(M_7\mathbf{a})\odot(M_8\mathbf{a}) + (M_9\mathbf{a})\odot(M_{10}\mathbf{a})]\end{aligned} \tag{4.1}$$

Where the shorthand matrices are defined from the pre-computed products of the POD basis and FEM sampling operators (see Section 2.3 and Appendix A): $M_1 = \Phi^\top\Theta_u^\top W \in \mathbb{R}^{r\times m}$, $M_2 = \Phi^\top\Theta_v^\top W \in \mathbb{R}^{r\times m}$, and the eight projection matrices $M_3,\dots,M_{10} \in \mathbb{R}^{m\times r}$ correspond to $\Theta_u\Phi$, $\partial_x\Theta_u\Phi$, $\Theta_v\Phi$, $\partial_y\Theta_u\Phi$, $\Theta_u\Phi$, $\partial_x\Theta_v\Phi$, $\Theta_v\Phi$, $\partial_y\Theta_v\Phi$, respectively. In batched form:

$$\mathbf{f} = \left[\left(\mathbf{a}M_3^\top\right)\odot\left(\mathbf{a}M_4^\top\right) + \left(\mathbf{a}M_5^\top\right)\odot\left(\mathbf{a}M_6^\top\right)\right]M_1^\top + \left[\left(\mathbf{a}M_7^\top\right)\odot\left(\mathbf{a}M_8^\top\right) + \left(\mathbf{a}M_9^\top\right)\odot\left(\mathbf{a}M_{10}^\top\right)\right]M_2^\top. \tag{4.2}$$

Each of the six matrix–vector products costs brm multiplications and $b(r-1)m$ additions. The element-wise products add bm multiplications, and the element-wise sums add bm additions. Finally, the two projections back to reduced space each cost brm multiplications and $br(m-1)$ additions.

In total, the hyper-reduction costs

$$(8brm + 4bm)\,\text{multiplications}, \qquad (8brm - 4bm - br)\,\text{additions},$$

that is $16brm - br$ operations overall.

4.1.3. Trade-off between methods

From these counts, hyper-reduction is cheaper whenever

$$m \le \frac{r(2r+1)}{16}.$$

For example, with $r = 20$, hyper-reduction is cheaper if $m < 50$; for $r = 40$, if $m < 200$; and for $r = 100$, if $m < 1256$.

These thresholds are broadly consistent with earlier empirical results, although small discrepancies remain. For instance, the 20-mode ROM was observed to be more costly than the 20-mode hyper-reduction with 200 Gauss points, despite the theoretical bound. Such deviations can be explained by hardware effects, parallelization strategies, and optimization details. For a more robust view of performance, Subsection 4.2 reports averaged wall-clock results for isolated nonlinear computations (Table 1).

4.2. Empirical cost scaling and asymptotic behavior on CPU

After establishing the theoretical operation counts, we now evaluate empirical runtimes. The goal is to test how the wall-clock cost of nonlinear evaluations scales with the number of POD modes, and to confirm whether the observed scaling agrees with theoretical predictions: cubic for the quadratic Galerkin ROM and linear for ECSW hyper-reduction.

We run the nonlinear evaluation functions introduced in Section 3 on a single Intel Xeon(R) Gold 5118 core. To obtain a robust estimate of runtime for each configuration, we perform 7×100 function calls.

Table 1. *Exact operation counts for quadratic nonlinearity evaluation using quadratic ROM and ECSW hyper-reduction*

	Quadratic ROM	Hyper-reduction
Multiplications	$br^3 + br^2$	$8brm + 4bm$
Additions	$br^3 - br$	$8brm - 4bm - br$
Total ops	$2br^3 + br^2 - br$	$16brm - br$

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

Each block of 100 runs is averaged to reduce measurement noise, and we then take the median of the seven block averages as the final reported value. This procedure combines the precision of averaging with the robustness of a median, limiting the impact of occasional outliers. For consistency with the deep learning code, all computations are batched with batch size $b = 256$.

The experiment covers:

- Quadratic ROMs with POD sizes between 5 and 250 modes (245 data points).
- Hyper-reduced ROMs over the same POD range, combined with Gauss-point ensemble sizes $m \in \{10, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000\}$, yielding $11 \times 245 = 2695$ configurations.

In total, this corresponds to 171,500 function calls for the quadratic ROM and 1,886,500 for the hyper-reduced models.

Figure 4 plots the results on log–log axes. For the quadratic ROM, the slope of the curve is close to cubic, with an empirical exponent of ≈ 3.25. The slightly super-cubic behavior can be attributed to hierarchical memory effects on the CPU. For hyper-reduction, we illustrate two representative cases ($m = 100$ and $m = 1000$). Their asymptotic slopes are ≈ 0.94 and ≈ 1.13, respectively, confirming the predicted linear complexity with respect to the POD size.

Beyond asymptotics, the point of intersection between the curves is of practical importance. Across all tested Gauss-point sizes, hyper-reduction becomes cheaper than the quadratic ROM at around 50 POD modes. For the case $m = 200$, which is most relevant to our applications, hyper-reduction already outperforms the quadratic ROM from about 10 POD modes onward. At 20 modes, the hyper-reduced

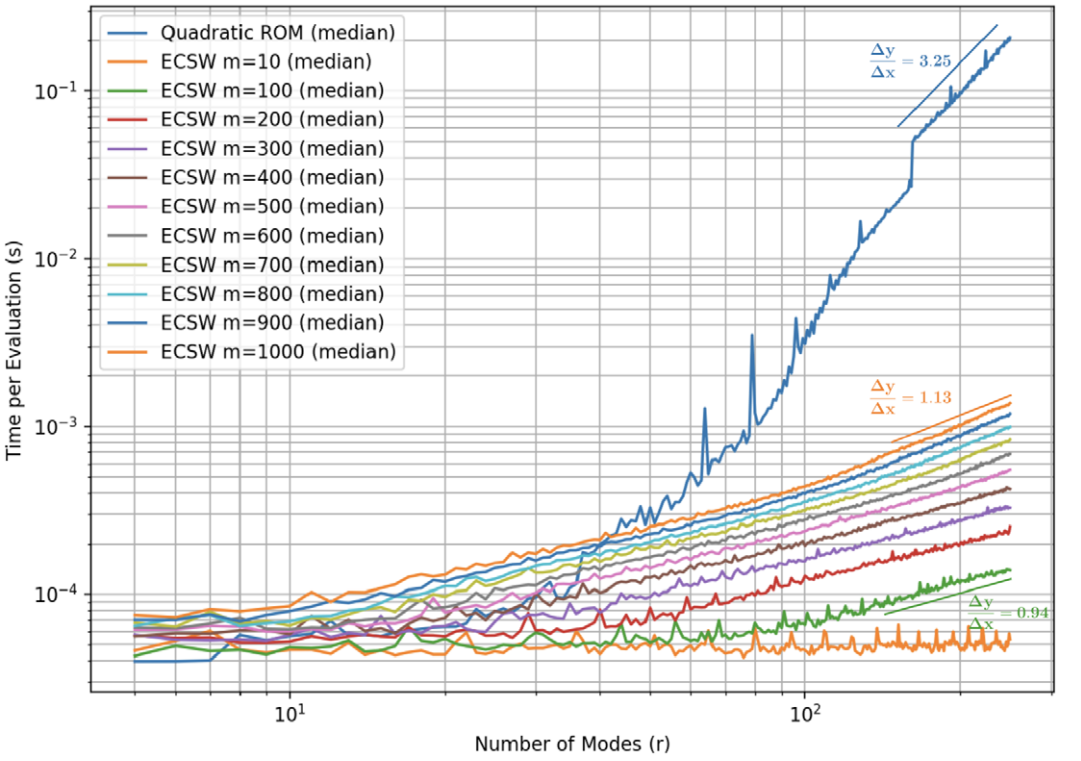


Figure 4. *Empirical wall-clock time (log–log scale) for nonlinear evaluation. Comparison between quadratic ROMs and ECSW hyper-reduced models with varying Gauss-point sizes, executed on CPU hardware.*

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

model requires 5×10^{-5} s per evaluation, compared to 7×10^{-5} s for the quadratic ROM. This crossover behavior agrees with our earlier tests.

These experiments confirm that, on CPU hardware, quadratic ROMs scale cubically with POD size, while ECSW hyper-reduction scales linearly. Although hyper-reduction can be more expensive at very small reduced spaces, its advantage grows rapidly with dimension, making it the more efficient choice for realistic POD sizes.

4.3. Empirical cost scaling and asymptotic behavior running on GPU hardware

As a final test, we run the same experiment with both models now running on our NVIDIA Tesla V100S 32GB VRAM GPU chip. This is the hardware we use to run our hybrid model training and inference. The same hardware where we observed slower run times for hyper-reduction compared to the quadratic ROM.

The test follows the same isolated-evaluation setup as on CPU, except that on GPU we report the median over 100 single evaluations together with the inter-quartile range, since the per-call launch jitter is itself of interest and is better shown directly than hidden inside block averages. The Gauss-point ensemble size grid search is also slightly different, with $m \in \{10,20,50,100,500,1000\}$. This more logarithmic grid-search scale allows us to more clearly see the effects at the extreme ends of the spectrum.

The results for the scaling for the new hardware experiment are very different from the previous ones. Analyzing first the cost of the hyper-reduction models, the cost is nearly constant with respect to both the size of the POD modes and the size of the Gauss-point ensemble, or at least the cost increases only at discrete values.

Concerning the quadratic ROM scaling, initially we see the same near-constant cost scaling with several discrete cost values, but then for models over ~ 80 POD modes the compute time blows up and follows a new cost scaling that seems exponential or at least asymptotically of high polynomial order of around $\mathcal{O}(r^{5.5})$.

These results broadly agree with the timings reported in Section 3.3. In both cases only the nonlinear evaluation itself is timed; what differs is the context surrounding the timed region. During trajectory integration, each right-hand-side evaluation is interleaved with the implicit linear solve and time-stepping bookkeeping, whereas in the present experiment it is called repeatedly in a tight loop. These two settings are not strictly equivalent, so a small difference in absolute cost is to be expected: the 18-mode quadratic ROM evaluates in 138 μs and the 18-mode, 200-Gauss-point hyper-reduced model in 184 μs, both marginally below the trajectory values of 153 and 208 μs. Crucially, the relative gap between the two methods is preserved: 33% here, close to the 36% measured during trajectory integration in Section 3.3.

Figure 5 also reports the experimental variability, plotting the median over 100 evaluations together with the shaded inter-quartile range (IQR), which is visually narrow throughout. The IQR peaks at about 28% of the median for the quadratic ROM (at $r = 5$) and at about 16% for the hyper-reduced model, confirming that the median is a reliable indicator of practical performance.

To ensure that these observations were not artifacts of a particular implementation, we repeated the profiling experiments with both implementations expressed as `torch.nn.Module` objects compiled with `torch.compile`, and also reproduced the benchmark in JAX. In all cases, the relative performance and scaling trends remained unchanged.

Several steps are needed to understand these results: why the hardware scaling appears almost linear at small sizes, and why the cost suddenly explodes at $r = 79$ for the quadratic ROM. We begin by examining how each method can be parallelized, and then turn to memory scaling and GPU hardware effects.

4.3.1. Parallelization of the nonlinearity evaluation

For the quadratic ROM, the lower compute time can be explained by two factors. First, as shown in the operation counts above, at typical sizes (e.g., $r = 20$ POD modes and $m = 200$ Gauss points), the quadratic ROM tensor evaluation requires fewer total operations than hyper-reduction. Second, GPU optimization reduces the effective cost even further.

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

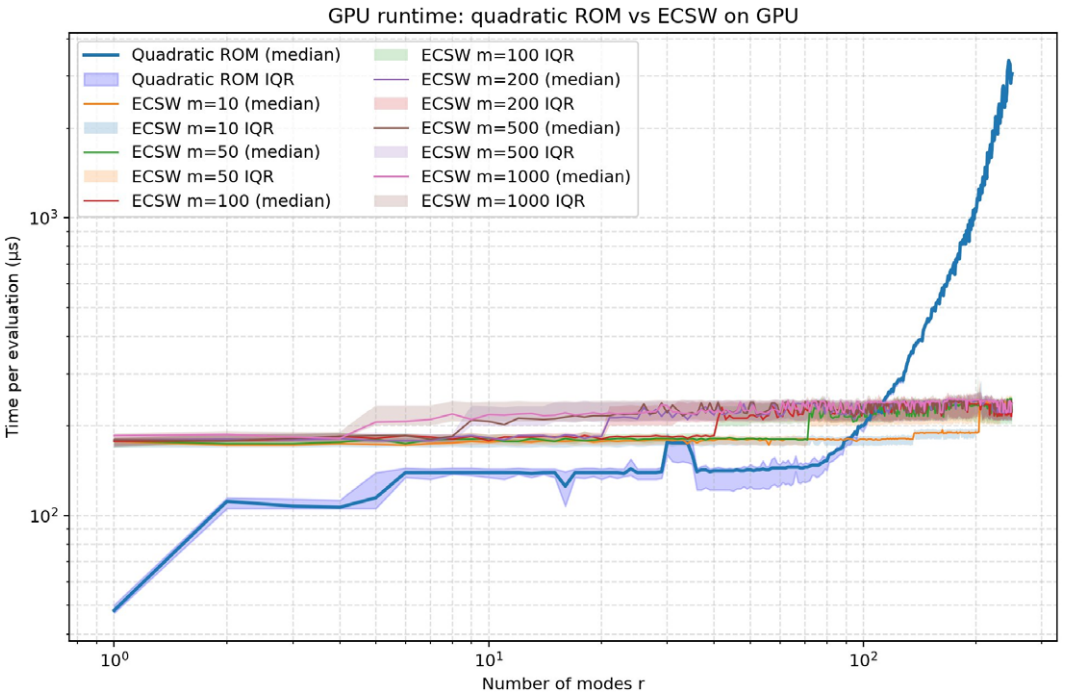


Figure 5. *Empirical wall-clock time (log–log scale) showing the computational-cost scaling of quadratic reduction vs hyper-reduction on GPU hardware. Results are reported as medians with inter-quartile ranges.*

The computation involves essentially two steps. First,

$$T_{ljk} = \sum_{i=1}^{r} N_{ijk} a_{li},$$

followed by

$$f_{lk} = \sum_{j=1}^{r} T_{ljk} a_{lj}.$$

In both steps, the br^2 (first) and br (second) repeated operations are non-sequential and can be executed in parallel. In practice, they are likely handled by a batched general matrix–multiplication (GEMM) kernel. Ignoring bandwidth limits, each step can be treated as a near constant-time operation. Thus, the quadratic ROM reduces to two sequential, highly optimized GPU kernels.

The hyper-reduction is more sequential by design. It can be broken down into three main stages: (1) projection into the empirical Gauss-point basis, (2) point-wise products and additions within this space, and (3) projection back into the POD subspace.

For the projection stage, the matrices can be stacked to exploit GPU parallelism. For example, M_3, M_4, M_5, M_6 are grouped into $M_{\text{large1}} \in \mathbb{R}^{4m \times r}$, and M_7, M_8, M_9, M_{10} into $M_{\text{large2}} \in \mathbb{R}^{4m \times r}$. In practice, we have 2 Gauss-point ensembles of slightly different sizes, so two separate stacks are required. This allows all projections to be evaluated efficiently using the highly optimized `torch.linear` routine (perhaps the highest optimized torch GPU function thanks to its role in neural networks), which can again be considered constant-time for small m and r.

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

Once the m-sized vectors are computed, the element-wise products are trivially parallelizable with compute unified device architecture (CUDA) kernels. The subsequent additions are equally efficient. Together, these operations can be treated as one or perhaps several constant-time extremely fast operations.

Finally, the projection back to POD space is performed by stacking M_1 and M_2 into $M_{\text{large3}} \in \mathbb{R}^{r \times 2m}$ and applying a batched `torch.linear`. This again corresponds to one constant-time operation.

When memory and bandwidth limits are ignored, the quadratic ROM reduces to two sequential constant-time kernels, while the hyper-reduction requires at least three, and typically more, sequential stages. This explains both the near-constant cost observed for small r and why hyper-reduction is consistently about 35% slower in this regime: it involves roughly $3/2$ as many sequential steps.

This sequential-stage argument can be complemented by an arithmetic intensity perspective. The quadratic ROM consists entirely of dense GEMM operations; at typical sizes ($r = 20$, $b = 256$), each contraction achieves roughly 9 FLOP/byte, placing it comfortably in the compute-efficient regime of the GPU. The ECSW GEMM stages (forward projections and back-projection) reach a comparable arithmetic intensity of about 8–9 FLOP/byte. However, between these GEMM stages, ECSW requires element-wise Hadamard products over $b \times m$ vectors, which perform a single floating-point operation per element while reading two operands and writing one result, yielding an arithmetic intensity of approximately 0.08 FLOP/byte—two orders of magnitude lower. Whether these heterogeneous operations are fused by the compiler into a single kernel (as observed with the XLA backend) or dispatched separately, the overall arithmetic intensity of the ECSW evaluation is diluted by the element-wise stages, contributing to its higher wall-clock cost despite comparable GEMM-level performance.

4.3.2. Hardware bandwidth limitations

We now account for hardware limits to explain the near-constant cost at small r and the sharp increase at $r \geq 79$ for the quadratic ROM. The key object is the intermediate tensor $T \in \mathbb{R}^{b \times r \times r}$ formed in the first contraction. Its size is $|T| = 4br^2$ bytes (float32). While $N \in \mathbb{R}^{r \times r \times r}$ is streamed slice-by-slice (working set $\mathcal{O}(r^2)$) and $\mathbf{a}, \mathbf{f} \in \mathbb{R}^{b \times r}$ are tiny, T must be repeatedly updated and then consumed. If T fits entirely in the GPU's L2 cache, all those updates and reads occur on-chip; dynamic random-access memory (DRAM) traffic is dominated by streaming the tensor N slice-by-slice from memory, and wall time appears flat. The threshold is set by L2 capacity S_{L2}:

$$4br^2 \lesssim S_{L2} \quad \Rightarrow \quad r \lesssim \sqrt{\frac{S_{L2}}{4b}}.$$

With $b = 256$ and $S_{L2} \approx 6{,}291{,}456$ B (V100, ~6 MB), this gives $r \lesssim 78.38$, this exactly matches our observed constant cost breaking point. Beyond this point, T no longer resides in L2. Kernels must tile T and perform read–modify–write to DRAM for each i-slice of N, introducing a dominant bandwidth term $\propto br^3$. This is why the cost "cliff" appears precisely around $r = 79$: below it, T reuse is on-chip, and cost is nearly constant; above it, repeated DRAM traffic for T drives the high-order scaling.

4.4. Guidelines for using hyper-reduction to speed up learning and inference in hybrid deep learning models

The initial motivation for adopting hyper-reduction in hybrid deep learning ROM architectures was its reported computational efficiency. While this remains theoretically true, our results show that scaling arguments alone do not justify its use. In practice, the tight coupling between the ROM and the neural network means both must run on the same hardware. Since deep learning gains enormous speed-ups on GPUs, it is natural to place the ROM there as well. The relevant question is therefore: given a target accuracy, POD size, GPU hardware, and the structure of the nonlinearity, does hyper-reduction actually accelerate the hybrid method?

Before giving practical guidelines, it is useful to discuss how the required quadrature support may evolve for more complex flows. ECSW is an empirical approximation on top of the quadratic POD–

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

Galerkin ROM; in general it cannot be more accurate than the underlying ROM, and coincides with it only when a sufficiently rich quadrature rule is retained. At high Reynolds numbers, two scenarios arise.

In the first, all turbulent scales are resolved through DNS. The Kolmogorov barrier (Ahmed and San, 2020) slows the decay of the POD singular spectrum, forcing the retention of a large number of modes r. Since Hernández et al. (2017) show that the number of quadrature points m scales between $\mathcal{O}(r)$ and $\mathcal{O}(r^2)$ for greedy non-negative cubature rules (including ECSW), for a given NNLS residual tolerance the number of retained quadrature points grows substantially with turbulence intensity.

In the second, one retains a moderate reduced basis (possibly larger than in the laminar regime studied here, but not drastically so) and closes the unresolved scales through a turbulence model (Lorenzi et al., 2016; Hijazi et al., 2019), a learned closure, or both. Because fewer scales need to be represented, the required number of POD modes r stays restrained; the same cubature scaling (Hernández et al., 2017) then implies that m also remains moderate, and ECSW preserves its cost advantage.

Our GPU profiling results support this second scenario. Section 4.3 shows that the hyper-reduced cost remains nearly constant for quadrature sizes up to $m \approx 10^3$, well above the values needed in our experiments. For moderately richer bases, the dominant operations reduce to batched $m \times r$ matrix–vector products that better exploit GPU parallelism; the advantage would diminish only when these products become large enough to saturate memory bandwidth or kernel parallelization capacity. These observations remain qualitative projections; dedicated turbulent benchmarks would be needed to confirm them.

From our experiments, we suggest the following:

- If your nonlinearity is more complex than the quadratic convective term, hyper-reduction is the only practical option.
- If your application involves a moderately large reduced basis (on the order of a hundred modes, as may arise in industrial turbulent configurations) and your chosen method resolves all retained POD modes through the physics-based operator (Barnett et al., 2023; Menier et al., 2023), then the quadratic ROM computational cost exceeds the efficient threshold. At this scale, the number of quadrature points required remains manageable, making ECSW computable and the preferred option.
- In contrast, for hybrid ROM–DL approaches where only a small subset of dominant POD modes is evolved through the physics-based operator and the remaining dynamics are captured by a neural correction (Wang et al., 2020; Menier et al., 2023; Lepage et al., 2025), the reduced dimension is intentionally kept very small. In this setting, the quadratic nonlinear term can already be evaluated extremely efficiently on GPU. As a result, the hyper-reduced formulation is less efficient than the standard tensorial ROM in this low-dimensional regime.

5. Conclusion

This work set out to assess the computational relevance of the ECSW hyper-reduction method in modern reduced-order modeling workflows. While hyper-reduction is traditionally introduced to accelerate nonlinear term evaluation, its practical benefit in GPU-accelerated and deep-learning-augmented settings remains unclear. Our objective was therefore not to improve model accuracy, but to rigorously quantify the numerical cost of ECSW and determine under which conditions it remains advantageous compared to standard tensorial ROM formulations.

After validating the implementation on a simple periodic flow configuration of the fluidic pinball at $Re = 30$, we focused exclusively on computational aspects. We derived exact operation counts for both tensorial and hyper-reduced formulations, showing that their theoretical complexities scale respectively as $\mathcal{O}(r^3)$ and $\mathcal{O}(rm)$, where r is the reduced dimension and m the number of retained quadrature points. Extensive numerical experiments on both CPU and GPU hardware confirmed these asymptotic trends but also revealed strong hardware dependencies. On CPUs, hyper-reduction behaves as expected, providing linear scaling and consistent speed-ups even for moderately sized reduced models. On GPUs, however, the picture is more nuanced: for small to medium reduced spaces—typical of hybrid ROM–deep learning

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

architectures—the quadratic tensorial formulation can in fact be faster, owing to its tighter integration with highly optimized batched matrix kernels.

Overall, our results demonstrate that ECSW hyper-reduction remains a powerful tool for CPU-based reduced-order modeling, especially for large reduced systems or higher-order nonlinearities. However, in GPU-based hybrid frameworks where the ROM must coexist with neural-network components, its computational advantage is not guaranteed. The study therefore provides practical guidance: hyper-reduction should be employed when the reduced system is sufficiently large, when nonlinear operators dominate the cost, or when CPU-based execution remains necessary. In contrast, for small latent spaces running on GPU hardware, standard tensorial reduction already offers near-optimal performance.

By clarifying these trade-offs, this work contributes a concrete benchmark for future hybrid ROMs and helps practitioners decide when hyper-reduction genuinely improves computational efficiency.

Data availability statement. The data, code, and materials supporting the findings of this study are openly available on GitHub at the following repository: `nicolaslepagecnam/ECSW-hyper-reduction-for-GPU`. This repository contains all scripts required to reproduce the simulations, reduced-order models, hyper-reduction weights, and GPU performance experiments.

Author contribution. Author contributions follow the CRediT taxonomy (https://www.casrai.org/credit.html). Conceptualization: All authors; Data curation: N.L., D.S.; Final approval: All authors; Formal Analysis: N.L., M.G.C., S.B., C.F., D.S.; Investigation: N.L.; Software: N.L., M.G.C.; Supervision: C.F., S.B., D.S., I.M., N.T.; Validation: N.L., M.G.C.; Visualization: N.L.; Writing – original draft preparation: N.L.; Writing – review and editing: All authors. All authors approved the final manuscript.

Funding statement. This work was supported by the French National Research Agency (ANR) under grant ANR-20-THIA-0002.

Competing interests. The authors declare that they have no known financial, professional, or personal relationships that could be perceived as influencing the research reported in this article.

Ethical standards. The research meets all ethical guidelines, including adherence to the legal requirements of the study country.

Use of Generative AI. Generative AI tools (specifically GPT-5) were used solely for minor proofreading and language refinement during the preparation of this manuscript. All scientific content, analyses, and conclusions are entirely the authors' own, and the authors take full responsibility for the final text.

References

Ahmed SE and San O (2020) Breaking the Kolmogorov barrier in model reduction of fluid flows. *Fluids 5*(1), 26. https://doi.org/10.3390/fluids5010026.

Barnett J, **Farhat C and Maday Y** (2023) Neural-network-augmented projection-based model order reduction for mitigating the Kolmogorov barrier to reducibility. *Journal of Computational Physics 492*, 112420. https://doi.org/10.1016/j.jcp.2023.112420.

Barrault M, **Maday Y**, **Nguyen NC and Patera AT** (2004) *An 'empirical interpolation' method: Application to efficient reduced-basis discretization of partial differential equations*, Comptes Rendus. *Mathématique 339*(9), 667–672.

Carlino MG (2021) *ADER scheme on Overset Grids with Compact Transmission and Hyper-reduction: Application to Incompressible Navier-Stokes Equations*, Theses, Université de Bordeaux, December.

Chapman T, **Avery P**, **Collins P and Farhat C** (2017) Accelerated mesh sampling for the hyper reduction of nonlinear computational models. *International Journal for Numerical Methods in Engineering 109*(12), 1623–1654. https://doi.org/10.1002/nme.5332.

Chaturantabut S and Sorensen DC (2010) Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing 32*(5), 2737–2764. https://doi.org/10.1137/090766498.

Everson R and Sirovich L (1995) Karhunen–Loève procedure for gappy data. *Journal of the Optical Society of America A 12*(8), 1657–1664. https://doi.org/10.1364/JOSAA.12.001657.

Hernández JA, **Caicedo MA and Ferrer A** (2017) Dimensional hyper-reduction of nonlinear finite element models via empirical cubature. *Computer Methods in Applied Mechanics and Engineering 313*, 687–722. https://doi.org/10.1016/j.cma.2016.10.022.

Hijazi S, **Stabile G**, **Mola A and Rozza G** (2019) Data-driven POD-Galerkin reduced order model for turbulent flows. *Journal of Computational Physics 416*, 109513. https://doi.org/10.1016/j.jcp.2020.109513.

Holmes P, **Lumley JL and Berkooz G** (1996) *Turbulence, Coherent Structures, Dynamical Systems and Symmetry, Cambridge Monographs on Mechanics*. Cambridge University Press. https://doi.org/10.1017/CBO9780511622700.

Lawson CL and Hanson RJ (1995) *Solving Least Squares Problems*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Lepage N, **Beneddine S**, **Fiorini C**, **Mortazavi I**, **Sipp D and Thome N** (2025) Hybrid AutoEncoder/Galerkin approach for nonlinear reduced order modelling. *Computers & Fluids 302*, 106811. https://doi.org/10.1016/j.compfluid.2025.106811.

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

Lorenzi S, **Cammi A**, **Luzzi L and Rozza G** (2016) POD-Galerkin method for finite volume approximation of Navier–stokes and RANS equations. *Computer Methods in Applied Mechanics and Engineering 311*, 151–179. https://doi.org/10.1016/j.cma.2016.08.006.

Lumley JL (1979) Computational modeling of turbulent flows. In *Advances in Applied Mechanics*, Vol. *18*. Elsevier, pp. 123–176. https://doi.org/10.1016/S0065-2156(08)70266-7.

Menier E, **Bucci MA**, **Yagoubi M**, **Mathelin L and Schoenauer M** (2023) CD-ROM: Complemented deep reduced order model. *Computer Methods in Applied Mechanics and Engineering 410*, 115985. https://doi.org/10.1016/j.cma.2023.115985.

Mori H (1965) Transport, collective motion, and Brownian motion. *Progress of Theoretical Physics 33*(3), 423–455. https://doi.org/10.1143/PTP.33.423.

Noack BR, **Afanasiev K**, **Morzyński M**, **Tadmor G and Thiele F** (2003) A hierarchy of low-dimensional models for the transient and post-transient cylinder wake. *Journal of Fluid Mechanics 497*, 335–363. https://doi.org/10.1017/S0022112003006694.

Riffaud S, **Bergmann M**, **Farhat C**, **Grimberg S and Iollo A** (2021) The DGDD method for reduced-order modeling of conservation laws. *Journal of Computational Physics 437*, 110336. https://doi.org/10.1016/j.jcp.2021.110336.

Sirovich L (1987) Turbulence and the dynamics of coherent structures. I. Coherent structures. *I. coherent structures, Quarterly of Applied Mathematics 45*(3), 561–571. https://doi.org/10.1090/qam/910462.

Takens F (1981) Detecting strange attractors in turbulence. In Rand D and Young L-S (eds), *Dynamical Systems and Turbulence, Warwick 1980.* Lecture Notes in Mathematics, vol. 898. Berlin, Heidelberg: Springer, pp. 366–381. https://doi.org/10.1007/BFb0091924.

Virtanen P, **Gommers R**, **Oliphant TE**, **Haberland M**, **Reddy T**, **Cournapeau D**, **Burovski E**, **Peterson P**, **Weckesser W**, **Bright J**, **van der Walt SJ**, **Brett M**, **Joshua Wilson KJM**, **Mayorov N**, **Andrew RJN**, **Jones E**, **Kern R**, **Eric Larson CJC**, **Polat İ**, **Feng Y**, **Moore EW**, **VanderPlas J**, **Laxalde D**, **Perktold J**, **Cimrman R**, **Ian Henriksen EAQ**, **Harris CR**, **Archibald AM**, **Ribeiro AH and Pedregosa F** (2020) Paul van Mulbregt, and SciPy 1.0 contributors, *SciPy 1.0: Fundamental algorithms for scientific computing in python*. *Nature Methods 17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2.

Wang Q, **Ripamonti N and Hesthaven JS** (2020) Recurrent neural network closure of parametric POD-Galerkin reduced-order models based on the Mori-Zwanzig formalism. *Journal of Computational Physics 410*, 109402. https://doi.org/10.1016/j.jcp.2020.109402.

Yaghoobi M, **Wu D and Davies ME** (2015) Fast non-negative orthogonal matching pursuit. *IEEE Signal Processing Letters 22*(9), 1229–1233. https://doi.org/10.1109/LSP.2015.2393637.

Zwanzig R (1961) Memory effects in irreversible thermodynamics. *Physics Review 124*(4), 983–992. https://doi.org/10.1103/PhysRev.124.983.

Appendix A. Detailed explanation of our implementation of the ECSW method

A.1. Quadrature rule for FEM and quantities to export

A.1.1. Quadrature in the finite element method

Consider a generic evolution problem in strong form

$$\partial_t u = \mathcal{L}u \quad \text{in}\Omega,$$

where $\mathcal{L}$ is a spatial differential operator. Introducing a test function φ_j and integrating over the domain yields the weak form

$$\int_\Omega \varphi_j \partial_t u d\Omega = \int_\Omega \varphi_j \mathcal{L}u d\Omega, \quad \forall \varphi_j$$

With the finite element approximation $u(x,t) = \sum_{i=1}^n u_i(t)\varphi_i(x)$, we obtain a semi-discrete system

$$\sum_{i=1}^n M_{ji}\dot{u}_i = \sum_{i=1}^n A_{ji}u_i, \quad M_{ji} = \int_\Omega \varphi_i\varphi_j d\Omega, \quad A_{ji} = \int_\Omega (\mathcal{L}\varphi_i)\varphi_j d\Omega.$$

The evaluation of these integrals is performed numerically using a quadrature rule.

A.1.2. Numerical quadrature

Over each triangular element K_e, the integral of a generic function f is approximated by a weighted sum over n_g Gauss points:

$$\int_{K_e} f(x,y)d\Omega \approx \sum_{\ell=1}^{n_g} \omega_{g,\ell} f(x_\ell, y_\ell).$$

The quadrature points (ξ_ℓ, η_ℓ) and weights $\omega_{g,\ell}$ are defined on the reference triangle,

$$T_{\text{ref}} = \{(\xi,\eta) \,|\, \xi \geq 0, \eta \geq 0, \xi + \eta \leq 1\},$$

and mapped to each physical element by

$$(x_\ell, y_\ell) = T_{K_e}(\xi_\ell, \eta_\ell), \quad \omega_\ell = |J_{K_e}|\omega_{g,\ell}.$$

In this work, we employ the 7-point Dunavant rule, which is exact for all polynomials up to degree 5 (see Figure A1).

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

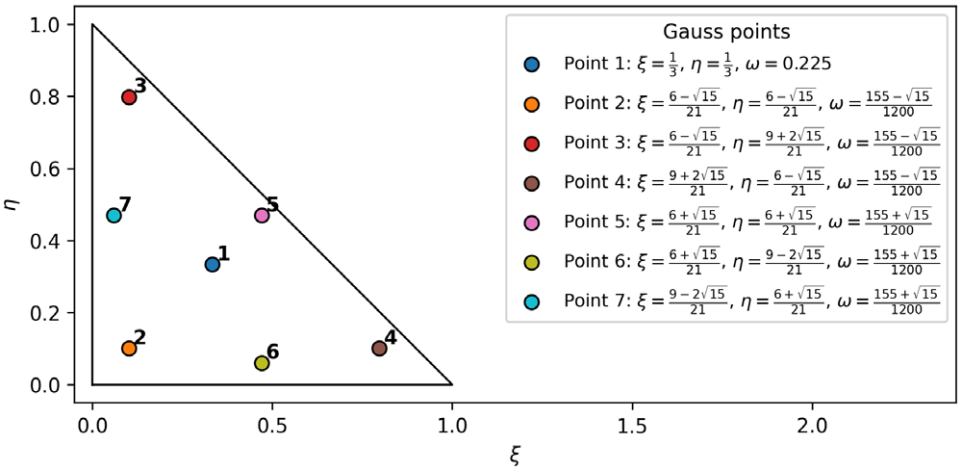


Figure A1. *Triangular reference element: Dunavant 7-point quadrature rule.*

A.1.3. Matrix representation

After applying the quadrature rule and inserting the finite element expansion, each weak-form term can be expressed as a weighted sum of basis evaluations:

$$A_{ji} \approx \sum_{\ell=1}^{\tilde{m}} \omega_\ell (\mathcal{L}\varphi_i)(x_\ell)\varphi_j(x_\ell).$$

Introducing the matrices

$$\Theta = [\varphi_i(x_\ell)]_{i,\ell}, \quad \Theta_{\mathcal{L}} = [(\mathcal{L}\varphi_i)(x_\ell)]_{i,\ell}, \quad W = \mathrm{diag}(\omega_1,\ldots,\omega_{\tilde{m}}),$$

the discrete operator can be written compactly as

$$A = \Theta^{\mathsf{T}} W \Theta_{\mathcal{L}}, \quad \left(\text{and} M = \Theta^{\mathsf{T}} W \Theta\right).$$

A.1.4. Quantities to export from FEM framework

Therefore, to reproduce the full quadrature algebra externally and enable ECSW hyper-reduction, the following data must be extracted from the FEM solver:

- the physical coordinates (x_ℓ, y_ℓ) of all Gauss quadrature points over the mesh,
- the corresponding quadrature weights ω_ℓ forming the diagonal matrix W,
- and the sampling matrices Θ, $\partial_x\Theta$, and $\partial_y\Theta$, containing the basis and derivative evaluations at those points.

These objects fully describe the numerical integration process and form the interface between the FEM solver and the ECSW procedure.

A.2. Description of the term to empirically approximate

A.2.1. Expressing the nonlinearity as a source term in weak form

Since the reduction of the linear system components is straightforward and computationally inexpensive, we focus our application of hyper-reduction exclusively on the nonlinear contributions.

The 2D incompressible Navier–Stokes equations are given by:

$$\partial_t \mathbf{w}_t + (\mathbf{w}_t \cdot \nabla)\mathbf{w}_t + \nabla p_t = \nu \Delta \mathbf{w}_t, \quad \nabla \cdot \mathbf{w}_t = 0, \tag{A.1}$$

where $\mathbf{w}_t = [u_t(x,y,t), v_t(x,y,t)]^{\mathsf{T}}$ is the velocity field and $p_t(x,y,t)$ is the pressure.

After lifting $\mathbf{w}_t(t) = \overline{\mathbf{w}} + \mathbf{w}(t)$, the explicit nonlinear convection term that appears in the equation governing $\mathbf{w}$ reads:

$$\mathbf{f}(\mathbf{w}) = \begin{bmatrix} u\partial_x u + v\partial_y u \\ u\partial_x v + v\partial_y v \end{bmatrix}. \tag{A.2}$$

However, in our code, as is often the case in computational fluid dynamics, this term is treated explicitly. This effectively makes the system linear, as the nonlinear term is now essentially a fixed source term that no longer depends on the unknown:

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

$$\mathbf{f}\left(\mathbf{w}^{t-\Delta t}\right)=\begin{bmatrix} u^{t-\Delta t}\partial_x u^{t-\Delta t}+v^{t-\Delta t}\partial_y u^{t-\Delta t} \\ u^{t-\Delta t}\partial_x v^{t-\Delta t}+v^{t-\Delta t}\partial_y v^{t-\Delta t} \end{bmatrix}. \tag{A.3}$$

A.2.2. Numerical quadrature

We now clarify how this source term is treated in the finite element framework. Since the velocity field has two components, u and v, and each component is associated with its own finite element space, we denote by $\{\varphi_i^u\}_{i=1}^{n_u}$ and $\{\varphi_i^v\}_{i=1}^{n_v}$ the basis functions corresponding to the u- and v-velocity components, respectively. The pressure component p does not appear in the convection term and is therefore excluded from this step.

The explicit nonlinear term $\mathbf{f}\left(\mathbf{w}^{t-\Delta t}\right)$ is projected independently onto the u and v test spaces, resulting in two sets of integrals:

$$f_i^u := \int_\Omega \varphi_i^u(x)\left[u^{t-\Delta t}\partial_x u^{t-\Delta t}+v^{t-\Delta t}\partial_y u^{t-\Delta t}\right]\mathrm{d}x, \quad i=1,\ldots,n_u, \tag{A.4}$$

$$f_i^v := \int_\Omega \varphi_i^v(x)\left[u^{t-\Delta t}\partial_x v^{t-\Delta t}+v^{t-\Delta t}\partial_y v^{t-\Delta t}\right]\mathrm{d}x, \quad i=1,\ldots,n_v. \tag{A.5}$$

These integrals are approximated using Gauss quadrature with $\tilde{m}$ points $\{x_\ell\}_{\ell=1}^{\tilde{m}}$ and weights $\{\omega_\ell\}_{\ell=1}^{\tilde{m}}$:

$$f_i^u \approx \sum_{\ell=1}^{\tilde{m}} \varphi_i^u(x_\ell)\left[u^{t-\Delta t}(x_\ell)\partial_x u^{t-\Delta t}(x_\ell)+v^{t-\Delta t}(x_\ell)\partial_y u^{t-\Delta t}(x_\ell)\right]\omega_\ell, \tag{A.6}$$

$$f_i^v \approx \sum_{\ell=1}^{\tilde{m}} \varphi_i^v(x_\ell)\left[u^{t-\Delta t}(x_\ell)\partial_x v^{t-\Delta t}(x_\ell)+v^{t-\Delta t}(x_\ell)\partial_y v^{t-\Delta t}(x_\ell)\right]\omega_\ell. \tag{A.7}$$

Mathematically, each field is associated with its own finite element space: $H^1(\Omega)$ for velocity components and $L^2(\Omega)$ for pressure. However, in practice, these fields are stored in a single global vector of degrees of freedom that spans the product space of all three variables. This means that the source term vector must match the structure of the full discrete unknown vector, which includes the pressure degrees of freedom—even though they do not receive any contribution from the convection term.

Accordingly, the global source vector $\mathbf{f}\in\mathbb{R}^n$, where $n=n_u+n_v+n_p$, is assembled by interleaving the u and v contributions and inserting zeros at the pressure dof positions:

$$\mathbf{f} := \left[f_1^u\, f_1^v\, 0\, f_2^u\, f_2^v\, 0\, \cdots\, f_{n_{\mathrm{dof}}/3}^u\, f_{n_{\mathrm{dof}}/3}^v\, 0\right]^\top.$$

We will express this finite element quadrature term with the matrices discussed in Appendix A.1.

Given the global finite element vector $\mathbf{w}=[\mathbf{u},\mathbf{v},\mathbf{p}]^\top\in\mathbb{R}^n$, which stacks the degrees of freedom of the streamwise velocity u, cross-stream velocity v, and pressure p, we define the following sampling operators acting on the $\tilde{m}$ Gauss points of the mesh:

- $\Theta_u,\Theta_v\in\mathbb{R}^{\tilde{m}\times n}$: interpolate the discrete velocity fields onto Gauss points, such that $(\Theta_u\mathbf{w})_\ell=u(x_\ell)$ and $(\Theta_v\mathbf{w})_\ell=v(x_\ell)$;
- $\partial_x\Theta_u,\partial_y\Theta_u,\partial_x\Theta_v,\partial_y\Theta_v\in\mathbb{R}^{\tilde{m}\times n}$: provide spatial derivatives of the velocity basis, for example, $(\partial_x\Theta_u\mathbf{w})_\ell=\partial_x u(x_\ell)$ and $\left(\partial_y\Theta_v\mathbf{w}\right)_\ell=\partial_y v(x_\ell)$;
- $W=\mathrm{diag}(\omega_1,\ldots,\omega_{\tilde{m}})\in\mathbb{R}^{\tilde{m}\times\tilde{m}}$: diagonal matrix of Gauss weights used for numerical integration.

Each Θ thus acts as a linear projection from the nodal degrees of freedom to weak form function or derivative evaluations at all quadrature points, filtered by variable component (u, v, or p).

Here is the complete matrix form where $\odot$ is the Hadamard product:

$$\mathbf{f}^u=\Theta_u^\top W\left(\Theta_u\mathbf{w}^{t-\Delta t}\odot\partial_x\Theta_u\mathbf{w}^{t-\Delta t}+\Theta_v\mathbf{w}^{t-\Delta t}\odot\partial_y\Theta_u\mathbf{w}^{t-\Delta t}\right) \tag{A.8}$$

$$\mathbf{f}^v=\Theta_v^\top W\left(\Theta_u\mathbf{w}^{t-\Delta t}\odot\partial_x\Theta_v\mathbf{w}^{t-\Delta t}+\Theta_v\mathbf{w}^{t-\Delta t}\odot\partial_y\Theta_v\mathbf{w}^{t-\Delta t}\right). \tag{A.9}$$

A.2.3. Reduced weak form in vector-valued FEM

We now introduce reduced-order modeling by constructing a low-dimensional approximation space from full-order snapshot data. Select $r\ll n$ POD modes $\{\boldsymbol{\phi}_j\}_{j=1}^r$ and define the reduced approximation as:

$$\mathbf{w}_h(x,t)\approx\sum_{j=1}^r\boldsymbol{\phi}_j(x)a_j(t), \tag{A.10}$$

where $\mathbf{a}(t)\in\mathbb{R}^r$ denotes the vector of reduced coefficients. Each POD mode $\boldsymbol{\phi}_j:\Omega\to\mathbb{R}^3$ is a vector-valued function containing degrees of freedom from all three finite element fields: the two velocity components and the pressure. We thus write:

$$\boldsymbol{\phi}_j(x)=\begin{bmatrix}\phi_j^u(x)\\ \phi_j^v(x)\\ \phi_j^p(x)\end{bmatrix}\in V_u\times V_v\times Q:=\mathbb{V}_h.$$

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

In other words, the reduced approximation $\mathbf{w}_h(x,t)$ lives in a reduced subspace of the full-order space:

$$\mathbb{V}_r := \operatorname{span}\{\boldsymbol{\phi}_1,\ldots,\boldsymbol{\phi}_r\} \subset \mathbb{V}_h = V_u \times V_v \times Q.$$

While the FE weak form treats each component separately—testing the equations against φ_i^u, φ_i^v, or φ_i^p depending on the PDE being discretized—the reduced-order model uses global modes $\boldsymbol{\phi}_j$ as both trial and test functions. Therefore, we define the reduced test space to be the same as the reduced trial space:

$$\mathcal{V}_r := \operatorname{span}\{\boldsymbol{\phi}_1,\ldots,\boldsymbol{\phi}_r\}, \quad \text{for } i = 1,\ldots,r.$$

Projecting the nonlinear term onto the reduced test space gives the reduced weak form:

$$\int_\Omega \boldsymbol{\phi}_i(x) \cdot \mathbf{f}(\mathbf{w}_h(x,t))\,\mathrm{d}x = 0, \quad i = 1,\ldots,r. \tag{A.11}$$

Using the reduced approximation $\mathbf{w}_h(x,t) \approx \sum_{j=1}^{r} \boldsymbol{\phi}_j(x) a_j(t)$ and applying Gauss quadrature, the i-th component of the nonlinear vector becomes:

$$f_i^{\text{red}} \approx \sum_{\ell=1}^{\tilde{m}} \boldsymbol{\phi}_i(x_\ell) \cdot \mathbf{f}\left(\sum_{j=1}^{r} \boldsymbol{\phi}_j(x_\ell) a_j\right) \omega_\ell, \quad i = 1,\ldots,r. \tag{A.12}$$

Using the matrix notations from Subsection A.2.2, we now express the entire reduced nonlinear term compactly using the POD basis and Gauss-point sampling matrices. We recall that the nonlinear convection term is evaluated explicitly using the velocity fields only, so only the u and v components of the reduced modes and sampling matrices appear.

We obtain:

$$\begin{aligned}\mathbf{f}_{\text{red}} = &\underbrace{\Phi^\top \Theta_u^\top W}_{\mathbb{R}^{r\times\tilde{m}}} \left(\underbrace{\Theta_u \Phi\, \mathbf{a}}_{\mathbb{R}^{\tilde{m}\times r}} \odot \underbrace{\partial_x \Theta_u \Phi\, \mathbf{a}}_{\mathbb{R}^{\tilde{m}\times r}} + \underbrace{\Theta_v \Phi\, \mathbf{a}}_{\mathbb{R}^{\tilde{m}\times r}} \odot \underbrace{\partial_y \Theta_u \Phi\, \mathbf{a}}_{\mathbb{R}^{\tilde{m}\times r}} \right) \\ &+ \underbrace{\Phi^\top \Theta_v^\top W}_{\mathbb{R}^{r\times\tilde{m}}} \left(\underbrace{\Theta_u \Phi\, \mathbf{a}}_{\mathbb{R}^{\tilde{m}\times r}} \odot \underbrace{\partial_x \Theta_v \Phi\, \mathbf{a}}_{\mathbb{R}^{\tilde{m}\times r}} + \underbrace{\Theta_v \Phi\, \mathbf{a}}_{\mathbb{R}^{\tilde{m}\times r}} \odot \underbrace{\partial_y \Theta_v \Phi\, \mathbf{a}}_{\mathbb{R}^{\tilde{m}\times r}} \right),\end{aligned} \tag{A.13}$$

This expression is now entirely written in terms of matrix–vector operations involving the reduced basis, the FEM interpolation matrices at Gauss points, and the reduced coordinates $\mathbf{a}$. It is therefore suitable for further acceleration using hyper-reduction techniques such as ECSW, which will be introduced in Section A.3.

A.3. Hyper-Reduction Targets

ECSW hyper-reduction acts directly on the discrete inner product defining the quadratic nonlinear term. By replacing the full quadrature matrix W with a sparse surrogate, supported only on a limited set of retained Gauss points, the cost of evaluating the nonlinear operator is drastically reduced. In practice, sparsifying the weighted inner product modifies three categories of quantities:

- Nonlinear function evaluations at Gauss points, obtained from the POD state through Θ_u, Θ_v, $\partial_x\Theta_u$, $\partial_y\Theta_u$, $\partial_x\Theta_v$, $\partial_y\Theta_v$.
- The quadrature weights, where its ECSW-compressed version replaces the full diagonal matrix W, supported only on the retained points.
- The reduced projection operators mapping nonlinear Gauss-point contributions back to the reduced space, such as $\Phi^\top\Theta_u^\top W$ and $\Phi^\top\Theta_v^\top W$, which now only involve the retained quadrature points and their modified weights.

Once hyper-reduction is applied, evaluating the nonlinear term requires accessing these quantities only at the retained Gauss points, yielding a significant reduction in online computational cost. Moreover, this sparsified structure preserves Galerkin consistency and leverages precomputations to eliminate any operations scaling with the full-order dimension n during the online phase.

A.4. Construction of the NNLS problems for 2D hyper-reduction

To reduce the computational cost of evaluating the explicit nonlinear convection term online, we apply the Energy-Conserving Sampling and Weighting (ECSW) method to sparsify the quadrature points and determine optimal empirical weights. As shown previously, the reduced nonlinear term involves matrix–vector operations combining the POD basis Φ, the FEM sampling matrices at Gauss points (e.g., Θ_u, $\partial_x\Theta_u$, etc.), and the reduced coordinates $\mathbf{a}$. To efficiently approximate this reduced nonlinear term, we now formulate two independent non-negative least squares (NNLS) problems targeting the two separate vector contributions to the nonlinear source term.

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press

A.4.1. Snapshot-based reformulation of the nonlinear term

Let $\mathbf{a}^{(k)} \in \mathbb{R}^r$ denote the reduced coefficients of snapshot $k = 1, \ldots, N_{\text{snap}}$. For each snapshot, we evaluate the reduced nonlinear term:

$$\mathbf{f}_{\text{red}}^{(k)} = \Phi^\top \Theta_u^\top W \mathbf{f}_{m,u}^{(k)} + \Phi^\top \Theta_v^\top W \mathbf{f}_{m,v}^{(k)} \in \mathbb{R}^r, \tag{A.14}$$

where the intermediate Gauss-point contributions $\mathbf{f}_{m,u}^{(k)}, \mathbf{f}_{m,v}^{(k)} \in \mathbb{R}^{\tilde{m}}$ are defined component-wise as:

$$\mathbf{f}_{m,u}^{(k)} = \left(\Theta_u \Phi \mathbf{a}^{(k)}\right) \odot \left(\partial_x \Theta_u \Phi \mathbf{a}^{(k)}\right) + \left(\Theta_v \Phi \mathbf{a}^{(k)}\right) \odot \left(\partial_y \Theta_u \Phi \mathbf{a}^{(k)}\right), \tag{A.15}$$

$$\mathbf{f}_{m,v}^{(k)} = \left(\Theta_u \Phi \mathbf{a}^{(k)}\right) \odot \left(\partial_x \Theta_v \Phi \mathbf{a}^{(k)}\right) + \left(\Theta_v \Phi \mathbf{a}^{(k)}\right) \odot \left(\partial_y \Theta_v \Phi \mathbf{a}^{(k)}\right). \tag{A.16}$$

A.4.2. Energy-conserving sampling and weighting via non-negative least squares

We now approximate the matrix–vector product $\Phi^\top \Theta_u^\top W \mathbf{f}_{m,u}^{(k)}$ using a sparse set of quadrature weights $\boldsymbol{\omega}_u \in \mathbb{R}^{\tilde{m}}$ by solving the following empirical approximation:

$$\mathbf{f}_{\text{red},u}^{(k)} \approx \Phi^\top \Theta_u^\top \operatorname{diag}\left(\mathbf{f}_{m,u}^{(k)}\right) \boldsymbol{\omega}_u. \tag{A.17}$$

To determine the optimal sparse weights, we collect all N_{snap} snapshots into a single least-squares problem. Define the stacked system:

$$\mathcal{A}_u = \begin{bmatrix} \Phi^\top \Theta_u^\top \operatorname{diag}\left(\mathbf{f}_{m,u}^{(1)}\right) \\ \vdots \\ \Phi^\top \Theta_u^\top \operatorname{diag}\left(\mathbf{f}_{m,u}^{(N_{\text{snap}})}\right) \end{bmatrix} \in \mathbb{R}^{rN_{\text{snap}} \times \tilde{m}}, \quad \mathbf{b}_u = \begin{bmatrix} \mathbf{f}_{\text{red},u}^{(1)} \\ \vdots \\ \mathbf{f}_{\text{red},u}^{(N_{\text{snap}})} \end{bmatrix} \in \mathbb{R}^{rN_{\text{snap}}}, \tag{A.18}$$

and solve the NNLS problem:

$$\min_{\boldsymbol{\omega}_u \geq 0} \|\mathcal{A}_u \boldsymbol{\omega}_u - \mathbf{b}_u\|_2^2. \tag{A.19}$$

An identical procedure is applied independently to the second contribution, corresponding to the v-component:

$$\mathbf{f}_{\text{red},v}^{(k)} \approx \Phi^\top \Theta_v^\top \operatorname{diag}\left(\mathbf{f}_{m,v}^{(k)}\right) \boldsymbol{\omega}_v, \tag{A.20}$$

with

$$\mathcal{A}_v = \begin{bmatrix} \Phi^\top \Theta_v^\top \operatorname{diag}\left(\mathbf{f}_{m,v}^{(1)}\right) \\ \vdots \\ \Phi^\top \Theta_v^\top \operatorname{diag}\left(\mathbf{f}_{m,v}^{(N_{\text{snap}})}\right) \end{bmatrix}, \quad \mathbf{b}_v = \begin{bmatrix} \mathbf{f}_{\text{red},v}^{(1)} \\ \vdots \\ \mathbf{f}_{\text{red},v}^{(N_{\text{snap}})} \end{bmatrix}, \tag{A.21}$$

and the corresponding NNLS problem:

$$\min_{\boldsymbol{\omega}_v \geq 0} \|\mathcal{A}_v \boldsymbol{\omega}_v - \mathbf{b}_v\|_2^2. \tag{A.22}$$

A.4.3. Remarks

Note that these two NNLS problems are solved once during the offline stage and yield sparse weight vectors $\boldsymbol{\omega}_u$ and $\boldsymbol{\omega}_v$ with many zero entries. The associated non-zero quadrature points constitute the empirical integration rule. This approach preserves Galerkin consistency and ensures that only a reduced set of quadrature points is needed during the online phase, significantly decreasing the cost of evaluating the nonlinear term. The result is a sparse, yet accurate approximation of the reduced nonlinear operator that scales independently of the full-order DOF size n.

Cite this article: Lepage N, Beneddine S, Carlino MG, Fiorini C, Mortazavi I, Sipp D and Thome N (2026). GPU-accelerated ECSW hyper-reduction: implementation and performance evaluation. *Data-Centric Engineering*, 7, e48. doi:10.1017/dce.2026.10079

https://doi.org/10.1017/dce.2026.10079 Published online by Cambridge University Press